



UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

Progetto di Basi di Dati A.A. 2024-2025

Elaborato di gruppo (Traccia 2)

Palma Michele – N86005009
Villano Sara – N86005045

Sommario

1.	Introduzione.....	5
1.1.	Descrizione del problema	5
2.	Progettazione concettuale	6
2.1.	Schema concettuale	6
3.	Ristrutturazione dello Schema Concettuale	7
3.1.	Analisi delle chiavi	7
3.2.	Analisi degli attributi derivati.....	8
3.3.	Analisi delle ridondanze	8
3.4.	Analisi degli attributi strutturati	8
3.5.	Analisi degli attributi a valore multiplo.....	9
3.6.	Analisi delle gerarchie di specializzazione.....	9
3.7.	Schema concettuale ristrutturato ER	10
3.8.	Schema concettuale ristrutturato UML	10
4.	Dizionari	11
4.1.	Dizionario delle classi	11
4.2.	Dizionario delle associazioni.....	14
4.3.	Dizionario dei vincoli	16
5.	Progettazione logica	21
5.1.	Schema logico.....	21
6.	Progettazione fisica.....	23
6.1.	Definizione tabelle.....	23
6.1.1.	Definizione della tabella Utente	23
6.1.2.	Definizione della tabella Organizzatore	23
6.1.3.	Definizione della tabella Giudice	24
6.1.4.	Definizione della tabella Invito	24
6.1.5.	Definizione della tabella Giudica	24
6.1.6.	Definizione della tabella Evento.....	25

6.1.7.	Definizione della tabella Iscrizione	25
6.1.8.	Definizione della tabella Partecipante	26
6.1.9.	Definizione della tabella Team	26
6.1.10.	Definizione della tabella Vota	26
6.1.11.	Definizione della tabella Documento.....	27
6.1.12.	Definizione della tabella Commenta.....	27
6.1.13.	Definizione della tabella Partecipazione.....	27
7.	Trigger	28
7.1.	Dizionario dei trigger.....	28
7.2.	Implementazione dei trigger.....	32
7.2.1.	Trigger enforce_eta_minima_utente.....	32
7.2.2.	Trigger enforce_prerequisiti_organizzatore	32
7.2.3.	Trigger enforce_prerequisiti_giudice	33
7.2.4.	Trigger enforce_setta_problema	33
7.2.5.	Trigger enforce_evento_futuro_dinamico	34
7.2.6.	Trigger enforce_assegnazione_team.....	34
7.2.7.	Trigger enforce_max_team_size.....	35
7.2.8.	Trigger enforce_invito_prima_inizio.....	35
7.2.9.	Trigger enforce_invito_precondizioni	36
7.2.10.	Trigger enforce_iscrizione_apertura	37
7.2.11.	Trigger enforce_iscrizione_ruoli	37
7.2.12.	Trigger enforce_max_iscritti_evento	38
7.2.13.	Trigger enforce_limite_team_periodo.....	38
7.2.14.	Trigger enforce_limite_upload_documento	39
7.2.15.	Trigger enforce_problema_pubblicato_upload	39
7.2.16.	Trigger enforce_commenta_periodo	40
7.2.17.	Trigger enforce_invito_giudica	40
7.2.18.	Trigger enforce_update_ruolo_giudice	41
7.2.19.	Trigger enforce_voto_post_evento.....	41

7.2.20.	Trigger enforce_unique_autore_problema	41
8.	Funzioni	42
8.1.	Dizionario delle funzioni	42
8.2.	Implementazione delle funzioni	45
8.2.1.	Funzione registra_utente.....	45
8.2.2.	Funzione crea_evento_e_organizzatore	46
8.2.3.	Funzione crea_invito_giudice.....	46
8.2.4.	Funzione accetta_invito_giudice	47
8.2.5.	Funzione rifiuta_invito_giudice	47
8.2.6.	Funzione iscrivi_utente_evento	48
8.2.7.	Funzione crea_team	48
8.2.8.	Funzione iscrivi_team.....	49
8.2.9.	Funzione team_con_autoassegnazione	50
8.2.10.	Funzione setta_ruolo_autore_problema	51
8.2.11.	Funzione setta_problema_evento	51
8.2.12.	Funzione corregge_problema_evento	52
8.2.13.	Funzione carica_documento_team.....	52
8.2.14.	Funzione commenta_documento	53
8.2.15.	Funzione assegna_voto_team_unico.....	53
8.2.16.	Funzione get_potenziali_giudici	54
8.2.17.	View classifica_finale_team.....	54

1. Introduzione

Il seguente elaborato descrive la progettazione concettuale di un database relazionale finalizzato alla gestione di eventi hackathon. Un hackathon è un evento collaborativo e competitivo in cui team di partecipanti si sfidano per sviluppare soluzioni innovative a problemi complessi, in un arco di tempo limitato, di solito 2 giorni. La crescente popolarità di questi eventi comporta la creazione di un'infrastruttura di dati robusta, scalabile e coerente, in grado di supportare tutte le fasi di un hackathon dal pre-evento alla valutazione finale dei progetti.

1.1. Descrizione del problema

L'obiettivo principale di questo progetto è tradurre i requisiti di un hackathon in uno schema di database logico e ottimizzato, il quale deve essere in grado di gestire in modo strutturato le seguenti dinamiche:

- Gestione dei ruoli utente: l'entità Utente deve poter assumere ruoli specializzati (Organizzatore, Partecipante, Giudice) in modo flessibile. Il database deve permettere a un singolo utente di essere associato a ruoli multipli in eventi diversi.
- Vita dell'evento: l'entità Evento deve contenere tutte le informazioni essenziali (titolo, sede, data inizio e fine registrazioni, data inizio e fine evento...) e deve essere collegata ad un Organizzatore specifico.
- Iscrizione e formazione dei team: il database deve tracciare le iscrizioni degli utenti a specifici eventi e la successiva formazione dei team. La struttura deve garantire che un partecipante appartenga ad un solo team per evento.
- Valutazione progressi: il sistema di valutazione è centrale. I giudici devono essere collegati ai team e ai documenti in modo da poter registrare voti e commenti in modo persistente e non ambiguo.

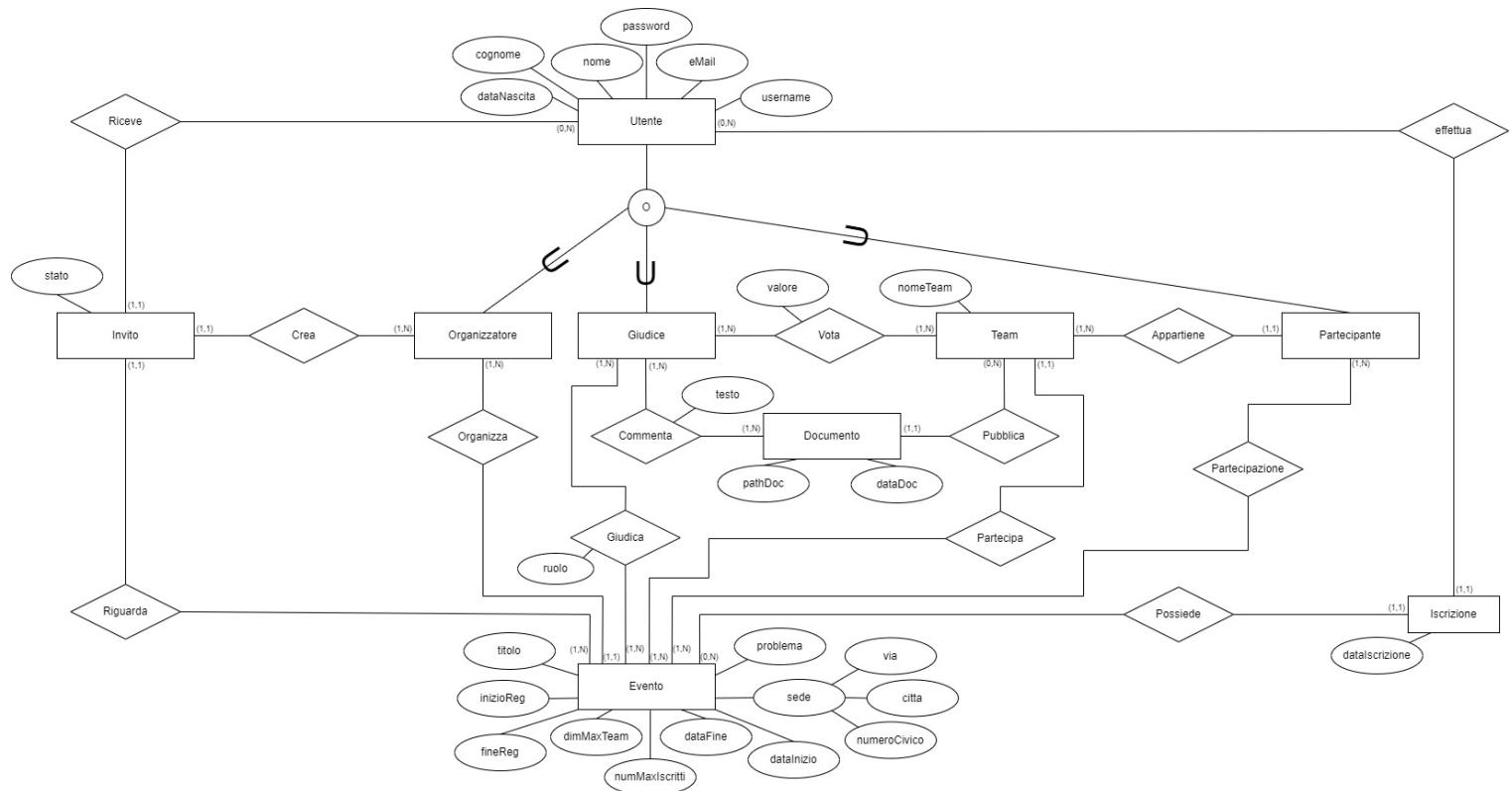
In sintesi, l'obiettivo è produrre un modello di database pulito e documentato che funga da scheletro per un sistema di gestione hackathon, garantendo che ogni informazione sia memorizzata in modo logico e facilmente accessibile.

2. Progettazione concettuale

Questo capitolo documenta la fase di progettazione concettuale del database per la gestione di eventi hackathon. L'obiettivo è tradurre i requisiti descritti nella traccia in uno schema concettuale, che sia indipendente dalla tecnologia di implementazione. Il risultato di questa fase è il diagramma entità-relazione (ER), che visualizza le entità principali del sistema, i loro attributi e le relazioni che le collegano.

Questo modello è la base per le successive fasi di progettazione logica e fisica e serve a validare la correttezza della struttura dei dati prima di procedere con la creazione effettiva del database.

2.1. Schema concettuale



3. Ristrutturazione dello Schema Concettuale

In questa fase si procede alla ristrutturazione dello schema concettuale elaborato precedentemente. L'obiettivo principale è ottimizzare il modello al fine di renderlo più efficiente e idoneo alla successiva traduzione in uno schema relazionale. La ristrutturazione è stata condotta seguendo i seguenti punti chiave:

1. Analisi delle chiavi
2. Analisi degli attributi derivati
3. Analisi delle ridondanze
4. Analisi degli attributi strutturati
5. Analisi degli attributi a valore multiplo
6. Analisi delle gerarchie di specializzazione

3.1. Analisi delle chiavi

Per garantire una gestione efficiente è risultata conveniente l'introduzione di chiavi primarie surrogate, cioè identificativi di tipo intero associati a ciascuna istanza delle entità. Le abbiamo introdotte per la maggior parte delle entità, abbiamo idUtente in Utente, idOrganizzatore in Organizzatore, idPartecipante in Partecipante, idInvito in Invito, idEvento in Evento, idTeam in Team, idDocumento in Documento e idIscrizione in Iscrizione. Questo approccio rende l'identificazione di ciascuna istanza computazionalmente meno dispendiosa e migliora l'efficienza del database. Per le associazioni con attributi come Vota, Commenta e Giudica è stata definita una chiave primaria composta. Ciò garantisce che ogni combinazione di idGiudice e idTeam in Vota, di idGiudice e idDocumento in Commenta e di idGiudice e idEvento in Giudica sia unica, impedendo così che un giudice possa votare lo stesso team, commentare lo stesso documento, o essere assegnato come giudice allo stesso evento più di una volta. Mentre per l'associazione Partecipazione è stata definita una chiave primaria composta (idPartecipante, idEvento) che garantisce che un partecipante sia registrato ad un determinato evento una sola volta.

3.2. Analisi degli attributi derivati

In base alla traccia, non sono stati identificati attributi derivati nel modello. L'analisi si è quindi concentrata su un'attenta valutazione di ogni attributo per assicurare che non fosse calcolabile da altri dati già presenti nel sistema, evitando così possibili ridondanze.

