Università degli Studi di Napoli Federico II

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

Elaborato di gruppo (Traccia 2)

25/07/2025

Petraccone Simone - N86005387
Picari Alessia - N86005131

Indice

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 degli studenti Petraccone Simone e Picari Alessia 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 di Hackathon.

1.1 Descrizione del problema

Verranno riportate progettazione e sviluppo di una base di dati relazionale, che implementi un sistema di gestione di gare (dette Hackathon) che permetta l'organizzazione, la partecipazione e la valutazione di ques'ultime. Il sistema tiene traccia degli utenti della piattaforma, che possono essere di tre tipologie: Concorrenti, Giudici, Organizzatori. In base al tipo di utente, la piattaforma permette di:
(per gli organizzatori) creare Hackathon, convocare giudici,
(per i giudici) dare/ricevere valutazioni/commenti,
(per i concorrenti) formare team e caricare documenti di progetto.

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

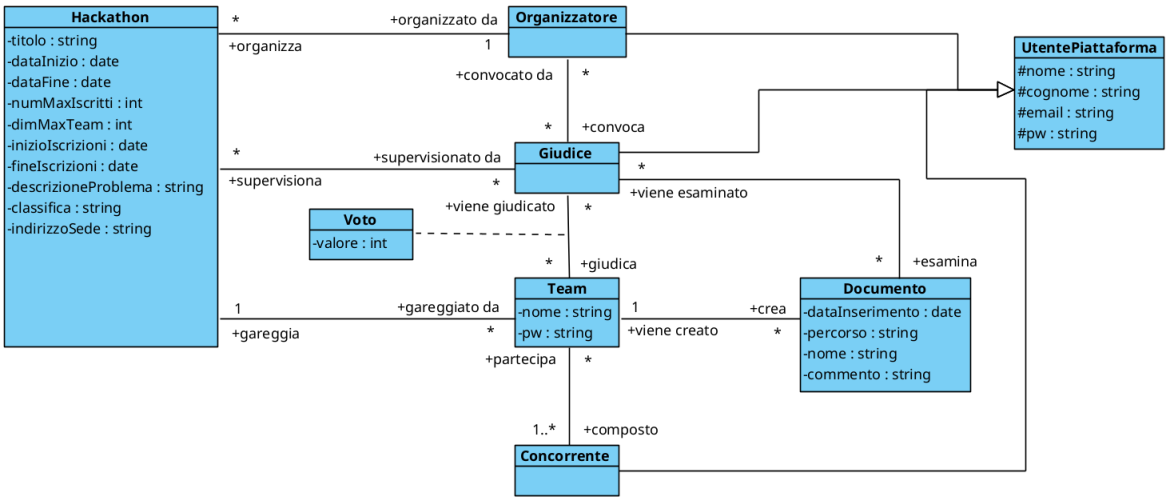

Figura 1: Diagramma UML iniziale

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

Ai fini dell'efficienza nella rappresentazione delle varie entità, nello specifico **Documento**, risulta conveniente l'introduzione di chiavi primarie surrogate, ovvero rappresentate da identificativi interi associati a ciascuna istanza.

2.2.2 Analisi degli attributi derivati

Per ottimizzare ulteriormente l'utilizzo delle risorse di calcolo, analizziamo gli eventuali attributi derivati, ovvero calcolabili da altri attributi delle entità. I più evidenti risultano essere in questo caso il numero di voti assegnati e il numero di iscritti in un Hackathon. Essi sono infatti calcolabili a partire dalle relazioni dell'Hackathon stesso: il numero di voti assegnati si può ricavare dal numero di voti assegnati dai giudici ai team iscritti all'Hackathon; mentre il numero di iscritti si può ottenere a partire dal numero di partecipazioni dei concorrenti ai team iscritti all'Hackathon. Ad un'attenta analisi risulta essere conveniente la storicizzazione di tali valori come attributi dell'Hackathon stesso, ai fini di una più efficiente valutazione dello stato di una determinata gara.

