|  |
| --- |
| Corso del Professore Silvio Barra |
| Basi di Dati – Progetto Hackathon |
| Università Degli Studi Di Napoli FEDERICO II |

|  |
| --- |
| di Francesco Grauso (N86005852)  Anno Accademico 2024/2025 |

Sommario

[1. Introduzione 4](#_Toc204340046)

[1.1 Descrizione del problema 4](#_Toc204340047)

[2. Progettazione Concettuale 4](#_Toc204340048)

[2.1. Class Diagram 5](#_Toc204340049)

[2.2. Ristrutturazione del Class Diagram 5](#_Toc204340050)

[2.2.1. Analisi delle chiavi 5](#_Toc204340051)

[2.2.2. Analisi degli attributi derivati 5](#_Toc204340052)

[2.2.3. Analisi delle ridondanze 6](#_Toc204340053)

[2.2.4. Analisi degli attributi strutturati 6](#_Toc204340054)

[2.2.5. Analisi degli attributi a valore multiplo 6](#_Toc204340055)

[2.2.6. Analisi delle gerarchie di specializzazione 6](#_Toc204340056)

[2.3. Class Diagram Ristrutturato 6](#_Toc204340057)

[2.4. Dizionario delle Classi 6](#_Toc204340058)

[2.5. Dizionario delle Associazioni 7](#_Toc204340059)

[2.6. Dizionario dei Vincoli 8](#_Toc204340060)

[3. Progettazione Logica 10](#_Toc204340061)

[3.1. Schema Logico 10](#_Toc204340062)

[4. Progettazione Fisica 11](#_Toc204340063)

[4.1. Definizione Tabelle 11](#_Toc204340064)

[4.1.1. Definizione della Tabella RUOLI\_UTENTE 11](#_Toc204340065)

[4.1.2. Definizione della Tabella STATI\_HACKATHON 12](#_Toc204340066)

[4.1.3. Definizione della Tabella STATI\_INVITO 12](#_Toc204340067)

[4.1.4. Definizione della Tabella RUOLI\_TEAM 12](#_Toc204340068)

[4.1.5. Definizione della Tabella UTENTI 12](#_Toc204340069)

[4.1.6. Definizione della Tabella HACKATHONS 12](#_Toc204340070)

[4.1.7. Definizione della Tabella TEAMS 13](#_Toc204340071)

[4.1.8. Definizione della Tabella REGISTRAZIONI 13](#_Toc204340072)

[4.1.9. Definizione della Tabella GIUDICI\_HACKATHON 14](#_Toc204340073)

[4.1.10. Definizione della Tabella PROBLEMI 14](#_Toc204340074)

[4.1.11. Definizione della Tabella PROGRESSI 14](#_Toc204340075)

[4.1.12. Definizione della Tabella COMMENTI 14](#_Toc204340076)

[4.1.13. Definizione della Tabella VOTI 15](#_Toc204340077)

[4.1.14. Definizione della Tabella INVITI\_TEAM 15](#_Toc204340078)

[4.1.15. Definizione della Tabella INVITI\_GIUDICE 15](#_Toc204340079)

[4.2. Implementazione dei vincoli 16](#_Toc204340080)

[4.2.1. Implementazione del Vincolo TRG01 16](#_Toc204340081)

[4.2.2. Implementazione del Vincolo TRG02 16](#_Toc204340082)

[4.2.3. Implementazione del Vincolo TRG03 17](#_Toc204340083)

[4.2.4. Implementazione del Vincolo TRG04 17](#_Toc204340084)

[4.2.5. Implementazione del Vincolo TRG05 18](#_Toc204340085)

[4.2.6. Implementazione del Vincolo TRG06 18](#_Toc204340086)

[4.2.7. Implementazione del Vincolo TRG07 19](#_Toc204340087)

[4.2.8. Implementazione del Vincolo TRG08 20](#_Toc204340088)

[4.2.9. Implementazione del Vincolo TRG09 20](#_Toc204340089)

[4.2.10. Implementazione del Vincolo TRG10 21](#_Toc204340090)

[4.2.11. Implementazione del Vincolo TRG11 21](#_Toc204340091)

[4.2.12. Implementazione del Vincolo TRG12 22](#_Toc204340092)

[4.3. Definizione Viste 22](#_Toc204340093)

[4.3.1. Definizione della Vista V\_CLASSIFICA\_HACKATHONS 22](#_Toc204340094)

# 1. Introduzione

Il seguente elaborato ha lo scopo di documentare la progettazione e lo sviluppo di una base di dati relazionale del DBMS PostgreSQL, ad opera dello studente Francesco Grauso del CdL in Informatica presso l’Università degli Studi di Napoli “Federico II”. Il database nasce come progetto a scopi valutativi per il corso di Basi di Dati, ed implementa un sistema di gestione hackathon.