3.3. Analisi delle ridondanze

Il processo di analisi delle ridondanze si è concentrato principalmente sul garantire che le informazioni non fossero duplicate all'interno dello schema. In particolare:

- Le informazioni anagrafiche degli utenti (nome, cognome, email, ecc.) sono centralizzate nell'entità Utente.
- Le entità dei ruoli (Organizzatore, Giudice, Partecipante) si collegano all'Utente tramite una chiave esterna (idUtente), evitando di dover duplicare le informazioni personali in più tabelle.

In particolare, l'analisi delle ridondanze ha riconosciuto la presenza di tabelle di associazione (Partecipazione e Giudica) che sebbene possano apparire ridondanti sono necessarie per risolvere le relazioni M:N tra i ruoli specializzati e l'entità Evento. Difatti questa struttura è fondamentale per:

- Giudica: in modo da tracciare l'assegnazione esatta di un giudice ad uno specifico evento e consentire l'introduzione di attributi come ruolo che serve per determinare quale giudice ha il compito di stabilire il problema da affrontare all'inizio dell'evento.
- Partecipazione: così da tenere traccia degli eventi a cui prendono parte dei determinati partecipanti.

3.4. Analisi degli attributi strutturati

Viene qui analizzata la presenza di eventuali attributi strutturati i quali non sono logicamente rappresentabili all'interno di un DBMS. Difatti attributi strutturati, come la sede dell'evento, sono stati decomposti in attributi più semplici. La sede è stata suddivisa in via, numero civico e città, rendendo i dati più gestibili e ricercabili in modo indipendente.

3.5. Analisi degli attributi a valore multiplo

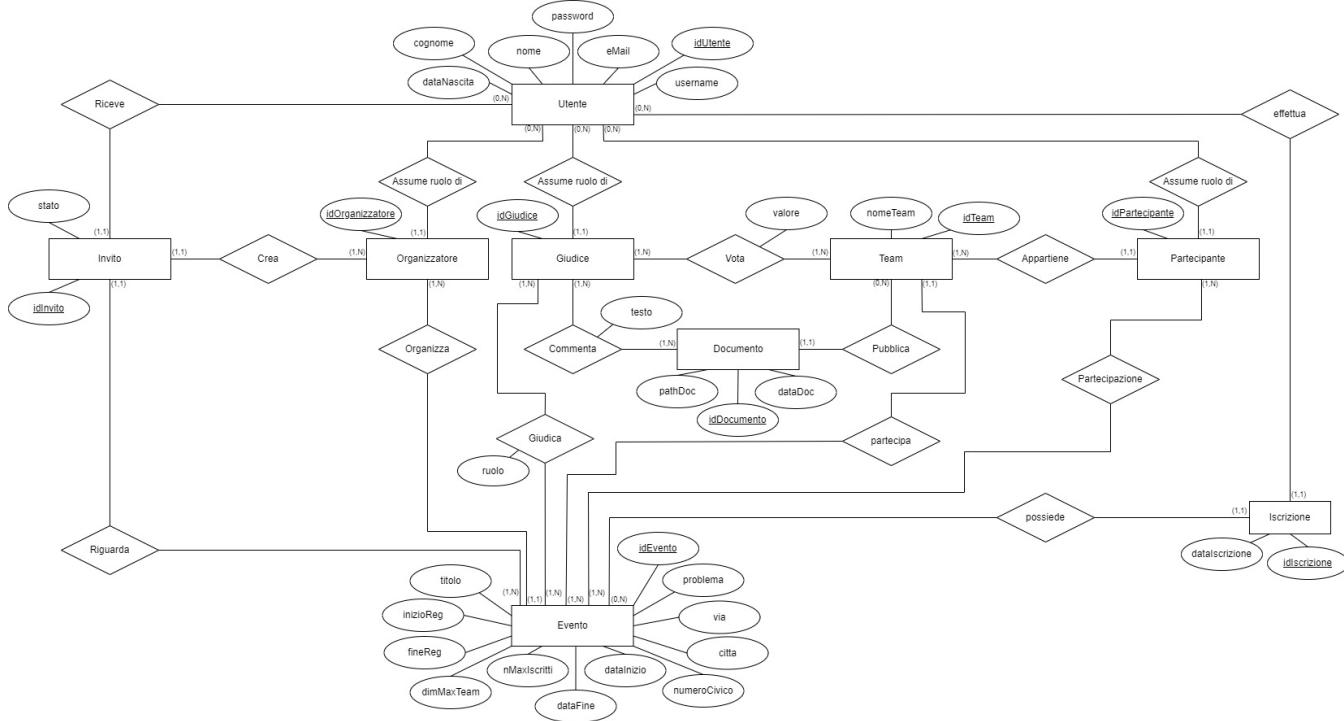
Non sono stati identificati attributi a valore multiplo nel modello. Tutti gli attributi sono stati definiti come a singolo valore, in modo da garantire la coerenza dei dati e facilitare la successiva fase di traduzione in uno schema relazionale.

3.6. Analisi delle gerarchie di specializzazione

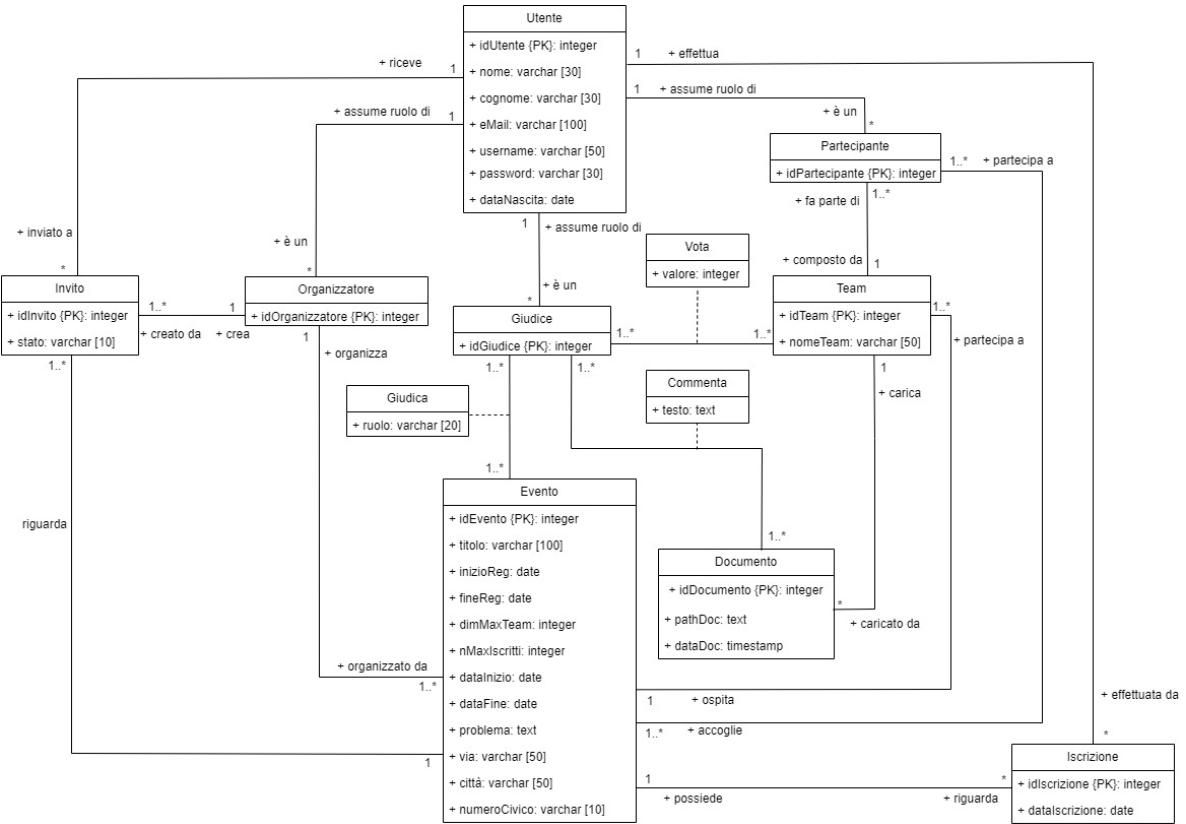
Infine, abbiamo analizzato la presenza di eventuali gerarchie di specializzazione, anch'esse non rappresentabili in un DBMS relazionale. È stata approfondita la gerarchia di specializzazione tra l'entità generica Utente e i ruoli specializzati (Organizzatore, Partecipante, Giudice). Abbiamo deciso di risolvere ciò utilizzando le associazioni, cioè, creando entità separate per ogni ruolo e collegandole all'entità Utente. Questa scelta è stata motivata da precise considerazioni:

- Flessibilità e dinamicità: un singolo Utente può assumere uno o più ruoli in momenti e contesti diversi (ad esempio può essere partecipante in un evento e giudice in un altro). Modellare i ruoli come entità separate collegate da associazioni risolve in modo elegante e flessibile questa relazione.
- Efficienza: centralizzando i dati anagrafici (nome, e-mail, password...) nell'entità utente, si evitano ridondanze e si semplificano gli aggiornamenti. Le entità dei ruoli contengono solo le informazioni specifiche di quel ruolo, mantenendo lo schema pulito e ottimizzato.
- Gestione delle funzionalità: questa struttura permette di tener traccia in modo chiaro e distinto delle funzionalità e delle azioni che ogni ruolo deve svolgere.
- Scalabilità: questo modello facilita l'aggiunta di nuovi ruoli in futuro senza dover alterare la struttura fondamentale della tabella Utente, garantendo una maggiore scalabilità del database.

3.7. Schema concettuale ristrutturato ER



3.8. Schema concettuale ristrutturato UML



4. Dizionari

4.1. Dizionario delle classi

Classe	Descrizione	Attributi
Utente	Rappresenta un utente generico registrato nel sistema. Può assumere ruoli differenti: partecipante, giudice, organizzatore.	<ul style="list-style-type: none">○ idUtente (int): identificativo univoco per ogni utente nel sistema.○ nome (varchar(30)): nome dell'utente.○ cognome (varchar(30)): cognome dell'utente.○ eMail (varchar(100)): e-mail dell'utente.○ username (varchar(50)): username dell'utente.○ password (varchar(30)): password di accesso al sistema.○ dataNascita (date): data di nascita dell'utente registrato.
Giudice	Rappresenta un utente con il ruolo di giudice dell'evento.	<ul style="list-style-type: none">○ idGiudice (int): identificativo univoco del giudice.
Partecipante	Rappresenta un utente con il ruolo di partecipante dell'evento.	<ul style="list-style-type: none">○ idPartecipante (int): identificativo univoco del partecipante.
Organizzatore	Rappresenta un utente con il ruolo di organizzatore dell'evento.	<ul style="list-style-type: none">○ idOrganizzatore (int): identificativo univoco dell'organizzatore.
Invito	Rappresenta un invito inviato da un organizzatore ad un utente, diventando così	<ul style="list-style-type: none">○ idInvito (int): identificativo univoco dell'invito.○ stato (varchar(20)): stato corrente dell'invito ('accettato',

	giudice dell'evento nel caso di accettazione dell'invito.	‘rifiutato’). Se è a null non è stato ancora né accettato, né rifiutato.
Evento	Rappresenta un hackathon specifico.	<ul style="list-style-type: none"> ○ idEvento (int): identificativo univoco dell'evento. ○ titolo (varchar(100)): nome dell'evento. ○ inizioReg (date): data di inizio del periodo di iscrizione. ○ fineReg (date): data di fine del periodo di iscrizione. ○ dimMaxTeam (int): numero massimo di membri che un team può avere. ○ nMaxIscritti (int): numero massimo di partecipanti ammessi all'evento. ○ dataInizio (date): data di inizio dell'evento. ○ dataFine (date): data di fine evento. ○ problema (text): descrizione del problema da affrontare durante l'evento. ○ via (varchar(50)): via della sede in cui si terrà l'evento. ○ numeroCivico (varchar(10)): numero civico della sede. ○ citta (varchar(50)): città in cui avrà luogo l'evento.
Team	Rappresenta un gruppo di partecipanti che si uniscono in team per partecipare all'hackathon.	<ul style="list-style-type: none"> ○ idTeam (int): identificativo univoco del team. ○ nomeTeam (varchar(50)): nome del team.

Iscrizione	Rappresenta l'iscrizione di un utente generico ad un evento, diventandone così un partecipante.	<ul style="list-style-type: none"> ○ idIscrizione (int): identificativo univoco dell'iscrizione. ○ dataIscrizione (date): data in cui è avvenuta l'iscrizione.
Documento	Rappresenta un documento caricato da un team per documentare i progressi del team sul problema da affrontare durante l'evento.	<ul style="list-style-type: none"> ○ idDocumento (int): identificativo univoco del documento. ○ pathDoc (text): percorso del file del documento. ○ dataDoc (timestamp): data in cui il documento è stato caricato.

