Università degli Studi di Napoli Federico II
Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione
Corso di Laurea in Informatica

**Basi di Dati**

Anno Accademico 2024–2025

Elaborato di gruppo – Traccia 2

# Progettazione e Implementazione di un Database Relazionale per la Gestione di Hackathon

**Candidati**

Minopoli Alessandro - N86004964
Megna Daniele - N86005120
Iodice Simone - N86005076

**Docenti**

Prof. Silvio Barra

Napoli, 16 settembre 2025

# Contents

# 1 Introduzione

Il presente elaborato ha l'obiettivo di documentare la progettazione e lo sviluppo di una base di dati relazionale in PostgreSQL, realizzata dagli studenti Minopoli Alessandro, Megna Daniele e Iodice Simone, iscritti al Corso di Laurea in Informatica presso l'Università degli Studi di Napoli "Federico II". Il database è stato sviluppato come progetto a fini valutativi per l'insegnamento di Basi di Dati e implementa un sistema di gestione degli Hackathon.

## 1.1 Descrizione del problema

Saranno presentati la progettazione e lo sviluppo di una base di dati relazionale, finalizzata all'implementazione di un sistema di gestione di gare (Hackathon). Il sistema consente l'organizzazione, la partecipazione e la valutazione di tali eventi.

La piattaforma gestisce diverse tipologie di utenti: **Partecipanti**, **Giudici** e **Organizzatori**. In funzione del ruolo ricoperto, sono previste specifiche funzionalità:

- **Organizzatori**: crea l'Hackaton e convoca i giudici;
- **Giudici**: valuta i Team e controlla i documenti;
- **Partecipanti**: formazione di team e caricamento dei documenti di progetto.

# 2 Progettazione Concettuale

In questo capitolo viene documentata la progettazione del database al livello di astrazione più elevato. A partire dall'analisi dei requisiti da soddisfare, si giunge alla definizione di uno schema concettuale, indipendente sia dalla struttura dei dati sia dalla loro implementazione fisica, rappresentato mediante un **Class Diagram UML**. Tale schema mette in evidenza le entità rilevanti del problema, le relazioni tra esse e gli eventuali vincoli da imporre.

## 2.1 Class Diagram

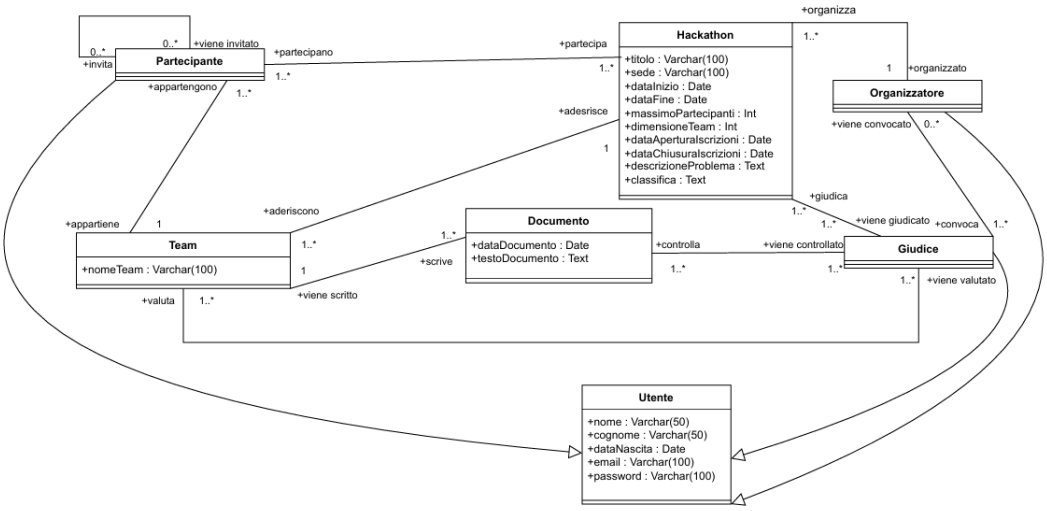


Figure 2.1: Diagramma del sistema Hackathon

## 2.2 Ristrutturazione del Class Diagram

Si effettua la ristrutturazione del Class Diagram con l'obiettivo di renderlo idoneo alla traduzione in schemi relazionali e di aumentarne l'efficienza. Tale ristrutturazione sarà articolata secondo i seguenti punti:

- Analisi delle chiavi
- Analisi degli attributi derivati

- Analisi delle ridondanze
- Analisi degli attributi strutturati
- Analisi degli attributi a valore multiplo
- Analisi delle gerarchie di specializzazione
- Partizionamento/Accorpamento delle entità o relazioni

### 2.2.1 Analisi delle chiavi

Nel modello sviluppato non sono state utilizzate chiavi naturali come identificatori primari.

Sebbene attributi come l'**email** degli utenti o il **titolo** dell'hackathon possano apparire univoci nel dominio, essi non sono stati ritenuti adatti come chiavi primarie in quanto:

- comportano **maggiore costo computazionale** nelle ricerche e nei confronti, poiché sono stringhe anziché valori numerici;
- sono **soggetti a modifiche** (ad esempio un utente può cambiare indirizzo email), compromettendo la stabilità della chiave;
- rendono più **pesanti gli indici** e meno efficienti le relazioni con chiavi esterne;
- espongono **dati sensibili** (come l'email) all'interno delle relazioni, riducendo la riservatezza del modello.

Per questo motivo, è stata adottata la soluzione di introdurre **chiavi surrogate numeriche (ID)** per tutte le entità.

Gli attributi potenzialmente univoci sono comunque mantenuti con **vincoli UNIQUE**, così da garantire la coerenza dei dati senza comprometterne l'efficienza.

### 2.2.2 Analisi degli attributi derivati

Nel modello sviluppato non sono stati introdotti in generale attributi derivati.

Unica eccezione è rappresentata dall'attributo **classifica** dell'entità **Hackathon**, che può essere considerato derivato in quanto non inserito manualmente ma calcolato automaticamente a partire dai voti assegnati ai team dai giudici.

La funzione `generazione_classifica` provvede infatti ad aggiornarlo in modo dinamico, garantendo che lo stato della competizione risulti sempre coerente con le valutazioni espresse.

### 2.2.3 Analisi delle ridondanze

Dopo un'attenta valutazione dello schema e grazie al corretto utilizzo dei vincoli e dei trigger implementati, possiamo concludere che non sussistono ridondanze effettive all'interno del modello.

### 2.2.4 Analisi degli attributi strutturati

È necessario analizzare e, se presenti, correggere eventuali attributi strutturati all'interno delle entità. Tali attributi non risultano infatti logicamente rappresentabili in un DBMS e devono pertanto essere eliminati o codificati in altra forma.

