Indice

1 Introduzione

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

1.1 Descrizione del problema

Il sistema da modellare deve consentire l'organizzazione degli eventi e la storicizzazione dei progetti realizzati dai vari team.
Nello specifico, la piattaforma permette la registrazione degli utenti, la formazione delle squadre e la pubblicazione degli elaborati. Ogni team lavora separatamente 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 i voti e genera la classifica finale. In questo modo, il sistema permette non solo di gestire e monitorare le attività in corso, ma anche di storicizzare i risultati ottenuti in ogni edizione passata.

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

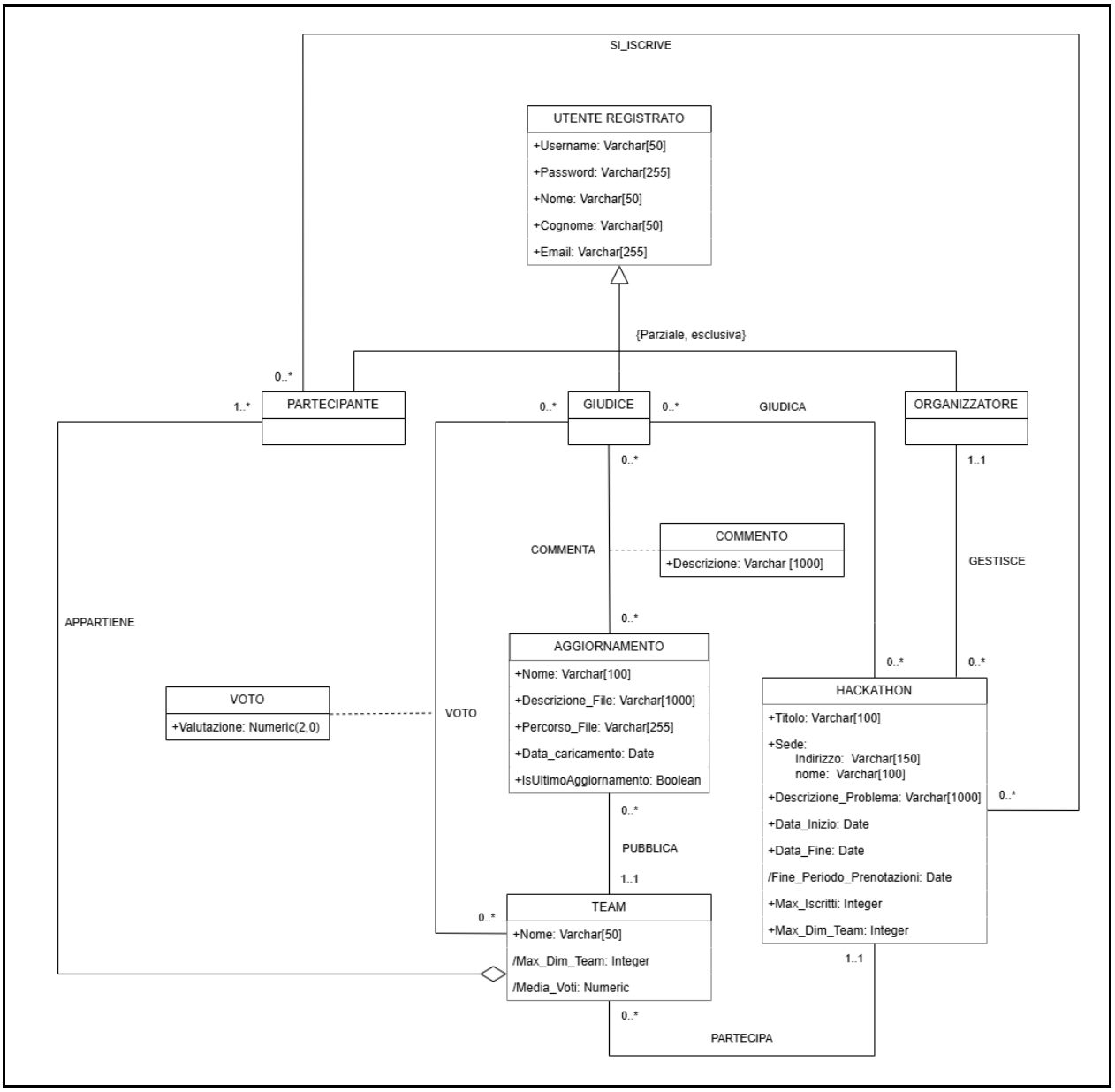

Figura 1: Class Diagram UML

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 così 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 fine periodo prenotazioni. 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** e **Organizzatore**) vengono eliminate e assorbite nell'entità padre **Utente Registrato** (rinominata per brevità in **Utente** nello schema finale).
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 mantenendo le rispettive associazioni (che verranno tradotte in **tabelle di associazione** nel modello logico). Di conse-

guenza, il ruolo assunto dall'utente non viene memorizzato staticamente, ma viene **dedotto dinamicamente**, verificando la presenza dell'istanza all'interno di queste tabelle.