4.2. Dizionario delle associazioni

Nome	Descrizione	Classi coinvolte
Assume ruolo di	Definisce la gerarchia di specializzazione: ogni istanza di ruolo deve essere associata esattamente ad un utente (1..1). Un utente può o non può (a seconda del ruolo) assumere quel ruolo specifico.	Utente [1] ruolo generico: indica l'entità base. Ruolo [*] ruolo specializzato: istanza del ruolo che l'utente assume (Organizzatore, Partecipante, Giudice).
Organizza	Definisce la responsabilità di un Organizzatore di gestire un Evento.	Evento [1..*] ruolo gestito: indica l'Evento o gli eventi gestiti dall'Organizzatore. Organizzatore [1] ruolo responsabile: indica l'Organizzatore dell'Evento.
Partecipazione	Stabilisce la partecipazione di un Partecipante ad un Evento.	Partecipante [1..*] ruolo partecipante: un Evento ha uno o più partecipanti. Evento [1..*] ruolo ospitante: un Partecipante partecipa a uno o più eventi.
Appartiene	Definisce l'appartenenza di un Partecipante ad uno specifico Team.	Team [1] ruolo ospitante: un Partecipante appartiene ad un solo Team. Partecipante [1..*] ruolo componente: un Team è composto da uno o più partecipanti.
Vota	Registra il voto assegnato da un Giudice ad un Team.	Giudice [1..*] ruolo votante: un Team riceve voti da uno o più giudici.

		Team [1..*] ruolo votato: un Giudice assegna voti a uno o più Team.
Commenta	Registra il commento di un Giudice ad un Documento pubblicato da un Team.	Giudice [1..*] ruolo commentatore: un Documento è commentato da uno o più giudici. Documento [1..*] ruolo commentato: un Giudice commenta uno o più documenti.
Pubblica	Registra la pubblicazione di un Documento da parte di un Team.	Documento [*] ruolo pubblicato: un Team pubblica zero o più documenti. Team [1] ruolo pubblicante: un Documento è pubblicato da un solo Team.
Giudica	Definisce la valutazione di un Evento da parte di un Giudice.	Evento [1..*] ruolo valutato: un Giudice valuta uno o più eventi. Giudice [1..*] ruolo valutatore: un Evento è valutato da uno o più giudici.
Partecipa	Rappresenta la partecipazione di un Team ad un Evento.	Team [1..*]: ad un Evento partecipano uno o più Team. Evento [1]: un Team partecipa ad un evento.

4.3. Dizionario dei vincoli

<i>Nome</i>	<i>Descrizione</i>
Unicità Email	L'indirizzo email di un Utente deve essere unico all'interno del sistema.
Unicità Username	L'username di un Utente deve essere unico all'interno di un sistema.
Formato Nome	Il nome di un Utente deve essere composto unicamente da caratteri alfabetici.
Formato Cognome	Il cognome di un Utente deve essere composto unicamente da carattere alfabetici.
Formato Email	L'indirizzo email di un Utente deve rispettare il formato standard (ad esempio parte_locale@dominio.estensione).
Formato Password	La password di un Utente deve rispettare una serie di requisiti minimi di sicurezza: essere composta da almeno 8 caratteri, avere almeno un carattere speciale, almeno un numero e almeno una lettera maiuscola.
Completezza Utente	Per potersi registrare, un Utente deve fornire nome, cognome, email, username, password e dataNascita.
Sequenza Evento	La data di inizio di un Evento (dataInizio) deve essere precedente alla sua data di fine (dataFine).

Sequenza Registrazione	La data di inizio delle registrazioni (inizioReg) deve essere precedente alla data di fine delle registrazioni (fineReg).
Tempistiche Hackathon	La data di fine delle registrazioni (fineReg) deve essere precedente alla data di inizio dell'Evento (dataInizio).
Integrità Voto	Il valore di un voto deve rientrare nell'intervallo che va da 1 a 10.
Unicità Voto	Un Giudice non può votare più di una volta lo stesso team. Vincolo già garantito dalla chiave primaria composta della tabella Vota (idGiudice, idTeam).
Unicità Commento	Un Giudice non può commentare più di una volta lo stesso Documento. Vincolo già garantito dalla chiave primaria composta della tabella Commenta (idGiudice, idDocumento).
Completezza Commento	Il commento di un Giudice non può essere vuoto.
Unicità Invito	Un Organizzatore non può invitare più di una volta lo stesso Utente per lo stesso Evento.
Integrità Referenziale Organizzatore	Un Organizzatore non può essere eliminato se è ancora associato a uno o più eventi. Questo assicura la permanenza dei dati storici.
Appartenenza Team	Un Partecipante può appartenere ad un solo Team in un determinato Evento.

Unicità Nome Team	Il nome di un Team deve essere unico all'interno dello stesso Evento.
Validità Dimensione Massima Team	La dimMaxTeam deve essere un valore intero positivo.
Validità Numero Massimo Iscritti	Il nMaxIscritti deve essere un valore intero positivo.
Validità Via	La Via deve essere composta da caratteri alfabetici, con possibili spazi e apostrofi.
Validità Città	La Città deve essere composta da caratteri alfabetici, con possibili spazi e apostrofi.
Validità Numero Civico	Il Numero Civico può contenere sia cifre che lettere.
Completezza Indirizzo	Per un Evento, gli attributi Via, NumeroCivico e Città non devono essere nulli. È fondamentale che un evento abbia una sede fisica con un indirizzo completo per essere localizzato correttamente dagli utenti.
Dominio Stato Invito	Il campo Stato dell'entità Invito deve accettare solo un set di valori predefiniti ('Accettato', 'Rifiutato').
Completezza Problema	Il campo problema dell'entità Evento deve contenere una descrizione significativa, con un numero di caratteri superiore a zero e una lunghezza minima

per garantire la completezza
dell'informazione.

Completezza Titolo	Il campo titolo dell'entità Evento non deve essere vuoto.
Unicità Evento	Non è consentito inserire due eventi che abbiano lo stesso titolo nella stessa dataInizio e nella stessa sede (via, numeroCivico, città).
Unicità Iscrizione	Non è consentito ad un Utente iscriversi più di una volta allo stesso Evento.
Completezza Nome Team	Non è possibile inserire il nome di un team vuoto.
Unicità Documento	Non è consentito ad un Team pubblicare più di una volta lo stesso Documento.
Formato PathDoc	Nell'inserimento del pathDoc bisogna rispettare un formato che permette path assoluti e relativi che finiscono con file che hanno una delle estensioni consentite.
Completezza PathDoc	Non è possibile inserire path che siano vuoti o composti da soli spazi bianchi.
Unicità Partecipazione	Un Partecipante non può registrarsi più volte come partecipante dello stesso Evento. Vincolo già garantito dalla chiave primaria composta della tabella Partecipazione (idPartecipante, idEvento).

Unicità Giudice

Un Giudice può essere assegnato allo stesso Evento massimo una volta. Vincolo già garantito dalla chiave primaria composta della tabella Giudica (idGiudice, idEvento).

Dominio Ruolo Giudice

Il campo ruolo dell'entità Giudica può accettare solo un set di valori predefiniti: ‘Giudice Standard’ e ‘Autore Problema’. Quando un Utente accetta un Invito e diventa Giudice di default è un Giudice Standard.

5. Progettazione logica

In questo capitolo tratteremo la seconda fase della progettazione, cioè la progettazione logica, che ci porta ad un livello di astrazione più concreto e vicino all'implementazione. Il modello concettuale precedentemente definito e strutturato sarà convertito in uno schema logico.

5.1. Schema logico

Di seguito è riportato lo schema logico della nostra base di dati. Le chiavi primarie sono indicate con una singola sottolineatura, mentre le chiavi esterne con una doppia sottolineatura.

- Utente (idUtente, nome, cognome, eMail, username, password, dataNascita)
 $\text{idUtente} \rightarrow \text{Utente.idUtente}$
- Giudice (idGiudice, idUtente)
 $\text{idUtente} \rightarrow \text{Utente.idUtente}$
- Organizzatore (idOrganizzatore, idUtente)
 $\text{idUtente} \rightarrow \text{Utente.idUtente}$
- Partecipante (idPartecipante, idUtente, idTeam)
 $\text{idUtente} \rightarrow \text{Utente.idUtente}$
 $\text{idTeam} \rightarrow \text{Team.idTeam}$
- Evento (idEvento, titolo, inizioReg, fineReg, dimMaxTeam, nMaxIscritti, dataInizio, dataFine, problema, via, numeroCivico, citta, idOrganizzatore)
 $\text{idOrganizzatore} \rightarrow \text{Organizzatore.idOrganizzatore}$
- Team (idTeam, nomeTeam, idEvento)
 $\text{idEvento} \rightarrow \text{Evento.idEvento}$
- Invito (idInvito, stato, idUtente, idEvento, idOrganizzatore)
 $\text{idUtente} \rightarrow \text{Utente.idUtente}$
 $\text{idEvento} \rightarrow \text{Evento.idEvento}$
 $\text{idOrganizzatore} \rightarrow \text{Organizzatore.idOrganizzatore}$

- Documento (idDocumento, path, data, idTeam)

idTeam → Team.idTeam
- Iscrizione (idIscrizione, dataIscrizione, idUtente, idEvento)

idUtente → Utente.idUtente

idEvento → Evento.idEvento
- Vota (valore, idGiudice, idTeam)

idGiudice → Giudice.idGiudice

idTeam → Team.idTeam
- Commenta (testo, idGiudice, idDocumento)

idGiudice → Giudice.idGiudice

idDocumento → Documento.idDocumento
- Partecipazione (idPartecipante, idEvento)

idPartecipante → Partecipante.idPartecipante

idEvento → Evento.idEvento
- Giudica (idGiudice, idEvento)

idGiudice → Giudice.idGiudice

idEvento → Evento.idEvento

6. Progettazione fisica

In questo capitolo verrà riportata l'implementazione dello schema logico precedentemente descritto nel DBMS PostgreSQL.

6.1. Definizione tabelle

Di seguito sono riportate le definizioni delle tabelle, dei loro vincoli intrarelazionali e di eventuali semplici strutture per la loro gestione.

6.1.1. Definizione della tabella Utente

```
CREATE TABLE Utente (
    idUtente INT PRIMARY KEY DEFAULT NEXTVAL('utente_id'),
    nome VARCHAR(30) NOT NULL,
    cognome VARCHAR(30) NOT NULL,
    dataNascita DATE NOT NULL,
    eMail VARCHAR(100) UNIQUE NOT NULL,
    username VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(30) NOT NULL,
    CONSTRAINT chk_nome CHECK (nome ~ '^[a-zA-Z\s]+$'),
    CONSTRAINT chk_cognome CHECK (cognome ~ '^[a-zA-Z\s]+$'),
    CONSTRAINT chk_email CHECK (email ~* '^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9+-]+\.[a-zA-Z]{2,4}$'),
    CONSTRAINT chk_password CHECK (password ~* '^(?=.*[A-Z])(?=.*[0-9])(?=.*[^a-zA-Z0-9\s]).{8,}$')
);

CREATE SEQUENCE utente_id
    START WITH 1
    INCREMENT BY 1
    MINVALUE 1
    NO MAXVALUE;
```

6.1.2. Definizione della tabella Organizzatore

```
CREATE TABLE Organizzatore (
    idOrganizzatore INT PRIMARY KEY DEFAULT NEXTVAL('organizzatore_id'),
    idUtente INT NOT NULL,
    FOREIGN KEY (idUtente) REFERENCES Utente (idUtente) ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE SEQUENCE organizzatore_id
    START WITH 1
    INCREMENT BY 1
    MINVALUE 1
    NO MAXVALUE;
```

6.1.3. Definizione della tabella Giudice

```
CREATE TABLE Giudice (
    idGiudice INT PRIMARY KEY DEFAULT NEXTVAL('giudice_id'),
    idUtente INT NOT NULL,
    FOREIGN KEY (idUtente) REFERENCES Utente (idUtente) ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE SEQUENCE giudice_id
    START WITH 1
    INCREMENT BY 1
    MINVALUE 1
    NO MAXVALUE;
```

6.1.4. Definizione della tabella Invito

```
CREATE TABLE Invito (
    idInvito INT PRIMARY KEY DEFAULT NEXTVAL('invito_id'),
    stato VARCHAR(10),
    idUtente INT NOT NULL,
    idEvento INT NOT NULL,
    idOrganizzatore INT NOT NULL,
    FOREIGN KEY (idUtente) REFERENCES Utente (idUtente) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (idEvento) REFERENCES Evento (idEvento) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (idOrganizzatore) REFERENCES Organizzatore (idOrganizzatore) ON DELETE CASCADE ON UPDATE CASCADE,
    UNIQUE (idUtente, idEvento),
    CONSTRAINT chk_statoInvito CHECK (stato IS NULL OR stato IN ('Accettato', 'Rifiutato'))
);

CREATE SEQUENCE invito_id
    START WITH 1
    INCREMENT BY 1
    MINVALUE 1
    NO MAXVALUE;
```