Nel modello concettuale proposto non sono stati individuati attributi di questo tipo, per cui non si rende necessaria alcuna modifica.

### 2.2.5 Analisi degli attributi a valore multiplo

Nel modello sviluppato non sono stati individuati attributi a valore multiplo. Tutti gli attributi definiti per le varie entità sono infatti di tipo atomico e possono assumere un unico valore per ogni istanza.

Ad esempio, un partecipante possiede una sola `email` e una sola `dataNascita`, un team è caratterizzato da un unico `nomeTeam`, e un documento è associato a una sola `dataDocumento`. Per questo motivo non è stato necessario introdurre ulteriori entità o relazioni per gestire attributi multivalore.

### 2.2.6 Analisi delle gerarchie di specializzazione

Occorre infine ristrutturare le eventuali gerarchie di specializzazione, poiché anch'esse non sono direttamente rappresentabili in un DBMS relazionale.

Nel modello proposto è presente un'unica gerarchia di specializzazione, quella relativa agli *Utenti della piattaforma*. L'entità generica *Utente* si specializza infatti in *Partecipante*, *Giudice* oppure *Organizzatore*.

Questa specializzazione è:

- **Totale**: ogni utente deve appartenere a una delle tre categorie;
- **Disgiunta**: un utente non può essere contemporaneamente partecipante, giudice e organizzatore.

Ai fini della rappresentazione logica abbiamo adottato la tecnica dell'**accorpamento del padre nei figli**, poiché si tratta di una specializzazione totale e disgiunta. L'entità *Utente* è stata quindi eliminata, mentre i suoi attributi e le sue associazioni sono stati trasferiti alle tre specializzazioni.

Il risultato è costituito da tre entità indipendenti: *Partecipante*, *Giudice* e *Organizzatore*, ciascuna dotata degli attributi originariamente ereditati da *Utente*.

### 2.2.7 Partizionamento/Accorpamento delle entità o delle associazioni

Nel modello sviluppato non è stato necessario effettuare **partizionamenti di entità**, in quanto tutte le entità individuate risultano già sufficientemente specifiche e prive di ambiguità semantiche. Allo stesso modo, non sono emersi casi in cui fosse opportuno accorpare più entità in una sola.

Sono stati invece effettuati diversi **partizionamenti di associazioni**, al fine di rappresentare in modo più accurato le informazioni aggiuntive collegate alle relazioni. In particolare:

- l'associazione tra *Giudice* e *Documento* è stata trasformata nell'entità *Controllo*, con attributo `dataControllo` e `commento`;
- l'associazione tra *Team* e *Giudice* è stata trasformata nell'entità *Voto*, con attributo `voto`;
- l'associazione tra *Giudice* e *Hackathon* è stata trasformata nell'entità *GiudiceHackathon*;
- l'associazione tra *Organizzatore* e *Giudice* è stata trasformata nell'entità *InvitoOrganizzatore*, con attributi `dataInvito` ed `esito`;
- l'associazione tra *Partecipante* e *Hackathon* è stata trasformata nell'entità *PartecipanteHackathon*, con attributo `dataIscrizione`;
- l'associazione ricorsiva tra partecipanti è stata trasformata nell'entità *InvitoPartecipante*, con attributi `dataInvito`, `motivazione` ed `esito`.

## 2.3 Class Diagram Ristrutturato

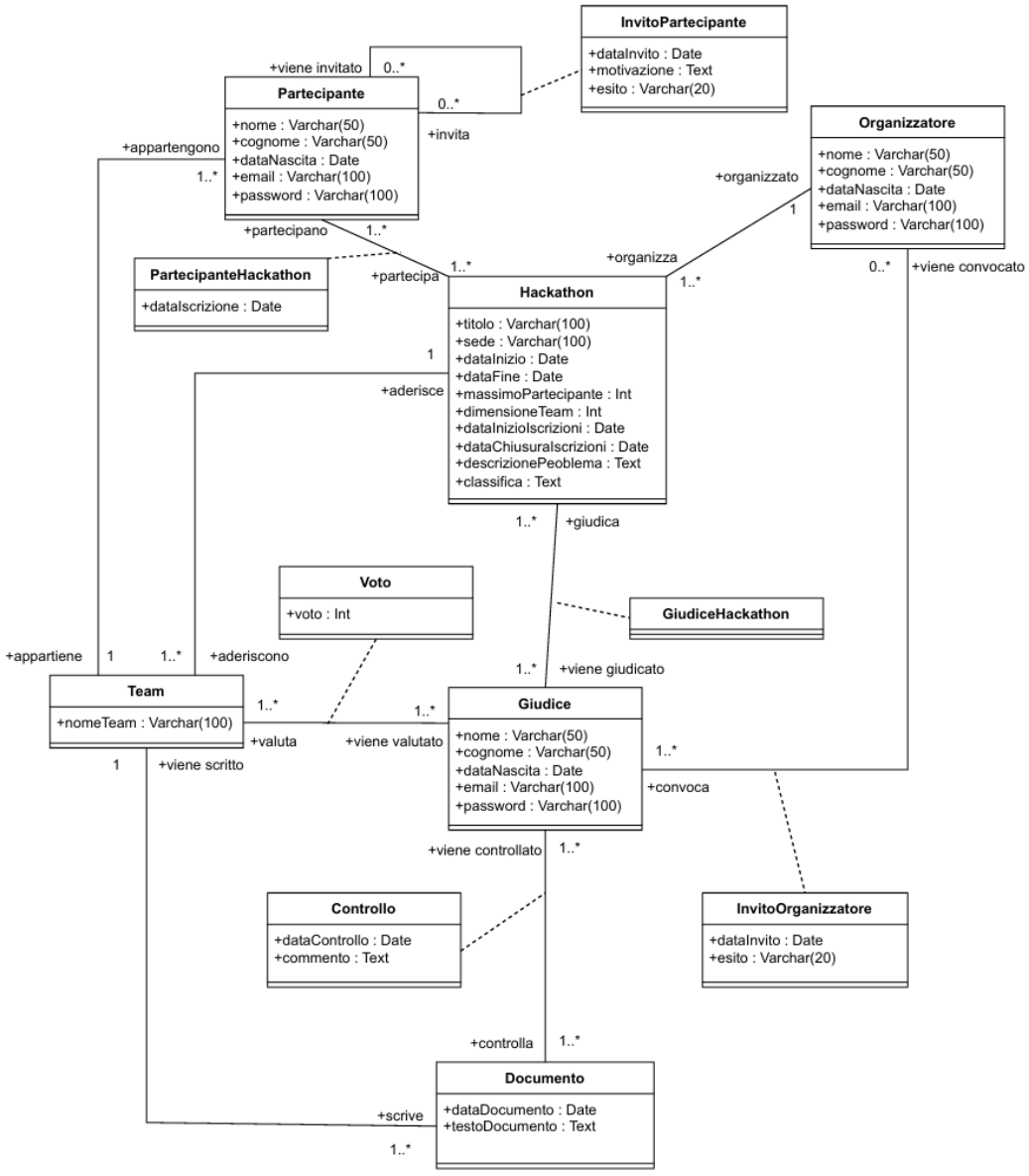


Figure 2.2: Diagramma del sistema Hackathon (ristrutturato)