2.3 Class Diagram ristrutturato

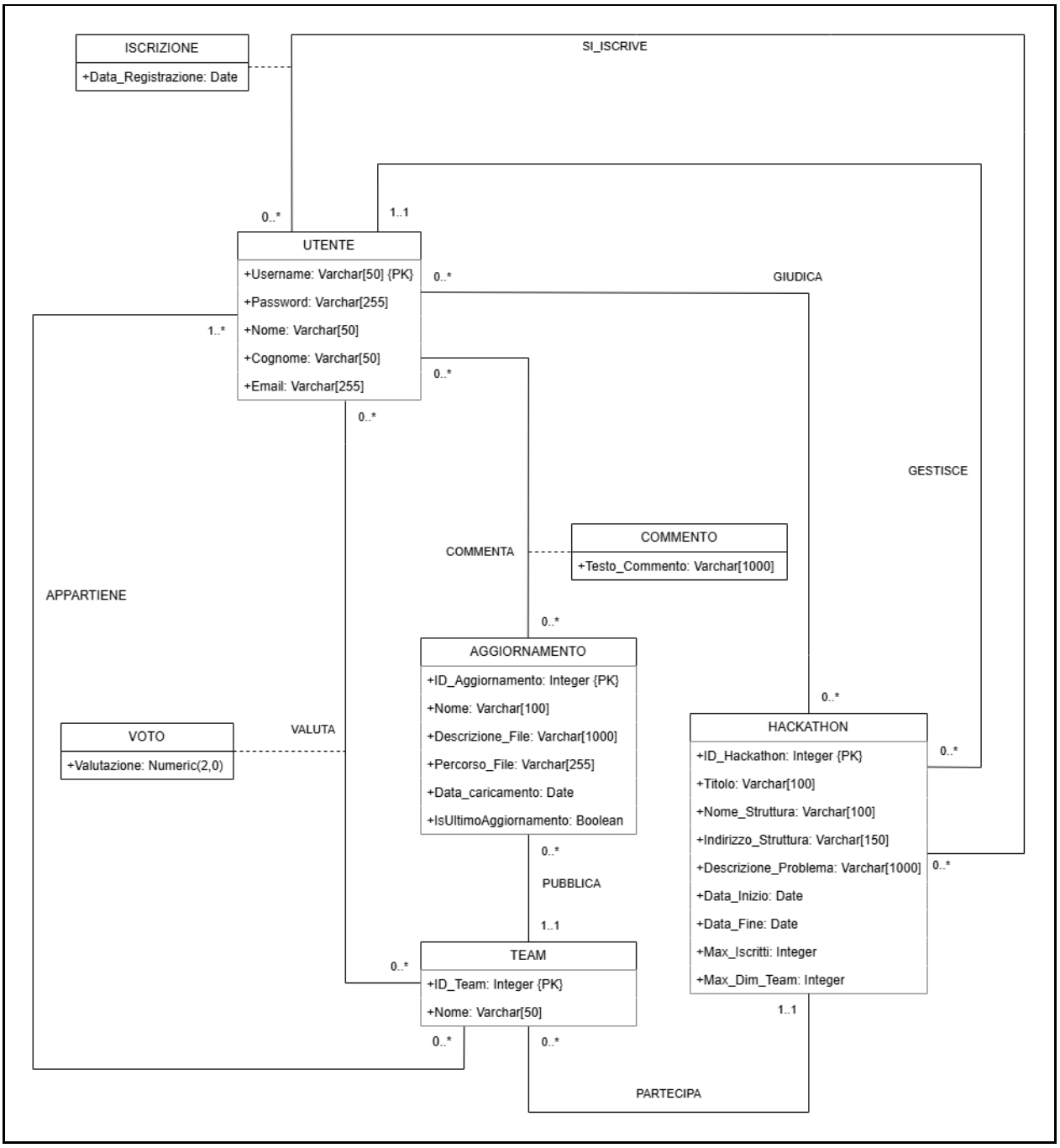

Figura 2: Class Diagram ristrutturato UML

2.4 Dizionario delle Classi

Classe	Descrizione	Attributi
UTENTE	Descrive ciascun utente iscritto alla piattaforma. Quest'ultimo può 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, che 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 utente (giudice), per un aggiornamento.	• **Testo_commento** (varchar(1000)): Contenuto del commento scritto dall'utente (giudice).
VOTO	Descrive un voto assegnato da un utente (giudice) ad un team a fine hackathon.	• **Valutazione**(numeric(2,0)): Voto assegnato da un utente (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 (partecipante) ad un determinato team dell'hackathon.	• **UTENTE[1..*]**: Indica un utente (partecipante) che si unisce ad un team. • **TEAM[0..*]**: Indica un team al quale un utente (partecipante) 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]**: Indica un team che carica un aggiornamento. • **AGGIORNAMENTO[0..*]**: Indica un aggiornamento che viene caricato da un team.
COMMENTA	Esprime la possibilità, da parte di un utente (giudice), di aggiungere un commento ad un aggiornamento pubblicato da un team.	• **UTENTE[0..*]**: Indica un utente (giudice) che commenta un aggiornamento di un team. • **AGGIORNAMENTO[0..*]**: Indica un aggiornamento che riceve un commento da parte di un utente (giudice). **Classe di associazione:** COMMENTO **Attributi:** Testo_commento: Indica il testo del commento di un utente (giudice) per un determinato aggiornamento.
VALUTA	Esprime la possibilità da parte di un utente (giudice) di valutare un team.	• **UTENTE[0..*]**: Indica un utente (giudice) che valuta un team. • **TEAM[0..*]**: Indica un team che viene valutato da un utente (giudice). **Classe di associazione:** VOTO **Attributi:** Valutazione: Indica la valutazione che un utente (giudice) attribuisce ad un team.

2.6 Dizionario dei vincoli

Nome	Descrizione
Validità nome	Il nome di un utente registrato alla piattaforma deve essere unicamente composto da caratteri.
Validità cognome	Il cognome di un utente registrato alla piattaforma deve essere unicamente composto da caratteri.
Validità email	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.
Immutabilità dell'associazione team-hackathon	Un team verrà creato esclusivamente per una certa competizione, non potrà passare da un hackathon ad un altro.
Autorizzazione commenti	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 partecipazione hackathon attiva	Partecipanti, giudici o organizzatori possono essere associati ad un singolo hackathon attivo. Il sistema conserverà lo storico completo delle partecipazioni passate per tutti i ruoli.
Termine massimo 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 né esser modificato e né esser eliminato.
Rilascio traccia problema	I giudici di un hackathon potranno pubblicare la traccia del problema da affrontare, ciò dovrà avvenire solo il giorno di inizio hackathon.

Validazione 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.
Validazione modifiche 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.
Chiusura 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.
Validazione inserimento e modifica 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.
Validazione dimensioni team e numero partecipanti	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.
Immutabilità dell'organizzatore	Un organizzatore non può ritirarsi dall'organizzazione di un hackathon e non può esser trasferire l'organizzazione ad un altro utente.

3 Progettazione Logica

In questo capitolo verrà analizzata 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.

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. Nelle tabelle di associazione (**Iscrizione, Giudica, Appartenenza, Voto** e **Commento**), in cui la chiave primaria è composta da chiavi esterne, gli attributi presentano la doppia sottolineatura vista la loro **duplice natura** (chiave primaria e chiave esterna).

- **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, ID_organizzatore)

  ID_organizzatore → Utente.username

- **Giudica** (ID_giudice, ID_hackathon)

  ID_giudice → Utente.username ; ID_hackathon → Hackathon.ID_hackathon

- **Team** (ID_team, nome, ID_hackathon)

  ID_hackathon → Hackathon.ID_hackathon

- **Aggiornamento** (ID_aggiornamento, nome, descrizione_file, percorso_file, data_caricamento, isUltimoAggiornamento, ID_team)

  ID_team → Team.ID_team

- **Voto** (ID_giudice, ID_team, valutazione)

  ID_giudice → Utente.username ; ID_team → Team.ID_team

- **Commento** (ID_giudice,ID_aggiornamento, testo_commento)

  ID_giudice → Utente.username ; ID_aggiornamento → Aggiornamento.ID_aggiornamento

- **Iscrizione** (ID_utente, ID_hackathon, data_registrazione)

  ID_utente → Utente.username ; ID_hackathon → Hackathon.ID_hackathon

- **Appartenenza** (ID_utente, ID_team)

  ID_utente → Utente.username ; ID_team → 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 ISCRIZIONE_UTENTE_HACKATHON (
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. )
```

4.1.4 Definizione della tabella GIUDICA-HACKATHON

```
1.  --Definizione tabella per assegnare il ruolo di giudice ad un
       utente per una certa hackathon.
2.  CREATE TABLE GIUDICA_HACKATHON (
3.      id_giudice VARCHAR(50) NOT NULL,
4.      id_hackathon INTEGER NOT NULL,
5.      CONSTRAINT giudicahackathon_pkey PRIMARY KEY (id_giudice,
           id_hackathon),
6.      CONSTRAINT fk_giudice FOREIGN KEY (id_giudice)
7.          REFERENCES UTENTE(username) MATCH SIMPLE
8.          ON UPDATE CASCADE
9.          ON DELETE CASCADE,
10.     CONSTRAINT fk_hackathon FOREIGN KEY (id_hackathon)
11.         REFERENCES HACKATHON(id_hackathon) MATCH SIMPLE
12.         ON UPDATE CASCADE
13.         ON DELETE CASCADE
14. )
```

4.1.5 Definizione della tabella TEAM

```
1.  --Definizione tabella per la creazione di un team.
2.  CREATE TABLE TEAM (
3.      id_team INTEGER PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
4.      nome VARCHAR(50) NOT NULL,
5.      id_hackathon INTEGER NOT NULL,
6.      CONSTRAINT fk_hackathon FOREIGN KEY (id_hackathon)
7.          REFERENCES HACKATHON(id_hackathon) MATCH SIMPLE
8.          ON UPDATE CASCADE
9.          ON DELETE CASCADE
10. )
```

4.1.6 Definizione della tabella APPARTENENZA-UTENTE-TEAM