2.2.3 Analisi delle ridondanze

Analizziamo ora l'eventuale presenza di associazioni ridondanti tra le varie entità, in maniera tale da evitare incoerenze nella rappresentazione logica dei dati. L'unico caso riscontrabile è dato dall'associazione tra Team e Concorrente: ogni team può avere infatti un qualunque numero (entro un determinatro valore specificato nell'Hackathon) di partecipanti, ma richiede sempre la presenza di uno e un solo Creatore. Quest'ultimo è però a sua volta un partecipante al team, motivo per cui risulta logicamente più corretto gestire l'associazione relativa al creatore, a questo punto evidentemente ridondante, con una serie di vincoli interrelazionali, nello specifico uno di unicità ed uno di totalità del creatore.

2.2.4 Analisi degli attributi strutturati

Vanno ora analizzati e concettualmente corretti eventuali attributi strutturati presenti nelle entità. Questi infatti non sono logicamente rappresentabili all'interno di un DBMS, e vanno quindi eliminati e codificati in altro modo. Fortunatamente, nella rappresentazione concettuale presentata non sono presenti attributi strutturati.

2.2.5 Analisi degli attributi a valore multiplo

Verifichiamo ora la presenza di eventuali attributi a valore multiplo, anch'essi non logicamente rappresentabili e quindi da eliminare nello schema concettuale ristrutturato. Un attributo a valore multiplo è il commento di un documento: un singolo documento può infatti ricevere più di un commento. Per correggere tale problema logico, rappresentiamo l'attributo tramite una nuova classe di associazione: Commento, così da poterlo assegnare come attributo dell'associazione Giudice - Documento.

2.2.6 Analisi delle gerarchie di specializzazione

Infine andiamo a ristrutturare eventuali gerarchie di specializzazione, altro elemento non rappresentabile in un DBMS relazionale. L'unica gerarchia di specializzazione nel Class Diagram presentato è quella riguardante gli Utenti piattaforma: il generico utente si specializza infatti in concorrente oppure in giudice oppure in organizzatore. Tale specializzazione è Totale (ogni Utente deve essere specializzato) e Disgiunta (un Utente non può essere sia concorrente che giudice che organizzatore)[1], dunque risulta conveniente ai fini della rappresentazione logica ricontestualizzare tale specializzazione tramite un "appiattimento": la generalizzazione UtentePiattaforma viene eliminata e cede i suoi attributi e le sue associazioni alle singole specializzazioni. Il risultato saranno quindi tre entità, Concorrente, Giudice e Organizzatore indipendenti l'una dall'altra.

[1]Nota: una persona può essere più tipi di utente (es: un organizzatore che è anche giudice), a patto che crei due utenze diverse, ovvero si identifichi due o tre volte ma con email differente (es: Mario Rossi può essere sia mariorossi@organizzatore.com che mariorossi@giudice.com).