## 2.4 Dizionario delle classi

Table 2.1: Dizionario delle classi

| **Classe** | **Descrizione** | **Attributi** |
|---|---|---|
| Organizzatore | Utente addetto alla creazione degli Hackathon ed invita i giudici a partecipare | idOrganizzatore (int) (PK)<br>nome (string)<br>cognome (string)<br>dataNascita (date)<br>email (string)<br>password (string) |
| Giudice | Utente addetto alla correzione dei team | idGiudice (int) (PK)<br>nome (string)<br>cognome (string)<br>dataNascita (date)<br>email (string)<br>password (string) |
| Partecipante | Utente addetto a partecipare all'Hackathon con un Team | idPartecipante (int) (PK)<br>nome (string)<br>cognome (string)<br>dataNascita (date)<br>email (string)<br>password (string) |
| Team | Squadra che gareggia ad un Hackathon | idTeam (int) (PK)<br>nomeTeam (string) |
| Documento | File caricati dal Team e verranno commentati dai giudici | idDocumento (int) (PK)<br>dataDocumento (date)<br>testoDocumento (string) |
| Hackathon | Competizione a cui si iscrivono i team | idHackathon (int) (PK)<br>titolo (string)<br>sede (string)<br>dataInizio (date)<br>dataFine (date)<br>massimoPartecipanti (int)<br>dimensioneTeam (int)<br>dataAperturaIscrizioni (date)<br>dataChiusuraIscrizioni (date)<br>classifica (string)<br>descrizioneProblema (string) |

## 2.5 Dizionario delle Associazioni

Table 2.2: Dizionario delle associazioni

| **Nome** | **Descrizione** | **Classi coinvolte** |
|---|---|---|
| Organizza | Descrive come l'organizzatore può creare diversi Hackathon | Organizzatore [1..*] ruolo organizza: indica l'organizzatore della competizione<br>Hackathon [1] ruolo organizzato: indica la competizione creata dall'organizzatore |
| Aderisce | Descrive i team che partecipano all'Hackathon | Team [1] ruolo aderisce: indica il team a quale competizione aderisce<br>Hackathon [1..*] ruolo aderiscono: indica i team che aderiscono alla competizione |
| Appartiene | Descrive come i partecipanti appartengono ai team | Partecipante [1] ruolo appartiene: indica a quale team appartiene<br>Team [1..*] ruolo appartengono: indica il team a cui appartengono i partecipanti |
| Scrive | Descrive il tema che scrive il documento | Team [1..*] ruolo scrive: indica il team che scrive il documento<br>Documento [1..*] ruolo viene scritto: indica il documento scritto dal team |

| **Nome** | **Descrizione** | **Classi coinvolte** |
|---|---|---|
| GiudiceHackathon | Descrive come i giudici possono giudicare l'Hackathon | Giudice [1..*] ruolo giudica: indica il giudice che giudica l'hackathon<br>Hackathon [1..*] ruolo viene giudicato: indica come viene giudicato dal giudice<br>GiudiceHackathon classe associativa: esprime la relazione tra i vari giudici e i vari Hackathon |
| Voto | Descrive come i team vengono valutati dai giudici | Giudice [1..*] ruolo valuta: indica come il giudice valuta i team<br>Team [1..*] ruolo viene valutato: indica il team valutato dal giudice<br>Voto classe associativa: indica la valutazione di un giudice ad un team con un voto che va da 0 a 10 |
| Controllo | Descrive come il giudice controlla il documento dei team | Giudice [1..*] ruolo controlla: indica il giudice che controlla il documento<br>Documento [1..*] ruolo viene controllato: indica il documento controllato dal giudice<br>Controllo classe associativa: indica il giudice che controlla il documento con dataControllo quando avviene e un commento da parte dei giudici opzionale |

| **Nome** | **Descrizione** | **Classi coinvolte** |
|---|---|---|
| InvitoOrganizzatore | Descrive come l'organizzatore invita il giudice | Organizzatore [1..*] ruolo convoca: indica l'organizzatore che invita il giudice<br>Giudice [0..*] ruolo viene convocato: indica il giudice invitato dall'organizzatore<br>InvitoOrganizzatore<br>classe associativa: indica l'organizzatore che invita il giudice con dataInvito e esito dove l'invito è stato accettato o rifiutato |
| InvitoPartecipante | Descrive come il partecipante invita un altro partecipante | Partecipante [0..*] ruolo invita: indica il partecipante che invita un altro partecipante<br>Partecipante [0..*] ruolo viene invitato: indica il Partecipante invitato da un altro<br>InvitoPartecipante classe associativa: indica l'invito di un partecipante a un altro partecipante con dataInvito quando viene invitato, motivazione dell'invito da parte del partecipante che invita e esito dove l'invito è stato accettato o rifiutato |

| Nome | Descrizione | Classi coinvolte |
|---|---|---|
| PartecipanteHackathon | Descrive come il partecipante partecipa all'Hackathon | Partecipante [1..*] ruolo partecipa: indica il Partecipante che partecipa all'Hackathon<br>Hackathon [1..*] ruolo partecipano: indica all'Hackathon quale partecipante partecipa<br>PartecipanteHackathon classe associativa: indica a quale Hackathon partecipa il partecipante con dataIscrizione |

## 2.6 Dizionario dei vincoli

Table 2.3: Dizionario dei vincoli

| Nome | Descrizione |
|---|---|
| `team_pkey` | L'idTeam deve essere univoco. |
| `team_idHackathon_fkey` | Ad ogni team deve essere associato l'Hackathon a cui partecipa. |
| `unique_nomeTeam_idHackathon` | La coppia nomeTeam e idHackathon deve essere univoca, così possono esistere due team con stesso nome ma in hackathon diversi. |
| `voto_pkey` | La coppia idGiudice e idTeam devono essere univoci per l'assegnazione del voto. |
| `voto_voto_check` | Il voto deve essere compreso tra 0 e 10. |
| `voto_idGiudice_fkey` | Ad ogni voto deve essere associato il giudice che dà il voto. |
| `voto_idTeam_fkey` | Ad ogni voto deve essere associato il team che riceve il voto. |
| `controllo_pkey` | La coppia idDocumento e idGiudice devono essere univoci per il controllo di un documento da parte di un giudice. |

