UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

DOCUMENTAZIONE PER PROGETTAZIONE BASE DI DATI

Progetto in Carico: Hackathon

CdL Triennale in Informatica
Corso di Basi di Dati I
GIOELE MANZONI
N86004562
LUCA LUCCI
N86005180

July 20, 2025

Anno Accademico: 2024/2025

Contents

1 Introduzione

Questa documentazione descriverà il processo di progettazione e sviluppo di un Database relazionale che gestirà il flusso di dati di un applicativo dedicato all'organizzazione di Hackathon. Questo è un progetto a cura degli studenti Gioele Manzoni e Luca Lucci del CdL di Informatica presso l'Università degli Studi di Napoli "Federico II".

1.1 Traccia del Progetto

Un hackathon, ovvero una "maratona di hacking", è un evento durante il quale team di partecipanti si sfidano per progettare e implementare nuove soluzioni basate su una certa tecnologia o mirate a un certo ambito applicativo. Ogni hackathon ha un titolo identificativo, si svolge in una certa sede e in un certo intervallo di tempo (solitamente 2 giorni) e ha un organizzatore specifico (registrato alla piattaforma). L'organizzatore seleziona un gruppo di giudici (selezionati tra gli utenti della piattaforma, invitandoli). Infine, l'organizzatore apre le registrazioni, che si chiuderanno 2 giorni prima dell'evento. Ogni evento avrà un numero massimo di iscritti e una dimensione massima del team. Durante il periodo di registrazione, gli utenti possono registrarsi per l'Hackathon di loro scelta (eventualmente registrandosi sulla piattaforma se non lo hanno già fatto). Una volta iscritti, gli utenti possono formare team. I team diventano definitivi quando si chiudono le iscrizioni. All'inizio dell'hackathon, i giudici pubblicano una descrizione del problema da affrontare. Durante l'hackathon, i team lavorano separatamente per risolvere il problema e devono caricare periodicamente gli aggiornamenti sui "progressi" sulla piattaforma come documento, che può essere esaminato e commentato dai giudici. Alla fine dell'hackathon, ogni giudice assegna un voto (da 0 a 10) a ciascun team e la piattaforma, dopo aver acquisito tutti i voti, pubblica le classifiche dei team.

Caratteristiche dell'Hackathon

Ogni Hackathon ha le seguenti caratteristiche:

- Un **titolo identificativo**;
- Una **sede** in cui si svolge;
- Un **intervallo di tempo**, solitamente di due giorni;
- Un **organizzatore specifico**, registrato sulla piattaforma.

Giudici e Registrazioni

- L'organizzatore seleziona un gruppo di **giudici**, invitandoli tra gli utenti registrati sulla piattaforma.
- L'organizzatore apre le **registrazioni**, che si chiudono due giorni prima dell'inizio dell'evento.
- Ogni evento prevede un **numero massimo di iscritti** e una **dimensione massima del team**.
- Durante il periodo di registrazione, gli utenti possono registrarsi all'Hackathon di loro scelta, previa registrazione sulla piattaforma se non ancora effettuata.

Formazione dei Team

- Una volta iscritti, gli utenti possono **formare team**.
- I team diventano **definitivi alla chiusura delle iscrizioni**.

Svolgimento dell'Hackathon

- All'inizio dell'evento, i giudici **pubblicano una descrizione del problema** da affrontare.
- Durante l'Hackathon, i team lavorano separatamente per risolvere il problema.
- I team devono **caricare periodicamente aggiornamenti sui progressi** tramite documenti sulla piattaforma.
- I documenti possono essere **esaminati e commentati dai giudici**.

Valutazione e Classifica

- Alla fine dell'Hackathon, ogni giudice assegna un **voto da 0 a 10** a ciascun team.
- La piattaforma, dopo aver acquisito tutti i voti, **pubblica le classifiche dei team**.

2 Progettazione Concettuale

2.1 Analisi delle Entità e degli Attributi

Seguendo la traccia, nella fase di progettazione concettuale sono state trovate le suddette entità:

2.1.1 Hackathon

Entità dedicata a tutte le maratone di Hacking organizzate.

- **Hackathon** ($\underline{Titolo\_identificativo}$, Descrizione_problema, Sede, Classifica, DataInizio_registrazioni, DataFine_registrazioni, DataInizio_Evento, DataFine_Evento, Num_iscrittiCorrente, MaxNum_membriTeam, MaxNum_iscritti)

2.1.2 Utente

Generalizzazione dedicata a tutti i tipi di utenti che è possibile avere all'interno della piattaforma. La generalizzazione è considerata come **DISGIUNTA PARZIALE**, poiché è possibile avere utenti della piattaforma che non sono né organizzatori né giudici.

- **Utente** ($\underline{Login}$, Password)

2.1.3 Organizzatore

Specializzazione dell'entità **Utente**, rappresentante gli organizzatori di maratone di Hacking. Non possiede alcun attributo specifico a sé stesso, la sua specializzazione definisce soltanto gli utenti con le giuste credenziali per poter gestire la piattaforma.

2.1.4 Giudice

Specializzazione dell'entità **Utente**, rappresentante i giudici che daranno le loro valutazioni ai documenti e supervisioneranno le Hackathon.

2.1.5 Team

Entità che definisce una squadra organizzata da un Utente e composta da N Utenti.

- **Team** ($\underline{NomeTeam}$, VotoFinale)

2.1.6 Documento

Entità debole che definisce un documento scritto da un team.

- **Documento** ($\underline{NomeTeam}$, Testo)

2.2 Analisi delle Relazioni

Qui verranno descritte tutte le relazioni e le specializzazioni presenti all'interno della struttura concettuale non ancora ristrutturata.

- *Organizzazione* (**Hackathon** - **Organizzatore**: 1 - N):
  Un Hackathon può essere organizzata da un solo organizzatore. Un organizzatore può organizzare più Hackathon.
- *Partecipazione* (**Team** - **Hackathon**: 1 - 2..N):
  Un Team può partecipare ad una sola Hackathon. Un Hackathon, per essere valida, deve avere un minimo di 2 Team fino ad un massimo di N.
- *Supervisione* (**Giudice** - **Hackathon**: 1..N - N):
  Un Giudice può supervisionare N Hackathon. Un Hackathon deve essere monitorata da almeno un giudice.

- *Appartenenza* (**Utente** - **Team**: 0..N - 1..N):
  Un Utente può partecipare ad uno o più team, o può non parteciparci affatto. Un Team deve essere composto da un minimo di un Utente fino ad un massimo di N (il limite di utenti appartenenti ad un Team è deciso dall'Hackathon alla quale si partecipa).
- *Invito* (**Organizzatore** - **Giudice**: 1..N - 1..N):
  Un organizzatore deve invitare un minimo di un utente fino ad un massimo di N utenti per essere giudici. Un giudice, per ricevere un invito, deve riceverlo da almeno un organizzatore.
- *Voto* (**Giudice** - **Team**: 1..N - 1..N):
  Un Giudice può esprimere una votazione ad un minimo di un Team. Un team può ricevere voti da almeno un giudice.
- *Valutazione* (**Giudice** - **Documento**: 1..N - 1..N):
  Un Giudice deve esprimere una valutazione per almeno un documento. Un documento deve ottenere una valutazione da almeno un giudice.
- *Pubblicazione* (**Documento** - **Team**: 1 - N):
  Relazione identificante per l'entità debole Documento, in quanto generato direttamente da un Team e non può esistere senza un associazione ad un Team.