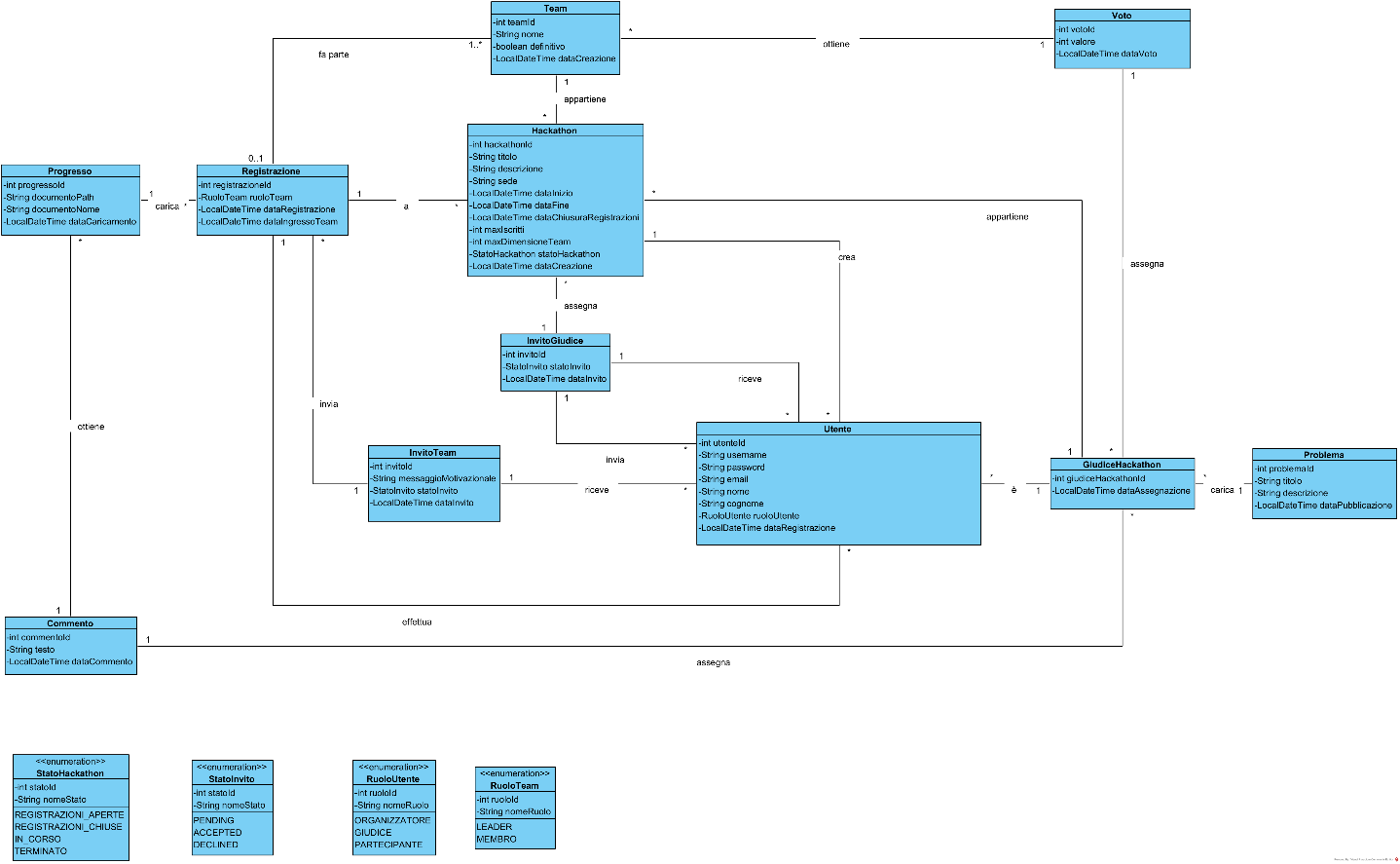
## 1.1 Descrizione del problema

Verranno riportate progettazione e sviluppo di una base di dati relazionale, che implementi un sistema di gestione hackathon. Il sistema tiene traccia dei partecipanti agli hackathon, identificando ruoli specifici per ognuno di essi (ORGANIZZATORE, GIUDICE, PARTECIPANTE). Ad ogni hackathon sono associati uno stato (REGISTRAZIONI\_APERTE, REGISTRAZIONI\_CHIUSE, IN\_CORSO, TERMINATO), una sede, date di inizio e fine, e limiti per il numero massimo di iscritti e la dimensione dei team. Il sistema permette inoltre la formazione di team, con un sistema di inviti tra partecipanti, la pubblicazione di problemi da parte dei giudici, il caricamento di progressi da parte dei team, e un sistema di valutazione finale con voti che consente di stilare classifiche.

# 2. Progettazione Concettuale

In questo capitolo documentiamo la progettazione del database al suo livello di astrazione più alto. Partendo dall’analisi dei requisiti da soddisfare, si arriverà ad uno schema concettuale indipendente dalla struttura dei dati e dall’implementazione fisica degli stessi, rappresentato con un Class Diagram UML. Quest’ultimo evidenzierà le entità rilevanti nel problema, oltre alle relazioni che intercorrono tra esse e gli eventuali vincoli da imporre.

## 2.1. Class Diagram



## 2.2. Ristrutturazione del Class Diagram

Si procede alla ristrutturazione del Class Diagram, al fine di rendere quest’ultimo idoneo alla traduzione in schemi relazionali e di migliorarne l’efficienza. La ristrutturazione procederà 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

### 2.2.1. Analisi delle chiavi

L'analisi delle chiavi è stata completata durante la fase di progettazione iniziale, con particolare attenzione all'introduzione di chiavi primarie surrogate per ottimizzare le performance e garantire l'univocità degli identificativi.

### 2.2.2. Analisi degli attributi derivati

L'analisi degli attributi derivati è stata effettuata identificando gli elementi calcolabili dinamicamente, come le classifiche degli hackathon derivate dai voti assegnati dai giudici ai team.

### 2.2.3. Analisi delle ridondanze

L'analisi delle ridondanze ha confermato l'assenza di associazioni ridondanti nel modello, garantendo un design pulito e privo di inconsistenze logiche.

### 2.2.4. Analisi degli attributi strutturati

L'analisi degli attributi strutturati non ha evidenziato la presenza di elementi complessi che richiedessero ristrutturazione per la compatibilità con il modello relazionale.

### 2.2.5. Analisi degli attributi a valore multiplo

L'analisi degli attributi a valore multiplo ha confermato che il modello non presenta attributi che necessitino di ristrutturazione per l'implementazione relazionale.