2.3 Class Diagram Ristrutturato

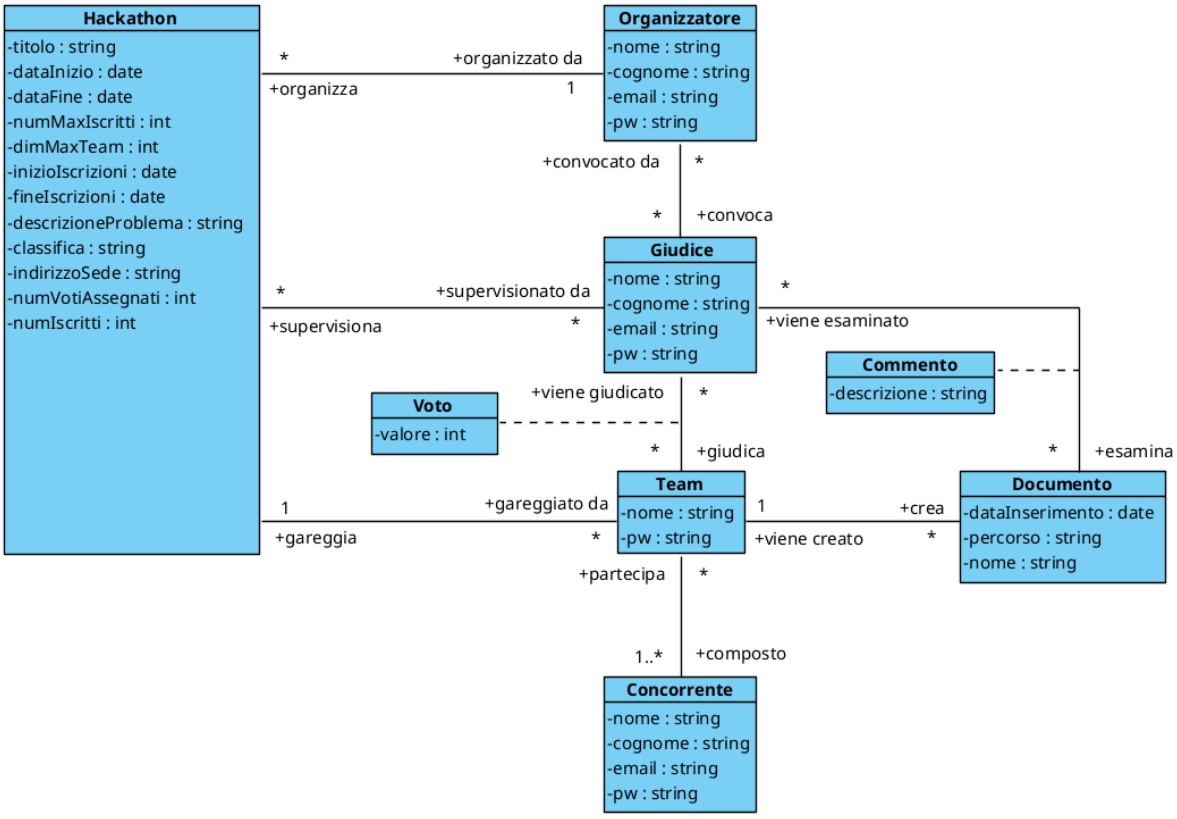

Figura 2: Diagramma UML ristrutturato

2.4 Dizionario delle Classi

Classe	Descrizione	Attributi
Concorrente	Utente che gareggia negli Hackathon tramite la partecipazione a un team	• email (string) (PK) • nome (string) • cognome (string) • pw (string)
Giudice	Utente addetto a giudicare i team partecipanti a determinati Hackathon	• email (string) (PK) • nome (string) • cognome (string) • pw (string)
Organizzatore	Utente addetto a creare Hackathon e assegnarvi giudici	• email (string) (PK) • nome (string) • cognome (string) • pw (string)
Team	Squadra gareggiante in un determinato Hackathon a cui sono iscrtitti almeno un concorrente (ossia il creatore)	• nome (string) (PK) • pw (string)
Documento	File caricati[2]dai team e valutati dai giudici	• idDocumento (int) (PK) • dataInserimento (date) • percorso (string) • nome (string)

[2]Nota: per semplicità, all'interno della base di dati verrà salvato solo il percorso del file.