6.1.5. Definizione della tabella Giudica

```
CREATE TABLE Giudica (
    idGiudice INT NOT NULL,
    idEvento INT NOT NULL,
    ruolo VARCHAR(20) DEFAULT 'Giudice Standard',
    PRIMARY KEY (idGiudice, idEvento),
    FOREIGN KEY (idGiudice) REFERENCES Giudice (idGiudice) ON DELETE RESTRICT ON UPDATE CASCADE,
    FOREIGN KEY (idEvento) REFERENCES Evento (idEvento) ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT chk_ruoloGiudice CHECK (ruolo IN ('Giudice Standard', 'Autore Problema'))
);
```

6.1.6. Definizione della tabella Evento

```
CREATE TABLE Evento (
    idEvento INT PRIMARY KEY DEFAULT NEXTVAL('evento_id'),
    titolo VARCHAR(100) NOT NULL,
    inizioReg DATE NOT NULL,
    fineReg DATE NOT NULL,
    dimMaxTeam INT NOT NULL,
    nMaxIscritti INT NOT NULL,
    dataInizio DATE NOT NULL,
    dataFine DATE NOT NULL,
    problema TEXT,
    via VARCHAR(50) NOT NULL,
    numeroCivico VARCHAR(10) NOT NULL,
    citta VARCHAR(50) NOT NULL,
    idOrganizzatore INT NOT NULL,
    FOREIGN KEY (idOrganizzatore) REFERENCES Organizzatore (idOrganizzatore) ON DELETE RESTRICT ON UPDATE CASCADE,
    UNIQUE (titolo, dataInizio, citta, numeroCivico, via),
    CONSTRAINT chk_titoloEvento CHECK (TRIM(titolo) <> ''),
    CONSTRAINT chk_sequenzaRegistrazione CHECK (inizioReg < fineReg),
    CONSTRAINT chk_tempisticheHackathon CHECK (fineReg < dataInizio),
    CONSTRAINT chk_sequenzaEvento CHECK (dataInizio < dataFine),
    CONSTRAINT chk_dimMaxTeam CHECK (dimMaxTeam > 0),
    CONSTRAINT chk_nMaxIscritti CHECK (nMaxIscritti > 0),
    CONSTRAINT chk_validitaCitta CHECK (citta ~* '^[a-zA-ZàééíòùÀÉÉÍÒÙ'\s]+$'),
    CONSTRAINT chk_validitaVia CHECK (via ~* '^[a-zA-ZàééíòùÀÉÉÍÒÙ'\s]+$'),
    CONSTRAINT chk_validitaNumeroCivico CHECK (numeroCivico ~ '^[a-zA-Z0-9]+$'),
    CONSTRAINT chk_problemaCompleto CHECK (TRIM(problema) <> '' AND LENGTH (TRIM(problema)) >=10)
);

CREATE SEQUENCE evento_id
    START WITH 1
    INCREMENT BY 1
    MINVALUE 1
    NO MAXVALUE;
```

6.1.7. Definizione della tabella Iscrizione

```
CREATE TABLE Iscrizione (
    idIscrizione INT PRIMARY KEY DEFAULT NEXTVAL('iscrizione_id'),
    dataIscrizione DATE NOT NULL,
    idUtente INT NOT NULL,
    idEvento INT NOT NULL,
    FOREIGN KEY (idUtente) REFERENCES Utente (idUtente) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (idEvento) REFERENCES Evento (idEvento) ON DELETE CASCADE ON UPDATE CASCADE,
    UNIQUE (idUtente, idEvento)
);

CREATE SEQUENCE iscrizione_id
    START WITH 1
    INCREMENT BY 1
    MINVALUE 1
    NO MAXVALUE;
```

6.1.8. Definizione della tabella Partecipante

```
CREATE TABLE Partecipante (
    idPartecipante INT PRIMARY KEY DEFAULT NEXTVAL('partecipante_id'),
    idUtente INT NOT NULL,
    idTeam INT,
    FOREIGN KEY (idUtente) REFERENCES Utente (idUtente) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (idTeam) REFERENCES Team (idTeam) ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE SEQUENCE partecipante_id
    START WITH 1
    INCREMENT BY 1
    MINVALUE 1
    NO MAXVALUE;
```

6.1.9. Definizione della tabella Team

```
CREATE TABLE Team (
    idTeam INT PRIMARY KEY DEFAULT NEXTVAL('team_id'),
    nomeTeam VARCHAR(50) NOT NULL,
    idEvento INT NOT NULL,
    FOREIGN KEY (idEvento) REFERENCES Evento (idEvento) ON DELETE CASCADE ON UPDATE CASCADE,
    UNIQUE (nomeTeam, idEvento),
    CONSTRAINT chk_nomeTeam CHECK (TRIM(nomeTeam) <> '')
);

CREATE SEQUENCE team_id
    START WITH 1
    INCREMENT BY 1
    MINVALUE 1
    NO MAXVALUE;
```

6.1.10. Definizione della tabella Vota

```
CREATE TABLE Vota (
    valore INT NOT NULL,
    idTeam INT NOT NULL,
    idGiudice INT NOT NULL,
    PRIMARY KEY (idTeam, idGiudice),
    FOREIGN KEY (idTeam) REFERENCES Team (idTeam) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (idGiudice) REFERENCES Giudice (idGiudice) ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT chk_valore CHECK (valore >= 0 AND valore <= 10)
);
```

6.1.11. Definizione della tabella Documento

```
CREATE TABLE Documento (
    idDocumento INT PRIMARY KEY DEFAULT NEXTVAL ('documento_id'),
    pathDoc TEXT NOT NULL,
    dataDoc TIMESTAMP NOT NULL,
    idTeam INT NOT NULL,
    FOREIGN KEY (idTeam) REFERENCES Team (idTeam) ON DELETE CASCADE ON UPDATE CASCADE,
    UNIQUE (pathDoc, idTeam),
    CONSTRAINT chk_pathDocumento CHECK (TRIM(pathDoc) <> ''
                                         AND pathDoc ~ '^\/?([^\/\0]+/?)*[^\/\0]+\.(pdf|docx?|docm|txt|pages|odt|rtf|tex)$')
);

CREATE SEQUENCE documento_id
    START WITH 1
    INCREMENT BY 1
    MINVALUE 1
    NO MAXVALUE;
```

6.1.12. Definizione della tabella Commenta

```
CREATE TABLE Commenta (
    idGiudice INT NOT NULL,
    idDocumento INT NOT NULL,
    testo TEXT NOT NULL,
    PRIMARY KEY (idGiudice, idDocumento),
    FOREIGN KEY (idGiudice) REFERENCES Giudice (idGiudice) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (idDocumento) REFERENCES Documento (idDocumento) ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT chk_testo CHECK (TRIM(testo) <> '')
);
```

6.1.13. Definizione della tabella Partecipazione

```
CREATE TABLE Partecipazione (
    idPartecipante INT NOT NULL,
    idEvento INT NOT NULL,
    PRIMARY KEY (idPartecipante, idEvento),
    FOREIGN KEY (idPartecipante) REFERENCES Partecipante (idPartecipante) ON DELETE RESTRICT ON UPDATE CASCADE,
    FOREIGN KEY (idEvento) REFERENCES Evento (idEvento) ON DELETE CASCADE ON UPDATE CASCADE
);
```

7. Trigger

In questo capitolo mostriamo le descrizioni ed implementazioni dei trigger necessari.

7.1. Dizionario dei trigger

<i>Nome</i>	<i>Descrizione</i>
enforce_eta_minima_utente	Il trigger è progettato per impedire la registrazione di utenti sotto i 14 anni, garantendo che la regola sull'età sia rispettata in ogni momento, indipendentemente dalla funzione che tenta di manipolare i dati.
enforce_prerequisiti_organizzatore	Il trigger garantisce che la specializzazione in Organizzatore sia conforme ai requisiti di responsabilità (maggiore età); bloccando l'inserimento o aggiornamento e impedendo che un minorenne assuma il ruolo di Organizzatore.
enforce_prerequisiti_giudice	Il trigger garantisce che il ruolo specializzato di Giudice sia limitato agli utenti che hanno raggiunto la maggiore età, bloccando così la promozione al ruolo di Giudice.
enforce_setta_problema	Il trigger gestisce l'aggiornamento del campo problema della tabella Evento, imponendo che l'aggiornamento del problema sia consentito solo ed esclusivamente nel giorno di inizio dell'evento.

enforce_evento_futuro_dinamico	Il trigger garantisce che la pianificazione degli eventi sia sempre corrente e che gli organizzatori non possano aggiornare i parametri temporali dell'evento in modo da farlo risultare già scaduto, mantenendo così l'integrità temporale del sistema.
enforce_assegnazione_team	Il trigger garantisce che un partecipante sia assegnato ad un solo team compatibile con l'evento e solo se l'evento non è ancora iniziato.
enforce_max_team_size	Il trigger garantisce che la dimensione di un team non superi la capacità massima definita dall'evento ospitante; bloccando l'operazione di assegnazione prima di un inserimento o aggiornamento nella tabella Partecipante.
enforce_invito_prima_inizio	Il trigger garantisce che i giudici possano essere invitati solo in una specifica finestra temporale. Assicura che l'atto di registrare un nuovo invito sia possibile solo prima dell'inizio dell'evento.
enforce_invito_precondizioni	Il trigger previene conflitti e autoassegnazioni inappropriate nel processo di invito di un giudice. Impedisce all'organizzatore di invitare come giudice dell'evento sé stesso e un utente che è già partecipante o giudice dell'evento.
enforce_iscrizione_apertura	Il trigger garantisce che la registrazione degli utenti nella tabella Iscrizione sia

	possibile solo durante l'intervallo di tempo stabilito per l'evento (da inizioReg a fineReg).
enforce_iscrizione_ruoli	Il trigger al momento dell'iscrizione previene che utenti con ruoli amministrativi (Organizzatore) o di valutazione (Giudice) possano partecipare come concorrenti dell'evento.
enforce_max_iscritti_evento	Il trigger garantisce che il numero di partecipanti non ecceda il numero massimo di iscritti prestabilito per l'evento.
enforce_limite_team_periodo	Il trigger garantisce che la creazione di un nuovo team sia consentita solo nel periodo di tempo stabilito (fino a prima dell'inizio dell'evento).
enforce_limite_upload_documento	Il trigger garantisce che la pubblicazione del documento da parte del team sia permessa solo durante il periodo di svolgimento effettivo dell'Hackathon.
enforce_problema_pubblicato_upload	Il trigger garantisce che i team non possano caricare documenti prima che il problema dell'Hackathon sia stato ufficialmente reso pubblico.
enforce_commento_periodo	Il trigger garantisce che l'azione di commentare un documento da parte di un giudice sia permessa solo dopo che il documento è stato effettivamente caricato dal team e prima della fine dell'evento.

enforce_invito_giudica

Il trigger garantisce che un giudice possa essere formalmente associato ad un evento solo se ha precedentemente accettato un invito valido per quell'evento. Pertanto garantisce che solo gli utenti che hanno dato esplicito consenso possano assumere il ruolo di giudice per un evento specifico.

enforce_update_ruolo_giudice

Il trigger garantisce che il ruolo di autore del problema sia assegnato al massimo il giorno prima dell'inizio ufficiale dell'evento.

enforce_voto_post_evento

Il trigger garantisce che l'assegnazione ufficiale del voto ad un team avvenga solo dopo la conclusione formale dell'Hackathon.

enforce_unique_autore_problema

Il trigger garantisce che in un dato evento esista al massimo un solo giudice con il ruolo di autore del problema. Impediamo così che più di un giudice abbia il permesso di definire o correggere il problema dell'evento.