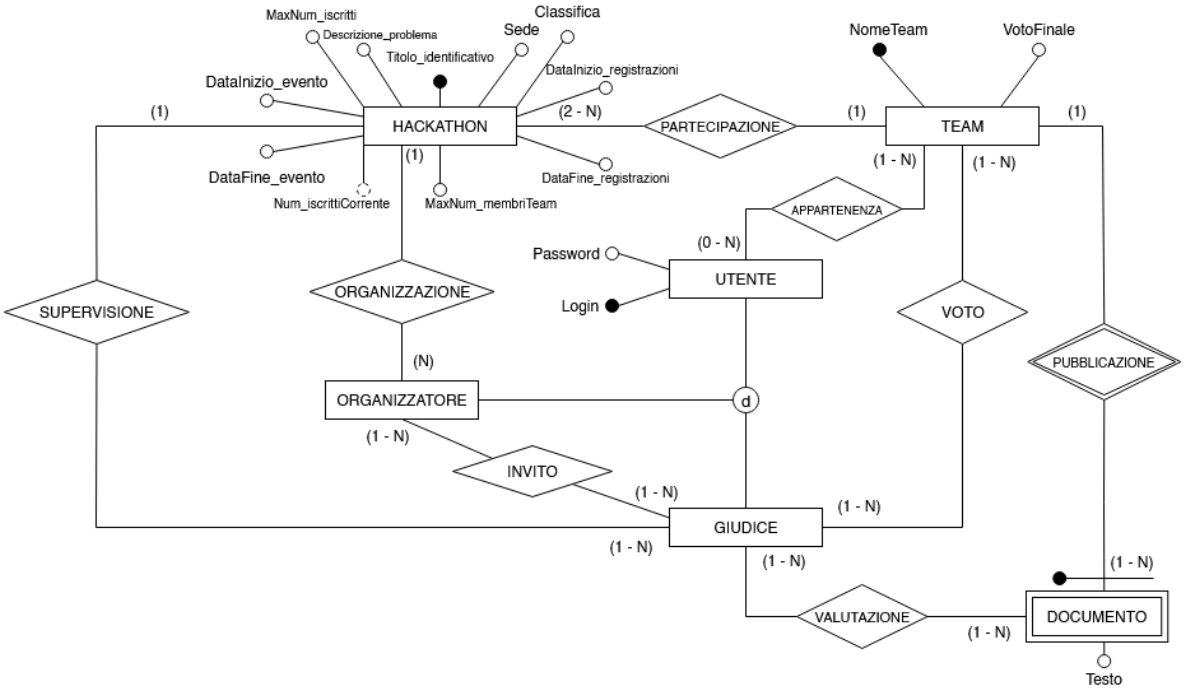


Figure 1: Grafico EER Concettuale

3 Ristrutturazione Modello Concettuale

3.1 Ristrutturazione del Modello Concettuale

Dopo aver analizzato i requisiti, le entità e le relazioni ed aver prodotto uno schema concettuale passeremo alla sua Ristrutturazione, seguendo i passaggi necessari elencati nelle prossime sottosezioni, ordinate in:

- Analisi delle Ridondanze
- Analisi delle Generalizzazioni/Gerarchie di Specializzazione
- Analisi degli Attributi Multivalore
- Analisi degli Attributi Strutturati
- Partizionamento/Accorpamento di Entità e Relazioni
- Analisi delle Chiavi

3.1.1 Analisi delle Ridondanze

- **Hackathon - NumIscritti_corrente**:
  Sebbene sia banalmente considerabile un attributo derivabile dal calcolo degli utenti iscritti ad un Hackathon è evidente che, per semplificarci la vita, sarebbe di gran lunga meglio avere a disposizione un attributo costantemente aggiornato che tenga traccia del numero totale degli iscritti a quell'Hackathon. In questo modo ci è possibile semplificare le operazioni di controllo sui limiti imposti dalla stessa per le iscrizioni.

3.1.2 Analisi delle Generalizzazioni/Gerarchie di Specializzazione

- **Gerarchia Disgiunta Parziale: Utente > Organizzatore, Giudice**:
  L'unica gerarchia all'interno della nostra progettazione è quella tra Utente, Organizzatore e Giudice. È definita come **disgiunta** in quanto un Utente non può essere sia Organizzatore che Giudice e come **parziale** in quanto possono esistere Utenti neutri che non siano specializzati in nessuno dei due titoli. Poiché abbiamo bisogno di fare una distinzione evidente tra i tre per livello di responsabilità si è deciso di renderle tre entità dipendenti: Organizzatore e Utente avranno entrambi il proprio metodo di accesso distinto (un Organizzatore può creare un account anche come Utente), mentre l'entità Giudice verrà gestita come una *"promozione"* concessa ad un Utente dopo aver accettato l'invito.

3.1.3 Analisi degli Attributi Multivalore

Non sono stati riscontrati attributi con valori multipli all'interno della nostra struttura.

3.1.4 Analisi degli Attributi Strutturati

Non sono stati riscontrati attributi con valori strutturati all'interno della nostra struttura.

