Hackathon

Federico Padulano N86005043
Fabrizio Padulano N86005058

Sommario

1 Introduzione

Il progetto descritto in questo documento si basa sulla modellazione e gestione di un'applicazione per la gestione di Hackathon. Il sistema prevede la registrazione di utenti, la gestione degli eventi, la creazione di team e l'interazione con i giudici durante e dopo la competizione.

1.1 Descrizione del problema

Un Hackathon è un evento competitivo, solitamente della durata di due giorni, che si svolge in una sede specifica ed è organizzato da un utente registrato alla piattaforma. L'organizzatore definisce il titolo dell'evento, la sede, la durata, il numero massimo di partecipanti e la dimensione massima dei team. Inoltre, l'organizzatore seleziona un gruppo di giudici, anch'essi utenti registrati alla piattaforma, invitandoli a partecipare come valutatori.

Una volta creato l'evento, vengono aperte le registrazioni, che si chiudono due giorni prima dell'inizio. Gli utenti interessati possono iscriversi all'Hackathon desiderato e, durante il periodo di registrazione, possono formare team. I team diventano definitivi alla chiusura delle iscrizioni.

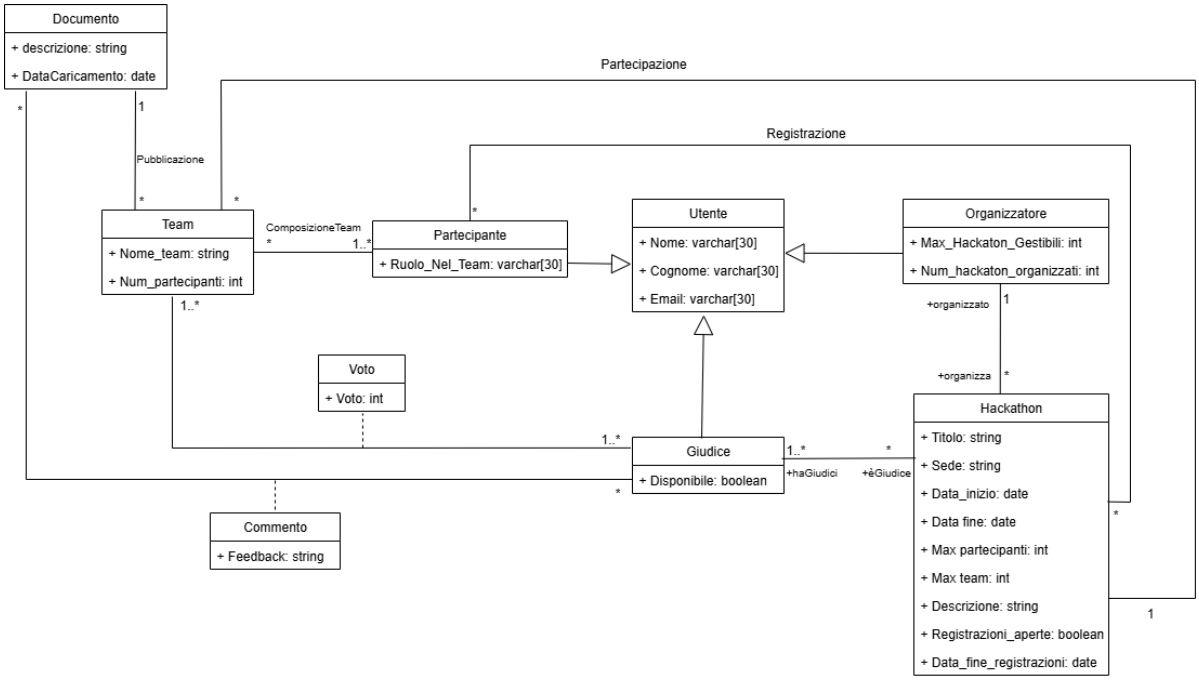
All'inizio dell'hackathon, i giudici pubblicano la descrizione del problema da affrontare. Durante l'evento, i team lavorano separatamente e caricano periodicamente documenti di aggiornamento dei loro progressi sulla piattaforma. Questi documenti possono essere esaminati e commentati dai giudici.

Alla fine dell'hackathon, ogni giudice assegna un voto, compreso tra 0 e 10, a ciascun team. Una volta raccolti tutti i voti, la piattaforma elabora e pubblica la classifica finale dei team partecipanti, in base ai punteggi ottenuti.