Classe	Descrizione	Attributi
Hackathon	Gara a cui possono iscriversi i team	• titolo (string) (PK) • dataInizio (date) • dataFine (date) • numMaxIscitti (int) • dimMaxTeam (int) • inizioIscrizioni (int) • fineIscrizioni (int) • descrizioneProblema (string) • classifica (string) • indirizzoSede (string) • numVotiAssegnati (int) • numIscritti (int)

Tabella 1: Schema delle classi del sistema

2.5 Dizionario delle Associazioni

Nome	Descrizione	Classi coinvolte
Organizza	Esprime l'organizzazione (ovvero la creazione) di più Hackathon da parte di un singolo organizzatore.	**Organizzatore** [*] ruolo organizza: indica l'organizzatore di un determinato Hackathon. **Hackathon** [1] ruolo organizzato da: indica l'Hackathon creato da un determinato organizzatore.
Convocazione	Esprime la convocazione di più giudici da parte di più organizzatori	**Organizzatore** [*] ruolo convoca: indica l'organizzatore che ha convocato un determinato giudice (per un specifico Hackathon). **Giudice** [*] ruolo convocato da: indica il giudice convocato da un determinato organizzatore.
Superivsione	Esprime la supervisione di più Hackathon da parte di più giudici	**Giudice** [*] ruolo supervisiona: indica il giudice assegnato ad un determinato Hackathon. **Hackathon** [*] ruolo supervisionato da: indica l'Hackathon supervisionato da un determinato giudice.
Voto	Esprime il giudizio di più team da parte di più giudici	**Giudice** [*] ruolo giudica: indica il giudice addetto alla valutazione di un determinato team. **Team** [*] ruolo giudicato da: indica il team valutato da un determinato giudice. **Voto** classe associativa: esprime la valutazione, tramite un numero intero tra 0 e 10 (estremi compresi).
Commento	Esprime il commento di più documenti da parte di più giudici	**Giudice** [*] ruolo esamina: indica il giudice addetto all'esaminazione di un determinato team. **Documento** [*] ruolo viene esaminato: indica il documento esaminato da un determinato giudice. **Commento** classe associativa: esprime il commento, tramite una stringa.

Nome	Descrizione	Classi coinvolte
Gareggia	Esprime la partecipazione da parte di più team ad un singolo Hackathon	**Team** [1] ruolo gareggia: indica il team gareggiante in un determinato Hackathon. **Hackathon** [*] ruolo gareggiato da: indica l'Hackathon in cui gareggia un determinato team.
Pubblica	Esprime la pubblicazione (creazione) da parte di un team di più documenti	**Team** [1] ruolo crea: indica il team creatore di un determinato documento. **Documento** [1] ruolo viene creato: indica il documento creato da un determinato team.
Partecipazione	Esprime la partecipazione da parte di almeno un concorrente a più team	**Concorrente** [*] ruolo partecipa: indica il concorrente partecipante ad un determinato team. **Team** [1..*] ruolo composto: indica il team composto da almeno un concorrente.

Tabella 2: Schema delle associazioni del sistema

2.6 Dizionario dei Vincoli

Nome	Descrizione
PK_emailUtente	Le email degli utenti (concorrenti, giudici e organizzatori) devono essere univoche.
PK_nomeTeam	I nomi dei team devono essere univoci in ciascun Hackathon (possono esistere due team con lo stesso nome ma partecipanti a due Hackathon diversi).
FK_titoloHackathon	Ogni team viene identificato anche dall'Hackathon a cui è iscritto (per poter distinguere due team con stesso nome in Hackathon diversi).
PK_titoloHackathon	I titoli degli Hackathon devono essere univoci.
FK_emaiOrganizzatore	Ogni Hackathon ha un organizzatore associato, ovvero il creatore stesso.
PK_idDocumento	I documenti vengono identificati mediante un valore numerico univoco (detto ID).
FK_nomeTeam	Ogni documento è associato al team che lo ha caricato in piattaforma.
FK_titoloHackathon	Ogni documento è associato ad un Hackathon, poiché il team è identificato sia dal nome che dal titolo dell'Hackathon a cui partecipa.
FK_Convocazione	Ogni convocazione avviene a partire da un organizzatore (emailOrganizzatore), su di un giudice (emailGiudice) per un Hackathon (titoloHackathon).
FK_Voto	Ogni voto viene assegnato da un giudice (emailGiudice), ad un team (nomeTeam) iscritto ad un Hackathon (titoloHackathon).
FK_Commento	Ogni commento viene assegnato da un giudice (emailGiudice), ad un documento (idDocumento).
FK_Concorrente_Team	Ogni concorrente (emailConcorrente) può partecipare a più team (nomeTeam) in Hackathon diversi (titoloHackathon).