| Nome | Descrizione |
|---|---|
| controllo_idDocumento_fkey | Ad ogni controllo deve essere associato il documento che viene controllato. |
| controllo_idGiudice_fkey | Ad ogni controllo deve essere associato il giudice che effettua il controllo. |
| documento_pkey | L'idDocumento deve essere univoco. |
| documento_idTeam_fkey | Ad ogni documento deve essere associato il team che lo scrive. |
| giudice_pkey | L'idGiudice deve essere univoco. |
| unique_email_giudice | L'email del giudice deve essere univoca, non possono esserci più giudici con la stessa email. |
| check_email_giudice | Controllo che l'email del giudice sia in formato valido. |
| giudicehackathon_pkey | La coppia idGiudice e idHackathon deve essere univoca per indicare il giudice che giudica l'Hackathon. |
| giudicehackathon_idGiudice_fkey | Ad ogni giudicehackathon deve essere associato il giudice che giudica l'Hackathon. |
| giudicehackathon_idHackathon_fkey | Ad ogni giudicehackathon deve essere associato l'Hackathon che viene giudicato. |
| hackathon_pkey | L'idHackathon deve essere univoco. |
| hackathon_idOrganizzatore_fkey | Ad ogni hackathon deve essere associato l'organizzatore che lo crea. |
| check_date_hackathon | La data di fine deve essere maggiore della data di inizio. |
| check_iscrizioni_periodo_hackathon | La chiusura iscrizioni deve essere successiva all'apertura. |
| check_dimensioneteam_hackathon | La dimensioneTeam deve essere $> 1$ (almeno 2 persone). |

| Nome | Descrizione |
|---|---|
| check_massimo partecipanti _hackathon | massimoPartecipanti deve essere $> 1$. |
| unique_titolo _hackathon | Il titolo di un Hackathon deve essere univoco. |
| invitoorganizzatore _pkey | La coppia mittente–destinatario (idOrganizzatore, idGiudice) deve essere univoca. |
| invitoorganizzatore_ mittente _fkey | Ogni invito ha un mittente (idOrganizzatore). |
| invitoorganizzatore_ destinatario _fkey | Ogni invito ha un destinatario (idGiudice). |
| invitoorganizzatore_ eventoInvitato _fkey | Ogni invito ha un evento invitato (idHackathon). |
| check_esito _invitoorganizzatore | L'esito può essere solo accettato o rifiutato. |
| unique_mittente _destinatario _eventoinvitato _invitoorganizzatore | La tripla mittente–destinatario–evento deve essere univoca. |
| organizzatore _pkey | L'idOrganizzatore deve essere univoco. |
| unique_email _organizzatore | L'email di un organizzatore deve essere unica. |
| check_email _organizzatore | L'email dell'organizzatore deve essere valida. |
| partecipante _pkey | L'idPartecipante deve essere univoco. |
| partecipante_idTeam _fkey | Se presente, ad ogni partecipante va associato il team. |
| unique_email _partecipante | L'email del partecipante deve essere univoca. |

| Nome | Descrizione |
|---|---|
| check_email_partecipante | L'email del partecipante deve essere valida. |
| partecipantehackathon_pkey | La coppia idPartecipante–idHackathon deve essere univoca. |
| partecipantehackathon_idHackathon_fkey | Ogni record ha l'idHackathon. |
| partecipantehackathon_idPartecipante_fkey | Ogni record ha l'idPartecipante. |
| invitopartecipante_pkey | La coppia mittente–destinatario deve essere univoca. |
| invitopartecipante_mittente_fkey | Ogni invito ha un mittente (idPartecipante). |
| invitopartecipante_destinatario_fkey | Ogni invito ha un destinatario (idPartecipante). |
| invitopartecipante_eventoInvitato_fkey | Ogni invito ha un evento invitato (idHackathon). |
| check_mittente_destinatario_invitopartecipante | Mittente e destinatario non possono coincidere. |
| check_esito_invitopartecipante | L'esito può essere solo accettato o rifiutato. |
| unique_mittente_destinatario_eventoinvitato_invitopartecipante | La tripla mittente–destinatario–evento deve essere univoca. |

# 3 Progettazione Logica

In questo capitolo affrontiamo la seconda fase della progettazione, scendendo a un livello di astrazione più basso rispetto al precedente. Lo schema concettuale viene tradotto, anche grazie alla ristrutturazione effettuata in precedenza, in uno schema logico, questa volta legato alla struttura dei dati scelta, nello specifico quella relazionale pura.

## 3.1 Schema Logico

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

- **Organizzatore** ( idOrganizzatore, nome, cognome, dataNascita, email, password )
- **Giudice** ( idGiudice, nome, cognome, dataNascita, email, password )
- **Partecipante** ( idPartecipante, nome, cognome, dataNascita, email, password, idTeam )
  idTeam → Team.idTeam
- **Hackathon** ( idHackathon, titolo, sede, dataInizio, dataFine, massimoPartecipanti, dimensioneTeam, dataAperturaIscrizioni, dataChiusuraIscrizioni, descrizioneProblema, classifica, idOrganizzatore )
  idOrganizzatore → Organizzatore.idOrganizzatore
- **Team** ( idTeam, nomeTeam, idHackathon )
  idHackathon → Hackathon.idHackathon
- **Documento** ( idDocumento, dataDocumento, testoDocumento, idTeam )
  idTeam → Team.idTeam
- **Voto** ( idTeam, idGiudice, voto )
  idTeam → Team.idTeam
  idGiudice → Giudice.idGiudice

- **Controllo** ( idDocumento, idGiudice, dataControllo, commento )
  idDocumento → Documento.idDocumento
  idGiudice → Giudice.idGiudice

- **GiudiceHackathon** ( idGiudice, idHackathon )
  idGiudice → Giudice.idGiudice
  idHackathon → Hackathon.idHackathon

- **PartecipanteHackathon** ( idPartecipante, idHackathon, dataIscrizione )
  idPartecipante → Partecipante.idPartecipante
  idHackathon → Hackathon.idHackathon

- **InvitoOrganizzatore** ( mittente, destinatario, dataInvito, eventoInvitato, esito )
  mittente → Organizzatore.idOrganizzatore
  destinatario → Giudice.idGiudice
  eventoInvitato → Hackathon.idHackathon

- **InvitoPartecipante** ( mittente, destinatario, dataInvito, motivazione, esito, eventoInvitato )
  mittente → Partecipante.idPartecipante
  destinatario → Partecipante.idPartecipante
  eventoInvitato → Hackathon.idHackathon

# 4 Progettazione Fisica

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

## 4.1 Definizione Tabelle

Di seguito sono riportate le definizioni delle tabelle, dei loro vincoli intrarelazionali e di eventuali semplici strutture per la loro gestione. Il codice è espresso in linguaggio SQL per PostgreSQL.