```
1.  --Definizione tabella per l'adesione di un partecipante ad un team
      .
2.  CREATE TABLE APPARTENENZA_UTENTE_TEAM (
3.      id_utente VARCHAR(50) NOT NULL,
4.      id_team INTEGER NOT NULL,
5.      CONSTRAINT appartenenzautenteteam_pkey PRIMARY KEY (id_utente,
          id_team),
6.      CONSTRAINT fk_team FOREIGN KEY (id_team)
7.          REFERENCES TEAM(id_team) MATCH SIMPLE
8.          ON UPDATE CASCADE
9.          ON DELETE CASCADE,
10.     CONSTRAINT fk_utente FOREIGN KEY (id_utente)
11.         REFERENCES UTENTE(username) MATCH SIMPLE
12.         ON UPDATE CASCADE
13.         ON DELETE CASCADE
14. )
```

4.1.7 Definizione della tabella AGGIORNAMENTO

```
1.  --Definizione tabella per la creazione di un aggiornamento.
2.  CREATE TABLE AGGIORNAMENTO (
3.      id_aggiornamento INTEGER PRIMARY KEY GENERATED ALWAYS AS
           IDENTITY,
4.      nome VARCHAR(100) NOT NULL,
5.      descrizione_file VARCHAR(1000) NOT NULL,
6.      percorso_file VARCHAR(255) NOT NULL,
7.      id_team INTEGER NOT NULL,
8.      data_caricamento DATE NOT NULL,
9.      isultimoaggiornamentocaricato BOOLEAN NOT NULL,
10.     CONSTRAINT fk_team FOREIGN KEY (id_team)
11.         REFERENCES TEAM(id_team) MATCH SIMPLE
12.         ON UPDATE CASCADE
13.         ON DELETE CASCADE
14. )
```

4.1.8 Definizione della tabella COMMENTO

```
1.  --Definizione tabella per la creazione di un commento.
2.  CREATE TABLE COMMENTO (
3.      testo_commento VARCHAR(1000) NOT NULL,
4.      id_giudice VARCHAR(50) NOT NULL,
5.      id_aggiornamento INTEGER NOT NULL,
6.      CONSTRAINT commento_pkey PRIMARY KEY (id_giudice,
           id_aggiornamento),
7.      CONSTRAINT fk_aggiornamento FOREIGN KEY (id_aggiornamento)
8.          REFERENCES AGGIORNAMENTO(id_aggiornamento) MATCH SIMPLE
9.          ON UPDATE CASCADE
10.         ON DELETE CASCADE,
11.     CONSTRAINT fk_giudice FOREIGN KEY (id_giudice)
12.         REFERENCES UTENTE(username) MATCH SIMPLE
13.         ON UPDATE CASCADE
14.         ON DELETE CASCADE
15. )
```

4.1.9 Definizione della tabella VOTO

```
1.  --Definizione tabella per la creazione di un voto.
2.  CREATE TABLE VOTO (
3.      valutazione NUMERIC(2,0) NOT NULL
4.      CHECK (valutazione >= 0 AND valutazione <= 10),
5.      id_giudice VARCHAR(50) NOT NULL,
6.      id_team INTEGER NOT NULL,
7.      CONSTRAINT voto_pkey PRIMARY KEY (id_giudice, id_team),
8.      CONSTRAINT fk_giudice FOREIGN KEY (id_giudice)
9.          REFERENCES UTENTE(username) MATCH SIMPLE
10.         ON UPDATE CASCADE
11.         ON DELETE CASCADE,
12.     CONSTRAINT fk_team FOREIGN KEY (id_team)
13.         REFERENCES TEAM(id_team) MATCH SIMPLE
14.         ON UPDATE CASCADE
15.         ON DELETE CASCADE
16. )
```

4.2 Implementazione dei vincoli

Nella sezione seguente vengono illustrate le implementazioni dei vincoli di integrità aggiuntivi, non esplicitamente dichiarati durante la fase di creazione delle tabelle.

4.2.1 Vincolo controllo ruolo partecipanti

Questo vincolo stabilisce che un utente non può iscriversi come partecipante se è già coinvolto in altre competizioni in corso, indipendentemente dal fatto che sia un giudice o che le stia gestendo come organizzatore.

```
1.  CREATE OR REPLACE FUNCTION check_ruoli_partecipante()
2.  RETURNS TRIGGER AS $$
3.  DECLARE
4.      isOrganizzatore INTEGER;
5.      check_giudice INTEGER;
6.  BEGIN
7.      SELECT COUNT(id_organizzatore) INTO isOrganizzatore
8.      FROM hackathon
9.      WHERE NEW.id_utente = id_organizzatore AND data_fine >
           CURRENT_DATE;
10.
11.     IF (isOrganizzatore > 0) THEN
12.         RAISE EXCEPTION 'Questo utente è un organizzatore di un
               hackathon attiva, non può essere iscritto come
               partecipante';
13.     END IF;
14.
15.     SELECT COUNT(*) INTO check_giudice
16.     FROM giudica_Hackathon gh
17.     JOIN hackathon h ON h.id_hackathon = gh.id_hackathon
18.     WHERE gh.id_giudice = NEW.id_utente AND h.data_fine >
           CURRENT_DATE;
19.
20.     IF (check_giudice > 0) THEN
21.         RAISE EXCEPTION 'Questo utente è già un giudice di un
               hackathon attiva, non puo essere iscritto come
               partecipante';
22.     END IF;
23.     RETURN NEW;
24. END;
25. $$LANGUAGE plpgsql;
26.
27. CREATE OR REPLACE TRIGGER trg_check_ruoli_partecipante
28. BEFORE INSERT ON iscrizione_utente_hackathon
29. FOR EACH ROW
30. EXECUTE FUNCTION check_ruoli_partecipante();
```

4.2.2 Vincolo validazione assegnazione giudici

Questo vincolo stabilisce che un utente non può essere nominato giudice se è già coinvolto in altre competizioni in corso, indipendentemente dal fatto che vi partecipi come concorrente o che le stia gestendo come organizzatore.

```
1.  CREATE OR REPLACE FUNCTION check_ruoli_giudice()
2.  RETURNS TRIGGER AS $$
3.  DECLARE
4.      check_partecipante INTEGER;
5.      isOrganizzatore INTEGER;
6.  BEGIN
7.      SELECT COUNT(*) INTO check_partecipante
8.      FROM iscrizione_Utente_Hackathon iuh
9.      JOIN hackathon h ON h.id_hackathon = iuh.id_hackathon
10.     WHERE iuh.id_utente = NEW.id_giudice AND h.data_fine >
           CURRENT_DATE;
11.
12.     IF (check_partecipante > 0) THEN
13.         RAISE EXCEPTION 'Questo utente è già iscritto come
               partecipante ad un hackathon attiva, non puo essere
               giudice';
14.     END IF;
15.
16.     SELECT COUNT(id_organizzatore) INTO isOrganizzatore
17.     FROM hackathon
18.     WHERE NEW.id_giudice = id_organizzatore AND data_fine >
           CURRENT_DATE;
19.
20.     IF (isOrganizzatore > 0) THEN
21.         RAISE EXCEPTION 'Questo utente è organizzatore di un
               hackathon attiva, non può essere iscritto come giudice'
               ;
22.     END IF;
23.
24.     RETURN NEW;
25. END;
26. $$LANGUAGE plpgsql;
27.
28. CREATE OR REPLACE TRIGGER trg_check_ruoli_giudice
29. BEFORE INSERT ON giudica_hackathon
30. FOR EACH ROW
31. EXECUTE FUNCTION check_ruoli_giudice()
```

4.2.3 Vincolo validazione creazione hackathon

Questo vincolo applica una serie di controlli al momento della creazione di un hackathon. In primo luogo, l'evento deve esser annunciato con almeno una settimana di anticipo e deve avere una durata di almeno quattordici giorni. In secondo luogo, un utente non può fondare una nuova hackathon se è già impegnato in un'altra competizione attiva, sia che vi figuri come organizzatore, sia come giudice o partecipante. Infine, non permetterà l'inserimento della traccia alla creazione.