3.1.5 Partizionamento/Accorpamento di Entità e Relazioni

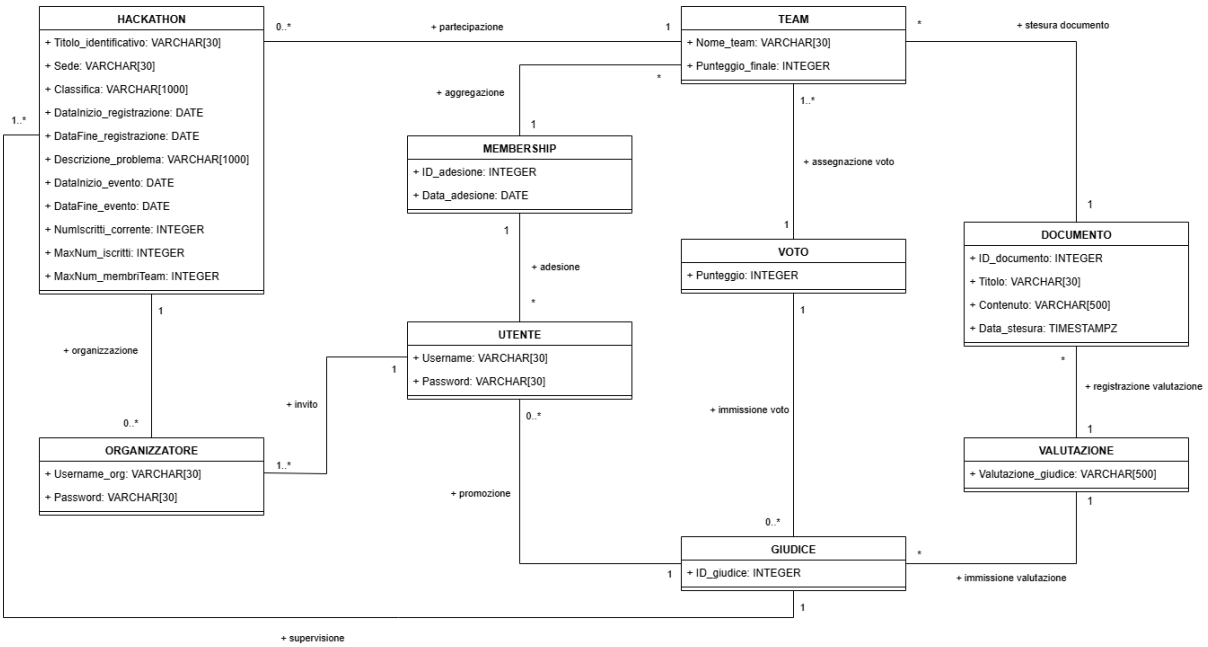
- **Invito** *(Organizzatore - Giudice [1..* - 1..*])*
  Questa relazione ha subito più cambiamenti:
  - È stata spostata da Giudice a Utente, in quanto adesso Giudice verrà trattato come un'entità debole che **estende** un Utente, e verrà creata solo all'accettazione dell'invito.
  - La relazione è stata ridefinita come 1 a N: Un Utente può ricevere un solo invito a partecipare ad un Hackathon e non può riceverne altri finché l'Hackathon alla quale sta partecipando come Giudice non sarà conclusa.
- **Voto** *(Giudice - Team [1..* - 1..*])*
  La relazione è stata spezzata con l'ausilio di un'entità associativa chiamata **Voto** che conserverà il punteggio dato dal Giudice al proprio interno. Un Team può ricevere più Voti da più Giudici, ma un Voto è deciso da un solo Giudice per un solo Team.
- **Appartenenza** *(Utente - Team [0..* - 1..*])*
  La relazione è stata spezzata con l'ausilio di un'entità associativa chiamata **Membership** che definisce l'appartenenza di un Utente ad un Team. Un Team è composto da più Membership di più Team, ma una Membership definisce la relazione tra un solo Utente con un solo Team.

- **Valutazione** *(Giudice - Documento [1..* - 1..*])*
  La relazione è stata spezzata con l'ausilio di un'entità associativa chiamata **Valutazione** che definisce la singola valutazione di un Giudice verso un Documento. Un Documento può essere valutato da più Giudici, ma la Valutazione è singola per il Documento.

3.1.6 Analisi delle Chiavi

- **Utente**: Si cambia il nome della chiave in Username poiché più appropriato. Rimane sufficiente per definire l'entità.
- **Organizzatore**: Discorso analogo a quello di Utente.
- **Hackathon**: Si mantiene il Titolo Identificativo come chiave principale, tenendo l'Username dell'organizzatore come chiave esterna. L'Hackathon sarà identificata anche dal proprio organizzatore.
- **Giudice**: Sarà un'entità debole definita sia dall'Utente al quale è legato che all'Hackathon per la quale l'Utente ha ricevuto l'invito. La sua chiave sarà composta dalle chiavi esterne delle due entità sopra citate.
- **Team**: Il nome del Team sarà parte della chiave composta, unita alla chiave esterna dell'Hackathon alla quale parteciperà.
- **Documento**: Nonostante sia essenzialmente un'entità debole generata dal Team al momento della sua stesura, si è comunque vista la necessità di una chiave surrogata (ID Autoincrementante, con l'ausilio del *SERIAL* di *Postgre*) per poter distinguere i vari documenti di un singolo Team. Sarà identificato tramite il Nome del Team e il Titolo dell'Hackathon come chiavi esterne.
- **Invito Giudice**: Questa entità verrà definita dalle chiavi dell'Organizzatore, l'Utente invitato e il Titolo dell'Hackathon alla quale l'utente parteciperà come Giudice.
- **Membership**: Questa entità verrà definita dalle chiavi dell'Utente e del Team, insieme ad una chiave surrogata (ID Autoincrementante, con l'ausilio del *SERIAL* di *Postgre*) che permetterà di distinguere le singole membership di un Team.
- **Voto**: Questa entità verrà definita dalle chiavi del Giudice e del Team.
- **Valutazione**: Questa entità verrà definita dalle chiavi del Giudice, del Team e del Documento da valutare.

3.2 Class Diagram Ristrutturato



3.3 Dizionario delle Classi

Classe	Descrizione	Attributi
HACKATHON	Evento organizzato a cui partecipano team.	▪ `Titolo_identificativo` (VARCHAR[30]): Titolo univoco che identifica un Hackathon; ▪ `Sede` (VARCHAR[30]): Sede dove si svolgerà l'Hackathon; ▪ `Classifica` (TEXT): Classifica finale con i posizionamenti dei Team, in ordine di punteggio complessivo; ▪ `DataInizio_registrazione` (DATE); ▪ `DataFine_registrazione` (DATE); ▪ `Descrizione_problema` (TEXT): Descrizione del problema da risolvere offerta dai giudici; ▪ `DataInizio_evento` (DATE); ▪ `DataFine_evento` (DATE); ▪ `NumIscritti_corrente` (INTEGER): Numero degli iscritti all'Hackathon aggiornato fino alla chiusura delle iscrizioni o al raggiungimento del tetto massimo; ▪ `MaxNum_iscritti` (INTEGER): Numero massimo di iscritti per un Hackathon; ▪ `MaxNum_membriTeam` (INTEGER): Numero massimo di membri per team iscritti ad una determinata Hackathon;
UTENTE	Utente registrato alla piattaforma.	▪ `Username` (VARCHAR[30]): Nome utente univoco; ▪ `Password` (VARCHAR[30])
ORGANIZZATORE	Utente con ruolo di organizzatore.	▪ `Username_org` (VARCHAR[30]): Nome utente univoco per organizzatore; ▪ `Password` (VARCHAR[30])
GIUDICE	Utente con ruolo di giudice.	
TEAM	Team partecipante a un hackathon.	▪ `Nome_team` (VARCHAR[30]): Nome univoco che identifica un team; ▪ `Punteggio_finale` (INTEGER): Punteggio complessivo del team dato dalla somma di tutti i punteggi ottenuti dai giudici;
MEMBERSHIP	Adesione di un utente a un team.	▪ `ID_adesione` (SERIAL): Codice identificativo per la singola adesione ad un Team; ▪ `Data_adesione` (DATE)
VOTO	Voto assegnato da un giudice a un team.	▪ `Punteggio` (INTEGER)
DOCUMENTO	Documento prodotto da un team.	▪ `ID_documento` (SERIAL): Codice identificativo per il singolo documento; ▪ `Titolo_doc` (VARCHAR[30]): Titolo del documento per riassumerne il contenuto; ▪ `Contenuto` (TEXT): Testo del documento; ▪ `Data_stesura` (TIMESTAMPTZ)
VALUTAZIONE	Valutazione scritta da un giudice.	▪ `Valutazione_giudice` (TEXT): Valutazione di un giudice per un documento;
INVITO_GIUDICE	Invito a diventare Giudice da parte di un Organizzatore verso un Utente	▪ `Stato_invito` (VARCHAR[20]): Stato di un invito, diviso in tre stati: L'invito è stato inviato all'utente e attende risposta (*Inviato*), l'invito è stato accettato (*Accettato*), L'invito è stato rifiutato (*Rifiutato*); ▪ `Data_invito` (DATE);