### 4.1.1 Definizione della Tabella ORGANIZZATORE

```
-- Definizione tabella
CREATE TABLE Organizzatore (
    idOrganizzatore SERIAL PRIMARY KEY,
    nome VARCHAR(50) NOT NULL,
    cognome VARCHAR(50) NOT NULL,
    dataNascita DATE NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE CHECK (email LIKE '%_@__%.
        __%'),
    password VARCHAR(50) NOT NULL
);
```

### 4.1.2 Definizione della Tabella GIUDICE

```
-- Definizione tabella
CREATE TABLE Giudice (
    idGiudice SERIAL PRIMARY KEY,
    nome VARCHAR(50) NOT NULL,
    cognome VARCHAR(50) NOT NULL,
    dataNascita DATE NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE CHECK (email LIKE '%_@__%.
        __%'),
```

```
    password VARCHAR(50) NOT NULL
);
```

### 4.1.3 Definizione della Tabella PARTECIPANTE

```
-- Definizione tabella
CREATE TABLE Partecipante (
    idPartecipante SERIAL PRIMARY KEY,
    nome VARCHAR(50) NOT NULL,
    cognome VARCHAR(50) NOT NULL,
    dataNascita DATE NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE CHECK (email LIKE '%_@__%.__%'),
    password VARCHAR(50) NOT NULL,
    idTeam INT,
    FOREIGN KEY (idTeam) REFERENCES Team(idTeam)
);
```

### 4.1.4 Definizione della Tabella HACKATHON

```
-- Definizione tabella
CREATE TABLE Hackathon (
    idHackathon SERIAL PRIMARY KEY,
    titolo VARCHAR(100) NOT NULL UNIQUE,
    sede VARCHAR(100) NOT NULL,
    dataInizio DATE NOT NULL,
    dataFine DATE NOT NULL,
    dataAperturaIscrizioni DATE NOT NULL,
    dataChiusuraIscrizioni DATE NOT NULL,
    descrizioneProblema TEXT,
    classifica TEXT,
    massimoPartecipanti INT NOT NULL CHECK (massimoPartecipanti > 1),
    dimensioneTeam INT NOT NULL CHECK (dimensioneTeam > 1),
    idOrganizzatore INT NOT NULL,
    FOREIGN KEY (idOrganizzatore) REFERENCES Organizzatore(idOrganizzatore),
    CHECK (dataInizio < dataFine),
    CHECK (dataAperturaIscrizioni < dataChiusuraIscrizioni)
);
```

### 4.1.5 Definizione della Tabella TEAM

```
-- Definizione tabella
CREATE TABLE Team (
    idTeam SERIAL PRIMARY KEY,
    nomeTeam VARCHAR(100) NOT NULL,
    idHackathon INT NOT NULL,
    FOREIGN KEY (idHackathon) REFERENCES Hackathon(idHackathon),
    CONSTRAINT unique_nome_team_hackathon UNIQUE (nomeTeam,
       idHackathon)
);
```

### 4.1.6 Definizione della Tabella DOCUMENTO

```
-- Definizione tabella
CREATE TABLE Documento (
    idDocumento SERIAL PRIMARY KEY,
    dataDocumento DATE NOT NULL,
    testoDocumento TEXT NOT NULL,
    idTeam INT NOT NULL,
    FOREIGN KEY (idTeam) REFERENCES Team(idTeam)
);
```

### 4.1.7 Definizione della Tabella VOTO

```
-- Definizione tabella
CREATE TABLE Voto (
    idTeam INT NOT NULL,
    idGiudice INT NOT NULL,
    voto INT NOT NULL CHECK (voto BETWEEN 0 AND 10),
    PRIMARY KEY (idTeam, idGiudice),
    FOREIGN KEY (idTeam) REFERENCES Team(idTeam),
    FOREIGN KEY (idGiudice) REFERENCES Giudice(idGiudice)
);
```

### 4.1.8 Definizione della Tabella GIUDICEHACKATHON

```
-- Definizione tabella
CREATE TABLE GiudiceHackathon (
    idGiudice INT NOT NULL,
```

```
    idHackathon INT NOT NULL,
    PRIMARY KEY (idGiudice, idHackathon),
    FOREIGN KEY (idGiudice) REFERENCES Giudice(idGiudice),
    FOREIGN KEY (idHackathon) REFERENCES Hackathon(idHackathon)
);
```

### 4.1.9 Definizione della Tabella CONTROLLO

```
-- Definizione tabella
CREATE TABLE Controllo (
    idDocumento INT NOT NULL,
    idGiudice INT NOT NULL,
    dataControllo DATE NOT NULL,
    commento TEXT,
    PRIMARY KEY (idDocumento, idGiudice),
    FOREIGN KEY (idDocumento) REFERENCES Documento(idDocumento),
    FOREIGN KEY (idGiudice) REFERENCES Giudice(idGiudice)
);
```

### 4.1.10 Definizione della Tabella PARTECIPANTEHACKATHON

```
-- Definizione tabella
CREATE TABLE PartecipanteHackathon (
    idPartecipante INT NOT NULL,
    idHackathon INT NOT NULL,
    dataIscrizione DATE NOT NULL,
    PRIMARY KEY (idPartecipante, idHackathon),
    UNIQUE (idPartecipante, idHackathon),
    FOREIGN KEY (idPartecipante) REFERENCES Partecipante(
       idPartecipante),
    FOREIGN KEY (idHackathon) REFERENCES Hackathon(idHackathon)
);
```

### 4.1.11 Definizione della Tabella INVITOPARTECIPANTE

```
-- Definizione tabella
CREATE TABLE InvitoPartecipante (
    mittente INT NOT NULL,
    destinatario INT NOT NULL,
    eventoInvitato INT NOT NULL,
```

```
    dataInvito DATE NOT NULL,
    motivazione TEXT,
    esito VARCHAR(20) NOT NULL CHECK (esito IN ('accettato','rifiutato')),
    PRIMARY KEY (mittente, destinatario, dataInvito),
    UNIQUE (mittente, destinatario, eventoInvitato),
    FOREIGN KEY (mittente) REFERENCES Partecipante(idPartecipante),
    FOREIGN KEY (destinatario) REFERENCES Partecipante(idPartecipante),
    FOREIGN KEY (eventoInvitato) REFERENCES Hackathon(idHackathon),
    CHECK (mittente <> destinatario)
);
```