2 Progettazione Concettuale

In questa sezione progettiamo il database partendo da un livello generale, senza preoccuparci ancora dei dettagli tecnici. Basandoci sui requisiti analizzati prima, abbiamo creato uno schema concettuale usando un diagramma UML. L'obiettivo è identificare gli elementi principali di un hackathon e come si collegano tra loro, assicurandoci che tutte le regole emerse dalla descrizione del problema siano rappresentate correttamente.

2.1 Class Diagram



2.2 Ristrutturazione del Class Diagram

In questa fase riorganizziamo il Class Diagram per prepararlo alla trasformazione in database relazionale e migliorarne la struttura. La ristrutturazione seguirà questa serie di passaggi:

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

2.2.1 Analisi delle ridondanze

Nel Class Diagram sono presenti alcune ridondanze a livello di attributi che, pur potenzialmente utili per ragioni di efficienza, risultano non necessarie poiché i valori che rappresentano possono essere calcolati dinamicamente a partire da altre associazioni o entità. In particolare:

- **Numero di partecipanti nel team** (`Num_partecipanti` nella classe `Team`): si tratta di un attributo ridondante, in quanto il numero dei partecipanti può essere ricavato direttamente contando le associazioni tra la classe `Team` e la classe `Partecipante` (tramite la relazione `ComposizioneTeam`).
- **Numero di hackathon organizzati** (`Num_hackaton_organizzati` nella classe `Organizzatore`): anche questo è un attributo ridondante, in quanto il numero di hackathon associati a un organizzatore può essere ottenuto contando le istanze della relazione tra `Organizzatore` e `Hackaton`.

In entrambi i casi, si tratta di attributi derivabili da altre informazioni già presenti nel modello. Per mantenere la coerenza e ridurre la complessità, sarebbe meglio evitare queste ridondanze e calcolare i dati al momento del bisogno.

2.2.2 Analisi delle gerarchie di specializzazione

Nel Class Diagram proposto, l'entità Utente rappresenta una generalizzazione da cui derivano tre sottoclassi: Giudice, Partecipante e Organizzatore. Questa gerarchia è totale (ogni utente appartiene obbligatoriamente a una sottoclasse) e disgiunta (un utente non può appartenere a più di una sottoclasse contemporaneamente). Tuttavia, i modelli relazionali non supportano direttamente le gerarchie di specializzazione, rendendo necessaria una ristrutturazione del modello.

Si è scelto di eliminare l'entità generale Utente, distribuendo i suoi attributi comuni (nome, cognome, email) direttamente all'interno delle entità specializzate: Giudice, Partecipante e Organizzatore. Questa decisione è stata presa per semplificare la modellazione del sistema, poiché ciascuna di queste entità ha relazioni differenti e specifiche con altre entità del modello. Ad esempio, il Partecipante è associato ai Team, l'Organizzatore gestisce gli Hackathon, mentre il Giudice valuta i progetti. Mantenere una struttura gerarchica con una superclasse comune avrebbe complicato la gestione di queste relazioni, richiedendo join frequenti o controlli aggiuntivi per distinguere tra i diversi tipi di utente. Ogni classe specializzata eredita quindi tali attributi, che diventano propri attributi interni delle entità Giudice, Partecipante e Organizzatore. Questa trasformazione non comporta alcuna perdita di informazione.

2.2.3 Analisi degli attributi a valore multiplo

Nel modello concettuale proposto, **non sono presenti attributi a valore multiplo**. Ogni concetto potenzialmente associabile a più elementi (ad esempio, i partecipanti di un team o i giudici di un hackathon) è correttamente modellato attraverso *relazioni uno-a-molti o molti-a-molti*, evitando così la necessità di rappresentare attributi multipli direttamente all'interno di un'entità.

2.2.4 Analisi degli attributi strutturati

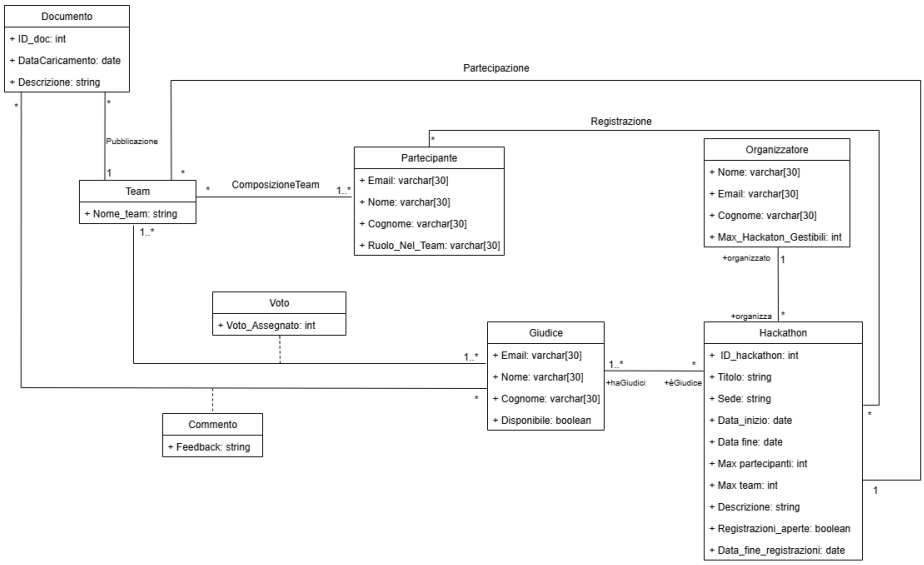
Gli attributi strutturati sono attributi composti da più componenti interne, spesso indicativi di una struttura dati più complessa.
Nel Class Diagram proposto **non sono presenti attributi strutturati**. Tutti gli attributi associati alle entità sono di tipo semplice e rappresentano dati elementari, pertanto non si rendono necessarie modifiche in questa fase.