7.2. Implementazione dei trigger

7.2.1. Trigger enforce_eta_minima_utente

```
CREATE OR REPLACE FUNCTION check_eta_minima_utente()
RETURNS TRIGGER AS $$ 
BEGIN
    IF NEW.dataNascita > CURRENT_DATE - INTERVAL '14 years' THEN
        RAISE EXCEPTION 'L''utente deve avere almeno 14 anni per essere registrato.';
    END IF;

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

```
CREATE TRIGGER enforce_eta_minima_utente
BEFORE INSERT OR UPDATE OF dataNascita ON Utente
FOR EACH ROW
EXECUTE FUNCTION check_eta_minima_utente();
```

7.2.2. Trigger enforce_prerequisiti_organizzatore

```
CREATE OR REPLACE FUNCTION check_prerequisiti_organizzatore()
RETURNS TRIGGER AS $$ 
DECLARE
    v_data_nascita DATE;
BEGIN
    SELECT dataNascita INTO v_data_nascita
    FROM Utente
    WHERE idUtente = NEW.idUtente;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Errore di integrità: L''ID Utente % non esiste.', NEW.idUtente;
    END IF;

    IF v_data_nascita > CURRENT_DATE - INTERVAL '18 years' THEN
        RAISE EXCEPTION 'L''utente deve avere almeno 18 anni per diventare Organizzatore.';
    END IF;

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

```
CREATE TRIGGER enforce_prerequisiti_organizzatore
BEFORE INSERT ON Organizzatore
FOR EACH ROW
EXECUTE FUNCTION check_prerequisiti_organizzatore();
```

7.2.3. Trigger enforce_prerequisiti_giudice

```
CREATE OR REPLACE FUNCTION check_prerequisiti_giudice()
RETURNS TRIGGER AS $$

DECLARE
    v_data_nascita DATE;
BEGIN
    SELECT dataNascita INTO v_data_nascita
    FROM Utente
    WHERE idUtente = NEW.idUtente;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Errore di integrità: L''ID Utente % non esiste.', NEW.idUtente;
    END IF;

    IF v_data_nascita > CURRENT_DATE - INTERVAL '18 years' THEN
        RAISE EXCEPTION 'L''utente deve avere almeno 18 anni per diventare Giudice.';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER enforce_prerequisiti_giudice
BEFORE INSERT ON Giudice
FOR EACH ROW
EXECUTE FUNCTION check_prerequisiti_giudice();
```

7.2.4. Trigger enforce_setta_problema

```
CREATE OR REPLACE FUNCTION check_setta_problema()
RETURNS TRIGGER AS $$

BEGIN
    IF CURRENT_DATE != OLD.dataInizio THEN
        RAISE EXCEPTION 'Il problema è gestibile solo il giorno di inizio evento (%). Data odierna: %.', OLD.dataInizio, CURRENT_DATE;
    END IF;

    IF NEW.problema IS NULL THEN
        RAISE EXCEPTION 'Il problema non può essere lasciato NULL。';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER enforce_setta_problema
BEFORE UPDATE OF problema ON Evento
FOR EACH ROW
EXECUTE FUNCTION check_setta_problema();
```

7.2.5. Trigger enforce_evento_futuro_dinamico

```
CREATE OR REPLACE FUNCTION check_evento_futuro_dinamico()
RETURNS TRIGGER AS $$

BEGIN
    IF NEW.inizioReg <= CURRENT_DATE THEN
        RAISE EXCEPTION 'La data di Inizio Registrazione (%) deve essere successiva alla data odierna (%).', NEW.inizioReg, CURRENT_DATE;
    END IF;

    IF NEW.dataInizio <= CURRENT_DATE THEN
        RAISE EXCEPTION 'La data di Inizio Evento (%) deve essere successiva alla data odierna (%).', NEW.dataInizio, CURRENT_DATE;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER enforce_evento_futuro_dinamico
BEFORE INSERT OR UPDATE OF inizioReg, dataInizio ON Evento
FOR EACH ROW
EXECUTE FUNCTION check_evento_futuro_dinamico();
```

7.2.6. Trigger enforce_assegnazione_team

```
CREATE OR REPLACE FUNCTION check_assegnazione_team()
RETURNS TRIGGER AS $$

DECLARE
    v_idEventoPartecipante INT;
    v_idEventoTeam INT;
    v_dataInizioEvento DATE;
BEGIN
    IF NEW.idTeam IS NULL THEN
        RETURN NEW;
    END IF;

    SELECT idEvento INTO v_idEventoTeam
    FROM Team
    WHERE idTeam = NEW.idTeam;

    SELECT idEvento INTO v_idEventoPartecipante
    FROM Partecipazione
    WHERE idPartecipante = NEW.idPartecipante AND idEvento = v_idEventoTeam;

    IF v_idEventoPartecipante IS NULL THEN
        RAISE EXCEPTION 'Errore di integrità: Il partecipante non è iscritto all''evento % ed è quindi incompatibile.', v_idEventoTeam;
    END IF;

    SELECT dataInizio INTO v_dataInizioEvento
    FROM Evento
    WHERE idEvento = v_idEventoTeam;

    IF CURRENT_DATE >= v_dataInizioEvento THEN
        RAISE EXCEPTION 'Impossibile assegnare il Team. L''Evento % è già iniziato il %.', v_idEventoTeam, v_dataInizioEvento;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER enforce_assegnazione_team
BEFORE UPDATE OF idTeam ON Partecipante
FOR EACH ROW
WHEN (old.idTeam IS DISTINCT FROM new.idTeam)
EXECUTE FUNCTION check_assegnazione_team();
```

7.2.7. Trigger enforce_max_team_size

```
CREATE OR REPLACE FUNCTION check_max_team_size()
RETURNS TRIGGER AS $$

DECLARE
    v_membri_correnti INT;
    v_dim_max_team INT;
    v_evento_id INT;

BEGIN
    SELECT COUNT(*) INTO v_membri_correnti
    FROM Partecipante
    WHERE idTeam = NEW.idTeam;

    SELECT idEvento INTO v_evento_id
    FROM Team
    WHERE idTeam = NEW.idTeam;

    SELECT dimMaxTeam INTO v_dim_max_team
    FROM Evento
    WHERE idEvento = v_evento_id;

    IF v_membri_correnti >= v_dim_max_team THEN
        RAISE EXCEPTION 'La dimensione massima del team per questo evento è stata raggiunta.';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER enforce_max_team_size
BEFORE INSERT OR UPDATE OF idTeam ON Partecipante
FOR EACH ROW
EXECUTE FUNCTION check_max_team_size();
```

7.2.8. Trigger enforce_invito_prima_inizio

```
CREATE OR REPLACE FUNCTION check_invito_prima_inizio()
RETURNS TRIGGER AS $$

DECLARE
    v_dataInizio DATE;

BEGIN
    SELECT dataInizio INTO v_dataInizio
    FROM Evento
    WHERE idEvento = NEW.idEvento;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Errore di integrità: L''evento (ID %) non esiste.', NEW.idEvento;
    END IF;

    IF CURRENT_DATE >= v_dataInizio THEN
        RAISE EXCEPTION 'Non è più possibile invitare giudici. L''evento è già iniziato il %.', v_dataInizio;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER enforce_invito_prima_inizio
BEFORE INSERT ON Invito
FOR EACH ROW
EXECUTE FUNCTION check_invito_prima_inizio();
```

7.2.9. Trigger enforce_invito_precondizioni

```
CREATE OR REPLACE FUNCTION check_invito_precondizioni()
RETURNS TRIGGER AS $$ 
DECLARE
    v_id_giudice INT;
    v_gia_giudice INT;
    v_idUtenteOrganizzatore INT;
    v_isPartecipante INT;
    v_dataNascita DATE;
BEGIN
    SELECT dataNascita INTO v_dataNascita
    FROM Utente
    WHERE idUtente = NEW.idUtente;

    IF v_dataNascita IS NULL OR v_dataNascita > CURRENT_DATE - INTERVAL '18 years' THEN
        RAISE EXCEPTION 'L''utente invitato (ID %) non ha l''età minima di 18 anni per essere Giudice.', NEW.idUtente;
    END IF;

    SELECT O.idUtente INTO v_idUtenteOrganizzatore
    FROM Organizzatore O JOIN Evento E ON O.idOrganizzatore = E.idOrganizzatore
    WHERE E.idEvento = NEW.idEvento;

    IF v_idUtenteOrganizzatore = NEW.idUtente THEN
        RAISE EXCEPTION 'Un organizzatore non può invitare sé stesso come giudice del proprio evento.';
    END IF;

    SELECT 1 INTO v_isPartecipante
    FROM Partecipazione PA JOIN Partecipante P ON PA.idPartecipante = P.idPartecipante
    WHERE P.idUtente = NEW.idUtente AND PA.idEvento = NEW.idEvento
    LIMIT 1;

    IF v_isPartecipante IS NOT NULL THEN
        RAISE EXCEPTION 'L''utente (ID %) è già partecipante all''evento (ID %) e non può essere invitato come giudice.', NEW.idUtente, NEW.idEvento;
    END IF;

    SELECT idGiudice INTO v_id_giudice
    FROM Giudice
    WHERE idUtente = NEW.idUtente;

    IF v_id_giudice IS NULL THEN
        RETURN NEW;
    END IF;

    SELECT 1 INTO v_gia_giudice
    FROM Giudica
    WHERE idGiudice = v_id_giudice AND idEvento = NEW.idEvento
    LIMIT 1;

    IF v_gia_giudice IS NOT NULL THEN
        RAISE EXCEPTION 'L''utente (ID %) è già un Giudice assegnato per l''evento (ID %). Impossibile inviare un nuovo invito.', NEW.idUtente, NEW.idEvento;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER enforce_invito_precondizioni
BEFORE INSERT ON Invito
FOR EACH ROW
EXECUTE FUNCTION check_invito_precondizioni();
```

7.2.10. Trigger enforce_iscrizione_apertura

```
CREATE OR REPLACE FUNCTION check_iscrizione_apertura()
RETURNS TRIGGER AS $$ 
DECLARE
    v_inizio_registrazioni DATE;
    v_fine_registrazioni DATE;
BEGIN
    SELECT inizioReg, fineReg INTO v_inizio_registrazioni, v_fine_registrazioni
    FROM Evento
    WHERE idEvento = NEW.idEvento;

    IF v_inizio_registrazioni IS NULL OR v_fine_registrazioni IS NULL THEN
        RAISE EXCEPTION 'Errore di sistema: Date di registrazione non definite per l''evento %.', NEW.idEvento;
    END IF;

    IF CURRENT_DATE < v_inizio_registrazioni THEN
        RAISE EXCEPTION 'Le registrazioni non sono ancora aperte. L''iscrizione è possibile solo a partire dal %.', v_inizio_registrazioni;
    END IF;

    IF CURRENT_DATE > v_fine_registrazioni THEN
        RAISE EXCEPTION 'Le registrazioni per questo evento sono già chiuse dal %.', v_fine_registrazioni;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER enforce_iscrizione_apertura
BEFORE INSERT ON Iscrizione
FOR EACH ROW
EXECUTE FUNCTION check_iscrizione_apertura();
```

7.2.11. Trigger enforce_iscrizione_ruoli

```
CREATE OR REPLACE FUNCTION check_iscrizione_ruoli()
RETURNS TRIGGER AS $$ 
DECLARE
    v_idUtenteOrganizzatore INT;
    v_isGiudiceEffettivo INT;
BEGIN
    SELECT O.idUtente INTO v_idUtenteOrganizzatore
    FROM Organizzatore O JOIN Evento E ON O.idOrganizzatore = E.idOrganizzatore
    WHERE E.idEvento = NEW.idEvento;

    IF v_idUtenteOrganizzatore = NEW.idUtente THEN
        RAISE EXCEPTION 'L''organizzatore non può iscriversi come partecipante al proprio evento.';
    END IF;

    SELECT 1 INTO v_isGiudiceEffettivo
    FROM Giudica g JOIN Giudice j ON g.idgiudice = j.idgiudice
    WHERE j.idutente = NEW.idUtente AND g.idevento = NEW.idEvento
    LIMIT 1;

    IF v_isGiudiceEffettivo IS NOT NULL THEN
        RAISE EXCEPTION 'L''utente con ID % è già un giudice effettivo per l''evento % e non può iscriversi come partecipante.', NEW.idUtente, NEW.idEvento;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER enforce_iscrizione_ruoli
BEFORE INSERT ON Iscrizione
FOR EACH ROW
EXECUTE FUNCTION check_iscrizione_ruoli();
```

7.2.12. Trigger enforce_max_iscritti_evento

```
CREATE OR REPLACE FUNCTION check_max_iscritti_evento()
RETURNS TRIGGER AS $$ 
DECLARE
    v_iscritti_correnti INT;
    v_max_iscritti INT;
BEGIN
    SELECT COUNT(*) INTO v_iscritti_correnti
    FROM Iscrizione
    WHERE idEvento = NEW.idEvento;

    SELECT nMaxIscritti INTO v_max_iscritti
    FROM Evento
    WHERE idEvento = NEW.idEvento;

    IF v_iscritti_correnti >= v_max_iscritti THEN
        RAISE EXCEPTION 'Numero massimo di iscritti per l''evento raggiunto.';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER enforce_max_iscritti_evento
BEFORE INSERT ON Iscrizione
FOR EACH ROW
EXECUTE FUNCTION check_max_iscritti_evento();
```

7.2.13. Trigger enforce_limite_team_periodo

```
CREATE OR REPLACE FUNCTION check_limite_team_periodo()
RETURNS TRIGGER AS $$ 
DECLARE
    v_inizioReg DATE;
    v_dataInizioEvento DATE;
BEGIN
    SELECT inizioReg, dataInizio INTO v_inizioReg, v_dataInizioEvento
    FROM Evento
    WHERE idEvento = NEW.idEvento;

    IF v_inizioReg IS NULL OR v_dataInizioEvento IS NULL THEN
        RAISE EXCEPTION 'Errore: Date Evento % non definite.', NEW.idEvento;
    END IF;

    IF CURRENT_DATE < v_inizioReg THEN
        RAISE EXCEPTION 'La creazione di Team è consentita solo dopo l''inizio delle registrazioni Evento (dal %).', v_dataInizioEvento;
    END IF;

    IF CURRENT_DATE >= v_dataInizioEvento THEN
        RAISE EXCEPTION 'Impossibile creare il Team. L''Evento % è già iniziato il %.', NEW.idEvento, v_dataInizioEvento;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER enforce_limite_team_periodo
BEFORE INSERT ON Team
FOR EACH ROW
EXECUTE FUNCTION check_limite_team_periodo();
```