### 4.1.12 Definizione della Tabella INVITOORGANIZZATORE

```
-- Definizione tabella
CREATE TABLE InvitoOrganizzatore (
    mittente INT NOT NULL,
    destinatario INT NOT NULL,
    eventoInvitato INT NOT NULL,
    dataInvito DATE NOT NULL,
    esito VARCHAR(20) NOT NULL CHECK (esito IN ('accettato','rifiutato')),
    PRIMARY KEY (mittente, destinatario, dataInvito),
    UNIQUE (mittente, destinatario, eventoInvitato),
    FOREIGN KEY (mittente) REFERENCES Organizzatore(idOrganizzatore),
    FOREIGN KEY (destinatario) REFERENCES Giudice(idGiudice),
    FOREIGN KEY (eventoInvitato) REFERENCES Hackathon(idHackathon)
);
```

## 4.2 Implementazione dei vincoli

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

### 4.2.1 Implementazione del vincolo Partecipante non può stare in più team nello stesso Hackathon

Avviene un controllo per vedere se il partecipante è presente in un team oppure no, perchè il partecipante può stare in un solo team se partecipa ad un Hackathon

```
CREATE OR REPLACE FUNCTION verifica_team_partecipante()
RETURNS TRIGGER AS $$
DECLARE
hackathon_corrente INT;
BEGIN
-- Se non assegnato un team, nessun controllo
IF NEW."idTeam" IS NULL THEN
RETURN NEW;
END IF;
 -- Recupera l'hackathon del team assegnato
SELECT "idHackathon" INTO hackathon_corrente
FROM "Team"
WHERE "idTeam" = NEW."idTeam";
 -- Controlla che il partecipante non sia in un altro team dello
    stesso hackathon
IF EXISTS (
SELECT 1
FROM "Partecipante" p
JOIN "Team" t ON p."idTeam" = t."idTeam"
WHERE p."idPartecipante" = NEW."idPartecipante"
AND t."idHackathon" = hackathon_corrente
AND (NEW."idTeam" IS DISTINCT FROM p."idTeam") -- evita falsi
   positivi quando resta nello stesso team
) THEN
RAISE EXCEPTION 'Il partecipante % partecipa in un altro team
   nello stesso hackathon',
NEW."idPartecipante";
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trg_verifica_team_partecipante
BEFORE INSERT OR UPDATE OF "idTeam" ON "Partecipante"
FOR EACH ROW
EXECUTE FUNCTION verifica_team_partecipante();
```

### 4.2.2 Implementazione del vincolo il team non può superare la dimensione massima stabilita dall'Hackathon

Avviene un controllo per vedere se la dimensione dei vari team che partecipano ad un Hackathon supera la dimensione massima stabilita dall'Hackathon a cui partecipano

```
CREATE OR REPLACE FUNCTION verifica_dimensione_team()
RETURNS TRIGGER AS $$
DECLARE
dimensione_massima INT;
numero_attuale INT;
BEGIN
-- Se il partecipante non assegnato a un team, saltare il
   controllo
IF NEW."idTeam" IS NULL THEN
RETURN NEW;
END IF;
 -- Recupera la dimensione massima dall'hackathon del team
SELECT h."dimensioneTeam"
INTO dimensione_massima
FROM "Hackathon" h
JOIN "Team" t ON t."idHackathon" = h."idHackathon"
WHERE t."idTeam" = NEW."idTeam";
 -- Se il team non esiste errore
IF dimensione_massima IS NULL THEN
RAISE EXCEPTION 'Il team % non associato a un hackathon valido',
   NEW."idTeam";
END IF;
 -- Conta i partecipanti attuali del team
SELECT COUNT(*) INTO numero_attuale
FROM "Partecipante"
WHERE "idTeam" = NEW."idTeam";
 -- Se ha raggiunto la dimensione massima errore
IF numero_attuale >= dimensione_massima THEN
RAISE EXCEPTION 'Il team % ha raggiunto la dimensione massima di
   % partecipanti',
NEW."idTeam", dimensione_massima;
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trg_verifica_dimensione_team
```

```
BEFORE INSERT OR UPDATE ON "Partecipante"
FOR EACH ROW
EXECUTE FUNCTION verifica_dimensione_team();
```

### 4.2.3 Implementazione del vincolo non superare il numero di partecipanti iscritti all'Hackathon

Avviene un controllo peer vedere se il numero di partecipanti ad un Hackathon supera la dimensione stabilita dall'Hackathon stesso

```
CREATE OR REPLACE FUNCTION verifica_limite_partecipanti()
RETURNS TRIGGER AS $$
DECLARE
max_partecipanti INTEGER;
partecipanti_attuali INTEGER;
BEGIN
-- Recupera il limite massimo dall'hackathon
SELECT "massimoPartecipanti"
INTO max_partecipanti
FROM "Hackathon"
WHERE "idHackathon" = NEW."idHackathon"
FOR UPDATE;
 -- Conta i partecipanti iscritti
SELECT COUNT(*)
INTO partecipanti_attuali
FROM "PartecipanteHackathon"
WHERE "idHackathon" = NEW."idHackathon";
 -- Se superato il limite errore
IF partecipanti_attuali >= max_partecipanti THEN
RAISE EXCEPTION 'L''hackathon % ha raggiunto il numero massimo di
    % partecipanti',
NEW."idHackathon", max_partecipanti;
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trg_verifica_limite_partecipanti
BEFORE INSERT OR UPDATE ON "PartecipanteHackathon"
FOR EACH ROW
EXECUTE FUNCTION verifica_limite_partecipanti();
```

### 4.2.4 Implementazione del vincolo Invito partecipante, il mittente deve essere iscritto ad un Hackathon prima di invitare il destinatario che non deve stare in nessun team e deve essere iscritto allo stesso Hackathon del mittente

Qundo il mittente invia un invito al destinatario per unirsi al suo team, avvengono una serie di controlli, perchè il mittente deve essere iscritto all'Hackaton e il destinatario deve essere iscritto allo stesso Hackathon del mittente e non avere nessun team

```
CREATE OR REPLACE FUNCTION verifica_invito_partecipante()
RETURNS TRIGGER AS $$
DECLARE
idHackathonMittente INT;
idTeamMittente INT;
idTeamDestinatario INT;
BEGIN
-- Recupera il team del mittente e l'hackathon di quel team
SELECT t."idTeam", t."idHackathon"
INTO idTeamMittente, idHackathonMittente
FROM "Team" t
JOIN "Partecipante" p ON p."idTeam" = t."idTeam"
WHERE p."idPartecipante" = NEW."mittente";
 -- Il mittente deve avere un team
IF idTeamMittente IS NULL THEN
RAISE EXCEPTION 'Il mittente % non appartiene a nessun team,
   quindi non pu  inviare inviti', NEW."mittente";
END IF;
 -- Il mittente deve invitare solo partecipanti per lo stesso
    hackathon
IF idHackathonMittente <> NEW."eventoInvitato" THEN
RAISE EXCEPTION 'Il mittente % non appartiene all''hackathon %',
   NEW."mittente", NEW."eventoInvitato";
END IF;
 -- Controlla se il destinatario ha un team
SELECT "idTeam"
INTO idTeamDestinatario
FROM "Partecipante"
WHERE "idPartecipante" = NEW."destinatario";
 IF idTeamDestinatario IS NOT NULL THEN
RAISE EXCEPTION 'Il destinatario % si trova in un team e non lo
   puoi invitare', NEW."destinatario";
```

```
END IF;
 -- Controlla che il destinatario sia iscritto allo stesso
    hackathon
IF NOT EXISTS (
SELECT 1
FROM "PartecipanteHackathon" ph
WHERE ph."idPartecipante" = NEW."destinatario"
AND ph."idHackathon" = NEW."eventoInvitato"
) THEN
RAISE EXCEPTION 'Destinatario % non iscritto all''hackathon %',
   NEW."destinatario", NEW."eventoInvitato";
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trg_verifica_invito_partecipante
BEFORE INSERT OR UPDATE ON "InvitoPartecipante"
FOR EACH ROW
EXECUTE FUNCTION verifica_invito_partecipante();
```