```
1.  CREATE OR REPLACE FUNCTION check_creazione_hackathon()
2.  RETURNS TRIGGER AS $$
3.  DECLARE
4.      controllo INTEGER;
5.      check_giudice INTEGER;
6.      check_partecipante INTEGER;
7.  BEGIN
8.      IF(NEW.data_inizio < (CURRENT_DATE + 7)) THEN
9.          RAISE EXCEPTION 'La data inizio della hackathon deve
               essere almeno a 7 giorni di distanza dalla data odierna
               ';
10.     END IF;
11.
12.     IF(NEW.data_fine < (NEW.data_inizio + 14)) THEN
13.         RAISE EXCEPTION 'La data di fine hackathon deve essere
               almeno a 14 giorni di distanza dalla data di inizio';
14.     END IF;
15.
16.     SELECT COUNT(*) INTO controllo
17.     FROM Hackathon
18.     WHERE ID_organizzatore = NEW.ID_organizzatore
19.     and data_fine > CURRENT_DATE;
20.
21.     IF(controllo>0) THEN
22.         RAISE EXCEPTION 'Questo utente gestisce già un hackathon';
23.     END IF;
24.
25.     SELECT COUNT(*) INTO check_giudice
26.     FROM giudica_Hackathon gh
27.     JOIN hackathon h ON h.id_hackathon = gh.id_hackathon
28.     WHERE NEW.id_organizzatore = gh.id_giudice AND h.data_fine >
           CURRENT_DATE;
29.
30.     IF(check_giudice>0) THEN
31.         RAISE EXCEPTION 'Questo utente è un giudice di un
               hackathon attiva, non può essere organizzatore';
32.     END IF;
```

```
33.
34.     SELECT COUNT(*) INTO check_partecipante
35.     FROM iscrizione_Utente_Hackathon iuh
36.     JOIN hackathon h ON h.id_hackathon = iuh.id_hackathon
37.     WHERE NEW.id_organizzatore = iuh.id_utente AND h.data_fine >
          CURRENT_DATE;
38.
39.     IF(check_partecipante > 0) THEN
40.         RAISE EXCEPTION 'Questo utente è un partecipante di un
              hackathon attiva, non può essere organizzatore';
41.     END IF;
42.
43.     NEW.descrizione_problema = 'La traccia verrà pubblicata il
          giorno di inizio Hackathon';
44.     RETURN NEW;
45. END;
46. $$LANGUAGE plpgsql;
47.
48. CREATE OR REPLACE TRIGGER trg_check_creazione_hackathon
49. BEFORE INSERT ON hackathon
50. FOR EACH ROW
51. EXECUTE FUNCTION check_creazione_hackathon()
```

4.2.4 Vincolo validazione creazione team

Questo vincolo agisce in fase di inserimento di un nuovo team, verificando innanzitutto che l'hackathon di destinazione esista realmente nel sistema. Successivamente, applica due controlli temporali: impedisce la creazione di team per hackathon già concluse e, soprattutto, fa rispettare la scadenza del periodo di iscrizioni bloccando la formazione di nuove squadre a meno di due giorni dall'inizio dell'evento.

```
1.  CREATE OR REPLACE FUNCTION check_creazione_team()
2.  RETURNS TRIGGER AS $$
3.  DECLARE
4.      check_esistenza_hackathon INTEGER;
5.      check_dataInizio DATE;
6.      check_dataFine DATE;
7.  BEGIN
8.      SELECT COUNT(*) INTO check_esistenza_hackathon
9.      FROM Hackathon
10.     WHERE id_hackathon = NEW.id_hackathon;
11.
12.     IF(check_esistenza_hackathon = 0) THEN
13.         RAISE EXCEPTION 'La hackathon inserita non è presente nel
              database';
14.     END IF;
```

```
15.
16.     SELECT data_inizio, data_fine INTO check_dataInizio,
            check_dataFine
17.     FROM Hackathon
18.     WHERE id_hackathon = NEW.id_hackathon;
19.
20.     IF (CURRENT_DATE >= check_dataFine) THEN
21.         RAISE EXCEPTION 'Questa hackathon è terminata, non è
                possibile creare altri team';
22.     END IF;
23.
24.     IF (CURRENT_DATE > (check_dataInizio-2)) THEN
25.         RAISE EXCEPTION 'Superato termine massimo di iscrizione
                utenti/team per questa hackathon';
26.     END IF;
27.
28.     RETURN NEW;
29. END;
30. $$LANGUAGE plpgsql;
31.
32. CREATE OR REPLACE TRIGGER trg_check_creazione_team
33. BEFORE INSERT ON team
34. FOR EACH ROW
35. EXECUTE FUNCTION check_creazione_team()
```

4.2.5 Vincolo validazione iscrizione all'hackathon

Lo scopo del seguente vincolo farà si che, prima di confermare l'iscrizione, il sistema registrerà automaticamente la data corrente. Successivamente, verifica due condizioni fondamentali: la prima assicura che l'utente si stia iscrivendo con il giusto preavviso, bloccando le richieste che avvengono a meno di due giorni dall'inizio della competizione. La seconda condizione protegge il limite di capienza dell'evento, impedendo ulteriori iscrizioni se è già stato raggiunto il numero massimo di partecipanti stabilito per quell'hackathon.

```
1.  CREATE OR REPLACE FUNCTION check_iscrizione_utente()
2.  RETURNS TRIGGER AS $$
3.  DECLARE
4.      check_date Date;
5.      conteggioUtenti INTEGER;
6.      maxIscrittiHackathon INTEGER;
7.  BEGIN
8.      SELECT h.data_inizio, h.max_iscritti into check_date,
            maxIscrittiHackathon
9.      FROM Hackathon h
10.     WHERE h.id_hackathon = NEW.id_hackathon;
11.
```

```
12.      NEW.data_registrazione := CURRENT_DATE;
13.      IF(New.data_registrazione > (check_date - 2)) THEN
14.          RAISE EXCEPTION 'Superato termine massimo di iscrizione
                per questa hackathon';
15.      END IF;
16.
17.      SELECT COUNT(id_utente) INTO conteggioUtenti
18.      FROM iscrizione_Utente_Hackathon
19.      WHERE id_hackathon = NEW.id_hackathon;
20.
21.      IF(conteggioUtenti = maxIscrittiHackathon) THEN
22.          RAISE EXCEPTION 'Questo Hackathon ha raggiunto il numero
                massimo di iscritti';
23.      END IF;
24.
25.      RETURN NEW;
26.  END;
27.  $$LANGUAGE plpgsql;
28.
29.  CREATE OR REPLACE TRIGGER trg_check_iscrizione_utente
30.  BEFORE INSERT ON iscrizione_utente_hackathon
31.  FOR EACH ROW
32.  EXECUTE FUNCTION public.check_iscrizione_utente()
```

4.2.6 Vincolo validazione inserimento commenti

Questo vincolo impedisce due scenari non consentiti: in primo luogo, evita che vengano aggiunti nuovi commenti ad hackathon già concluse. In secondo luogo, impedisce ad utenti non registrati come giudici per quella competizione di andare ad inserire commenti. Se una di queste due regole viene violata, il database interrompe l'operazione restituendo un'eccezione esplicativa.