Nome	Descrizione
Validità Voto	Il valore numerico del voto deve essere compreso tra 0 e 10 (estremi compresi).
Validità Numero massimo iscritti all'Hackathon	Il numero massimo di iscritti in un Hackathon deve essere almeno maggiore di 1 (in un Hackathon ci devono poter essere almeno due concorrenti).
Validità Numero massimo partecipanti ai team dell'Hackathon	Il numero massimo di partecipanti nei team di un Hackathon deve essere almeno maggiore di 1 (nei team iscritti in un Hackathon, vi devono poter partecipare almeno due concorrenti).
Validità Data inizio iscrizioni	Le iscrizioni devono iniziare almeno un giorno dopo la creazione dell'Hackathon.
Coerenza Data fine iscrizioni	La iscrizioni devono chiudiresi o il giorno stesso in cui si aprono (le iscrizioni durano un giorno solo) oppure a partire dal giorno successivo (le iscirzioni durano più giorni).
Coerenza Data inizio gara	L'inizio della gara deve avvenire almeno due giorni dopo la fine delle iscrizioni (tempo necessario ai giuidici per inserire la descrizione del problema dell'Hackathon).
Coerenza Data fine gara	La fine della gara deve avvenire o il giorno stesso in cui si inizia la gara (la gara dura un giorno solo) oppure a partire dal giorno successivo (la gara dura più giorni).
Validità Email	Le email degli utenti devono rispettare un formato ben preciso: almeno una lettera minuscola (iniziale del nome e ulteriori lettere in caso di email già esistente), seguita/e da almeno una lettera minuscola (iniziale del cognome e ulteriori lettere in caso di email già esistente), seguita/e da zero o più cifre numeriche (nel caso in cui esista già una email con stesso nome e cognome, dunque si aggiunge un contatore numerico), il tutto seguito da @concorrente.com (@giudice.com per i giudici e @organizzatore.com per gli organizzatori).

Tabella 3: Schema dei vincoli del sistema

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.

- Organizzatore (email, nome, cognome, pw)
- Giudice (email, nome, cognome, pw)
- Concorrente (email, nome, cognome, pw)
- Hackathon (titolo, dataInizio, dataFine, numMaxIscritti, dimMaxTeam, inizioIscrizioni, fineIscrizioni, descrizioneProblema, classifica, indirizzoSede, numVotiAssegnati, numIscritti, emailOrganizzatore)
  emailOrganizzatore → Organizzatore.email
- Team (nome, titoloHackathon, pw, creatore)
  titoloHackathon → Hackathon.titolo ; creatore → Concorrente.email
- Documento (idDocumento, dataInserimento, percorso, nome, nomeTeam, titoloHackathon)
  nomeTeam → Team.nome ; titoloHackathon → Team.titoloHackathon
- Convocazione (emailOrganizzatore, emailGiudice, titoloHackathon)
  emailOrganizzatore → Organizzatore.email ; emailgiudice → Giudice.email ; titoloHackathon → Hackathon.titolo
- Commento (emailGiudice, idDocumento), descrizione
  emailGiudice → Giudice.email ; idDocumento → Documento.idDocumento
- Voto (emailGiudice, nomeTeam, titoloHackathon, valore)
  emailGiudice → Giudice.email ; nomeTeam → Team.nome ; titoloHackathon → Team.titoloHackathon
- Concorrente_Team (emailConcorrente, nomeTeam, titoloHackathon)
  emailConcorrente → Concorrente.email, nomeTeam → Team.nome ; titoloHackathon → Team.titoloHackathon