### 4.2.5 Implementazione del vincolo su invito organizzatore perchè il mittente può invitare solo agli Hackathon creati da lui

Quando viene inviato l'invito, avviene un controllo per capire se il mittente sta invitando il destinatario all'Hackathon creato da lui

```
CREATE OR REPLACE FUNCTION verifica_invito_organizzatore()
RETURNS TRIGGER AS $$
BEGIN
-- L'organizzatore mittente deve aver creato l'hackathon
IF NOT EXISTS (
SELECT 1
FROM "Hackathon" h
WHERE h."idHackathon" = NEW. "eventoInvitato"
AND h."idOrganizzatore" = NEW.mittente
) THEN
RAISE EXCEPTION 'L''organizzatore % non ha creato l''hackathon %'
   , NEW.mittente, NEW.eventoInvitato;
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER trg_verifica_invito_organizzatore
BEFORE INSERT OR UPDATE ON "InvitoOrganizzatore"
FOR EACH ROW
EXECUTE FUNCTION verifica_invito_organizzatore();
```

### 4.2.6 Implementazione del vincolo data di controllo dei giudici deve essere situata tra la data di inizio e fine dell'Hackathon

Qunado il giudice cotrolla i documenti, avviene un controllo sulla data perchè i documenti possono essere controllati solo durnante l'Hackathon, quindi tra la data di inizio e la data di fine dell'Hackathon

```
CREATE OR REPLACE FUNCTION verifica_data_controllo()
RETURNS TRIGGER AS $$
DECLARE
data_inizio DATE;
data_fine DATE;
BEGIN
SELECT h."dataInizio", h."dataFine"
INTO data_inizio, data_fine
FROM "Hackathon" h
JOIN "Team" t ON t."idHackathon" = h."idHackathon"
JOIN "Documento" d ON d."idTeam" = t."idTeam"
WHERE d."idDocumento" = NEW."idDocumento";
 -- Se non ha trovato nessuna corrispondenza
IF NOT FOUND THEN
RAISE EXCEPTION 'Documento % non collegato ad alcun hackathon
   valido', NEW."idDocumento";
END IF;
 -- Controllo finestra temporale
IF NEW."dataControllo" NOT BETWEEN data_inizio AND data_fine THEN
RAISE EXCEPTION 'Data del controllo % non valida. Deve essere tra
    % e %',
NEW."dataControllo", data_inizio, data_fine;
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trg_verifica_data_controllo
BEFORE INSERT OR UPDATE ON "Controllo"
FOR EACH ROW
```

```
28 EXECUTE FUNCTION verifica_data_controllo();
```

### 4.2.7 Implementazione del vincolo controllo data iscrizione partecipanti tra data inizio e fine data iscrizioni dell'Hackaton

Qunado un partecipante si iscrive all'Hackathon avviene un controllo, perchè la data di iscrizione all'Hackathon da parte del partecipante deve essere tra data apertura iscrizioni e data chiusura iscrizioni dell'Hackathon

```
1  CREATE OR REPLACE FUNCTION verifica_data_iscrizione()
2  RETURNS TRIGGER AS $$
3  DECLARE
4  apertura DATE;
5  chiusura DATE;
6  BEGIN
7  -- Recupero date apertura e chiusura iscrizioni dell'hackathon
8  SELECT h."dataAperturaIscrizioni", h."dataChiusuraIscrizioni"
9  INTO apertura, chiusura
10 FROM "Hackathon" h
11 WHERE h."idHackathon" = NEW."idHackathon";
12  -- Controllo che l'hackathon esiste
13 IF apertura IS NULL OR chiusura IS NULL THEN
14 RAISE EXCEPTION 'Hackathon % non trovato per la verifica
      iscrizione', NEW."idHackathon";
15 END IF;
16  -- Controllo che la data di iscrizione sia  n e l l intervallo
       valido
17 IF NEW."dataIscrizione" < apertura OR NEW."dataIscrizione" >
      chiusura THEN
18 RAISE EXCEPTION 'Data di iscrizione % non valida. Deve essere tra
       % e %',
19 NEW."dataIscrizione", apertura, chiusura;
20 END IF;
21 RETURN NEW;
22 END;
23 $$ LANGUAGE plpgsql;
24 CREATE TRIGGER trg_verifica_data_iscrizione
25 BEFORE INSERT OR UPDATE ON "PartecipanteHackathon"
26 FOR EACH ROW
27 EXECUTE FUNCTION verifica_data_iscrizione();
```

### 4.2.8 Implementazione del vincolo data chiusura iscrizioni si deve chiudere due giorni prima della data di inizio dell'Hackaton

Quando si chiudono le iscrizioni di un Hackathon avviene un controllo su data di chiusura delle iscrizioni, perchè devono essere chiuse due giorni prima della data di inizio dell'Hackathon

```
CREATE OR REPLACE FUNCTION verifica_chiusura_iscrizioni()
RETURNS TRIGGER AS $$
BEGIN
-- Controlla che la chiusura iscrizioni sia almeno 2 giorni prima
    dell'inizio
IF NEW."dataChiusuraIscrizioni" > NEW."dataInizio" - INTERVAL '2
   days' THEN
RAISE EXCEPTION 'La chiusura iscrizioni % deve avvenire almeno 2
   giorni prima dell''inizio %',
NEW."dataChiusuraIscrizioni", NEW."dataInizio";
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trg_verifica_chiusura_iscrizioni
BEFORE INSERT OR UPDATE ON "Hackathon"
FOR EACH ROW
EXECUTE FUNCTION verifica_chiusura_iscrizioni();
```

## 4.3 Funzioni

Di seguito sono riportate le stored function che si è deciso di implementare per semplificare alcuni aspetti dell'utilizzo della base di dati