```
1.  CREATE OR REPLACE FUNCTION check_permesso_commento()
2.  RETURNS TRIGGER AS $$
3.  DECLARE
4.      idHackathon INTEGER;
5.      check_giudice INTEGER;
6.      dataFine DATE;
7.  BEGIN
8.      SELECT h.id_hackathon, h.data_fine INTO idHackathon, dataFine
9.      FROM aggiornamento a
10.     JOIN team t ON a.id_team = t.id_team
11.     JOIN hackathon h ON t.id_hackathon = h.id_hackathon
12.     WHERE a.id_aggiornamento = NEW.id_aggiornamento;
13.
14.     IF (dataFine <= CURRENT_DATE) THEN
```

```
15.          RAISE EXCEPTION 'Hackathon terminata, non è possibile
                inserire nuovi commenti';
16.      END IF;
17.
18.      SELECT COUNT(*) INTO check_giudice
19.      FROM giudica_Hackathon
20.      WHERE id_giudice = NEW.id_giudice
21.          AND id_hackathon = idHackathon;
22.
23.      IF (check_giudice = 0) THEN
24.          RAISE EXCEPTION 'Solo i giudici possono scrivere commenti
                agli aggiornamenti dei TEAM';
25.      END IF;
26.
27.      RETURN NEW;
28.  END;
29.  $$LANGUAGE plpgsql;
30.
31.  CREATE OR REPLACE TRIGGER trg_check_permesso_commento
32.  BEFORE INSERT ON commento
33.  FOR EACH ROW
34.  EXECUTE FUNCTION check_permesso_commento()
```

4.2.7 Vincolo validazione aggiornamenti

Questo vincolo assicura che un team possa caricare aggiornamenti solo mentre l'hackathon è in corso. Inoltre, registra automaticamente la data odierna e contrassegna la nuova consegna come 'ultimo aggiornamento', disattivando contemporaneamente questo stato per tutte le consegne precedenti dello stesso team.

```
1.  CREATE OR REPLACE FUNCTION check_dataandset_ultimoaggiornamento()
2.  RETURNS TRIGGER AS $$
3.  DECLARE
4.      dataFineHackathon date;
5.      dataInizioHackathon date;
6.  BEGIN
7.      NEW.data_caricamento := CURRENT_DATE;
8.      IF (NEW.data_caricamento <> CURRENT_DATE) THEN
9.          RAISE NOTICE 'Attenzione: La data caricamento non può
                essere diversa da quella odierna. Data odierna
                impostata come data di caricamento.';
10.     END IF;
11.
12.     SELECT h.data_fine, h.data_inizio INTO dataFineHackathon,
            dataInizioHackathon
13.     FROM hackathon h
```

```
14.      JOIN team t ON t.id_hackathon = h.id_hackathon
15.      WHERE t.id_team = NEW.id_team;
16.
17.      IF(dataInizioHackathon > NEW.data_caricamento)THEN
18.          RAISE EXCEPTION 'Questa Hackathon non è ancora iniziata,
                non è possibile caricare nuovi aggiornamenti';
19.      END IF;
20.
21.      IF(NEW.data_caricamento >= dataFineHackathon) THEN
22.          RAISE EXCEPTION 'Questa Hackathon è terminata, non è
                possibile caricare nuovi aggiornamenti';
23.      END IF;
24.
25.      UPDATE aggiornamento
26.      SET isUltimoAggiornamentoCaricato = FALSE
27.      WHERE id_team = NEW.id_team AND isUltimoAggiornamentoCaricato
            = TRUE;
28.
29.      NEW.isUltimoAggiornamentoCaricato := TRUE;
30.
31.      RETURN NEW;
32.  END;
33.  $$LANGUAGE plpgsql;
34.
35.  CREATE OR REPLACE TRIGGER trg_check_dataandset_ultimoaggiornamento
36.  BEFORE INSERT ON aggiornamento
37.  FOR EACH ROW
38.  EXECUTE FUNCTION check_dataandset_ultimoaggiornamento()
```

4.2.8 Vincolo validazione inserimento voti

Questo vincolo garantisce che i voti vengano inseriti solo ad hackathon terminata e solo dai giudici assegnati a quella specifica competizione. Inoltre, assegna automaticamente la valutazione '0' d'ufficio ai team che non hanno mai caricato alcun elaborato, avvisando tramite una notifica.

```
1.  CREATE OR REPLACE FUNCTION check_inserimento_voto()
2.  RETURNS TRIGGER AS $$
3.  DECLARE
4.      checkDate date;
5.      idHackathonTeam INTEGER;
6.      checkHackathonUguale INTEGER;
7.      check_aggiornamentiTeam INTEGER;
8.  BEGIN
9.      SELECT h.id_hackathon, h.data_fine INTO idHackathonTeam,
            checkDate
```

```
10.      FROM team t
11.      JOIN hackathon h ON t.id_hackathon = h.id_hackathon
12.      WHERE t.id_team = NEW.id_team;
13.
14.      IF(checkDate > CURRENT_DATE) THEN
15.          RAISE EXCEPTION 'Hackathon ancora in corso (Un giudice può
                 valutare un team soltanto ad hackathon conclusa)';
16.      END IF;
17.
18.      SELECT COUNT (*) INTO checkHackathonUguale
19.      FROM giudica_hackathon
20.      WHERE id_giudice = NEW.id_giudice AND id_hackathon =
            idHackathonTeam;
21.
22.      IF(checkHackathonUguale = 0) THEN
23.          RAISE EXCEPTION 'Il giudice ed il team non appartengono
                alla stessa Hackathon';
24.      END IF;
25.
26.      SELECT COUNT(*) INTO check_aggiornamentiTeam
27.      FROM aggiornamento
28.      WHERE id_team = NEW.id_team;
29.
30.      IF(check_aggiornamentiTeam = 0) THEN
31.          NEW.valutazione = 0;
32.          RAISE NOTICE 'Attenzione: Il team non ha caricato alcun
                elaborato. Voto impostato a 0.';
33.      END IF;
34.
35.      RETURN NEW;
36.  END;
37.  $$LANGUAGE plpgsql;
38.
39.  CREATE OR REPLACE TRIGGER trg_check_inserimento_voto
40.  BEFORE INSERT ON voto
41.  FOR EACH ROW
42.  EXECUTE FUNCTION check_inserimento_voto()
```

4.2.9 Vincolo validazione appartenenza team

Questo vincolo stabilisce che un utente possa unirsi a un team solo se è già iscritto alla relativa hackathon, se il team non è al completo e se vengono rispettate le scadenze temporali (almeno 2 giorni prima del l'inizio); inoltre, impedisce all'utente di partecipare se è già membro di un'altra squadra in un evento in corso.