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 ORGANIZZATORE

```
--Definizione tabella
CREATE TABLE ORGANIZZATORE
(
    email VARCHAR(150) NOT NULL,
    nome VARCHAR(50) NOT NULL,
    cognome VARCHAR(50) NOT NULL,
    pw VARCHAR(255) NOT NULL,
    CONSTRAINT organizzatore_pkey PRIMARY KEY (email),
    CONSTRAINT chk_email CHECK (
     email ~ '^[a-z]{1,}[a-z]{1,}[0-9]*@organizzatore\.com$')
);
```

4.1.2 Definizione della Tabella GIUDICE

```
--Definizione tabella
CREATE TABLE GIUDICE
(
    email VARCHAR(150) NOT NULL,
    nome VARCHAR(50) NOT NULL,
    cognome VARCHAR(50) NOT NULL,
    pw VARCHAR(255) NOT NULL,
    CONSTRAINT giudice_pkey PRIMARY KEY (email),
    CONSTRAINT chk_email CHECK (
     email ~ '^[a-z]{1,}[a-z]{1,}[0-9]*@giudice\.com$')
);
```

4.1.3 Definizione della Tabella CONCORRENTE

```
--Definizione tabella
CREATE TABLE CONCORRENTE
(
    email VARCHAR(150) NOT NULL,
    nome VARCHAR(50) NOT NULL,
    cognome VARCHAR(50) NOT NULL,
    pw VARCHAR(255) NOT NULL,
    CONSTRAINT concorrente_pkey PRIMARY KEY (email),
    CONSTRAINT chk_email CHECK (
     email ~ '^[a-z]{1,}[a-z]{1,}[0-9]*@concorrente\.com$')
);
```

4.1.4 Definizione della Tabella HACKATHON

```
--Definizione tabella
CREATE TABLE HACKATHON
(
    titolo VARCHAR(100) NOT NULL,
    datainizio DATE,
    datafine DATE,
    nummaxiscritti INT,
    dimmaxteam INT,
    inizioiscrizioni DATE,
    fineiscrizioni DATE,
    descrizioneproblema VARCHAR(250),
    classifica VARCHAR(500),
    indirizzosede VARCHAR(50),
    creatore VARCHAR(50),
    numiscritti INT,
    numvotiassegnati INT,
    CONSTRAINT hackathon_pkey PRIMARY KEY (titolo),
    CONSTRAINT chk_durata CHECK (datafine >= datainizio),
    CONSTRAINT chk_iscrizioni CHECK (fineiscrizioni >=
       inizioiscrizioni),
    CONSTRAINT chk_nummaxiscritti CHECK (nummaxiscritti > 1),
    CONSTRAINT chk_tempoiscrizioni CHECK (datainizio > (
       fineiscrizioni + INTERVAL '2 days'))
);
```

4.1.5 Definizione della Tabella TEAM

```
--Definizione tabella
CREATE TABLE TEAM
(
    nome VARCHAR(50) NOT NULL,
    titolohackathon VARCHAR(100) NOT NULL,
    pw VARCHAR(255) NOT NULL,
    creatore VARCHAR(50),
    CONSTRAINT pk_team PRIMARY KEY (nome, titolohackathon),
    CONSTRAINT fk_team_hackathon FOREIGN KEY (titolohackathon)
       REFERENCES hackathon(titolo)
);
```

4.1.6 Definizione della Tabella DOCUMENTO

```
--Definizione tabella
CREATE TABLE DOCUMENTO
(
   iddocumento INT NOT NULL,
   percorso VARCHAR(100) NOT NULL,
   nometeam VARCHAR(50) NOT NULL,
   titolohackathon(100) NOT NULL,
   dataInserimento DATE,
   nome VARCHAR(100),
   CONSTRAINT pk_documento PRIMARY KEY (iddocumento)
);
```