### 2.2.6. Analisi delle gerarchie di specializzazione

L'analisi delle gerarchie di specializzazione ha portato alla gestione dei ruoli utente tramite un sistema di enumerazioni relazionali, eliminando la necessità di strutture gerarchiche complesse.

## 2.3. Class Diagram Ristrutturato

Il Class Diagram è stato completamente ristrutturato secondo le analisi effettuate nei punti precedenti, risultando idoneo alla traduzione in schema relazionale e ottimizzato per l'implementazione nel DBMS PostgreSQL.

## 2.4. Dizionario delle Classi

Entità Principali:

|  |  |  |
| --- | --- | --- |
| **Classe** | **Descrizione** | **Attributi** |
| Utente | Rappresenta un utente del sistema | utenteId, username, password, email, nome, cognome, ruoloUtente, dataRegistrazione |
| Hackathon | Evento di programmazione | hackathonId, titolo, descrizione, sede, dataInizio, dataFine, dataChiusuraRegistrazioni, maxIscritti, maxDimensioneTeam, statoHackathon, dataCreazione |
| Team | Gruppo di partecipanti | teamId, nome, definitivo, dataCreazione |
| Registrazione | Partecipazione di un utente a un hackathon | registrazioneId, ruoloTeam, dataRegistrazione, dataIngressoTeam |
| GiudiceHackathon | Giudice assegnato a un hackathon | giudiceHackathonId, dataAssegnazione |
| Voto | Valutazione di un team | votoId, valore, dataVoto |
| Progresso | Documento caricato dal team | progressoId, documentoPath, documentoNome, dataCaricamento |
| Commento | Feedback su un progresso | commentoId, testo, dataCommento |
| Problema | Sfida proposta nell'hackathon | problemaId, titolo, descrizione, dataPubblicazione |
| InvitoTeam | Invito a partecipare a un team | invitoId, messaggioMotivazionale, statoInvito, dataInvito |
| InvitoGiudice | Invito a giudicare un hackathon | invitoId, statoInvito, dataInvito |

Enumerazioni:

|  |  |  |  |
| --- | --- | --- | --- |
| **Enumerazione** | **Descrizione** | **Attributi** | **Valori** |
| RuoloUtente | Ruolo dell'utente nel sistema | ruoloId, nomeRuolo | ORGANIZZATORE, GIUDICE, PARTECIPANTE |
| StatoHackathon | Stato dell'hackathon | statoId, nomeStato | REGISTRAZIONI\_APERTE, REGISTRAZIONI\_CHIUSE, IN\_CORSO, TERMINATO |
| RuoloTeam | Ruolo nel team | ruoloId, nomeRuolo | LEADER, MEMBRO |
| StatoInvito | Stato dell’invito | statoId, nomeStato | PENDING, ACCEPTED, DECLINED |

## 2.5. Dizionario delle Associazioni

|  |  |  |  |
| --- | --- | --- | --- |
| **Associazione** | **Entità Coinvolte** | **Cardinalità** | **Descrizione** |
| crea | Utente → Hackathon | 1:N | Un Utente può creare uno o più Hackathon |
| appartiene | Team → Hackathon | N:1 | Un Team deve appartenere a un solo Hackathon |
| fa parte | Registrazione → Team | N:1 | Una Registrazione può far parte di un solo Team |
| effettua | Utente → Registrazione | 1:N | Un Utente può effettuare una o più registrazioni |
| è | Utente → GiudiceHackathon | 1:N | Un Utente può essere uno o più giudice di Hackathon |
| appartiene | GiudiceHackathon → Hackathon | N:1 | Un Giudice Hackathon deve appartenere a un solo Hackathon |
| assegna | GiudiceHackathon → Voto | 1:N | Un Giudice Hackathon può assegnare uno o più Voti |
| ottiene | Team → Voto | 1:N | Team può ottenere uno o più Voti |
| carica | Registrazione → Progresso | 1:N | Una Registrazione può caricare uno o più Progressi |
| ottiene | Progresso → Commento | 1:N | Un progresso può ottenere uno o più Commenti |
| assegna | GiudiceHackathon → Commento | 1:N | Un Giudice Hackathon può assegnare uno o più Commenti |
| carica | GiudiceHackathon → Problema | 1:N | Un Giudice Hackathon può caricare uno o più Problemi |
| invia | Registrazione → InvitoTeam | 1:N | Una Registrazione può inviare uno o più Inviti Team |
| riceve | Utente → InvitoTeam | 1:N | Un Utente può ricevere uno o più Inviti Team |
| invia | Utente → InvitoGiudice | 1:N | Un Utente può invitare uno o più Inviti Giudice |
| riceve | Utente → InvitoGiudice | 1:N | Un Utente può ricevere uno o più Inviti Giudice |
| assegna | Hackathon → InvitoGiudice | 1:N | Un Hackathon può assegnare uno o più Invito Giudice |
| a | Registrazione → Hackathon | N:1 | Una Registrazione deve (essere assegnata) a un solo Hackathon |

## 2.6. Dizionario dei Vincoli

Vincoli di Dominio (CHECK):

|  |  |
| --- | --- |
| **Nome Vincolo** | **Descrizione** |
| chk\_email\_valida | L'email di un utente deve rispettare il formato standard con @ e dominio valido |
| chk\_username\_lunghezza | Lo username di un utente deve essere lungo almeno 3 caratteri |
| chk\_password\_lunghezza | La password di un utente deve essere lunga almeno 6 caratteri |
| chk\_ordine\_date | La data di fine di un hackathon deve essere posteriore alla data di inizio |
| chk\_deadline\_registrazione | La data di chiusura registrazioni deve essere antecedente o uguale alla data di inizio |
| chk\_ruolo\_con\_team | Il ruolo e l’appartenenza a un team devono essere entrambi null o entrambi not null |
| chk\_no\_auto\_invito | L’invitante utente deve essere diverso dall’invitato utente (in inviti\_giudice) |