2.2.5 Analisi delle chiavi

Per garantire un'identificazione univoca delle entità principali del sistema Hackathon, è importante distinguere tra **chiavi naturali** e **chiavi surrogate**. Le prime sono attributi già significativi nel dominio applicativo (come l'email), mentre le seconde sono identificatori artificiali. Nel nostro progetto si è scelto di adottare entrambe le soluzioni in base alla stabilità e semplicità dell'attributo. Di seguito, le chiavi candidate per ciascuna entità:

- **Hackathon**:
  - Chiave surrogata: ID_hackathon
- **Team**:
  - Chiave naturale: nome_team
- **Giudice**:
  - Chiave naturale: email
- **Organizzatore**:
  - Chiave naturale: email
- **Partecipante**:
  - Chiave naturale: email
- **Documento**:
  - Chiave surrogata: ID_doc

2.3 Class Diagram Ristrutturato



2.4 Dizionario delle Classi

Classe	Descrizione	Attributi
Partecipante	Utente che partecipa a un hackathon ed è associato a un team.	• **Email** (string): Chiave primaria. Email univoca del partecipante. • **Nome** (string): Nome del partecipante. • **Cognome** (string): Cognome del partecipante. • **RuoloNelTeam** (string): Ruolo assegnato nel team.

Classe	Descrizione	Attributi
Giudice	Utente che valuta i team partecipanti a un hackathon.	• **Email** (string): Chiave primaria. Email univoca del giudice. • **Nome** (string): Nome del giudice. • **Cognome** (string): Cognome del giudice. • **Disponibile** (boolean): Indica se il giudice può essere invitato a nuovi hackathon.
Organizzatore	Utente che crea e gestisce uno o più hackathon.	• **Email** (string): Chiave primaria. Email univoca dell'organizzatore. • **Nome** (string): Nome dell'organizzatore. • **Cognome** (string): Cognome dell'organizzatore. • **Max_Hackaton_Gestibili** (int): Indica il numero massimo di Hackaton gestibili.

Classe	Descrizione	Attributi
Hackathon	Evento competitivo organizzato da un utente, con partecipanti e giudici.	• **ID_hackathon** (int): Chiave primaria. Identificativo univoco. • **Titolo** (string): Nome dell'evento. • **Sede** (string): Luogo dell'evento. • **Data_inizio** (date): Inizio dell'evento. • **Data_fine** (date): Fine dell'evento. • **Num_max_partecipanti** (int): Numero massimo di partecipanti. • **Num_max_team** (int): Dimensione massima per team. • **Descrizione** (string): Descrizione dell'evento. • **Registrazioni_aperte** (boolean): Flag che indica se le registrazioni all'hackathon sono attualmente aperte (TRUE) o chiuse (FALSE). • **Data_fine_registrazioni** (date): Data limite oltre la quale non è più possibile iscriversi all'hackathon.
Team	Raggruppamento di partecipanti che collaborano per competere all'interno di un hackathon.	• **Nome_team** (string): Chiave primaria. Nome identificativo del team.

Classe	Descrizione	Attributi
Voto	Valutazione assegnata da un giudice a un team alla fine di un hackathon.	• **Voto_Assegnato** (int): Punteggio assegnato (da 0 a 10).
Documento	Rappresenta un aggiornamento caricato da un team durante un hackathon.	• **ID_doc** (int): Chiave primaria. Identificativo univoco del documento. • **Descrizione** (text): Contenuto o riferimento al file caricato. • **DataCaricamento** (date): Data in cui è stato caricato il documento.
Commento	Osservazione lasciata da un giudice su un documento caricato da un team.	• **Feedback** (text): Testo del commento inserito dal giudice.