```
1.  CREATE OR REPLACE FUNCTION
       check_inserimento_appartenenzautenteteam()
2.  RETURNS TRIGGER AS $$
3.  DECLARE
4.      conteggioUtenti INTEGER;
5.      maxIscrittiTeam INTEGER;
6.      idHackathonCorrente INTEGER;
7.      checkIscrizione INTEGER;
8.      hDataInizio DATE; hDataFine DATE;
9.      checkUtenteImpegnato INTEGER;
10. BEGIN
11.     SELECT h.max_dim_team, h.id_hackathon, h.data_inizio, h.
           data_fine INTO maxIscrittiTeam, idHackathonCorrente,
           hDataInizio, hDataFine
12.     FROM hackathon h
13.     JOIN team t ON t.id_hackathon = h.id_hackathon
14.     WHERE t.id_team = NEW.id_team;
15.
16.     SELECT COUNT(*) INTO checkIscrizione
17.     FROM iscrizione_Utente_Hackathon
18.     WHERE id_utente = NEW.id_utente AND
19.           id_hackathon = idHackathonCorrente;
20.
21.     IF(hDataFine <= CURRENT_DATE) THEN
22.         RAISE EXCEPTION 'Hackathon terminata';
23.     END IF;
24.
25.     IF(hDataInizio < CURRENT_DATE + 2) THEN
26.         RAISE EXCEPTION 'Termine massimo per iscrizione ad un team
               superato';
27.     END IF;
28.
29.     IF(checkIscrizione = 0) THEN
30.         RAISE EXCEPTION 'Utente non iscritto a questa Hackathon';
31.     END IF;
32.
33.     SELECT COUNT(id_utente) INTO conteggioUtenti
34.     FROM appartenenza_Utente_Team
35.     WHERE id_team = NEW.id_team;
```

```
36.
37.     IF(conteggioUtenti = maxIscrittiTeam) THEN
38.         RAISE EXCEPTION 'Questo Team ha raggiunto il numero
               massimo di membri';
39.     END IF;
40.
41.     SELECT COUNT(*) INTO checkUtenteImpegnato
42.     FROM appartenenza_Utente_Team aut
43.     JOIN team t ON aut.id_team = t.id_team
44.     JOIN hackathon h ON t.id_hackathon = h.id_hackathon
45.     WHERE h.data_fine > CURRENT_DATE
46.             AND aut.id_utente = NEW.id_utente;
47.
48.     IF(checkUtenteImpegnato > 0) THEN
49.         RAISE EXCEPTION 'Questo utente fa parte di un team di cui
               la hackathon è ancora in corso';
50.     END IF;
51.
52.     RETURN NEW;
53. END;
54. $$LANGUAGE plpgsql;
55.
56. CREATE OR REPLACE TRIGGER
       trg_check_inserimento_appartenenzautenteteam
57. BEFORE INSERT ON appartenenza_utente_team
58. FOR EACH ROW
59. EXECUTE FUNCTION check_inserimento_appartenenzautenteteam()
```

4.2.10 Vincolo blocco modifiche/eliminazioni aggiornamento

Questo vincolo stabilisce che: in un aggiornamento, una volta caricato, possa esser modificato solo lo stato di ultimo aggiornamento, e che un aggiornamento non può esser eliminato.

```
1.  CREATE OR REPLACE FUNCTION blocco_updateordelete_aggiornamento()
2.  RETURNS TRIGGER AS $$
3.  BEGIN
4.      IF(TG_OP = 'UPDATE') THEN
5.          IF(NEW.isUltimoaggiornamentocaricato <> OLD.
               isUltimoAggiornamentocaricato) THEN
6.              RETURN NEW;
7.          END IF;
8.          RAISE EXCEPTION 'Operazione negata: gli aggiornamenti non
               possono essere modificati una volta inviati. Creane uno
               nuovo.';
9.      END IF;
10.
```

```
11.     IF(TG_OP = 'DELETE') THEN
12.         RAISE EXCEPTION 'Operazione negata: gli aggiornamenti non
               possono essere eliminati.';
13.     END IF;
14. END;
15. $$LANGUAGE plpgsql;
16.
17. CREATE OR REPLACE TRIGGER trg_blocco_updateordelete_aggiornamento
18. BEFORE DELETE OR UPDATE ON aggiornamento
19. FOR EACH ROW
20. EXECUTE FUNCTION blocco_updateordelete_aggiornamento()
```

4.2.11 Vincolo validazione cambio/abbandono team

Questo vincolo regola i trasferimenti e le uscite dalle squadre: permette a un utente di cambiare team solo all'interno della stessa hackathon, verso una squadra non al completo e rigorosamente entro le scadenze (almeno 2 giorni prima dell'inizio); inoltre, impedisce qualsiasi modifica o rimozione di partecipanti per le hackathon già concluse.

```
1.  CREATE OR REPLACE FUNCTION
       check_updateordelete_appartenenzautenteteam()
2.  RETURNS TRIGGER AS $$
3.  DECLARE
4.      hackathon_old INTEGER; hackathon_new INTEGER;
5.      checkDataInizio Date; checkDataFine Date;
6.      maxIscrittiTeam INTEGER; conteggioUtenti INTEGER;
7.  BEGIN
8.      IF(TG_OP = 'UPDATE') THEN
9.          IF(NEW.id_utente <> OLD.id_utente) THEN
10.             RAISE EXCEPTION 'Gli utenti non possono essere
                   modificati. Operazione Bloccata.';
11.         END IF;
12.
13.         SELECT id_hackathon INTO hackathon_old
14.         FROM team
15.         WHERE id_team = OLD.id_team;
16.
17.         SELECT h.id_hackathon, h.data_inizio, h.max_dim_team, h.
               data_fine
18.         INTO hackathon_new, checkDataInizio, maxIscrittiTeam,
               checkDataFine
19.         FROM team t
20.         JOIN hackathon h ON t.id_hackathon = h.id_hackathon
21.         WHERE t.id_team = NEW.id_team;
22.
23.         IF (hackathon_old <> hackathon_new) THEN
```

```
24.             RAISE EXCEPTION 'La hackathon del nuovo team deve
                    essere la stessa del team precedente. Operazione
                    Bloccata.';
25.         END IF;
26.
27.         IF (checkDataFine <= CURRENT_DATE) THEN
28.             RAISE EXCEPTION 'Hackathon terminata, non è possibile
                    apportare modifiche ad un hackathon archiviata.';
29.         END IF;
30.
31.         IF (checkDataInizio < (CURRENT_DATE+2)) THEN
32.             RAISE EXCEPTION 'Termine massimo delle iscrizioni
                    superato, Operazione Bloccata';
33.         END IF;
34.
35.         SELECT COUNT(id_utente) INTO conteggioUtenti
36.         FROM appartenenza_Utente_Team
37.         WHERE id_team = NEW.id_team;
38.
39.         IF(conteggioUtenti = maxIscrittiTeam) THEN
40.             RAISE EXCEPTION 'Il nuovo team scelto ha raggiunto il
                    numero massimo di membri, Operazione Bloccata';
41.         END IF;
42.         RETURN NEW;
43.     END IF;
44.     IF(TG_OP = 'DELETE') THEN
45.         SELECT h.data_fine INTO checkDataFine
46.         FROM team t
47.         JOIN hackathon h ON t.id_hackathon = h.id_hackathon
48.         WHERE t.id_team = OLD.id_team;
49.
50.         IF (checkDataFine <= CURRENT_DATE) THEN
51.             RAISE EXCEPTION 'Non è possibile eliminare un
                    partecipante di un team per hackathon archiviati';
52.         END IF;
53.         RETURN OLD;
54.     END IF;
55. END;
56. $$LANGUAGE plpgsql;
57. CREATE OR REPLACE TRIGGER
       trg_check_updateordelete_appartenenzautenteteam
58.     BEFORE DELETE OR UPDATE ON appartenenza_utente_team
59.     FOR EACH ROW
60.     EXECUTE FUNCTION check_updateordelete_appartenenzautenteteam()
```

4.2.12 Vincolo validazione modifica/eliminazione commenti

Questo vincolo regola le modifiche e le eliminazioni dei commenti: impedisce qualsiasi modifica o cancellazione se l'hackathon associata è già terminata; inoltre, in caso di modifica a evento in corso, vieta rigorosamente di riassegnare il commento a un altro giudice o a un elaborato diverso.