4.1.7 Definizione della Tabella CONVOCAZIONE

```
--Definizione tabella
CREATE TABLE CONVOCAZIONE
(
    emailorganizzatore VARCHAR(150) NOT NULL,
    emailgiudice VARCHAR(150) NOT NULL,
    titolohackathon VARCHAR(50) NOT NULL,
    CONSTRAINT pk_convocazione PRIMARY KEY (emailorganizzatore,
       emailgiudice, titolohackathon),
    CONSTRAINT fk_convocazione_giudice FOREIGN KEY (emailgiudice)
       REFERENCES giudice(email) ON DELETE CASCADE,
    CONSTRAINT fk_convocazione_hackathon FOREIGN KEY (
       titolohackathon) REFERENCES hackathon(titolo),
    CONSTRAINT fk_convocazione_organizzatore FOREIGN KEY (
       emailorganizzatore) REFERENCES public.organizzatore(email)
       ON DELETE CASCADE
);
```

4.1.8 Definizione della Tabella COMMENTO

```
--Definizione tabella
CREATE TABLE COMMENTO
(
    emailgiudice VARCHAR(150) NOT NULL,
    iddocumento INT NOT NULL,
    commento VARCHAR(50) NOT NULL,
    CONSTRAINT fk_commento_giudice FOREIGN KEY (emailgiudice)
       REFERENCES giudice(email) ON DELETE CASCADE
);
```

4.1.9 Definizione della Tabella VOTO

```
--Definizione tabella
CREATE TABLE VOTO
(
    emailgiudice VARCHAR(150) NOT NULL,
    nometeam VARCHAR(50) NOT NULL,
    titolohackathon VARCHAR(50) NOT NULL,
    valore INT NOT NULL,
    CONSTRAINT pk_voto PRIMARY KEY (emailgiudice, nometeam,
       titolohackathon),
    CONSTRAINT voto_check CHECK (((voto >= 0) AND (voto <= 10))),
    CONSTRAINT fk_voto_giudice FOREIGN KEY (emailgiudice)
       REFERENCES giudice(email) ON DELETE CASCADE,
    CONSTRAINT fk_voto_team FOREIGN KEY (nometeam, titolohackathon)
        REFERENCES team(nome, titolohackathon
);
```

4.1.10 Definizione della Tabella CONCORRENTE_TEAM

```
--Definizione tabella
CREATE TABLE CONCORRENTE_TEAM
(
    emailconcorrente VARCHAR(150) NOT NULL,
    nometeam VARCHAR(50) NOT NULL,
    titolohackathon VARCHAR(150) NOT NULL
    CONSTRAINT pk_concorrente_team PRIMARY KEY (emailconcorrente,
       nometeam, titolohackathon),
    CONSTRAINT fk_concorrente_emailconcorrente FOREIGN KEY (
       emailconcorrente) REFERENCES concorrente(email) ON DELETE
       CASCADE,
    CONSTRAINT fk_concorrente_team FOREIGN KEY (nometeam,
       titolohackathon) REFERENCES team(nome, titolohackathon)

);
```

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 Validità Data inizio iscrizioni

Quando viene creato un Hackathon, l'inizio delle iscrizioni deve avvenire almeno il giorno successivo alla creazione della gara.

```
CREATE FUNCTION check_inizioiscrizioni()
RETURNS TRIGGER
AS $$
BEGIN
    IF NEW.inizioiscrizioni < CURRENT_DATE + INTERVAL '1 day' THEN
        RAISE EXCEPTION 'Le iscrizioni devono iniziare almeno
           domani';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_check_inizioiscrizioni
BEFORE INSERT OR UPDATE ON hackathon
FOR EACH ROW
EXECUTE FUNCTION public.check_inizioiscrizioni();
```