3.4 Dizionario delle Associazioni

Nome Associazione	Entità Associate	Descrizione Associazione
Partecipazione	HACKATHON ↔ TEAM	▪ Tipo: Uno-a-molti [0..1 - *] ▪ Un Hackathon può avere più Team partecipanti. Un Team deve partecipare ad un Hackathon ▪ Chiave esterna: `Titolo_hackathon` in TEAM ▪ Vincolo: Ogni Team partecipa a un solo Hackathon
Organizzazione	ORGANIZZATORE ↔ HACKATHON	▪ Tipo: Uno-a-molti [0..* - 1] ▪ Un organizzatore può organizzare più Hackathon, o può non organizzarne. Un Hackathon è organizzata da un solo organizzatore ▪ Chiave esterna: `Organizzatore` in HACKATHON ▪ Vincolo: Ciascun Hackathon può essere organizzata da una sola persona
Supervisione	HACKATHON ↔ GIUDICE	▪ Tipo: Uno-a-molti [1..* - 1] ▪ Un Hackathon può avere più Giudici assegnati ▪ Chiave esterna: `Titolo_hackathon` in GIUDICE ▪ Vincolo: Ogni Giudice è assegnato a un solo Hackathon
Invito Giudice	ORGANIZZATORE ↔ UTENTE	▪ Tipo: Uno-a-molti [1..* - 1] ▪ Un organizzatore deve invitare almeno un utente a diventare giudice ▪ Vincolo: Un utente può ricevere un solo invito alla volta per essere giudice di un Hackathon
Promozione	UTENTE ↔ GIUDICE	▪ Tipo: Uno-a-molti [0..* - 1] ▪ Un Utente può essere promosso fino ad N volte per diventare Giudice. Un Giudice è correlato ad un solo Utente per una sola Hackathon. ▪ Vincolo: Un utente può ricevere un solo invito alla volta per essere giudice di un Hackathon
Aggregazione	TEAM ↔ MEMBERSHIP	▪ Tipo: Uno-a-molti [1..* - 1] ▪ Un Team può avere più adesioni (Membership) ▪ Chiave esterna: `Nome_team` in MEMBERSHIP ▪ Vincolo: Ogni Membership è per un solo Team
Adesione	UTENTE ↔ MEMBERSHIP	▪ Tipo: Uno-a-molti [* - 1] ▪ Un Utente può appartenere a più Team (tramite Membership) ▪ Chiave esterna: `Username_utente` in MEMBERSHIP ▪ Vincolo: Ogni Membership è di un solo Utente ed è rivolta ad un solo Team
Stesura Documento	TEAM ↔ DOCUMENTO	▪ Tipo: Uno-a-molti [* - 1] ▪ Un Team può produrre più Documenti ▪ Chiavi esterne: `Nome_team` e `Titolo_hackathon` in DOCUMENTO ▪ Vincolo: Ogni Documento è prodotto da un solo Team

Nome Associazione	Entità Associate	Descrizione Associazione
Registrazione Valutazione	DOCUMENTO ↔ VALUTAZIONE	▪ Tipo: Uno-a-molti [* - 1] ▪ Un Documento può ricevere più Valutazioni ▪ Chiave esterna: `ID_documento` in VALUTAZIONE ▪ Vincolo: Ogni Valutazione è per un solo Documento
Immissione Valutazione	GIUDICE ↔ VALUTAZIONE	▪ Tipo: Uno-a-molti [* - 1] ▪ Un Giudice può scrivere più Valutazioni ▪ Chiavi esterne: `Username_giudice` e `Titolo_hackathon` in VALUTAZIONE ▪ Vincolo: Ogni Valutazione è scritta da un solo Giudice
Immissione Voto	GIUDICE ↔ VOTO	▪ Tipo: Uno-a-molti [* - 1] ▪ Un Giudice può assegnare più Voti ▪ Chiavi esterne: `Username_giudice` e `Titolo_hackathon` in VOTO ▪ Vincolo: Ogni Voto è assegnato da un solo Giudice
Assegnazione Voto	TEAM ↔ VOTO	▪ Tipo: Uno-a-molti [1..* - 1] ▪ Un Team può ricevere più Voti ▪ Chiavi esterne: `Nome_team` e `Titolo_hackathon` in VOTO ▪ Vincolo: Ogni Voto è assegnato a un solo Team

3.5 Dizionario dei Vincoli

Nome Vincolo	Descrizione
Password Sicura	La password di un utente deve avere almeno una lettera maiuscola, una lettera minuscola, un carattere numerico, un carattere speciale e che non sia più breve di 8 caratteri
Chiusura Registrazioni per Data	Le registrazioni devono chiudersi 2 giorni prima dell'evento: `DataFine_registrazione` = `DataInizio_evento` - 2 giorni
Chiusura Registrazioni per Limite	Le registrazioni devono chiudersi quando viene raggiunto il limite massimo di iscritti all'Hackathon
Coerenza Date	Il periodo di registrazione all'evento non può né combaciare ne susseguire il periodo di durata dell'evento stesso. Le registrazioni possono avvenire solo prima dell'evento e si chiudono esattamente due giorni prima. Le date devono essere di per sé coerenti anche nei singoli casi: l'apertura delle registrazioni non può avvenire dopo la sua chiusura e viceversa, discorso analogo per l'evento.
Coerenza Data di Adesione	Non può esserci una data di adesione in membership che vada oltre la chiusura delle registrazioni o prima dell'apertura delle registrazioni.
Range Voti	I voti assegnati dai giudici devono essere interi compresi tra 0 e 10
Numero Minimo Membri Team	Ogni team deve avere almeno 2 membri per essere valido
Numero Massimo Membri Team	Un team non può superare il numero massimo di membri definito nell'Hackathon
Documentazione Obbligatoria	Ogni team deve aver caricato almeno un documento per essere valutato
Unicità Voto Giudice	Un giudice non può votare più volte lo stesso team
Classifica Automatica	La classifica deve essere generata automaticamente come somma dei punteggi
Completezza Valutazioni	Tutti i giudici devono aver votato prima della pubblicazione della classifica
Blocco Team	I team non possono essere modificati dopo la chiusura delle registrazioni