2.5 Dizionario delle Associazioni

Nome	Descrizione	Classi coinvolte
Registrazione	Esprime l'iscrizione di un Partecipante a un Hackathon.	**Partecipante** [1..1] partecipa a: partecipante iscritto. **Hackathon** [1..*] ha partecipanti: evento associato al partecipante.
Organizzazione	Descrive l'organizzazione di uno o più Hackathon da parte di un Organizzatore.	**Organizzatore** [1..1] organizza: utente che crea l'hackathon. **Hackathon** [0..*] è organizzato da: evento gestito da quell'utente.
Valutazione	Associazione molti-a-molti tra Giudice e Team con voto numerico.	**Giudice** [1..*] valuta: giudice che assegna il voto. **Team** [1..*] è valutato da: team che riceve voti. **Voto** è la classe associativa con attributo `Voto_Assegnato (0 a 10)`.

Nome	Descrizione	Classi coinvolte
Composizione Team	Associa un Partecipante a un Team, indicando il ruolo.	**Partecipante** [0..*] è membro di: partecipante assegnato a un team. **Team** [1..*] è composto da: team formato da uno o più partecipanti.
Giudici Invitati	Collega ogni Giudice agli Hackathon a cui è stato invitato.	**Giudice** [1..*] partecipa come valutatore. **Hackathon** [0..*] ha giudici: hackathon con uno o più giudici.
Pubblicazione	Associa ogni Documento a un Team.	**Team** [1..1] carica: team autore del documento. **Documento** [0..*] pubblicato da: aggiornamenti caricati durante l'evento.
Commento	Ogni Giudice può commentare documenti caricati da Team.	**Giudice** [0..*] autore del commento. **Documento** [0..*] riceve commenti. **Commento** è la classe associativa con attributo `commento`.
Partecipazione	Relazione tra un Team e l'Hackathon a cui appartiene.	**Team** [0..*] team creato per un hackathon. **Hackathon** [1..1] ha team: ogni evento ospita più team.

3 Progettazione Logica

In questa sezione viene presentata la trasformazione del modello concettuale (UML) in uno schema logico basato sul modello relazionale. Ogni entità, relazione e attributo viene convertito secondo le regole di mapping standard, mantenendo la coerenza semantica del modello di partenza.

3.1 Traduzione delle Entità

Le entità individuate nel Class Diagram vengono mappate in tabelle relazionali. Ogni entità mantiene la propria chiave primaria (naturale o surrogata) e gli attributi elementari.

- **Partecipante(<u>Email</u>, Nome, Cognome, RuoloNelTeam)**
- **Hackathon(<u>ID_hackathon</u>, Titolo, Sede, Data_inizio, Data_fine, Num_max_partecipanti, Num_max_team, Descrizione,Registrazioni_aperte, Data_fine_registrazioni, Email_organizzatore)**
- **Registrazione(<u>Email_partecipante</u>, <u>ID_hackathon</u>)**
  Email_partecipante → Partecipante.Email
  ID_hackathon → Hackaton.ID_hackathon

- **Organizzatore(Email, Nome, Cognome, Max_Hackaton_Gestibili)**
- **Giudice(Email, Nome, Cognome, Disponibile)**
- **InvitoGiudice(Email_giudice, ID_hackathon)**
  Email_giudice → Giudice.Email
  ID_hackathon → Hackaton.ID_hackathon
- **Team(Nome_team, ID_hackathon)**
- **Documento(ID_doc, Descrizione, DataCaricamento, Nome_team)**
- **Partecipante_Team(Email_partecipante, Nome_team)**
  Email_partecipante → Partecipante.Email
  Nome_team → Team.Nome_team
- **Commento(Email_giudice, ID_doc, commento)**
  Email_giudice → Giudice.Email
  ID_doc → Documento.ID_doc
- **Voto(Email_giudice, Nome_team, Voto)**
  Email_giudice → Giudice.Email
  Nome_team → Team.Nome_team

3.2 Dizionario dei Vincoli

Vincolo	**Descrizione**
PK_EmailUtenti	Gli attributi `Email` in `Partecipante`, `Organizzatore` e `Giudice` devono essere univoci.
PK_NomeTeam	L'attributo `Nome_team` è chiave primaria della tabella `Team`. Nessun nome duplicato.
FK_PartecipanteTeam	Ogni partecipazione collega un Partecipante e un Team. `Email_partecipante` e `Nome_team` sono chiavi esterne.
FK_DocumentoTeam	Ogni Documento è associato a un Team esistente. `Nome_team` è chiave esterna.
FK_CommentoDocGiudice	Ogni commento è collegato a un documento e a un giudice. `ID_doc` e `Email_giudice` sono chiavi esterne.