Vincoli di Unicità (UNIQUE):

|  |  |
| --- | --- |
| **Nome Vincolo** | **Descrizione** |
| unique\_username | Lo username di ogni utente deve essere univoco nel sistema |
| unique\_email | L'email di ogni utente deve essere univoca nel sistema |
| uq\_nome\_team\_per\_hackathon | Il nome di un team deve essere univoco all'interno dello stesso hackathon |
| uq\_utente\_per\_hackathon | Un utente può registrarsi una sola volta per ogni hackathon |
| uq\_giudice\_per\_hackathon | Un giudice può essere assegnato una sola volta per ogni hackathon |
| uq\_voto\_per\_giudice\_team | Un giudice può votare ogni team una sola volta |
| uq\_invito\_team | Un invito team deve essere univoco per coppia invitante-invitato |
| uq\_invito\_giudice | Un invito giudice deve essere univoco per terna invitante-invitato-hackathon |

Vincoli Comportamentali (TRIGGER):

|  |  |  |  |
| --- | --- | --- | --- |
| **Trigger** | **Nome Trigger** | **Nome Funzione** | **Descrizione** |
| TRG01 | trg\_01\_verifica\_organizzatore | check\_organizzatore\_role() | Solo organizzatori possono creare hackathon |
| TRG02 | trg\_02\_verifica\_giudice | check\_giudice\_role() | Solo utenti con ruolo giudice possono essere assegnati |
| TRG03 | trg\_03\_unico\_leader | check\_single\_leader() | Un solo leader per team |
| TRG04 | trg\_04\_dimensione\_team | check\_team\_size() | Verifica dimensione massima team |
| TRG05 | trg\_05\_membro\_team\_progresso | check\_team\_member\_progress() | Solo membri con team possono caricare progressi |
| TRG06 | trg\_06\_deadline\_registrazione | check\_registration\_deadline() | No registrazioni dopo la deadline |
| TRG07 | trg\_07\_max\_partecipanti | check\_max\_participants() | Verifica limite iscritti |
| TRG08 | trg\_08\_team\_definitivo\_voto | check\_team\_definitivo() | Solo team definitivi possono essere votati |
| TRG09 | trg\_09\_stato\_hackathon\_voto | check\_hackathon\_status\_vote() | Voti solo durante o dopo hackathon |
| TRG10 | trg\_10\_problema\_in\_corso | check\_hackathon\_in\_corso() | Problemi solo durante hackathon |
| TRG11 | trg\_11\_leader\_invita | check\_leader\_invite() | Solo leader possono invitare al team |
| TRG12 | trg\_12\_auto\_join\_team | auto\_join\_team() | Auto-join team su accettazione invito |

# 3. Progettazione Logica

In questo capitolo tratteremo la seconda fase della progettazione, scendendo ad un livello di astrazione più basso rispetto al precedente.

Lo schema concettuale verrà tradotto, anche grazie alla predisposizione conseguente la ristrutturazione, in uno schema logico, questa volta dipendente dalla struttura dei dati prescelta, nello specifico quella relazionale pura.

## 3.1. Schema Logico

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

* **ruoli\_utente** *(ruolo\_id, name\_ruolo)*
* **stati\_hackathon** *(stato\_id, nome\_stato)*
* **stati\_invito** *(stato\_id, nome\_stato)*
* **ruoli\_team** *(ruolo\_id, name\_ruolo)*
* **utenti** *(utente\_id, username, email, password, nome, cognome, ruolo\_fk\_ruoli\_utente, data\_registrazione)*
* **hackathon** *(hackathon\_id, titolo, descrizione, sede, data\_inizio, data\_fine, data\_chiusura\_reg, max\_iscritti, max\_membri\_team, organizzatore\_fk\_utenti, stato\_fk\_stati\_hackathon, data\_creazione)*
* **teams** *(team\_id, nome, hackathon\_fk\_hackathons, data\_creazione, definitivo)*
* **registrazioni** *(registrazione\_id, partecipante\_fk\_utenti, hackathon\_fk\_hackathons, team\_fk\_teams,* *ruolo\_fk\_ruoli\_team, data\_registrazione, data\_ingresso\_team)*
* **giudici\_hackathon** *(giudice\_hackathon\_id, hackathon\_fk\_hackathons, giudice\_fk\_utenti, data\_assegnazione)*
* **problemi** *(problema\_id, giudice\_hack\_fk\_giudici\_hackathon, titolo, descrizione, data\_pubblicazione)*
* **progressi** *(progresso\_id, registrazione\_fk\_registrazioni, documento\_path, documento\_nome, data\_caricamento)*
* **commenti** *(commento\_id, progresso\_fk\_progressi, giudice\_hack\_fk\_giudici\_hackathon, testo, data\_commento)*
* **voti** *(voto\_id, team\_fk\_teams, giudice\_hack\_fk\_giudici\_hackathon, valore, data\_voto)*
* **inviti\_team** *(invito\_id, invitante\_reg\_fk\_registrazioni, invitato\_fk\_utenti, messaggio, stato\_fk\_stati\_invito, data\_invito)*
* **inviti\_giudice** *(invito\_id, invitante\_fk\_utenti, invitato\_fk\_utenti, hackathon\_fk\_hackathons, stato\_fk\_stati\_invito, data\_invito)*