4 Progettazione Logica

4.1 Schema Logico

- Hackathon(Titolo_identificativo, Organizzatore, Sede, Classifica, DataInizio_registrazione, DataFine_registrazione, Descrizione_problema, DataInizio_evento, DataFine_evento, NumIscritti_corrente, MaxNum_iscritti, MaxNum_membriTeam)

  ```
  Organizzatore ⇒ Organizzatore.Username_org
  ```

- Utente(Username, Password)
- Organizzatore(Username_org, Password)
- Giudice(Username_utente, Titolo_hackathon)

  ```
  Username_utente ⇒ Utente.Username
  Titolo_hackathon ⇒ Hackathon.Titolo_identificativo
  ```

- Team(Nome_team, Titolo_hackathon, Punteggio_finale)

  ```
  Titolo_hackathon ⇒ Hackathon.Titolo_identificativo
  ```

- Documento(ID_documento, Nome_team, Titolo_hackathon, Titolo_doc, Contenuto, Data_stesura)

  ```
  Nome_team ⇒ Team.Nome_team
  Titolo_hackathon ⇒ Hackathon.Titolo_identificativo
  ```

- Invito_giudice(Username_organizzatore, Username_utente, Titolo_hackathon, Stato_invito, Data_invito)

  ```
  Username_organizzatore ⇒ Organizzatore.Username_org
  Username_utente ⇒ Utente.Username
  Titolo_hackathon ⇒ Hackathon.Titolo_identificativo
  ```

- Membership(ID_adesione, Username_utente, Team_appartenenza, Titolo_hackathon, Data_adesione)

  ```
  Username_utente ⇒ Utente.Username
  Team_appartenenza ⇒ Team.Nome_team
  Titolo_hackathon ⇒ Hackathon.Titolo_identificativo
  ```

- Voto(Username_giudice, Titolo_hackathon, Team_votato, Punteggio)

  ```
  Username_giudice ⇒ Giudice.Username_utente
  Titolo_hackathon ⇒ Giudice.Titolo_hackathon
  Team_votato ⇒ Team.Nome_team
  ```

- Valutazione(Username_giudice,Titolo_hackathon, Team_valutato, ID_documento, Valutazione_giudice)

  ```
  Username_giudice ⇒ Giudice.Username_utente
  Titolo_hackathon ⇒ Giudice.Titolo_hackathon
  Team_valutato ⇒ Team.Nome_team
  ID_documento ⇒ Documento.ID_documento
  ```

5 Progettazione Fisica

5.1 Definzione Tabelle

In questa sezione verranno elencate le definizioni di tutte le tabelle che comporranno la nostra struttura.

5.1.1 Definizione di Organizzatore

```
1   CREATE TABLE ORGANIZZATORE
2   (
3     Username_org VARCHAR(30) PRIMARY KEY,
4     Password VARCHAR(30) NOT NULL,
5     -- Controllo validita password
6     CONSTRAINT chk_password_complexity
7     CHECK (
8       -- Lunghezza minima 8 caratteri
9       LENGTH(Password) >= 8 AND
10      -- Almeno una lettera maiuscola
11      Password ~ '[A-Z]' AND
12      -- Almeno una lettera minuscola
13      Password ~ '[a-z]' AND
14      -- Almeno un numero
15      Password ~ '[0-9]' AND
16      -- Almeno un carattere speciale tra quelli consentiti
17      Password ~ '[!@#$%^&*()\-_=+{};:,<.>/?]'
18    )
19  );
```

5.1.2 Definizione di Utente

```
1   CREATE TABLE UTENTE
2   (
3     Username VARCHAR(30) PRIMARY KEY,
4     Password VARCHAR(30) NOT NULL,
5     -- Controllo validita password
6     CONSTRAINT chk_password_complexity
7     CHECK (
8       -- Lunghezza minima 8 caratteri
9       LENGTH(Password) >= 8 AND
10      -- Almeno una lettera maiuscola
11      Password ~ '[A-Z]' AND
12      -- Almeno una lettera minuscola
13      Password ~ '[a-z]' AND
14      -- Almeno un numero
15      Password ~ '[0-9]' AND
16      -- Almeno un carattere speciale tra quelli consentiti
17      Password ~ '[!@#$%^&*()\-_=+{};:,<.>/?]'
18    )
19  );
```

5.1.3 Definizione di Hackathon

```
CREATE TABLE HACKATHON
(
  Titolo_identificativo VARCHAR(30) PRIMARY KEY,
  Organizzatore VARCHAR(30) NOT NULL,
  Sede VARCHAR(30) NOT NULL,
  Classifica TEXT,
  DataInizio_registrazione DATE NOT NULL,
  DataFine_registrazione DATE NOT NULL,
  DataInizio_evento DATE NOT NULL,
  DataFine_evento DATE NOT NULL,
  Descrizione_problema TEXT NOT NULL,
  NumIscritti_corrente INTEGER,
  MaxNum_iscritti INTEGER NOT NULL,
  MaxNum_membriTeam INTEGER NOT NULL,

  FOREIGN KEY (Organizzatore) REFERENCES ORGANIZZATORE(Username_org) ON DELETE
      RESTRICT,

  -- Vincolo: la registrazione termina almeno 2 giorni prima dell'inizio dell'
      evento
  CHECK (DataFine_registrazione <= DataInizio_evento - INTERVAL '2 days'),

  -- Vincolo: l'intera registrazione deve avvenire prima dell'evento
  CHECK (DataFine_registrazione < DataInizio_evento AND DataInizio_registrazione
      < DataInizio_evento),

  -- Vincolo: le date devono risultare coerenti
  CHECK (DataInizio_registrazione < DataFine_registrazione AND DataInizio_evento
      < DataFine_evento)
);
```

5.1.4 Definizione di Giudice

```
CREATE TABLE GIUDICE
(
  Username_utente VARCHAR(30) NOT NULL,
  Titolo_hackathon VARCHAR(30) NOT NULL,

  PRIMARY KEY (Username_utente, Titolo_hackathon),

  FOREIGN KEY (Username_utente) REFERENCES UTENTE (Username)
  ON DELETE CASCADE,
  FOREIGN KEY (Titolo_hackathon) REFERENCES HACKATHON (Titolo_identificativo)
  ON DELETE CASCADE
);
```

5.1.5 Definizione di Team

```
CREATE TABLE TEAM
(
  Nome_team VARCHAR(30) UNIQUE NOT NULL,
  Punteggio_finale INTEGER,
  Titolo_hackathon VARCHAR(30) NOT NULL,

  PRIMARY KEY(Nome_team, Titolo_hackathon),

  FOREIGN KEY (Titolo_hackathon) REFERENCES HACKATHON (Titolo_identificativo)
  ON DELETE CASCADE
);
```