```
1.  CREATE OR REPLACE FUNCTION check_deleteorupdate_commento()
2.  RETURNS TRIGGER AS $$
3.  DECLARE
4.      DataFine date;
5.  BEGIN
6.      SELECT h.data_fine INTO DataFine
7.      FROM hackathon h
8.      JOIN team t ON h.id_hackathon = t.id_hackathon
9.      JOIN aggiornamento a ON t.id_team = a.id_team
10.     WHERE a.id_aggiornamento = OLD.id_aggiornamento;
11.
12.     IF(TG_OP = 'DELETE') THEN
13.         IF (DataFine <= CURRENT_DATE) THEN
14.             RAISE EXCEPTION 'Non è possibile eliminare un commento
                    di un hackathon terminato.';
15.         END IF;
16.         RETURN OLD;
17.     END IF;
18.
19.     IF(TG_OP = 'UPDATE') THEN
20.         IF(OLD.id_giudice <> NEW.id_giudice OR OLD.
               id_aggiornamento <> NEW.id_aggiornamento) THEN
21.             RAISE EXCEPTION 'Operazione negata: Non è possibile
                    modificare il giudice/aggiornamento di un commento'
                    ;
22.         END IF;
23.         IF(DataFine <= CURRENT_DATE) THEN
24.             RAISE EXCEPTION 'Operazione negata: Non è possibile
                    modificare un commento ad hackathon conclusa';
25.         END IF;
26.         RETURN NEW;
27.     END IF;
28. END;
29. $$LANGUAGE plpgsql;
30.
31. CREATE OR REPLACE TRIGGER trg_check_deleteorupdate_commento
32. BEFORE DELETE OR UPDATE ON commento
33. FOR EACH ROW
34. EXECUTE FUNCTION check_deleteorupdate_commento()
```

4.2.13 Vincolo blocco modifiche/eliminazione giudici

Questo vincolo rende le assegnazioni dei giudici immutabili (bloccando del tutto le operazioni di aggiornamento) e regola le rimozioni: impedisce di eliminare un giudice dall'incarico se l'hackathon associata è già terminata.

```
1.  CREATE OR REPLACE FUNCTION check_updateordelete_giudicahackathon()
2.  RETURNS TRIGGER AS $$
3.  DECLARE
4.      DataFine date;
5.  BEGIN
6.      IF(TG_OP = 'UPDATE') THEN
7.          RAISE EXCEPTION 'Operazione negata: Non è possibile
               modificare i dati';
8.      END IF;
9.
10.     IF (TG_OP = 'DELETE') THEN
11.         SELECT data_fine INTO DataFine
12.         FROM hackathon
13.         WHERE id_hackathon = OLD.id_hackathon;
14.
15.         IF(DataFine <= CURRENT_DATE) THEN
16.             RAISE EXCEPTION 'Non è possibile eliminare un giudice
                   di una hackathon terminata ed archiviata.';
17.         END IF;
18.
19.         RETURN OLD;
20.     END IF;
21. END;
22. $$LANGUAGE plpgsql;
23.
24. CREATE OR REPLACE TRIGGER
       trg_check_updateordelete_giudicahackathon
25. BEFORE DELETE OR UPDATE ON giudica_hackathon
26. FOR EACH ROW
27. EXECUTE FUNCTION check_updateordelete_giudicahackathon()
```

4.2.14 Vincolo validazione modifiche/eliminazione hackathon

Questo vincolo: impedisce qualsiasi modifica ad eventi già conclusi e vieta il trasferimento del ruolo di organizzatore. Regola inoltre la pubblicazione della traccia (bloccandola prima della data di inizio) e le modifiche alle date, consentite solo a evento non ancora iniziato e nel rispetto dei termini di preavviso (7 giorni) e durata (14 giorni). Infine, permette la cancellazione dell'evento solo ed esclusivamente se non è ancora iniziato.

```
1.  CREATE OR REPLACE FUNCTION check_updateordelete_hackathon()
2.  RETURNS TRIGGER AS $$
3.  BEGIN
4.      IF(TG_OP = 'UPDATE') THEN
5.          IF(OLD.data_fine <= CURRENT_DATE ) THEN
6.              RAISE EXCEPTION 'Questo hackathon è concluso. Non è
                    possibile apportare modifiche';
7.          END IF;
8.
9.          IF(NEW.descrizione_problema <> OLD.descrizione_problema
                AND OLD.data_inizio > CURRENT_DATE) THEN
10.             RAISE EXCEPTION 'Non è possibile pubblicare la traccia
                    del problema prima della data di inizio';
11.         END IF;
12.
13.         IF (NEW.id_organizzatore <> OLD.id_organizzatore) THEN
14.             RAISE EXCEPTION 'Non è permesso trasferire il ruolo di
                    organizzatore di questa hackathon a un altro
                    utente.';
15.         END IF;
16.
17.         IF (NEW.data_inizio <> OLD.data_inizio OR NEW.data_fine <>
                OLD.data_fine) THEN
18.             IF (OLD.data_inizio <= CURRENT_DATE) THEN
19.                 RAISE EXCEPTION 'Non è possibile modificare le
                        date di un hackathon già iniziato.';
20.             END IF;
21.             IF (NEW.data_inizio <> OLD.data_inizio AND NEW.
                    data_inizio < (CURRENT_DATE + 7)) THEN
22.                  RAISE EXCEPTION 'La nuova data di inizio deve
                        essere ad almeno 7 giorni di distanza da
                        oggi.';
23.             END IF;
24.             IF (NEW.data_fine < (NEW.data_inizio + 14)) THEN
25.                 RAISE EXCEPTION 'La durata dell''hackathon deve
                        rimanere di almeno 14 giorni.';
26.             END IF;
27.         END IF;
```

```
28.          RETURN NEW;
29.      END IF;
30.
31.      IF(TG_OP = 'DELETE') THEN
32.          IF(OLD.data_inizio <= CURRENT_DATE) THEN
33.              RAISE EXCEPTION 'Non è possibile eliminare un
                    Hackathon in corso oppure terminato ed archiviato';
34.          END IF;
35.          RETURN OLD;
36.      END IF;
37.  END;
38.  $$LANGUAGE plpgsql;
39.
40.  CREATE OR REPLACE TRIGGER trg_check_updateordelete_hackathon
41.  BEFORE DELETE OR UPDATE ON hackathon
42.  FOR EACH ROW
43.  EXECUTE FUNCTION check_updateordelete_hackathon();
```

4.2.15 Vincolo: gestione annullamento iscrizione hackathon

Questo vincolo gestisce l'annullamento dell'iscrizione di un partecipante: prima di tutto, impedisce la rimozione se l'hackathon è già terminata, tutelando. Se invece l'evento non è ancora concluso, l'utente può disiscriversi ed il trigger provvederà a rimuovere automaticamente l'utente anche dalla sua eventuale squadra, garantendo che non rimangano dati orfani o incoerenti nel database.