# 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.

### 4.1.1. Definizione della Tabella RUOLI\_UTENTE

|  |
| --- |
| CREATE TABLE ruoli\_utente (  ruolo\_id INTEGER PRIMARY KEY,  nome\_ruolo VARCHAR(50) NOT NULL UNIQUE ); |

### 4.1.2. Definizione della Tabella STATI\_HACKATHON

|  |
| --- |
| CREATE TABLE stati\_hackathon (  stato\_id INTEGER PRIMARY KEY,  nome\_stato VARCHAR(50) NOT NULL UNIQUE ); |

### 4.1.3. Definizione della Tabella STATI\_INVITO

|  |
| --- |
| CREATE TABLE stati\_invito (  stato\_id INTEGER PRIMARY KEY,  nome\_stato VARCHAR(50) NOT NULL UNIQUE ); |

### 4.1.4. Definizione della Tabella RUOLI\_TEAM

|  |
| --- |
| CREATE TABLE ruoli\_team (  ruolo\_id INTEGER PRIMARY KEY,  nome\_ruolo VARCHAR(20) NOT NULL UNIQUE ); |

### 4.1.5. Definizione della Tabella UTENTI

|  |
| --- |
| CREATE TABLE utenti (  utente\_id SERIAL PRIMARY KEY,  username VARCHAR(50) NOT NULL UNIQUE,  email VARCHAR(100) NOT NULL UNIQUE,  password VARCHAR(255) NOT NULL,  nome VARCHAR(50) NOT NULL,  cognome VARCHAR(50) NOT NULL,  ruolo\_fk\_ruoli\_utente INTEGER NOT NULL REFERENCES ruoli\_utente (ruolo\_id),  data\_registrazione TIMESTAMP DEFAULT *CURRENT\_TIMESTAMP*,  CONSTRAINT chk\_email\_valida CHECK (email ~\* '^[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$' ) ,  CONSTRAINT chk\_username\_lunghezza  CHECK (LENGTH(username) >= 3),  CONSTRAINT chk\_password\_lunghezza  CHECK (LENGTH(password) >= 6) ); |

### 4.1.6. Definizione della Tabella HACKATHONS

|  |
| --- |
| CREATE TABLE hackathons (  hackathon\_id SERIAL PRIMARY KEY,  titolo VARCHAR(200) NOT NULL,  descrizione TEXT,  sede VARCHAR(200) NOT NULL,  data\_inizio TIMESTAMP NOT NULL,  data\_fine TIMESTAMP NOT NULL,  data\_chiusura\_reg TIMESTAMP NOT NULL,  max\_iscritti INTEGER NOT NULL CHECK (max\_iscritti > 0),  max\_membri\_team INTEGER NOT NULL CHECK (max\_membri\_team >= 2),  organizzatore\_fk\_utenti INTEGER NOT NULL REFERENCES utenti (utente\_id),  stato\_fk\_stati\_hackathon INTEGER NOT NULL REFERENCES stati\_hackathon (stato\_id) DEFAULT 1,  data\_creazione TIMESTAMP DEFAULT *CURRENT\_TIMESTAMP*,  CONSTRAINT chk\_ordine\_date CHECK (data\_inizio < data\_fine),  CONSTRAINT chk\_deadline\_registrazione CHECK (data\_chiusura\_reg <= data\_inizio) ); |

### 4.1.7. Definizione della Tabella TEAMS

|  |
| --- |
| CREATE TABLE teams (  team\_id SERIAL PRIMARY KEY,  nome VARCHAR(100) NOT NULL,  hackathon\_fk\_hackathons INTEGER NOT NULL REFERENCES hackathons (hackathon\_id) ON DELETE CASCADE,  definitivo BOOLEAN DEFAULT FALSE,  data\_creazione TIMESTAMP DEFAULT *CURRENT\_TIMESTAMP*,  CONSTRAINT uq\_nome\_team\_per\_hackathon UNIQUE (nome, hackathon\_fk\_hackathons) ); |

### 4.1.8. Definizione della Tabella REGISTRAZIONI

|  |
| --- |
| CREATE TABLE registrazioni (  registrazione\_id SERIAL PRIMARY KEY,  partecipante\_fk\_utenti INTEGER NOT NULL REFERENCES utenti (utente\_id),  hackathon\_fk\_hackathons INTEGER NOT NULL REFERENCES hackathons (hackathon\_id) ON DELETE CASCADE,  team\_fk\_teams INTEGER REFERENCES teams (team\_id) ON DELETE SET NULL,  ruolo\_fk\_ruoli\_team INTEGER REFERENCES ruoli\_team (ruolo\_id),  data\_registrazione TIMESTAMP DEFAULT *CURRENT\_TIMESTAMP*,  data\_ingresso\_team TIMESTAMP,  CONSTRAINT uq\_utente\_per\_hackathon UNIQUE (partecipante\_fk\_utenti, hackathon\_fk\_hackathons),  CONSTRAINT chk\_ruolo\_con\_team CHECK ((team\_fk\_teams IS NULL AND ruolo\_fk\_ruoli\_team IS NULL) OR  (team\_fk\_teams IS NOT NULL AND ruolo\_fk\_ruoli\_team IS NOT NULL)) ); |

### 4.1.9. Definizione della Tabella GIUDICI\_HACKATHON

|  |
| --- |
| CREATE TABLE giudici\_hackathon (  giudice\_hackathon\_id SERIAL PRIMARY KEY,  hackathon\_fk\_hackathons INTEGER NOT NULL REFERENCES hackathons (hackathon\_id) ON DELETE CASCADE,  giudice\_fk\_utenti INTEGER NOT NULL REFERENCES utenti (utente\_id),  data\_assegnazione TIMESTAMP DEFAULT *CURRENT\_TIMESTAMP*,  CONSTRAINT uq\_giudice\_per\_hackathon UNIQUE (hackathon\_fk\_hackathons, giudice\_fk\_utenti) ); |