5.1.6 Definizione di Documento

```
CREATE TABLE DOCUMENTO
(
  ID_documento SERIAL PRIMARY KEY,
  Nome_team VARCHAR(30) NOT NULL,
  Titolo_hackathon VARCHAR(30) NOT NULL,
  Titolo_doc VARCHAR(30) NOT NULL,
  Contenuto TEXT NOT NULL,
  -- MODIFICATO: Usiamo TIMESTAMPTZ per data e ora con fuso orario
  Data_stesura TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP,

  FOREIGN KEY (Nome_team, Titolo_hackathon) REFERENCES TEAM (Nome_team,
      Titolo_hackathon)
    ON DELETE CASCADE
);
```

5.1.7 Definizione di Membership

```
CREATE TABLE MEMBERSHIP
(
  ID_adesione SERIAL PRIMARY KEY,
  Username_utente VARCHAR(30) NOT NULL,
  Team_appartenenza VARCHAR(30) NOT NULL,
  Titolo_hackathon VARCHAR(30) NOT NULL,
  Data_adesione DATE NOT NULL,

  UNIQUE (Username_utente, Team_appartenenza,
      Titolo_hackathon),

  FOREIGN KEY (Username_utente)
    REFERENCES UTENTE (Username)
    ON DELETE CASCADE,
  FOREIGN KEY (Team_appartenenza, Titolo_hackathon)
    REFERENCES TEAM (Nome_team, Titolo_hackathon)
    ON DELETE CASCADE
);
```

5.1.8 Definizione di Voto

```
CREATE TABLE VOTO
(
  Username_giudice VARCHAR(30) NOT NULL,
  Titolo_hackathon VARCHAR(30) NOT NULL,
  Team_votato VARCHAR(30) NOT NULL,
  Punteggio INTEGER,

  UNIQUE (Username_giudice, Titolo_hackathon,
    Team_votato),

  FOREIGN KEY (Username_giudice, Titolo_hackathon)
    REFERENCES GIUDICE (Username_utente, Titolo_hackathon)
    ON DELETE CASCADE,
  FOREIGN KEY (Team_votato, Titolo_hackathon)
    REFERENCES TEAM (Nome_team, Titolo_hackathon)
    ON DELETE CASCADE,

  CHECK (Punteggio >= 0 AND Punteggio <= 10)
);
```

5.1.9 Definizione di Valutazione

```
CREATE TABLE VALUTAZIONE
(
  ID_documento INTEGER,
  Username_giudice VARCHAR(30) NOT NULL,
  Titolo_hackathon VARCHAR(30) NOT NULL,
  Team_valutato VARCHAR(30) NOT NULL,
  Valutazione_giudice TEXT,

  UNIQUE(ID_documento, Username_giudice,
    Titolo_hackathon, Team_valutato),

  FOREIGN KEY (ID_documento)
    REFERENCES DOCUMENTO (ID_documento)
    ON DELETE CASCADE,
  FOREIGN KEY (Team_valutato, Titolo_hackathon)
    REFERENCES TEAM (Nome_team, Titolo_hackathon)
    ON DELETE CASCADE,
  FOREIGN KEY (Username_giudice, Titolo_hackathon)
    REFERENCES GIUDICE (Username_utente, Titolo_hackathon)
    ON DELETE CASCADE
);
```

5.1.10 Definizione di Invito Giudice

```
CREATE TABLE INVITO_GIUDICE
(
  Username_organizzatore VARCHAR(30) NOT NULL,
  Username_utente VARCHAR(30) NOT NULL,
  Titolo_hackathon VARCHAR(30) NOT NULL,
  Data_invito DATE NOT NULL DEFAULT CURRENT_DATE,
  Stato_invito VARCHAR(20) NOT NULL
    CHECK (Stato_invito IN ('Inviato', 'Accettato', 'Rifiutato')),

  PRIMARY KEY (Username_utente, Titolo_hackathon),

  FOREIGN KEY (Username_organizzatore) REFERENCES ORGANIZZATORE(Username_org) ON
      DELETE CASCADE,
  FOREIGN KEY (Username_utente) REFERENCES UTENTE(Username) ON DELETE CASCADE,
  FOREIGN KEY (Titolo_hackathon) REFERENCES HACKATHON(Titolo_identificativo) ON
      DELETE CASCADE
);
```

5.2 Implementazioni Vincoli

In questa sezione verranno riportate le implementazioni dei vincoli più complessi che non sono già stati indicati nelle definizioni delle tabelle.

5.2.1 Implementazione Vincolo di Creazione Giudice su Accettazione Invito

```
CREATE OR REPLACE FUNCTION aggiungi_giudice()
RETURNS TRIGGER AS $$
BEGIN
  -- Controlla se lo stato dell'invito e' diventato 'Accettato'
  IF NEW.Stato_invito = 'Accettato' THEN
    -- Inserisce il nuovo giudice, solo se non esiste gia'
    INSERT INTO GIUDICE (Username_utente, Titolo_hackathon)
    VALUES (NEW.Username_utente, NEW.Titolo_hackathon)
    ON CONFLICT DO NOTHING; -- evita errore se gia' presente
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_aggiungi_giudice
AFTER UPDATE ON INVITO_GIUDICE
FOR EACH ROW
WHEN (OLD.Stato_invito IS DISTINCT FROM NEW.Stato_invito
  AND NEW.Stato_invito = 'Accettato')
EXECUTE FUNCTION aggiungi_giudice();
```

5.2.2 Implementazione Vincolo di Controllo sull'unicità di un Utente come Giudice

```
-- Funzione per verificare la sovrapposizione di date tra hackathon
CREATE OR REPLACE FUNCTION verifica_giudice_sovrapposizione()
RETURNS TRIGGER AS $$
DECLARE
  nuovo_inizio DATE;
  nuovo_fine DATE;
  conteggio INTEGER;
BEGIN
  -- Recupera le date dell'evento hackathon per cui l'utente sta diventando
      giudice
  SELECT h.DataInizio_evento, h.DataFine_evento
  INTO nuovo_inizio, nuovo_fine
  FROM HACKATHON h
  WHERE h.Titolo_identificativo = NEW.Titolo_hackathon;

  -- Verifica se l'utente e' gia' giudice per un altro hackathon con date
      sovrapposte
  SELECT COUNT(*)
  INTO conteggio
  FROM GIUDICE g
  JOIN HACKATHON h ON g.Titolo_hackathon = h.Titolo_identificativo
  WHERE g.Username_utente = NEW.Username_utente
  AND g.Titolo_hackathon <> NEW.Titolo_hackathon
  AND (
    -- Verifica sovrapposizione date
    (h.DataInizio_evento <= nuovo_fine AND h.DataFine_evento >= nuovo_inizio)
  );

  -- Se c'e' sovrapposizione, genera un errore
  IF conteggio > 0 THEN
  RAISE EXCEPTION 'L''utente % non puo'' essere giudice per questo hackathon
    perche'' e'' gia'' giudice per un hackathon con date sovrapposte',
    NEW.Username_utente;
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Trigger per controllare la sovrapposizione quando un utente diventa giudice
```

```
39    CREATE TRIGGER trigger_verifica_giudice_sovrapposizione
40    BEFORE INSERT OR UPDATE ON GIUDICE
41    FOR EACH ROW
42    EXECUTE FUNCTION verifica_giudice_sovrapposizione();
```