```
1.  CREATE OR REPLACE FUNCTION delete_partecipante_team()
2.  RETURNS TRIGGER AS $$
3.  DECLARE
4.      DataFine Date;
5.  BEGIN
6.      SELECT data_fine INTO DataFine
7.      FROM Hackathon
8.      WHERE id_hackathon = OLD.id_hackathon;
9.
10.     IF(DataFine <= CURRENT_DATE) THEN
11.         RAISE EXCEPTION 'Non è possibile eliminare un utente da un
                hackathon terminato, poichè archiviato.';
12.     END IF;
13.
14.     DELETE FROM appartenenza_Utente_team
15.     WHERE id_utente = OLD.id_utente AND id_team IN (
16.           SELECT id_team
17.           FROM team
18.           WHERE id_hackathon = OLD.id_hackathon );
19.     RETURN OLD;
```

```
20. END;
21. $$LANGUAGE plpgsql;
22.
23. CREATE OR REPLACE TRIGGER trg_delete_partecipante_team
24. BEFORE DELETE ON iscrizione_utente_hackathon
25. FOR EACH ROW
26. EXECUTE FUNCTION delete_partecipante_team()
```

4.2.16 Vincolo: blocco modifica iscrizioni

Questo vincolo rende le iscrizioni degli utenti alle hackathon completamente immutabili: blocca qualsiasi tentativo di modifica ai dati di un'iscrizione già registrata, garantendo che le informazioni originali non possano essere alterate.

```
1. CREATE OR REPLACE FUNCTION blocco_update_iscrizioneutentehackathon
      ()
2. RETURNS TRIGGER AS $$
3. BEGIN
4.     RAISE EXCEPTION 'Operazione negata: Non è possibile modificare
           i dati di un iscrizione esistente';
5. END;
6. $$LANGUAGE plpgsql;
7.
8. CREATE OR REPLACE TRIGGER
      trg_blocco_update_iscrizioneutentehackathon
9. BEFORE UPDATE ON iscrizione_utente_hackathon
10. FOR EACH ROW
11. EXECUTE FUNCTION blocco_update_iscrizioneutentehackathon();
```

4.2.17 Vincolo: validazione modifica/eliminazione team

Questo vincolo regola la gestione delle squadre: impedisce di trasferire un team da un'hackathon all'altra e vieta qualsiasi modifica o eliminazione se la competizione è già terminata. Inoltre, permette di rinominare la squadra solo ed esclusivamente prima che l'evento abbia inizio, congelando il nome a competizione avviata.

```
1.  CREATE OR REPLACE FUNCTION check_updateordelete_team()
2.  RETURNS TRIGGER AS $$
3.  DECLARE
4.      dataInizio date;
5.      dataFine date;
6.  BEGIN
7.      SELECT data_inizio, data_fine INTO dataInizio, dataFine
8.      FROM hackathon
9.      WHERE id_hackathon = OLD.id_hackathon;
10.
11.     IF (TG_OP = 'UPDATE') THEN
12.         IF(OLD.id_hackathon <> NEW.id_hackathon ) THEN
13.             RAISE EXCEPTION 'Non è permesso modificare a che
                    hackathon un team partecipa';
14.         END IF;
15.
16.         IF (dataFine <= CURRENT_DATE) THEN
17.             RAISE EXCEPTION 'Questo hackathon è concluso. I dati
                    del team sono archiviati e non modificabili.';
18.         END IF;
19.
20.         IF (NEW.nome <> OLD.nome AND dataInizio <= CURRENT_DATE)
               THEN
21.             RAISE EXCEPTION 'Non è possibile cambiare il nome del
                    team a competizione iniziata.';
22.         END IF;
23.         RETURN NEW;
24.     END IF;
25.
26.     IF (TG_OP = 'DELETE') THEN
27.         IF (dataFine <= CURRENT_DATE) THEN
28.             RAISE EXCEPTION 'Questa hackathon è concluso. I dati
                    del team sono archiviati e non è possibile
                    eliminarli.';
29.         END IF;
30.         RETURN OLD;
31.     END IF;
32. END;
33. $$LANGUAGE plpgsql;
```

```
34.
35. CREATE OR REPLACE TRIGGER trg_check_updateordelete_team
36. BEFORE DELETE OR UPDATE ON team
37. FOR EACH ROW
38. EXECUTE FUNCTION check_updateordelete_team();
```

4.2.18 Vincolo: blocco modifica/eliminazione voto

Questo vincolo rende le valutazioni espresse dai giudici completamente immutabili: blocca qualsiasi tentativo di modifica o cancellazione di un voto già inserito a sistema, garantendo l'inalterabilità dei risultati.

```
1.  CREATE OR REPLACE FUNCTION check_updateordelete_voto()
2.  RETURNS TRIGGER AS $$
3.  BEGIN
4.      IF(TG_OP = 'UPDATE') THEN
5.          RAISE EXCEPTION 'Operazione negata: Non è possibile
                modificare un voto';
6.      END IF;
7.
8.      IF(TG_OP = 'DELETE') THEN
9.          RAISE EXCEPTION 'Operazione negata: Non è possibile
                eliminare un voto';
10.     END IF;
11. END;
12. $$LANGUAGE plpgsql;
13.
14. CREATE OR REPLACE TRIGGER trg_check_updateordelete_voto
15. BEFORE DELETE OR UPDATE ON voto
16. FOR EACH ROW
17. EXECUTE FUNCTION check_updateordelete_voto()
```

4.3 Definizione View

Nelle sezioni seguenti sono riportate nel dettaglio le istruzioni per la creazione di diverse View. Questi strumenti si rivelano fondamentali per semplificare l'interrogazione di dati che cambiano dinamicamente, offrendo una rappresentazione sempre aggiornata e coerente.

4.3.1 Definizione della view TOTALE_TEAM

```
1. --Definizione view per contare quanti team ha una certa hackathon.
2. CREATE OR REPLACE VIEW TOTALE_TEAM AS
3.     SELECT id_hackathon,
4.     COUNT(id_team) AS numero_team
5.     FROM TEAM
6.     GROUP BY id_hackathon
```

4.3.2 Definizione della view TOTALE_GIUDICI

```
1. --Definizione view per contare il numero di giudici di un
      hackathon.
2. CREATE OR REPLACE VIEW TOTALE_GIUDICI AS
3.     SELECT id_hackathon,
4.     COUNT(id_giudice) AS numero_giudici
5.     FROM GIUDICA_HACKATHON
6.     GROUP BY id_hackathon
```

4.3.3 Definizione della view HACKATHON_VALUTATI

```
1. --Definizione view per controllare se un hackathon ha ricevuto
      tutti i voti, dunque se ogni giudice ne ha assegnato uno ad
      ogni team.
2. CREATE OR REPLACE VIEW HACKATHON_VALUTATI AS
3.     SELECT t.id_hackathon
4.     FROM VOTO v
5.     JOIN TEAM t ON v.id_team = t.id_team
6.     JOIN TOTALE_TEAM tt ON t.id_hackathon = tt.id_hackathon
7.     JOIN TOTALE_GIUDICI tg ON t.id_hackathon = tg.id_hackathon
8.     GROUP BY t.id_hackathon, tt.numero_team, tg.numero_giudici
9.     HAVING COUNT(v.id_team) = (tt.numero_team * tg.numero_giudici)
```

4.3.4 Definizione della view CLASSIFICA

```
1.  --Definizione view per calcolare la classifica di un hackathon.
2.  CREATE OR REPLACE VIEW CLASSIFICA AS
3.      SELECT t.id_hackathon,
4.          t.id_team,
5.          t.nome AS nome_team,
6.          AVG(v.valutazione)::numeric(4,2) AS media_voti
7.      FROM TEAM t
8.      JOIN VOTO v ON t.id_team = v.id_team
9.      JOIN HACKATHON_VALUTATI hv ON t.id_hackathon = hv.id_hackathon
10.     GROUP BY t.id_hackathon, t.id_team, t.nome
11.     ORDER BY t.id_hackathon, (AVG(v.valutazione)::numeric(4,2))
           DESC
```