### 4.1.10. Definizione della Tabella PROBLEMI

|  |
| --- |
| CREATE TABLE problemi (  problema\_id SERIAL PRIMARY KEY,  giudice\_hack\_fk\_giudici\_hackathon INTEGER NOT NULL REFERENCES giudici\_hackathon (giudice\_hackathon\_id) ON DELETE CASCADE,  titolo VARCHAR(200) NOT NULL,  descrizione TEXT NOT NULL,  data\_pubblicazione TIMESTAMP DEFAULT *CURRENT\_TIMESTAMP* ); |

### 4.1.11. Definizione della Tabella PROGRESSI

|  |
| --- |
| CREATE TABLE progressi (  progresso\_id SERIAL PRIMARY KEY,  registrazione\_fk\_registrazioni INTEGER NOT NULL REFERENCES registrazioni (registrazione\_id) ON DELETE CASCADE,  documento\_path VARCHAR(500),  documento\_nome VARCHAR(200),  data\_caricamento TIMESTAMP DEFAULT *CURRENT\_TIMESTAMP* ); |

### 4.1.12. Definizione della Tabella COMMENTI

|  |
| --- |
| CREATE TABLE commenti (  commento\_id SERIAL PRIMARY KEY,  progresso\_fk\_progressi INTEGER NOT NULL REFERENCES progressi (progresso\_id) ON DELETE CASCADE,  giudice\_hack\_fk\_giudici\_hackathon INTEGER NOT NULL REFERENCES giudici\_hackathon (giudice\_hackathon\_id),  testo TEXT NOT NULL,  data\_commento TIMESTAMP DEFAULT *CURRENT\_TIMESTAMP* ); |

### 4.1.13. Definizione della Tabella VOTI

|  |
| --- |
| CREATE TABLE voti (  voto\_id SERIAL PRIMARY KEY,  team\_fk\_teams INTEGER NOT NULL REFERENCES teams (team\_id) ON DELETE CASCADE,  giudice\_hack\_fk\_giudici\_hackathon INTEGER NOT NULL REFERENCES giudici\_hackathon (giudice\_hackathon\_id),  valore INTEGER NOT NULL CHECK (valore >= 1 AND valore <= 10),  data\_voto TIMESTAMP DEFAULT *CURRENT\_TIMESTAMP*,  CONSTRAINT uq\_voto\_per\_giudice\_team UNIQUE (team\_fk\_teams, giudice\_hack\_fk\_giudici\_hackathon) ); |

### 4.1.14. Definizione della Tabella INVITI\_TEAM

|  |
| --- |
| CREATE TABLE inviti\_team (  invito\_id SERIAL PRIMARY KEY,  invitante\_reg\_fk\_registrazioni INTEGER NOT NULL REFERENCES registrazioni (registrazione\_id) ON DELETE CASCADE,  invitato\_fk\_utenti INTEGER NOT NULL REFERENCES utenti (utente\_id) ON DELETE CASCADE,  messaggio TEXT,  stato\_fk\_stati\_invito INTEGER DEFAULT 1 REFERENCES stati\_invito (stato\_id),  data\_invito TIMESTAMP DEFAULT *CURRENT\_TIMESTAMP*,  CONSTRAINT uq\_invito\_team UNIQUE (invitante\_reg\_fk\_registrazioni, invitato\_fk\_utenti) ); |

### 4.1.15. Definizione della Tabella INVITI\_GIUDICE

|  |
| --- |
| CREATE TABLE inviti\_giudice (  invito\_id SERIAL PRIMARY KEY,  invitante\_fk\_utenti INTEGER NOT NULL REFERENCES utenti (utente\_id) ON DELETE CASCADE,  invitato\_fk\_utenti INTEGER NOT NULL REFERENCES utenti (utente\_id) ON DELETE CASCADE,  hackathon\_fk\_hackathons INTEGER NOT NULL REFERENCES hackathons (hackathon\_id) ON DELETE CASCADE,  stato\_fk\_stati\_invito INTEGER DEFAULT 1 REFERENCES stati\_invito (stato\_id),  data\_invito TIMESTAMP DEFAULT *CURRENT\_TIMESTAMP*,  CONSTRAINT uq\_invito\_giudice UNIQUE (invitante\_fk\_utenti, invitato\_fk\_utenti, hackathon\_fk\_hackathons),  CONSTRAINT chk\_no\_auto\_invito CHECK (invitante\_fk\_utenti != invitato\_fk\_utenti ) ); |

## 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 TRG01

|  |
| --- |
| CREATE OR REPLACE FUNCTION check\_organizzatore\_role() RETURNS TRIGGER AS $$ BEGIN  IF NOT EXISTS (  SELECT 1 FROM utenti  WHERE utente\_id = NEW.organizzatore\_fk\_utenti  AND ruolo\_fk\_ruoli\_utente = 1  ) THEN  RAISE EXCEPTION 'Solo gli organizzatori possono creare hackathon'; END IF; RETURN NEW; END; $$ LANGUAGE plpgsql;  CREATE TRIGGER trg\_01\_verifica\_organizzatore  BEFORE INSERT OR UPDATE ON hackathons FOR EACH ROW EXECUTE FUNCTION check\_organizzatore\_role(); |