Vincolo	Descrizione
FK_Voto	Ogni voto è assegnato da un giudice a un team. `Email_giudice`, `Nome_team` sono chiavi esterne.
FK_Registrazione	Ogni registrazione collega un partecipante e un hackathon. `Email_partecipante` e `ID_hackathon` sono chiavi esterne.
FK_InvitoGiudice	Ogni invito collega un giudice a un evento. `Email_giudice`, `ID_hackathon` sono chiavi esterne.
FK_HackathonOrg	Ogni hackathon è organizzato da un utente. `Email_organizzatore` è chiave esterna.
Voto valido	L'attributo `Voto_Assegnato` deve essere compreso tra 0 e 10.
Data Caricamento Valida	La data di caricamento di un documento deve essere compresa tra l'inizio e la fine dell'hackathon.
Num Max Partecipanti	Numero di partecipanti iscritti a un hackathon non può superare `Num_max_partecipanti`.
Num Max Team	Numero di team iscritti a un hackathon non può superare `Num_max_team`.
Partecipazioni non sovrapposte	Un partecipante non può iscriversi a due hackathon con date che si sovrappongono.
Partecipazioni a team diversi di diversi hackathon	Un partecipante può far parte di più team, a condizione che questi appartengano a hackathon differenti. Non è quindi consentito che uno stesso partecipante sia membro di più team all'interno dello stesso hackathon.

Vincolo	Descrizione
Registrazioni con deadline	Non possono esistere hackathon con registrazioni aperte senza una deadline esplicitamente definita.

4 Progettazione Fisica

Di seguito viene riportata la definizione delle tabelle in linguaggio SQL, in accordo con la progettazione logica.

4.1 Definizione Tabelle

```
CREATE TABLE Partecipante (
    Email VARCHAR(100) PRIMARY KEY,
    Nome VARCHAR(50) NOT NULL,
    Cognome VARCHAR(50) NOT NULL,
    RuoloNelTeam VARCHAR(30)
);

CREATE TABLE Organizzatore (
    Email VARCHAR(100) PRIMARY KEY,
    Nome VARCHAR(50) NOT NULL,
    Cognome VARCHAR(50) NOT NULL,
    Max Hackaton Gestibili INT NOT NULL
);

CREATE TABLE Giudice (
    Email VARCHAR(100) PRIMARY KEY,
    Nome VARCHAR(50) NOT NULL,
    Cognome VARCHAR(50) NOT NULL,
    Disponibile BOOLEAN DEFAULT TRUE
);

CREATE TABLE Hackathon (
    ID_hackathon SERIAL PRIMARY KEY,
    Titolo VARCHAR(100) NOT NULL,
    Sede VARCHAR(100) NOT NULL,
    Data_inizio DATE NOT NULL,
    Data_fine DATE NOT NULL,
    Num_max_partecipanti INT,
    Num_max_team INT,
    Descrizione TEXT,
    Registrazioni_aperte BOOLEAN NOT NULL DEFAULT FALSE,
    Data_fine_registrazioni DATE,
    Email_organizzatore VARCHAR(100) NOT NULL,
    CONSTRAINT fk_organizzatore FOREIGN KEY (Email_organizzatore) REFERENCES Organizzatore(Email),
    CONSTRAINT chk_max_partecipanti CHECK (Num_max_partecipanti > 0),
    CONSTRAINT chk_max_team CHECK (Num_max_team > 0),
    CONSTRAINT chk_deadline_if_registrazioni_aperte
             CHECK ( NOT (Registrazioni_aperte = TRUE AND Data_fine_registrazioni IS NULL))
);
```

```
CREATE TABLE Team (
    Nome_team VARCHAR(50) PRIMARY KEY,
    ID_hackathon INT NOT NULL,
    CONSTRAINT fk_hackathon_team FOREIGN KEY (ID_hackathon) REFERENCES Hackathon(ID_hackathon)
);

CREATE TABLE Partecipante_Team (
    Email_partecipante VARCHAR(100) NOT NULL,
    Nome_team VARCHAR(50) NOT NULL,
    PRIMARY KEY (Email_partecipante, Nome_team),
    CONSTRAINT fk_partecipante FOREIGN KEY (Email_partecipante) REFERENCES Partecipante(Email),
    CONSTRAINT fk_team FOREIGN KEY (Nome_team) REFERENCES Team(Nome_team)
);

CREATE TABLE Documento (
    ID_doc SERIAL PRIMARY KEY,
    Descrizione TEXT NOT NULL,
    DataCaricamento DATE NOT NULL,
    Nome_team VARCHAR(50) NOT NULL,
    CONSTRAINT fk_team_doc FOREIGN KEY (Nome_team) REFERENCES Team(Nome_team)
);

CREATE TABLE Commento (
    Email_giudice VARCHAR(100) NOT NULL,
    ID_doc INT NOT NULL,
    Commento TEXT NOT NULL,
    PRIMARY KEY (Email_giudice, ID_doc),
    CONSTRAINT fk_commento_doc FOREIGN KEY (ID_doc) REFERENCES Documento(ID_doc),
    CONSTRAINT fk_commento_giudice FOREIGN KEY (Email_giudice) REFERENCES Giudice(Email)
);

CREATE TABLE Voto (
    Email_giudice VARCHAR(100) NOT NULL,
    Nome_team VARCHAR(50) NOT NULL,
    Voto INT NOT NULL,
    PRIMARY KEY (Email_giudice, Nome_team),
    CONSTRAINT fk_voto_giudice FOREIGN KEY (Email_giudice) REFERENCES Giudice(Email),
    CONSTRAINT fk_voto_team FOREIGN KEY (Nome_team) REFERENCES Team(Nome_team),
    CONSTRAINT chk_voto_range CHECK (Voto BETWEEN 0 AND 10)
);

CREATE TABLE Registrazione (
    Email_partecipante VARCHAR(100) NOT NULL,
    ID_hackathon INT NOT NULL,
    PRIMARY KEY (Email_partecipante, ID_hackathon),
    CONSTRAINT fk_registrazione_part FOREIGN KEY (Email_partecipante) REFERENCES Partecipante(Email),
    CONSTRAINT fk_registrazione_hack FOREIGN KEY (ID_hackathon) REFERENCES Hackathon(ID_hackathon)
);

CREATE TABLE InvitoGiudice (
    Email_giudice VARCHAR(100) NOT NULL,
    ID_hackathon INT NOT NULL,
    PRIMARY KEY (Email_giudice, ID_hackathon),
    CONSTRAINT fk_invito_giudice FOREIGN KEY (Email_giudice) REFERENCES Giudice(Email),
    CONSTRAINT fk_invito_hack FOREIGN KEY (ID_hackathon) REFERENCES Hackathon(ID_hackathon)
);
```