5.2.3 Implementazione Vincolo di Controllo sulla validità di Adesione di un Utente ad un Team

```
1     -- Funzione per verificare se un'adesione e' valida
2     CREATE OR REPLACE FUNCTION verifica_adesione_valida()
3     RETURNS TRIGGER AS $$
4     DECLARE
5       data_fine_registrazione DATE;
6       num_membri_attuali INTEGER;
7       max_membri INTEGER;
8       num_iscritti_corrente INTEGER;
9       max_iscritti INTEGER;
10    BEGIN
11      -- Recupera tutti i dati necessari dall'hackathon
12      SELECT h.DataFine_registrazione, h.MaxNum_membriTeam,
13      COALESCE(h.NumIscritti_corrente, 0), h.MaxNum_iscritti
14      INTO data_fine_registrazione, max_membri, num_iscritti_corrente, max_iscritti
15      FROM HACKATHON h
16      WHERE h.Titolo_identificativo = NEW.Titolo_hackathon;
17
18      -- Verifica se la data attuale e' successiva alla data di fine registrazione
19      IF CURRENT_DATE > data_fine_registrazione THEN
20        RAISE EXCEPTION 'Non e'' possibile aderire al team:
21          le registrazioni per l''hackathon "%" sono chiuse dal %',
22        NEW.Titolo_hackathon, data_fine_registrazione;
23      END IF;
24
25      -- Verifica se e' stato raggiunto il numero massimo
26      -- di iscritti all'hackathon
27      IF num_iscritti_corrente >= max_iscritti THEN
28        RAISE EXCEPTION 'Non e'' possibile aderire al team:
29          l''hackathon "%" ha raggiunto il numero massimo di partecipanti (%)',
30        NEW.Titolo_hackathon, max_iscritti;
31      END IF;
32
33      -- Conta il numero di membri attuali nel team
34      SELECT COUNT(*)
35      INTO num_membri_attuali
36      FROM MEMBERSHIP m
37      WHERE m.Team_appartenenza = NEW.Team_appartenenza
38      AND m.Titolo_hackathon = NEW.Titolo_hackathon;
39
40      -- Verifica se il team e' gia' al completo
41      IF num_membri_attuali >= max_membri THEN
42        RAISE EXCEPTION 'Non e'' possibile aderire al team "%":
43          il team ha gia'' raggiunto il numero massimo di membri (%)',
44        NEW.Team_appartenenza, max_membri;
45      END IF;
46
47      -- Tutto ok, incrementa il contatore di iscritti
48      UPDATE HACKATHON
49      SET NumIscritti_corrente = COALESCE(NumIscritti_corrente, 0) + 1
50      WHERE Titolo_identificativo = NEW.Titolo_hackathon;
51
52      -- Se tutte le verifiche sono passate, permetti l'inserimento
53      RETURN NEW;
54    END;
55    $$ LANGUAGE plpgsql;
56
57    -- Trigger per controllare l'adesione al team
58    CREATE TRIGGER trigger_verifica_adesione_valida
59    BEFORE INSERT ON MEMBERSHIP
60    FOR EACH ROW
61    EXECUTE FUNCTION verifica_adesione_valida();
```

5.2.4 Implementazione Vincolo di Controllo sui Team senza membri sufficienti

```
-- Funzione per verificare e cancellare team incompleti
CREATE OR REPLACE FUNCTION elimina_team_incompleti()
RETURNS void AS $$
DECLARE
  hackathon_record RECORD;
BEGIN
  -- Trova solo gli hackathon la cui data di fine registrazione e' OGGI
  FOR hackathon_record IN (
    SELECT h.Titolo_identificativo
    FROM HACKATHON h
    WHERE h.DataFine_registrazione = CURRENT_DATE
  ) LOOP
    -- Elimina direttamente i team con meno di 2 membri
    DELETE FROM TEAM t
    WHERE t.Titolo_hackathon = hackathon_record.Titolo_identificativo
    AND (
      SELECT COUNT(*)
      FROM MEMBERSHIP m
      WHERE m.Team_appartenenza = t.Nome_team
      AND m.Titolo_hackathon = t.Titolo_hackathon
    ) < 2;

    RAISE NOTICE 'Eliminati tutti i team incompleti per l''hackathon "%"',
    hackathon_record.Titolo_identificativo;
  END LOOP;

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

5.2.5 Implementazione Vincolo di Controllo sui Team senza Documenti caricati

```
-- Funzione per verificare che un team abbia caricato
-- almeno un documento prima di ricevere un voto
CREATE OR REPLACE FUNCTION verifica_documento_caricato()
RETURNS TRIGGER AS $$
DECLARE
  documenti_count INTEGER;
BEGIN
  -- Conta quanti documenti ha caricato il team
  SELECT COUNT(*)
  INTO documenti_count
  FROM DOCUMENTO d
  WHERE d.nome_team = NEW.team_votato
  AND d.Titolo_hackathon = NEW.Titolo_hackathon;

  -- Se il team non ha caricato alcun documento,
  -- impedisci l'inserimento del voto
  IF documenti_count = 0 THEN
    RAISE EXCEPTION 'Impossibile votare il team "%":
      non ha caricato alcun documento per l''hackathon "%"',
    NEW.team_votato, NEW.Titolo_hackathon;
  END IF;

  -- Se la verifica e' passata, permetti l'inserimento del voto
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Trigger per controllare che un team abbia
-- caricato documenti prima di ricevere un voto
CREATE TRIGGER trigger_verifica_documento_caricato
BEFORE INSERT ON VOTO
FOR EACH ROW
EXECUTE FUNCTION verifica_documento_caricato();
```

5.2.6 Implementazione Funzione di Generazione della classifica finale con annessi vincoli sulla fine dell'evento e sul controllo dei giudici senza voto

```
-- Funzione per generare la classifica finale dell'hackathon
CREATE OR REPLACE FUNCTION genera_classifica_hackathon(titolo_hack VARCHAR(30))
RETURNS TEXT AS $$
DECLARE
  classifica_text TEXT := '';
  team_record RECORD;
  posizione INTEGER := 1;
  data_fine_evento DATE;
  num_giudici INTEGER;
  num_team INTEGER;
  num_voti INTEGER;
  voti_attesi INTEGER;
BEGIN
  -- Verifica che l'hackathon esista
  IF NOT EXISTS (SELECT 1 FROM HACKATHON WHERE Titolo_identificativo =
      titolo_hack) THEN
    RETURN 'Errore: Hackathon non trovato';
  END IF;

  -- Recupera la data di fine evento dell'hackathon
  SELECT DataFine_evento INTO data_fine_evento
  FROM HACKATHON
  WHERE Titolo_identificativo = titolo_hack;