### 4.2.2. Implementazione del Vincolo TRG02

|  |
| --- |
| CREATE OR REPLACE FUNCTION check\_giudice\_role() RETURNS TRIGGER AS $$ BEGIN  IF NOT EXISTS (  SELECT 1 FROM utenti  WHERE utente\_id = NEW.giudice\_fk\_utenti  AND ruolo\_fk\_ruoli\_utente = 2  ) THEN  RAISE EXCEPTION 'Solo utenti con ruolo GIUDICE possono essere assegnati'; END IF; RETURN NEW; END; $$ LANGUAGE plpgsql;  CREATE TRIGGER trg\_02\_verifica\_giudice  BEFORE INSERT OR UPDATE ON giudici\_hackathon FOR EACH ROW EXECUTE FUNCTION check\_giudice\_role(); |

### 4.2.3. Implementazione del Vincolo TRG03

|  |
| --- |
| CREATE OR REPLACE FUNCTION check\_single\_leader() RETURNS TRIGGER AS $$ BEGIN  IF NEW.ruolo\_fk\_ruoli\_team = 1 AND EXISTS (  SELECT 1 FROM registrazioni  WHERE team\_fk\_teams = NEW.team\_fk\_teams  AND ruolo\_fk\_ruoli\_team = 1  AND registrazione\_id != COALESCE(NEW.registrazione\_id, -1)  ) THEN  RAISE EXCEPTION 'Un team può avere un solo leader'; END IF; RETURN NEW; END; $$ LANGUAGE plpgsql;  CREATE TRIGGER trg\_03\_unico\_leader  BEFORE INSERT OR UPDATE ON registrazioni FOR EACH ROW EXECUTE FUNCTION check\_single\_leader(); |

### 4.2.4. Implementazione del Vincolo TRG04

|  |
| --- |
| CREATE OR REPLACE FUNCTION check\_team\_size() RETURNS TRIGGER AS $$ DECLARE v\_membri\_attuali INTEGER;  v\_max\_membri INTEGER; BEGIN  IF NEW.team\_fk\_teams IS NOT NULL AND  (TG\_OP = 'INSERT' OR NEW.team\_fk\_teams IS DISTINCT FROM OLD.team\_fk\_teams) THEN SELECT *COUNT*(\*), h.max\_membri\_team INTO v\_membri\_attuali, v\_max\_membri FROM registrazioni r  JOIN teams t ON t.team\_id = r.team\_fk\_teams  JOIN hackathons h ON h.hackathon\_id = t.hackathon\_fk\_hackathons WHERE r.team\_fk\_teams = NEW.team\_fk\_teams GROUP BY h.max\_membri\_team; IF v\_membri\_attuali >= v\_max\_membri THEN  RAISE EXCEPTION 'Team ha raggiunto la dimensione massima di % membri', v\_max\_membri; END IF; END IF; RETURN NEW; END; $$ LANGUAGE plpgsql;  CREATE TRIGGER trg\_04\_dimensione\_team  BEFORE INSERT OR UPDATE ON registrazioni FOR EACH ROW EXECUTE FUNCTION check\_team\_size(); |

### 4.2.5. Implementazione del Vincolo TRG05

|  |
| --- |
| CREATE OR REPLACE FUNCTION check\_team\_member\_progress() RETURNS TRIGGER AS $$ BEGIN  IF NOT EXISTS (  SELECT 1 FROM registrazioni  WHERE registrazione\_id = NEW.registrazione\_fk\_registrazioni  AND team\_fk\_teams IS NOT NULL  ) THEN  RAISE EXCEPTION 'Solo membri di un team possono caricare progressi'; END IF; RETURN NEW; END; $$ LANGUAGE plpgsql;  CREATE TRIGGER trg\_05\_membro\_team\_progresso  BEFORE INSERT  ON progressi  FOR EACH ROW EXECUTE FUNCTION check\_team\_member\_progress(); |

### 4.2.6. Implementazione del Vincolo TRG06

|  |
| --- |
| CREATE OR REPLACE FUNCTION check\_registration\_deadline() RETURNS TRIGGER AS $$ BEGIN  IF EXISTS (  SELECT 1 FROM hackathons  WHERE hackathon\_id = NEW.hackathon\_fk\_hackathons  AND data\_chiusura\_reg < CURRENT\_TIMESTAMP  ) THEN  RAISE EXCEPTION 'Registrazioni chiuse per questo hackathon'; END IF; RETURN NEW; END; $$ LANGUAGE plpgsql;  CREATE TRIGGER trg\_06\_deadline\_registrazione  BEFORE INSERT  ON registrazioni  FOR EACH ROW EXECUTE FUNCTION check\_registration\_deadline(); |

### 4.2.7. Implementazione del Vincolo TRG07

|  |
| --- |
| CREATE OR REPLACE FUNCTION check\_max\_participants() RETURNS TRIGGER AS $$ DECLARE v\_iscritti\_attuali INTEGER;  v\_max\_iscritti INTEGER; BEGIN SELECT *COUNT*(\*), h.max\_iscritti INTO v\_iscritti\_attuali, v\_max\_iscritti FROM registrazioni r  JOIN hackathons h ON h.hackathon\_id = r.hackathon\_fk\_hackathons WHERE r.hackathon\_fk\_hackathons = NEW.hackathon\_fk\_hackathons GROUP BY h.max\_iscritti; IF v\_iscritti\_attuali >= v\_max\_iscritti THEN  RAISE EXCEPTION 'Raggiunto numero massimo iscritti'; END IF; RETURN NEW; END; $$ LANGUAGE plpgsql;  CREATE TRIGGER trg\_07\_max\_partecipanti  BEFORE INSERT  ON registrazioni  FOR EACH ROW EXECUTE FUNCTION check\_max\_participants(); |