4.2 Implementazione dei vincoli

Di seguito sono riportate le implementazioni dei vincoli che non sono già stati mostrati nelle definizioni delle tabelle.

4.2.1 Implementazione del vincolo Partecipazioni a team diversi di diversi hackathon

Un partecipante può far parte di più team, a condizione che questi appartengano a hackathon differenti. Non è quindi consentito che uno stesso partecipante sia membro di più team all'interno dello stesso hackathon. Questo vincolo viene implementato mediante un trigger SQL su `Partecipazione_Team`.

```
CREATE OR REPLACE FUNCTION check_unique_participant_per_hackathon()
RETURNS trigger AS $$
DECLARE
    hackathon_id INT;
BEGIN
    SELECT id_hackathon INTO hackathon_id
    FROM Team WHERE Nome_Team = NEW.Nome_Team;

    IF EXISTS (
        SELECT 1 FROM Partecipante_Team PT
        JOIN Team T ON PT.Nome_Team = T.Nome_Team
        WHERE PT.Email_partecipante = NEW.Email_partecipante
          AND T.id_hackathon = hackathon_id
    ) THEN
        RAISE EXCEPTION 'Il partecipante è già in un team per questo hackathon';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_check_unique_participant
BEFORE INSERT ON Partecipante_Team
FOR EACH ROW
EXECUTE FUNCTION check_unique_participant_per_hackathon();
```

4.2.2 Implementazione del vincolo DataCaricamento valida

La data di caricamento di un documento deve essere compresa tra la data di inizio e la data di fine dell'hackathon associato al team autore del documento.

```
CREATE OR REPLACE FUNCTION check_data_caricamento()
RETURNS TRIGGER AS $$
DECLARE
    data_inizio DATE;
    data_fine DATE;
BEGIN
    SELECT H.Data_inizio, H.Data_fine
    INTO data_inizio, data_fine
    FROM Team T
    JOIN Hackathon H ON T.ID_hackathon = H.ID_hackathon
    WHERE T.Nome_team = NEW.Nome_team;

    IF NEW.DataCaricamento < data_inizio OR NEW.DataCaricamento > data_fine THEN
```

```
        RAISE EXCEPTION 'La data di caricamento del documento
        non è compresa tra le date dell'hackathon';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_data_caricamento
BEFORE INSERT OR UPDATE ON Documento
FOR EACH ROW
EXECUTE FUNCTION check_data_caricamento();
```