### 4.3.1 Funzione: Generazione classifica

Funzione che calcola la classifica finale dell'Hackathon e inserisce il risultato nell'attributo classifica di Hackathon

```
CREATE OR REPLACE FUNCTION generazione_classifica(p_idHackathon
   INT)
RETURNS TEXT AS $$
DECLARE
classifica_testo TEXT := '';
pos INT := 1;
```

```
rec RECORD;
BEGIN
FOR rec IN
SELECT t.nomeTeam, AVG(v.voto)::NUMERIC AS mediaVoti
FROM "Team" t
LEFT JOIN "Voto" v ON t.idTeam = v.idTeam
WHERE t.idHackathon = p_idHackathon
GROUP BY t.idTeam, t.nomeTeam
ORDER BY AVG(v.voto) DESC
LOOP
classifica_testo := classifica_testo || pos || '. ' || rec.
   nomeTeam || ' (' || rec.mediaVoti || ')' || E'\n';
pos := pos + 1;
END LOOP;
 -- Aggiorna  l attributo  classifica in Hackathon
UPDATE "Hackathon"
SET classifica = classifica_testo
WHERE idHackathon = p_idHackathon;
RETURN classifica_testo;
END;
$$ LANGUAGE plpgsql;
```

### 4.3.2 Funzione: Media voto di un team

Funzione che restituisce la media dei voti di un team

```
CREATE OR REPLACE FUNCTION media_voti_team(p_idTeam INT)
RETURNS NUMERIC AS $$
DECLARE
media NUMERIC;
BEGIN
SELECT AVG(voto)::NUMERIC
INTO media
FROM Voto
WHERE idTeam = p_idTeam;
 RETURN media;
END;
$$ LANGUAGE plpgsql;
```

### 4.3.3 Funzione: Documenti scritti da un team

Funzione che restituisce tutti i documenti scritti da un team

```sql
CREATE OR REPLACE FUNCTION documenti_team(
p_idTeam INT,
p_idHackathon INT
)
RETURNS TEXT AS $$
DECLARE
rec RECORD;
risultati TEXT := '';
totale INT;
BEGIN
-- Conta il numero totale di documenti del team nell'hackathon
SELECT COUNT(*) INTO totale
FROM "Documento" d
JOIN "Team" t ON d.idTeam = t.idTeam
WHERE t.idTeam = p_idTeam
AND t.idHackathon = p_idHackathon;
 risultati := 'Totale documenti: ' || totale || E'\n\n';
 -- Recupera i documenti del team
FOR rec IN
SELECT d.idDocumento, d.dataDocumento, d.testoDocumento
FROM "Documento" d
JOIN "Team" t ON d.idTeam = t.idTeam
WHERE t.idTeam = p_idTeam
AND t.idHackathon = p_idHackathon
ORDER BY d.dataDocumento
LOOP
risultati := risultati || 'ID Documento: ' || rec.idDocumento
|| ', Data: ' || rec.dataDocumento
|| ', Testo: ' || rec.testoDocumento || E'\n';
END LOOP;
RETURN risultati;
END;
$$ LANGUAGE plpgsql;
```

### 4.3.4 Funzione: Dati dei partecipanti di un team

Funzione che restituisce i dati dei partecipanti di un team

```sql
CREATE OR REPLACE FUNCTION lista_partecipanti_team(p_idTeam INT)
RETURNS TEXT AS $$
DECLARE
```

```
rec RECORD;
risultati TEXT := '';
BEGIN
risultati := 'Partecipanti del team ' || p_idTeam || E'\n\n';
 FOR rec IN
SELECT idPartecipante, nome, cognome, email, dataNascita
FROM Partecipante
WHERE idTeam = p_idTeam
ORDER BY cognome, nome
LOOP
risultati := risultati
|| 'ID: ' || rec.idPartecipante
|| ', Nome: ' || rec.nome
|| ', Cognome: ' || rec.cognome
|| ', Email: ' || rec.email
|| ', Data di nascita: ' || rec.dataNascita
|| E'\n';
END LOOP;
RETURN risultati;
END;
$$ LANGUAGE plpgsql;
```

### 4.3.5 Funzione: Commenti da parte dei giudici

Funzione che restituisce i dati dei documenti e i commenti fa da un giudice

```
CREATE OR REPLACE FUNCTION statistiche_giudice(p_idGiudice INT)
RETURNS TEXT AS $$
DECLARE
rec RECORD;
risultati TEXT := '';
BEGIN
 risultati := 'Statistiche giudice ID: ' || p_idGiudice || E'\n\n
    ';
 FOR rec IN
SELECT d.idDocumento,
d.dataDocumento,
d.testoDocumento,
c.commento
FROM Controllo c
JOIN Documento d ON c.idDocumento = d.idDocumento
WHERE c.idGiudice = p_idGiudice
```

```
ORDER BY d.dataDocumento
LOOP
risultati := risultati
|| 'ID Documento: ' || rec.idDocumento
|| ', Data: ' || rec.dataDocumento
|| ', Testo: ' || rec.testoDocumento
|| ', Commento: ' || COALESCE(rec.commento, 'Nessun commento')
|| E'\n';
END LOOP;
RETURN risultati;
END;
$$ LANGUAGE plpgsql;
```

### 4.3.6 Funzione: Numero di partecipanti e team di un Hackathon

Funzione che restituisce il numero di partecipanti e team totali di un Hackathon dopo la data di chiusura delle iscrizioni

```
CREATE OR REPLACE FUNCTION statistiche_hackathon(p_idHackathon
   INT)
RETURNS TEXT AS $$
DECLARE
totale_partecipanti INT;
totale_team INT;
risultati TEXT := '';
chiusura DATE;
BEGIN
-- Controlla la data di chiusura iscrizioni
SELECT dataChiusuraIscrizioni INTO chiusura
FROM "Hackathon"
WHERE idHackathon = p_idHackathon;
 IF CURRENT_DATE < chiusura THEN
RAISE EXCEPTION 'Le iscrizioni per l''hackathon % non sono ancora
    chiuse (chiusura prevista: %)',
p_idHackathon, chiusura;
END IF;
 -- Conta i partecipanti
SELECT COUNT(*) INTO totale_partecipanti
FROM "PartecipanteHackathon"
WHERE idHackathon = p_idHackathon;
 -- Conta i team
SELECT COUNT(*) INTO totale_team
```

```
23 FROM "Team"
24 WHERE idHackathon = p_idHackathon;
25  -- Costruisce il testo dei risultati
26 risultati := 'Hackathon ID: ' || p_idHackathon || E'\n'
27 || 'Totale partecipanti: ' || totale_partecipanti || E'\n'
28 || 'Totale team: ' || totale_team;
29 RETURN risultati;
30 END;
31 $$ LANGUAGE plpgsql;
```