### 4.2.8. Implementazione del Vincolo TRG08

|  |
| --- |
| CREATE OR REPLACE FUNCTION check\_team\_definitivo() RETURNS TRIGGER AS $$ BEGIN  IF NOT EXISTS (  SELECT 1 FROM teams  WHERE team\_id = NEW.team\_fk\_teams  AND definitivo = TRUE  ) THEN  RAISE EXCEPTION 'Si possono votare solo team definitivi'; END IF; RETURN NEW; END; $$ LANGUAGE plpgsql;  CREATE TRIGGER trg\_08\_team\_definitivo\_voto  BEFORE INSERT  ON voti  FOR EACH ROW EXECUTE FUNCTION check\_team\_definitivo(); |

### 4.2.9. Implementazione del Vincolo TRG09

|  |
| --- |
| CREATE OR REPLACE FUNCTION check\_hackathon\_status\_vote() RETURNS TRIGGER AS $$ BEGIN  IF NOT EXISTS (  SELECT 1 FROM teams t  JOIN hackathons h ON h.hackathon\_id = t.hackathon\_fk\_hackathons  WHERE t.team\_id = NEW.team\_fk\_teams  AND h.stato\_fk\_stati\_hackathon IN (3, 4)  ) THEN  RAISE EXCEPTION 'Voti permessi solo durante o dopo hackathon'; END IF; RETURN NEW; END; $$ LANGUAGE plpgsql;  CREATE TRIGGER trg\_09\_stato\_hackathon\_voto  BEFORE INSERT  ON voti  FOR EACH ROW EXECUTE FUNCTION check\_hackathon\_status\_vote(); |

### 4.2.10. Implementazione del Vincolo TRG10

|  |
| --- |
| CREATE OR REPLACE FUNCTION check\_hackathon\_in\_corso() RETURNS TRIGGER AS $$ DECLARE v\_hackathon\_id INTEGER; BEGIN SELECT hackathon\_fk\_hackathons INTO v\_hackathon\_id FROM giudici\_hackathon WHERE giudice\_hackathon\_id = *COALESCE*(NEW.giudice\_hack\_fk\_giudici\_hackathon, OLD.giudice\_hack\_fk\_giudici\_hackathon); IF NOT EXISTS (  SELECT 1 FROM hackathons  WHERE hackathon\_id = v\_hackathon\_id  AND stato\_fk\_stati\_hackathon = 3  ) THEN  RAISE EXCEPTION 'Operazione permessa solo durante hackathon in corso'; END IF; RETURN *COALESCE*(NEW, OLD); END; $$ LANGUAGE plpgsql;  CREATE TRIGGER trg\_10\_problema\_in\_corso  BEFORE INSERT OR UPDATE OR DELETE ON problemi FOR EACH ROW EXECUTE FUNCTION check\_hackathon\_in\_corso(); |

### 4.2.11. Implementazione del Vincolo TRG11

|  |
| --- |
| REATE OR REPLACE FUNCTION check\_leader\_invite() RETURNS TRIGGER AS $$ BEGIN  IF NOT EXISTS (  SELECT 1 FROM registrazioni  WHERE registrazione\_id = NEW.invitante\_reg\_fk\_registrazioni  AND ruolo\_fk\_ruoli\_team = 1  ) THEN  RAISE EXCEPTION 'Solo il leader può invitare membri'; END IF; RETURN NEW; END; $$ LANGUAGE plpgsql;  CREATE TRIGGER trg\_11\_leader\_invita  BEFORE INSERT  ON inviti\_team  FOR EACH ROW EXECUTE FUNCTION check\_leader\_invite(); |

### 4.2.12. Implementazione del Vincolo TRG12

|  |
| --- |
| CREATE OR REPLACE FUNCTION auto\_join\_team() RETURNS TRIGGER AS $$ DECLARE v\_team\_id INTEGER;  v\_hackathon\_id INTEGER; BEGIN  IF NEW.stato\_fk\_stati\_invito = 2 AND OLD.stato\_fk\_stati\_invito = 1 THEN SELECT r.team\_fk\_teams, r.hackathon\_fk\_hackathons INTO v\_team\_id, v\_hackathon\_id FROM registrazioni r WHERE r.registrazione\_id = NEW.invitante\_reg\_fk\_registrazioni; UPDATE registrazioni SET team\_fk\_teams = v\_team\_id,  ruolo\_fk\_ruoli\_team = 2,  data\_ingresso\_team = *CURRENT\_TIMESTAMP* WHERE partecipante\_fk\_utenti = NEW.invitato\_fk\_utenti  AND hackathon\_fk\_hackathons = v\_hackathon\_id; END IF; RETURN NEW; END; $$ LANGUAGE plpgsql;  CREATE TRIGGER trg\_12\_auto\_join\_team  AFTER UPDATE  ON inviti\_team  FOR EACH ROW EXECUTE FUNCTION auto\_join\_team(); |

## 4.3. Definizione Viste

Di seguito sono riportate le definizioni delle viste.

### 4.3.1. Definizione della Vista V\_CLASSIFICA\_HACKATHONS

|  |
| --- |
| CREATE VIEW v\_classifica\_hackathons AS SELECT h.hackathon\_id,  h.titolo AS hackathon,  t.team\_id,  t.nome AS team,  *AVG*(v.valore) AS media\_voti,  *COUNT*(DISTINCT v.voto\_id) AS num\_voti,  *COUNT*(DISTINCT r.partecipante\_fk\_utenti) AS num\_membri FROM hackathons h  JOIN teams t ON t.hackathon\_fk\_hackathons = h.hackathon\_id  LEFT JOIN voti v ON v.team\_fk\_teams = t.team\_id  LEFT JOIN registrazioni r ON r.team\_fk\_teams = t.team\_id WHERE t.definitivo = TRUE GROUP BY h.hackathon\_id, h.titolo, t.team\_id, t.nome ORDER BY h.hackathon\_id, *AVG*(v.valore) DESC NULLS LAST; |