4.2.3 Vincolo: Numero massimo di partecipanti

Un hackathon non può accettare un numero di partecipanti superiore al valore di `Num_max_partecipanti` definito al momento della creazione dell'evento.

```
CREATE OR REPLACE FUNCTION check_max_partecipanti()
RETURNS TRIGGER AS $$
DECLARE
    partecipanti_attuali INTEGER;
    max_partecipanti INTEGER;
BEGIN
    SELECT COUNT(*) INTO partecipanti_attuali
    FROM Registrazione
    WHERE ID_hackathon = NEW.ID_hackathon;

    SELECT Num_max_partecipanti INTO max_partecipanti
    FROM Hackathon
    WHERE ID_hackathon = NEW.ID_hackathon;

    IF partecipanti_attuali >= max_partecipanti THEN
        RAISE EXCEPTION 'Numero massimo di partecipanti per questo hackathon già raggiunto.';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_check_max_partecipanti
BEFORE INSERT ON Registrazione
FOR EACH ROW
EXECUTE FUNCTION check_max_partecipanti();
```

4.2.4 Vincolo: Numero massimo di team

Il numero di team iscritti a un hackathon non può superare il limite definito nel campo `Num_max_team`.

```
CREATE OR REPLACE FUNCTION check_max_team()
RETURNS TRIGGER AS $$
DECLARE
    team_correnti INTEGER;
    max_team INTEGER;
BEGIN
```

```
    SELECT COUNT(*) INTO team_correnti
    FROM Team
    WHERE ID_hackathon = NEW.ID_hackathon;

    SELECT Num_max_team INTO max_team
    FROM Hackathon
    WHERE ID_hackathon = NEW.ID_hackathon;

    IF team_correnti >= max_team THEN
        RAISE EXCEPTION 'Numero massimo di team per questo hackathon già raggiunto.';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_check_max_team
BEFORE INSERT ON Team
FOR EACH ROW
EXECUTE FUNCTION check_max_team();
```

4.2.5 Vincolo: Partecipazioni sovrapposte

Un partecipante non può iscriversi a due hackathon le cui date si sovrappongono.

```
CREATE OR REPLACE FUNCTION check_hackathon_overlap()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1
        FROM Registrazione R
        JOIN Hackathon H1 ON R.ID_hackathon = H1.ID_hackathon
        JOIN Hackathon H2 ON H2.ID_hackathon = NEW.ID_hackathon
        WHERE R.Email_partecipante = NEW.Email_partecipante
          AND R.ID_hackathon <> NEW.ID_hackathon
          AND H1.Data_inizio <= H2.Data_fine
          AND H2.Data_inizio <= H1.Data_fine
    ) THEN
        RAISE EXCEPTION 'Non è possibile iscriversi a due hackathon con date sovrapposte.';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_check_hackathon_overlap
BEFORE INSERT OR UPDATE ON Registrazione
FOR EACH ROW
EXECUTE FUNCTION check_hackathon_overlap();
```

4.2.6 Vincolo: Data di caricamento documento

La data di caricamento di un documento deve essere compresa tra la data di inizio e quella di fine dell'hackathon collegato al team autore del documento.

```
CREATE OR REPLACE FUNCTION check_data_caricamento_documento()
```

```
RETURNS TRIGGER AS $$
DECLARE
    data_inizio DATE;
    data_fine DATE;
BEGIN
    SELECT H.Data_inizio, H.Data_fine
    INTO data_inizio, data_fine
    FROM Team T
    JOIN Hackathon H ON T.ID_hackathon = H.ID_hackathon
    WHERE T.Nome_team = NEW.Nome_team;

    IF NEW.DataCaricamento < data_inizio OR NEW.DataCaricamento > data_fine THEN
        RAISE EXCEPTION 'La data di caricamento non è compresa tra la data di inizio
        e quella di fine dell'hackathon.
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_check_data_caricamento
BEFORE INSERT OR UPDATE ON Documento
FOR EACH ROW
EXECUTE FUNCTION check_data_caricamento_documento();
```