7.2.14. Trigger enforce_limite_upload_documento

```
CREATE OR REPLACE FUNCTION check_limite_upload_documento()
RETURNS TRIGGER AS $$ 
DECLARE
    v_dataInizio DATE;
    v_dataFine DATE;
BEGIN
    SELECT E.dataInizio, E.dataFine INTO v_dataInizio, v_dataFine
    FROM Team T JOIN Evento E ON T.idEvento = E.idEvento
    WHERE T.idTeam = NEW.idTeam;

    IF NOT FOUND OR v_dataInizio IS NULL OR v_dataFine IS NULL THEN
        RAISE EXCEPTION 'Errore di sistema: Impossibile recuperare le date dell''evento associate al team %.', NEW.idteam;
    END IF;

    IF CURRENT_DATE < v_dataInizio THEN
        RAISE EXCEPTION 'Il caricamento dei documenti è consentito solo a partire dall''inizio dell''evento (dal %).', v_dataInizio;
    END IF;

    IF CURRENT_DATE > v_dataFine THEN
        RAISE EXCEPTION 'Impossibile caricare documenti. L''evento è terminato il %.', v_dataFine;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER enforce_limite_upload_documento
BEFORE INSERT ON Documento
FOR EACH ROW
EXECUTE FUNCTION check_limite_upload_documento();
```

7.2.15. Trigger enforce_problema_pubblicato_upload

```
CREATE OR REPLACE FUNCTION check_problema_pubblicato_upload()
RETURNS TRIGGER AS $$ 
DECLARE
    v_idEvento INT;
    v_problema_settato TEXT;
BEGIN
    SELECT idEvento INTO v_idEvento
    FROM Team
    WHERE idTeam = NEW.idTeam;

    SELECT problema INTO v_problema_settato
    FROM Evento
    WHERE idEvento = v_idEvento;

    IF v_problema_settato IS NULL THEN
        RAISE EXCEPTION 'La consegna dei documenti non è permessa finché il problema per l''Evento % non è stato ufficialmente pubblicato.', v_idEvento;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER enforce_problema_pubblicato_upload
BEFORE INSERT ON Documento
FOR EACH ROW
EXECUTE FUNCTION check_problema_pubblicato_upload();
```

7.2.16. Trigger enforce_commento_periodo

```
CREATE OR REPLACE FUNCTION check_commento_periodo()
RETURNS TRIGGER AS $$ 
DECLARE
    v_data_upload TIMESTAMP;
    v_data_fine_evento DATE;
BEGIN
    SELECT D.dataDoc, E.dataFine INTO v_data_upload, v_data_fine_evento
    FROM Documento D JOIN Team T ON D.idTeam = T.idTeam JOIN Evento E ON T.idEvento = E.idEvento
    WHERE D.idDocumento = NEW.idDocumento;

    IF v_data_fine_evento IS NULL THEN
        RAISE EXCEPTION 'Errore: Impossibile trovare i limiti temporali per l''Evento associato al Documento %.', NEW.idDocumento;
    END IF;

    IF CURRENT_TIMESTAMP < v_data_upload THEN
        RAISE EXCEPTION 'Non è possibile commentare un documento prima che sia stato caricato (Data/Ora Upload: %).', v_data_upload;
    END IF;

    IF CURRENT_DATE > v_data_fine_evento THEN
        RAISE EXCEPTION 'Non è possibile commentare. L''Evento è terminato il %.', v_data_fine_evento;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER enforce_commento_periodo
BEFORE INSERT ON Commenta
FOR EACH ROW
EXECUTE FUNCTION check_commento_periodo();
```

7.2.17. Trigger enforce_invito_giudica

```
CREATE OR REPLACE FUNCTION check_invito_giudica()
RETURNS TRIGGER AS $$ 
DECLARE
    v_idUtente INT;
    v_invito_accettato INT;
BEGIN
    SELECT idUtente INTO v_idUtente
    FROM Giudice
    WHERE idGiudice = NEW.idGiudice;

    SELECT 1 INTO v_invito_accettato
    FROM Invito
    WHERE idUtente = v_idUtente AND idEvento = NEW.idEvento AND stato = 'Accettato'
    LIMIT 1;

    IF v_invito_accettato IS NULL THEN
        RAISE EXCEPTION 'Non è possibile creare un collegamento in Giudica. È richiesto un invito accettato dall''utente per l''Evento %.', NEW.idEvento;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER enforce_invito_giudica
BEFORE INSERT ON Giudica
FOR EACH ROW
EXECUTE FUNCTION check_invito_giudica();
```

7.2.18. Trigger enforce_update_ruolo_giudice

```
CREATE OR REPLACE FUNCTION check_update_ruolo_giudice()
RETURNS TRIGGER AS $$ 
DECLARE
    v_data_inizio DATE;
BEGIN
    IF NEW.ruolo IS NOT DISTINCT FROM OLD.ruolo THEN
        RETURN NEW;
    END IF;

    IF NEW.ruolo = 'Autore Problema' THEN
        SELECT dataInizio INTO v_data_inizio
        FROM Evento
        WHERE idEvento = NEW.idEvento;

        IF CURRENT_DATE >= v_data_inizio THEN
            RAISE EXCEPTION 'Il ruolo "Autore Problema" deve essere assegnato al massimo il giorno prima dell''inizio evento (%).', v_data_inizio;
        END IF;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER enforce_update_ruolo_giudice
BEFORE UPDATE OF ruolo ON Giudica
FOR EACH ROW
EXECUTE FUNCTION check_update_ruolo_giudice();
```

7.2.19. Trigger enforce_voto_post_evento

```
CREATE OR REPLACE FUNCTION check_voto_post_evento()
RETURNS TRIGGER AS $$ 
DECLARE
    v_dataFineEvento DATE;
BEGIN
    SELECT E.dataFine INTO v_dataFineEvento
    FROM Team T JOIN Evento E ON T.idEvento = E.idEvento
    WHERE T.idTeam = NEW.idTeam;

    IF CURRENT_DATE <= v_dataFineEvento THEN
        RAISE EXCEPTION 'Il voto per l''Evento del Team % può essere assegnato solo DOPO la sua conclusione dal %.', NEW.idTeam, v_dataFineEvento;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER enforce_voto_post_evento
BEFORE INSERT OR UPDATE OF valore ON Vota
FOR EACH ROW
EXECUTE FUNCTION check_voto_post_evento();
```

7.2.20. Trigger enforce_unique_autore_problema

```
CREATE OR REPLACE FUNCTION check_unique_autore_problema()
RETURNS TRIGGER AS $$ 
DECLARE
    v conteggio_auor INT;
BEGIN
    IF NEW.ruolo = 'Autore Problema' THEN
        SELECT COUNT(*) INTO v conteggio_auor
        FROM Giudica
        WHERE idEvento = NEW.idEvento AND ruolo = 'Autore Problema' AND idGiudice <> NEW.idGiudice;

        IF v conteggio_auor >= 1 THEN
            RAISE EXCEPTION 'Violazione Unicità Ruolo: L''Evento % ha già un Giudice assegnato come Autore Problema.', NEW.idEvento;
        END IF;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER enforce_unique_autore_problema
BEFORE INSERT OR UPDATE OF ruolo ON Giudica
FOR EACH ROW
EXECUTE FUNCTION check_unique_autore_problema();
```

8. Funzioni

In questo capitolo mostriamo le descrizioni e le implementazioni di tutte le funzioni necessarie.

8.1. Dizionario delle funzioni

<i>Nome</i>	<i>Descrizione</i>
registra_utente	La funzione è progettata per la creazione di un nuovo account utente nel sistema, garantendo la validità dei dati prima dell'inserimento. Questa funzione è anche soggetta al trigger <code>enforce_eta_minima_utente</code> .
crea_evento_e_organizzatore	La funzione è progettata per la creazione di un nuovo evento e per la promozione dell'utente che lo crea al ruolo di Organizzatore. Questa funzione è soggetta ai trigger <code>enforce_evento_futuro_dinamico</code> e <code>enforce_prerequisiti_organizzatore</code> .
crea_invito_giudice	La funzione è progettata per gestire la creazione formale di un invito ad un utente per ricoprire il ruolo di giudice in un evento specifico. Questa funzione è soggetta ai trigger <code>enforce_invito_prima_inizio</code> e <code>enforce_invito_precondizioni</code> .
accetta_invito_giudice	La funzione è progettata per gestire l'intero processo per cui un utente, dopo aver ricevuto un invito, lo accetta formalmente, diventando un giudice effettivo per l'evento.

Questa funzione è soggetta al trigger
enforce_invito_giudica.

rifiuta_invito_giudice	La funzione è progettata per gestire il processo con cui un utente rifiuta un invito a ricoprire il ruolo di giudice, aggiornando lo stato dell'invito nel database.
iscrivi_utente_evento	La funzione è progettata per gestire l'iscrizione iniziale di un utente come partecipante ad un evento. Questa funzione è soggetta ai trigger enforce_iscrizione_apertura, enforce_max_iscritti_evento e enforce_iscrizione_ruoli.
crea_team	La funzione è progettata per permettere ad un partecipante di creare un nuovo team per un evento, a meno che non sia già in un team per quell'evento. Questa funzione è soggetta al trigger enforce_limite_team_periodo.
iscrivi_team	La funzione è progettata per assegnare un partecipante esistente ad un team esistente. Questa funzione è soggetta ai trigger enforce_max_team_size e enforce_assegnazione_team.
team_con_autoassegnazione	La funzione è progettata per gestire l'assegnazione automatica a team monomembro per i partecipanti di un evento che non sono ancora componenti di altri team.

setta_ruolo_autore_problema	<p>La funzione è progettata per assegnare il ruolo di autore del problema ad un giudice già associato all'evento.</p> <p>Questa funzione è soggetta ai trigger <code>enforce_unique_autore_problema</code> e <code>enforce_update_ruolo_giudice</code>.</p>
setta_problema_evento	<p>La funzione è progetta per definire il problema di un evento.</p> <p>Questa funzione è soggetta al trigger <code>enforce_setta_problema</code>.</p>
corregge_problema_evento	<p>La funzione è progettata per permettere di correggere il problema di un evento.</p> <p>Questa funzione è soggetta al trigger <code>enforce_setta_problema</code>.</p>
carica_documento_team	<p>La funzione è progettata per gestire il caricamento di un documento da parte di un partecipante per conto del proprio team.</p> <p>Questa funzione è soggetta ai trigger <code>enforce_limite_upload_documento</code> e <code>enforce_problema_pubblicato_upload</code>.</p>
commenta_documento	<p>La funzione è progettata per permettere ad un giudice di aggiungere un commento ad un documento consegnato da un team.</p> <p>Questa funzione è soggetta al trigger <code>enforce_commento_periodo</code>.</p>
assegna_voto_team_unico	<p>La funzione è progettata per gestire l'assegnazione di un voto da parte di un giudice ad un team per un evento.</p> <p>Questa funzione è soggetta al trigger <code>enforce_voto_post_evento</code>.</p>

get_potenziali_giudici

La funzione è progettata per fornire al'Organizzatore dell'evento una lista filtrata e dinamica degli utenti idonei a ricevere un invito per il ruolo di Giudice per un evento specifico.

Questa funzione è fondamentale per semplificare il processo di invito, garantendo che l'organizzatore non invii accidentalmente inviti a persone non qualificate o già impegnate in quell'evento.