  -- Verifica che l'hackathon sia terminato
  IF CURRENT_DATE < data_fine_evento THEN
    RETURN 'Errore: Non e'' possibile generare la classifica prima della fine
        dell''hackathon';
  END IF;

  -- Conta il numero di giudici per questo hackathon
  SELECT COUNT(*) INTO num_giudici
  FROM GIUDICE
  WHERE Titolo_hackathon = titolo_hack;

  -- Conta il numero di team per questo hackathon
  SELECT COUNT(*) INTO num_team
  FROM TEAM
  WHERE Titolo_hackathon = titolo_hack;

  -- Conta il numero totale di voti espressi
  SELECT COUNT(*) INTO num_voti
  FROM VOTO
  WHERE Titolo_hackathon = titolo_hack;

  -- Calcola il numero di voti attesi (ogni giudice deve votare ogni team)
  voti_attesi := num_giudici * num_team;

  -- Verifica se tutti i giudici hanno espresso il proprio voto per tutti i team
  IF num_voti < voti_attesi THEN
    RETURN 'Errore: Non e'' possibile generare la classifica. Mancano ' ||
      (voti_attesi - num_voti) || ' voti su ' || voti_attesi || ' attesi. Tutti i
          giudici devono votare tutti i team.';
  END IF;

  -- Aggiorna il punteggio finale di ciascun team
  UPDATE TEAM t
  SET Punteggio_finale = (
    SELECT COALESCE(SUM(v.Punteggio), 0)
    FROM VOTO v
    WHERE v.Team_votato = t.Nome_team
    AND v.Titolo_hackathon = t.Titolo_hackathon
  )
  WHERE t.Titolo_hackathon = titolo_hack;

  -- Costruisce la stringa della classifica
  FOR team_record IN (
    SELECT
      t.Nome_team,
      t.Punteggio_finale,
```

```
68          MIN(d.Data_stesura) as prima_consegna
69        FROM TEAM t
70        LEFT JOIN DOCUMENTO d ON t.Nome_team = d.Nome_team AND t.Titolo_hackathon = d
              .Titolo_hackathon
71        WHERE t.Titolo_hackathon = titolo_hack
72        GROUP BY t.Nome_team, t.Punteggio_finale
73        ORDER BY
74          t.Punteggio_finale DESC, -- Regola 1: Punteggio piu' alto prima
75          prima_consegna ASC,      -- Regola 2 (Spareggio): Timestamp di consegna
                anteriore prima
76          t.Nome_team ASC          -- Regola 3 (Spareggio finale): Ordine alfabetico
77      ) LOOP
78        classifica_text := classifica_text ||
79                posizione || ' ' ||
80                team_record.Nome_team || ' ' ||
81                COALESCE(team_record.Punteggio_finale, 0) || E'\n';
82        posizione := posizione + 1;
83      END LOOP;
84
85      -- Rimuovi l'ultimo carattere newline se la classifica non e' vuota
86      IF LENGTH(classifica_text) > 0 THEN
87        classifica_text := SUBSTRING(classifica_text, 1, LENGTH(classifica_text) - 1)
              ;
88      END IF;
89
90      -- Aggiorna il campo Classifica nella tabella HACKATHON
91      UPDATE HACKATHON
92      SET Classifica = classifica_text
93      WHERE Titolo_identificativo = titolo_hack;
94
95      -- Registra l'utente e la data/ora di generazione della classifica
96      RAISE NOTICE 'Classifica generata da % il % UTC',
97            CURRENT_USER, TO_CHAR(CURRENT_TIMESTAMP AT TIME ZONE 'UTC', 'YYYY-MM-DD
                HH24:MI:SS');
98
99      RETURN classifica_text;
100   END;
101   $$ LANGUAGE plpgsql;
```

5.3 Implementazione Funzioni Aggiuntive

5.3.1 Implementazione Funzione per la rimozione di un Utente da un Team

```
-- Funzione per decrementare il contatore
-- quando un utente viene rimosso da un team
CREATE OR REPLACE FUNCTION gestisci_rimozione_iscritto()
RETURNS TRIGGER AS $$
BEGIN
  -- Decrementa il contatore degli iscritti correnti
  UPDATE HACKATHON
  SET NumIscritti_corrente = GREATEST(COALESCE(NumIscritti_corrente, 0) - 1, 0)
  WHERE Titolo_identificativo = OLD.Titolo_hackathon;

  RETURN OLD;
END;
$$ LANGUAGE plpgsql;

-- Crea trigger per decrementare il contatore quando un membro viene rimosso
CREATE TRIGGER trigger_gestisci_rimozione_iscritto
AFTER DELETE ON MEMBERSHIP
FOR EACH ROW
EXECUTE FUNCTION gestisci_rimozione_iscritto();
```

5.3.2 Implementazione Funzione per la rimozione di un Team da un Hackathon

```
-- Funzione per gestire l'eliminazione di team completi
CREATE OR REPLACE FUNCTION gestisci_eliminazione_team()
RETURNS TRIGGER AS $$
DECLARE
  num_membri INTEGER;
BEGIN
  -- Conta quanti membri ha il team
  SELECT COUNT(*) INTO num_membri
  FROM MEMBERSHIP
  WHERE Team_appartenenza = OLD.Nome_team
  AND Titolo_hackathon = OLD.Titolo_hackathon;

  -- Decrementa il contatore degli iscritti per ogni membro del team
  -- Nota: facciamo l'update direttamente qui perche' il team
  -- e tutti i suoi membri verranno eliminati insieme
  -- a causa del vincolo ON DELETE CASCADE
  UPDATE HACKATHON
  SET NumIscritti_corrente = GREATEST(COALESCE(NumIscritti_corrente, 0) -
      num_membri, 0)
  WHERE Titolo_identificativo = OLD.Titolo_hackathon;

  RETURN OLD;
END;
$$ LANGUAGE plpgsql;

-- Crea trigger per gestire l'eliminazione di team completi
CREATE TRIGGER trigger_gestisci_eliminazione_team
BEFORE DELETE ON TEAM
FOR EACH ROW
EXECUTE FUNCTION gestisci_eliminazione_team();
```

5.3.3 Implementazione Funzione per l'inizializzazione del contatore degli iscritti

```
1   -- Funzione per inizializzare il contatore degli iscritti
2   CREATE OR REPLACE FUNCTION inizializza_contatore_iscritti()
3   RETURNS TRIGGER AS $$
4   BEGIN
5     -- Imposta il contatore degli iscritti a 0 se e' NULL
6     NEW.NumIscritti_corrente := 0;
7     RETURN NEW;
8   END;
9   $$ LANGUAGE plpgsql;
10
11  -- Trigger per inizializzare il contatore
12  CREATE TRIGGER trigger_inizializza_contatore_iscritti
13  BEFORE INSERT ON HACKATHON
14  FOR EACH ROW
15  EXECUTE FUNCTION inizializza_contatore_iscritti();
```