4.2.7 Vincolo: Numero massimo di partecipanti per team

Ogni team può contenere al massimo un numero di partecipanti pari al valore `Num_max_team` specificato nell'hackathon associato.

```
CREATE OR REPLACE FUNCTION check_max_partecipanti_per_team()
RETURNS TRIGGER AS $$
DECLARE
    partecipanti_correnti INTEGER;
    max_per_team INTEGER;
BEGIN
    SELECT COUNT(*) INTO partecipanti_correnti
    FROM Partecipante_team
    WHERE Nome_team = NEW.Nome_team;

    SELECT H.Num_max_team INTO max_per_team
    FROM Team T
    JOIN Hackathon H ON T.ID_hackathon = H.ID_hackathon
    WHERE T.Nome_team = NEW.Nome_team;

    IF partecipanti_correnti >= max_per_team THEN
        RAISE EXCEPTION 'Il team % ha già raggiunto il numero massimo di partecipanti (%).'
        , NEW.Nome_team, max_per_team;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_check_max_partecipanti_per_team
BEFORE INSERT ON partecipante_team
FOR EACH ROW
EXECUTE FUNCTION check_max_partecipanti_per_team();
```

5 Funzioni e Procedure

In questa sezione vengono presentate alcune funzionalità avanzate implementate nel sistema, utili a supportare automazioni, analisi e manutenzione dei dati.

5.1 Funzione: Calcolo della media voti di un team

Restituisce la media dei voti assegnati a un team.

```
CREATE OR REPLACE FUNCTION media_voti_team(nome TEXT)
RETURNS NUMERIC AS $$
DECLARE
    media NUMERIC;
BEGIN
    SELECT AVG(voto) INTO media
    FROM Voto
    WHERE Nome_team = nome;

    RETURN media;
END;
$$ LANGUAGE plpgsql;
```

5.2 Procedura: Chiusura automatica delle registrazioni scadute

Aggiorna lo stato delle registrazioni per tutti gli hackathon la cui deadline è superata.

```
CREATE OR REPLACE PROCEDURE chiudi_registrazioni_scadute()
LANGUAGE plpgsql
AS $$
BEGIN
    UPDATE Hackathon
    SET Registrazioni_aperte = FALSE
    WHERE Registrazioni_aperte = TRUE
      AND CURRENT_DATE > Data_fine_registrazioni;
END;
$$;
```

5.3 Funzione: Classifica finale dei team per hackathon

Funzione che restituisce una stringa con i team ordinati per media dei voti in un dato hackathon.

```
CREATE OR REPLACE FUNCTION classifica_team(id_evento INT)
RETURNS TEXT
AS $$
DECLARE
    risultato TEXT := '';
    r RECORD;
BEGIN
    FOR r IN (
        SELECT V.Nome_team, AVG(V.voto) AS media
```

```
        FROM Voto V
        JOIN Team T ON V.Nome_team = T.Nome_team
        WHERE T.ID_hackathon = id_evento
        GROUP BY V.Nome_team
        ORDER BY media DESC
    )
    LOOP
        risultato := risultato || r.Nome_team || ' ' || r.media || ', ';
    END LOOP;

    risultato := RTRIM(risultato, ', ');
    RETURN risultato;
END;
$$ LANGUAGE plpgsql;
```

5.4 Funzione: Elenco dei partecipanti a un hackathon

Restituisce una stringa con i partecipanti registrati a un determinato hackathon, nel formato `Nome Cognome Email`, separati da virgole.

```
CREATE OR REPLACE FUNCTION partecipanti_hackathon(id_evento Hackathon.ID_hackathon%TYPE)
RETURNS TEXT
AS $$
DECLARE
    s_out TEXT := '';
    p RECORD;
BEGIN
    FOR p IN (
        SELECT Nome, Cognome, Email
        FROM Partecipante P
        JOIN Registrazione R ON P.Email = R.Email_partecipante
        WHERE R.ID_hackathon = id_evento
    )
    LOOP
        s_out := s_out || p.Nome || ' ' || p.Cognome || ' ' || p.Email || ', ';
    END LOOP;

    s_out := rtrim(s_out);
    s_out := rtrim(s_out, ',');

    RETURN s_out;
END;
$$ LANGUAGE plpgsql;
```

5.5 Funzione: Elenco documenti caricati da un team

Restituisce una stringa con gli identificativi dei documenti caricati da un team, ordinati per data di caricamento e separati da virgole.

```
CREATE OR REPLACE FUNCTION documenti_team(nome_team Team.Nome_team%TYPE)
RETURNS TEXT
AS $$
DECLARE
    s_out TEXT := '';
    doc RECORD;
```

```
BEGIN
    FOR doc IN (
        SELECT ID_doc
        FROM Documento
        WHERE Nome_team = nome_team
        ORDER BY DataCaricamento
    )
    LOOP
        s_out := s_out || doc.ID_doc || ', ';
    END LOOP;
    s_out := rtrim(s_out);
    s_out := rtrim(s_out,',');
    RETURN s_out;
END;
$$ LANGUAGE plpgsql;
```