8.2. Implementazione delle funzioni

8.2.1. Funzione `registra_utente`

```
CREATE OR REPLACE FUNCTION registra_utente(p_nome VARCHAR, p_cognome VARCHAR, p_eMail VARCHAR,
                                            p_username VARCHAR, p_password VARCHAR, p_dataNascita DATE)
RETURNS INT AS $$

DECLARE
    v_nuovoIdUtente INT;
BEGIN
    INSERT INTO Utente (nome, cognome, eMail, username, password, dataNascita)
    VALUES (p_nome, p_cognome, p_eMail, p_username, p_password, p_dataNascita)
    RETURNING idUtente INTO v_nuovoIdUtente;

    RETURN v_nuovoIdUtente;
EXCEPTION
    WHEN uniqueViolation THEN
        RAISE EXCEPTION 'Registrazione fallita: L''email o l''username forniti sono già in uso da un altro account.';
    WHEN notNullViolation THEN
        RAISE EXCEPTION 'Errore: Mancano dati obbligatori per la registrazione.';
    WHEN OTHERS THEN
        RAISE EXCEPTION 'Errore durante la registrazione: %', SQLERRM;
END;
$$ LANGUAGE plpgsql;
```

8.2.2. Funzione crea_evento_e_organizzatore

```
CREATE OR REPLACE FUNCTION crea_evento_e_organizzatore (p_idUtente INT, p_titolo VARCHAR, p_inizioReg DATE, p_fineReg DATE, p_dimMaxTeam INT, p_nMaxIscritti INT, p_dataInizio DATE, p_dataFine DATE, p_via VARCHAR, p_numeroCivico VARCHAR, p_citta VARCHAR)
RETURNS TEXT AS $$

DECLARE
    v_idOrganizzatore INT;
BEGIN
    SELECT idOrganizzatore INTO v_idOrganizzatore
    FROM Organizzatore
    WHERE idUtente = p_idUtente;

    IF v_idOrganizzatore IS NULL THEN
        INSERT INTO Organizzatore (idUtente)
        VALUES (p_idUtente)
        RETURNING idOrganizzatore INTO v_idOrganizzatore;
    END IF;

    INSERT INTO Evento (titolo, inizioReg, fineReg, dimMaxTeam, nMaxIscritti, dataInizio, dataFine, problema, via, numeroCivico, citta, idOrganizzatore)
    VALUES (p_titolo, p_inizioReg, p_fineReg, p_dimMaxTeam, p_nMaxIscritti, p_dataInizio, p_dataFine, NULL, p_via, p_numeroCivico, p_citta, v_idOrganizzatore);

    RETURN 'Evento creato con successo e utente registrato come organizzatore.';
EXCEPTION
    WHEN OTHERS THEN
        RAISE EXCEPTION 'Errore durante la creazione dell''evento o la promozione a organizzatore: %', SQLERRM;
END;
$$ LANGUAGE plpgsql;
```

8.2.3. Funzione crea_invito_giudice

```
CREATE OR REPLACE FUNCTION crea_invito_giudice(p_idOrganizzatore INT, p_idUtenteInvitato INT, p_idEvento INT)
RETURNS TEXT AS $$

DECLARE
    v_organizzatore_evento INT;
    v_utente_esiste BOOLEAN;
BEGIN
    SELECT EXISTS (SELECT 1
                   FROM Utente
                   WHERE idUtente = p_idUtenteInvitato) INTO v_utente_esiste;

    IF NOT v_utente_esiste THEN
        RAISE EXCEPTION 'Errore: L''ID Utente Invitato % non esiste.', p_idUtenteInvitato;
    END IF;

    SELECT idOrganizzatore INTO v_organizzatore_evento
    FROM Evento
    WHERE idEvento = p_idEvento;

    IF v_organizzatore_evento IS NULL THEN
        RAISE EXCEPTION 'Errore: L''evento con ID % non esiste.', p_idEvento;
    END IF;

    IF v_organizzatore_evento != p_idOrganizzatore THEN
        RAISE EXCEPTION 'Autorizzazione negata: L''organizzatore con ID % non gestisce l''evento %.', p_idOrganizzatore, p_idEvento;
    END IF;

    INSERT INTO Invito (idUtente, idEvento, idOrganizzatore, stato)
    VALUES (p_idUtenteInvitato, p_idEvento, p_idOrganizzatore, NULL);

    RETURN 'Invito creato con successo per l''utente ' || p_idUtenteInvitato || ' all''evento ' || p_idEvento || '.';
EXCEPTION
    WHEN uniqueViolation THEN
        RAISE EXCEPTION 'Invito già esistente: L''utente è già stato invitato (o è già giudice) per questo evento.';
    WHEN foreign_keyViolation THEN
        RAISE EXCEPTION 'Errore di integrità: L''utente invitato o l''organizzatore non sono validi.';
END;
$$ LANGUAGE plpgsql;
```

8.2.4. Funzione accetta_invito_giudice

```
CREATE OR REPLACE FUNCTION accetta_invito_giudice(p_idInvito INT)
RETURNS TEXT AS $$

DECLARE
    v_idUtente INT;
    v_idEvento INT;
    v_nuovoIdGiudice INT;
    v_idGiudiceEsistente INT;

BEGIN
    SELECT idUtente, idEvento INTO v_idUtente, v_idEvento
    FROM Invito
    WHERE idInvito = p_idInvito AND stato IS NULL;

    IF v_idUtente IS NULL THEN
        RAISE EXCEPTION 'L''invito non è valido o è già stato gestito.';
    END IF;

    SELECT idGiudice INTO v_idGiudiceEsistente
    FROM Giudice
    WHERE idUtente = v_idUtente;

    IF v_idGiudiceEsistente IS NULL THEN
        INSERT INTO Giudice (idUtente)
        VALUES (v_idUtente)
        RETURNING idGiudice INTO v_nuovoIdGiudice;
    ELSE
        v_nuovoIdGiudice := v_idGiudiceEsistente;
    END IF;

    UPDATE Invito
    SET stato = 'Accettato'
    WHERE idInvito = p_idInvito;

    INSERT INTO Giudica (idGiudice, idEvento)
    VALUES (v_nuovoIdGiudice, v_idEvento);

    RETURN 'Invito accettato. Utente promosso a giudice per l''evento.';
END;
$$ LANGUAGE plpgsql;
```

8.2.5. Funzione rifiuta_invito_giudice

```
CREATE OR REPLACE FUNCTION rifiuta_invito_giudice(p_idInvito INT)
RETURNS TEXT AS $$

DECLARE
    v_stato_attuale VARCHAR;

BEGIN
    SELECT stato INTO v_stato_attuale
    FROM Invito
    WHERE idInvito = p_idInvito;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Errore: Invito non trovato con ID .', p_idInvito;
    END IF;

    IF v_stato_attuale IS NULL THEN
        UPDATE Invito
        SET stato = 'Rifiutato'
        WHERE idInvito = p_idInvito;
        RETURN 'Invito rifiutato con successo.';
    ELSIF v_stato_attuale = 'Rifiutato' THEN
        RAISE EXCEPTION 'L''utente ha già rifiutato l''invito.';
    ELSIF v_stato_attuale = 'Accettato' THEN
        RAISE EXCEPTION 'Impossibile rifiutare: l''invito è già stato accettato.';
    ELSE
        RAISE EXCEPTION 'Invito trovato';
    END IF;
END;
$$ LANGUAGE plpgsql;
```

8.2.6. Funzione iscrivi_utente_evento

```
CREATE OR REPLACE FUNCTION iscrivi_utente_evento(p_idUtente INT, p_idEvento INT)
RETURNS TEXT AS $$

DECLARE
    v_idPartecipanteFinale INT;
BEGIN
    IF NOT EXISTS (SELECT 1
                   FROM Evento
                   WHERE idEvento = p_idEvento) THEN
        RAISE EXCEPTION 'Errore: L''evento con ID % non esiste.', p_idEvento;
    END IF;

    INSERT INTO Iscrizione (idUtente, idEvento, dataIscrizione)
    VALUES (p_idUtente, p_idEvento, CURRENT_DATE);

    INSERT INTO Partecipante (idUtente, idTeam)
    VALUES (p_idUtente, NULL)
    RETURNING idPartecipante INTO v_idPartecipanteFinale;

    INSERT INTO Partecipazione (idPartecipante, idEvento)
    VALUES (v_idPartecipanteFinale, p_idEvento);

    RETURN 'Iscrizione completata con successo per l''evento ' || p_idEvento || '.';
EXCEPTION
    WHEN unique_violation THEN
        RAISE EXCEPTION 'Attenzione: L''utente con ID % è già iscritto a questo evento.', p_idUtente;
    WHEN foreign_keyViolation THEN
        RAISE EXCEPTION 'Errore di integrità: Impossibile iscrivere. ID Utente, Evento o dati di partecipazione non validi.';
END;
$$ LANGUAGE plpgsql;
```

8.2.7. Funzione crea_team

```
CREATE OR REPLACE FUNCTION crea_team(p_idPartecipante INT, p_idEvento INT, p_nomeTeam VARCHAR)
RETURNS TEXT AS $$

DECLARE
    v_idUtenteAssociato INT;
    v_CurrentTeamId INT;
    v_idTeamNuovo INT;
    v_messaggioIscrizione VARCHAR;
BEGIN
    SELECT idUtente, idTeam INTO v_idUtenteAssociato, v_CurrentTeamId
    FROM Partecipante
    WHERE idPartecipante = p_idPartecipante;

    IF v_idUtenteAssociato IS NULL THEN
        RAISE EXCEPTION 'Errore: L''ID Partecipante % non è valido.', p_idPartecipante;
    END IF;

    IF v_CurrentTeamId IS NOT NULL AND EXISTS (SELECT 1
                                                FROM Team
                                                WHERE idTeam = v_CurrentTeamId AND idEvento = p_idEvento) THEN
        RAISE EXCEPTION 'Il Partecipante % è già membro di un Team per l''Evento % e non può creare uno nuovo.', p_idPartecipante, p_idEvento;
    END IF;

    IF NOT EXISTS (SELECT 1
                   FROM Iscrizione
                   WHERE idUtente = v_idUtenteAssociato AND idEvento = p_idEvento) THEN
        RAISE EXCEPTION 'Il Partecipante non risulta iscritto all''Evento %.', p_idEvento;
    END IF;

    INSERT INTO Team (nomeTeam, idEvento)
    VALUES (p_nomeTeam, p_idEvento)
    RETURNING idTeam INTO v_idTeamNuovo;

    SELECT iscrivi_team(p_idPartecipante, v_idTeamNuovo, p_idEvento) INTO v_messaggioIscrizione;

    RETURN 'Team "' || p_nomeTeam || '" creato con successo per l''evento ' || p_idEvento || '. Risultato iscrizione: ' || v_messaggioIscrizione;
EXCEPTION
    WHEN unique_violation THEN
        RAISE EXCEPTION 'Errore: Esiste già un team chiamato "%" per l''evento %.', p_nomeTeam, p_idEvento;
    WHEN foreign_keyViolation THEN
        RAISE EXCEPTION 'Errore di integrità: L''ID evento non è valido.';
    WHEN OTHERS THEN
        RAISE EXCEPTION 'Errore durante la creazione o iscrizione al team: %', SQLERRM;
END;
$$ LANGUAGE plpgsql;
```

8.2.8. Funzione iscrivi_team

```
CREATE OR REPLACE FUNCTION iscrivi_team(p_idPartecipante INT, p_idTeam INT, p_idEvento INT)
RETURNS TEXT AS $$

DECLARE
    v_idUtenteAssociato INT;
    v_CurrentTeamId INT;
    v_teamEventoId INT;

BEGIN
    SELECT idUtente, idTeam INTO v_idUtenteAssociato, v_CurrentTeamId
    FROM Partecipante
    WHERE idPartecipante = p_idPartecipante;

    IF v_idUtenteAssociato IS NULL THEN
        RAISE EXCEPTION 'Errore: L''ID Partecipante % non è valido o non è registrato.', p_idPartecipante;
    END IF;

    SELECT idEvento INTO v_teamEventoId
    FROM Team
    WHERE idTeam = p_idTeam;

    IF v_teamEventoId IS NULL THEN
        RAISE EXCEPTION 'Errore: Il team con ID % non esiste.', p_idTeam;
    ELSIF v_teamEventoId != p_idEvento THEN
        RAISE EXCEPTION 'Il Team % non appartiene all''Evento %.', p_idTeam, p_idEvento;
    END IF;

    IF v_CurrentTeamId IS NOT NULL THEN
        IF v_CurrentTeamId = p_idTeam THEN
            RETURN 'Attenzione: Il partecipante è già membro di questo team.';
        ELSE
            RAISE EXCEPTION 'Il partecipante è già membro del Team % .', v_CurrentTeamId;
        END IF;
    END IF;

    UPDATE Partecipante
    SET idTeam = p_idTeam
    WHERE idPartecipante = p_idPartecipante;

    RETURN 'Il partecipante ' || p_idPartecipante || ' è stato iscritto con successo al Team ' || p_idTeam || '.';
EXCEPTION
    WHEN foreign_keyViolation THEN
        RAISE EXCEPTION 'Errore di integrità: ID Utente o Team non validi.';
    WHEN OTHERS THEN
        RAISE EXCEPTION 'Errore durante l''iscrizione al team: %', SQLERRM;
END;
$$ LANGUAGE plpgsql;
```

8.2.9. Funzione team_con_autoassegnazione

```
CREATE OR REPLACE FUNCTION team_con_autoassegnazione(p_idEvento INT)
RETURNS TEXT AS $$

DECLARE
    v_dataInizioEvento DATE;
    v_partecipante_record RECORD;
    v_idTeamNuovo INT;
    v conteggio_assegnazioni INT := 0;
    v_team_name VARCHAR;

BEGIN
    SELECT dataInizio INTO v_dataInizioEvento
    FROM Evento
    WHERE idEvento = p_idEvento;

    IF v_dataInizioEvento IS NULL THEN
        RAISE EXCEPTION 'Errore: Data di inizio evento non definita per ID %.', p_idEvento;
    END IF;

    IF CURRENT_DATE >= v_dataInizioEvento THEN
        RAISE EXCEPTION 'Impossibile assegnare team automaticamente. L''Evento % è già iniziato (Inizio: %).', p_idEvento, v_dataInizioEvento;
    END IF;

    FOR v_partecipante_record IN (SELECT P.idPartecipante, U.username
                                    FROM Partecipazione PA JOIN Partecipante P ON PA.idPartecipante = P.idPartecipante
                                    JOIN Utente U ON P.idUtente = U.idUtente
                                    WHERE PA.idEvento = p_idEvento AND P.idTeam IS NULL)
    LOOP
        v_team_name := 'Team Autogestito - ' || v_partecipante_record.username;

        INSERT INTO Team (nomeTeam, idEvento)
        VALUES (v_team_name, p_idEvento)
        RETURNING idTeam INTO v_idTeamNuovo;

        UPDATE Partecipante
        SET idTeam = v_idTeamNuovo
        WHERE idPartecipante = v_partecipante_record.idPartecipante;

        v conteggio_assegnazioni := v conteggio_assegnazioni + 1;
    END LOOP;

    IF v conteggio_assegnazioni > 0 THEN
        RETURN 'Sono stati creati ' || v conteggio_assegnazioni || ' team mono-membro per i partecipanti orfani dell''Evento ' || p_idEvento || '.';
    ELSE
        RETURN 'Nessun partecipante orfano trovato per l''Evento ' || p_idEvento || '. Tutti sono già in un team.';
    END IF;
END;
$$ LANGUAGE plpgsql;
```

8.2.10. Funzione setta_ruolo_autore_problema

```
CREATE OR REPLACE FUNCTION setta_ruolo_autore_problema(p_idOrganizzatore INT, p_idGiudice INT, p_idEvento INT)
RETURNS TEXT AS $$
DECLARE
    v_idOrganizzatoreEvento INT;
    v_data_inizio DATE;
BEGIN
    SELECT idOrganizzatore, dataInizio INTO v_idOrganizzatoreEvento, v_data_inizio
    FROM Evento
    WHERE idEvento = p_idEvento;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Errore: Evento con ID % non trovato.', p_idEvento;
    END IF;

    IF v_idOrganizzatoreEvento != p_idOrganizzatore THEN
        RAISE EXCEPTION 'Accesso negato: L''Organizzatore % non è l''organizzatore responsabile dell''Evento %.', p_idOrganizzatore, p_idEvento;
    END IF;

    UPDATE Giudica
    SET ruolo = 'Autore Problema'
    WHERE idGiudice = p_idGiudice AND idEvento = p_idEvento;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Errore: Il Giudice % non è assegnato all''Evento %.', p_idGiudice, p_idEvento;
    END IF;

    RETURN 'Ruolo "Autore Problema" assegnato con successo al Giudice ' || p_idGiudice || ' per l''Evento ' || p_idEvento || ' (Deadline: ' || v_data_inizio || ')';
EXCEPTION
    WHEN OTHERS THEN
        RAISE EXCEPTION 'Errore durante l''assegnazione del ruolo: %', SQLERRM;
END;
$$ LANGUAGE plpgsql;
```

8.2.11. Funzione setta_problema_evento

```
CREATE OR REPLACE FUNCTION setta_problema_evento(p_idGiudice INT, p_idEvento INT, p_nuovoProblema TEXT)
RETURNS TEXT AS $$
DECLARE
    v_ruolo_giudice VARCHAR;
BEGIN
    SELECT G.ruolo INTO v_ruolo_giudice
    FROM Giudica G
    WHERE G.idGiudice = p_idGiudice AND G.idEvento = p_idEvento;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Errore: Giudice % non assegnato all''Evento %.', p_idGiudice, p_idEvento;
    END IF;

    IF v_ruolo_giudice != 'Autore Problema' THEN
        RAISE EXCEPTION 'Il Giudice % (Ruolo: %) non ha il permesso di settare il problema.', p_idGiudice, v_ruolo_giudice;
    END IF;

    UPDATE Evento
    SET problema = p_nuovoProblema
    WHERE idEvento = p_idEvento;

    RETURN 'Problema settato con successo per l''Evento ' || p_idEvento || ' dall''Autore Problema (ID: ' || p_idGiudice || ').';
EXCEPTION
    WHEN OTHERS THEN
        RAISE EXCEPTION 'Errore durante l''impostazione del problema: %', SQLERRM;
END;
$$ LANGUAGE plpgsql;
```

8.2.12. Funzione corregge_problema_evento

```
CREATE OR REPLACE FUNCTION corregge_problema_evento(p_idGiudice INT, p_idEvento INT, p_testoCorrezione TEXT)
RETURNS TEXT AS $$

DECLARE
    v_ruolo_giudice VARCHAR(50);
BEGIN
    SELECT ruolo INTO v_ruolo_giudice
    FROM Giudica
    WHERE idGiudice = p_idGiudice AND idEvento = p_idEvento;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Errore: Giudice % non assegnato all''Evento % o evento non trovato.', p_idGiudice, p_idEvento;
    END IF;

    IF v_ruolo_giudice != 'Autore Problema' THEN
        RAISE EXCEPTION 'Accesso negato: Il Giudice % non ha il ruolo di "Autore Problema" per correggere il problema.', p_idGiudice;
    END IF;

    UPDATE Evento
    SET problema = p_testoCorrezione
    WHERE idEvento = p_idEvento;

    RETURN 'Problema per l''Evento ' || p_idEvento || ' corretto con successo dal Giudice Autore (ID: ' || p_idGiudice || ').';
EXCEPTION
    WHEN OTHERS THEN
        RAISE EXCEPTION 'Errore durante la correzione del problema: %', SQLERRM;
END;
$$ LANGUAGE plpgsql;
```

8.2.13. Funzione carica_documento_team

```
CREATE OR REPLACE FUNCTION carica_documento_team(p_idPartecipante INT, p_idEvento INT, p_pathDoc TEXT)
RETURNS TEXT AS $$

DECLARE
    v_idTeamAssociato INT;
    v_teamEventId INT;
BEGIN
    SELECT idTeam INTO v_idTeamAssociato
    FROM Partecipante
    WHERE idPartecipante = p_idPartecipante;

    IF v_idTeamAssociato IS NULL THEN
        RAISE EXCEPTION 'Il partecipante % non è assegnato a nessun team e non può caricare documenti.', p_idPartecipante;
    END IF;

    SELECT idEvento INTO v_teamEventId
    FROM Team
    WHERE idTeam = v_idTeamAssociato;

    IF v_teamEventId != p_idEvento THEN
        RAISE EXCEPTION 'Il Team % non è il team corretto per l''Evento .', v_idTeamAssociato, p_idEvento;
    END IF;
    |

    INSERT INTO Documento (pathDoc, dataDoc, idTeam)
    VALUES (p_pathDoc, NOW(), v_idTeamAssociato);

    RETURN 'Documento '' || p_pathDoc || '' caricato con successo per il Team ' || v_idTeamAssociato || '.';
EXCEPTION
    WHEN unique_violation THEN
        RAISE EXCEPTION 'Errore: Il documento con questo percorso è già stato caricato per il Team .', v_idTeamAssociato;
    WHEN foreign_keyViolation THEN
        RAISE EXCEPTION 'Errore di integrità: ID Team, Partecipante o Evento non validi.';
    WHEN OTHERS THEN
        RAISE EXCEPTION 'Errore durante il caricamento del documento: %', SQLERRM;
END;
$$ LANGUAGE plpgsql;
```

8.2.14. Funzione commenta_documento

```

CREATE OR REPLACE FUNCTION commenta_documento(p_idGiudice INT, p_idDocumento INT, p_testoCommento TEXT)
RETURNS TEXT AS $$
DECLARE
    v_team_associato INT;
    v_evento_documento INT;
    v_giudice_assegnato_evento BOOLEAN;
BEGIN
    SELECT D.idTeam, T.idEvento INTO v_team_associato, v_evento_documento
    FROM Documento D JOIN Team T ON D.idTeam = T.idTeam
    WHERE D.idDocumento = p_idDocumento;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Errore: Il documento con ID % non esiste o non è collegato a un team valido.', p_idDocumento;
    END IF;

    SELECT EXISTS (
        SELECT 1
        FROM Giudica
        WHERE idGiudice = p_idGiudice AND idEvento = v_evento_documento
    ) INTO v_giudice_assegnato_evento;

    IF v_giudice_assegnato_evento IS FALSE THEN
        RAISE EXCEPTION 'Il Giudice % non è autorizzato a valutare documenti dell''Evento %.', p_idGiudice, v_evento_documento;
    END IF;

    INSERT INTO Commenta (idDocumento, testo, idGiudice)
    VALUES (p_idDocumento, p_testoCommento, p_idGiudice);

    RETURN 'Commento inserito con successo sul Documento ' || p_idDocumento || ' (Evento ' || v_evento_documento || ') dal Giudice ' || p_idGiudice || '.';
EXCEPTION
    WHEN OTHERS THEN
        RAISE EXCEPTION 'Errore durante l''inserimento del commento: %', SQLERRM;
END;
$$ LANGUAGE plpgsql;

```

8.2.15. Funzione assegna_voto_team_unico

```

CREATE OR REPLACE FUNCTION assegna_voto_team_unico(p_idGiudice INT, p_idTeam INT, p_valore INT)
RETURNS TEXT AS $$
DECLARE
    v_idEvento INT;
    v_voto_esistente BOOLEAN;
BEGIN
    SELECT T.idEvento INTO v_idEvento
    FROM Team T
    WHERE T.idTeam = p_idTeam;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Errore: Il Team % non è valido o non è associato ad alcun Evento.', p_idTeam;
    END IF;

    SELECT EXISTS (
        SELECT 1
        FROM Vota
        WHERE idGiudice = p_idGiudice AND idTeam = p_idTeam
    ) INTO v_voto_esistente;

    IF v_voto_esistente THEN
        RAISE EXCEPTION 'Il Giudice % ha già espresso il voto per il Team % e non può votare nuovamente o aggiornare il voto.', p_idGiudice, p_idTeam;
    END IF;

    IF NOT EXISTS (
        SELECT 1
        FROM Giudica G
        WHERE G.idGiudice = p_idGiudice AND G.idEvento = v_idEvento
    ) THEN
        RAISE EXCEPTION 'Il Giudice % non è assegnato all''Evento %.', p_idGiudice, v_idEvento;
    END IF;

    INSERT INTO Vota (idGiudice, idTeam, valore)
    VALUES (p_idGiudice, p_idTeam, p_valore);

    RETURN 'Voto ' || p_valore || ' inserito con successo per il Team ' || p_idTeam || ' (Evento ' || v_idEvento || ').';
EXCEPTION
    WHEN OTHERS THEN
        RAISE EXCEPTION 'Errore durante l''assegnazione del voto: %', SQLERRM;
END;
$$ LANGUAGE plpgsql;

```

8.2.16. Funzione get_potenziali_giudici

```
CREATE OR REPLACE FUNCTION get_potenziali_giudici(p_idEvento INT)
RETURNS TABLE (idUtente INT, nome VARCHAR, cognome VARCHAR, username VARCHAR) AS $$ 
BEGIN
    RETURN QUERY
    SELECT U.idUtente, U.nome, U.cognome, U.username
    FROM Utente U
    WHERE U.dataNascita <= CURRENT_DATE - INTERVAL '18 years'
        AND NOT EXISTS (SELECT 1
                          FROM Giudice G JOIN Giudica J ON G.idGiudice = J.idGiudice
                          WHERE G.idUtente = U.idUtente AND J.idEvento = p_idEvento)
    AND NOT EXISTS (SELECT 1
                      FROM Partecipante P JOIN Partecipazione PA ON P.idPartecipante = PA.idPartecipante
                      WHERE P.idUtente = U.idUtente AND PA.idEvento = p_idEvento)
    AND U.idUtente NOT IN (SELECT O.idUtente
                           FROM Organizzatore O JOIN Evento E ON O.idOrganizzatore = E.idOrganizzatore
                           WHERE E.idEvento = p_idEvento)
    AND NOT EXISTS (SELECT 1
                      FROM Invito I
                      WHERE I.idUtente = U.idUtente AND I.idEvento = p_idEvento AND I.stato = 'Rifiutato');
END;
$$ LANGUAGE plpgsql;
```

8.2.17. View classifica_finale_team

```
CREATE OR REPLACE VIEW classifica_finale_team AS
    SELECT E.idEvento, E.titolo, T.idTeam, T.nomeTeam,
           CAST(AVG(V.valore) AS DECIMAL(10, 2)) AS punteggio_medio
    FROM Vota V JOIN Team T ON V.idTeam = T.idTeam JOIN Evento E ON T.idEvento = E.idEvento
    GROUP BY E.idEvento, E.titolo, T.idTeam, T.nomeTeam
    ORDER BY E.idEvento, punteggio_medio DESC;
```

Questa view è stata progettata per visualizzare i risultati definitivi di un evento una volta che il periodo di votazione è concluso.