4.2.2 Implementazione del vincolo Validità Data inserimento documento

Un documento può essere inserito esclusivamente durante il periodo di gara (tra l'inizio della gara e la fine di essa).

```
CREATE FUNCTION check_datainserimento_valid()
RETURNS TRIGGER
AS $$
DECLARE
    datainizio DATE;
    datafine DATE;
BEGIN
    SELECT h.datainizio, h.datafine
    INTO datainizio, datafine
    FROM hackathon h
    WHERE h.titolo = NEW.titolohackathon;
    IF NEW.datainserimento < datainizio OR NEW.datainserimento >
       datafine THEN
        RAISE EXCEPTION 'Non si deve inserire un documento prima
           che inizi la gara o dopo la sua fine';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_check_datainserimento_valid
BEFORE INSERT OR UPDATE ON documento
FOR EACH ROW
EXECUTE FUNCTION public.check_datainserimento_valid();
```

4.2.3 Implementazione del vincolo Unicità Partecipazione ad Hackathon

Un concorrente può partecipare ad al più un team per ogni Hackathon.

```
CREATE FUNCTION check_unique_partecipazione_hackathon()
RETURNS TRIGGER
AS $$
BEGIN
    IF EXISTS (
        SELECT 1 FROM concorrente_team ct
        WHERE ct.titolohackathon = NEW.titolohackathon
        AND ct.emailconcorrente = NEW.emailconcorrente
    ) THEN
        RAISE EXCEPTION 'Il partecipante gareggia in un team per
           questo Hackathon, quindi non deve iscriversi ad un altro
           team';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_check_unique_partecipazione_hackathon
BEFORE INSERT ON public.concorrente_team
FOR EACH ROW
EXECUTE FUNCTION public.check_unique_partecipazione_hackathon();
```

4.2.4 Implementazione del vincolo Dimensione massima dei team di un Hackathon

Ogni team di un determinato Hackathon ha un numero massimo di partecipanti. Quando un nuovo partecipante vuole iscriversi ad un team, bisogna verificare che esso non sia al completo.

```
CREATE FUNCTION check_dim_max_team_hackathon()
RETURNS TRIGGER
AS $$
DECLARE
    num_concorrenti INTEGER;
    dim_max_team INTEGER;
BEGIN
    SELECT COUNT(*)
    INTO num_concorrenti
    FROM concorrente_team
    WHERE titolohackathon = NEW.titolohackathon
    AND nometeam = NEW.nometeam;
    SELECT dimmaxteam
    INTO dim_max_team
    FROM hackathon
    WHERE titolo = NEW.titolohackathon;
    IF num_concorrenti >= dim_max_team THEN
        RAISE EXCEPTION 'Non si deve inserire un nuovo partecipante
            in questo team. Numero massimo di partecipanti per
            questo team raggiunto.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_check_dim_max_team_hackathon
BEFORE INSERT ON concorrente_team
FOR EACH ROW
EXECUTE FUNCTION check_dim_max_team_hackathon();
```

4.2.5 Implementazione del vincolo Numero massimo di concorrenti di un Hackathon

Ogni Hackathon ha un numero massimo di concorrenti che possono partecipare. Quando un nuovo partecipante vuole iscriversi ad esso, bisogna verificare che non sia al completo.

```
CREATE FUNCTION check_num_max_concorrenti_hackathon()
RETURNS TRIGGER
AS $$
DECLARE
    num_iscritti INTEGER;
    num_max_iscritti INTEGER;
BEGIN
    SELECT numiscritti, nummaxiscritti
    INTO num_iscritti, num_max_iscritti
    FROM hackathon
    WHERE titolo = NEW.titolohackathon;
    IF num_iscritti >= num_max_iscritti THEN
        RAISE EXCEPTION 'Non si deve inserire un nuovo partecipante
            in questo team. Numero massimo di partecipanti per
            questo Hackathon raggiunto.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_check_num_max_concorrenti_hackathon
BEFORE INSERT ON concorrente_team
FOR EACH ROW
EXECUTE FUNCTION check_num_max_concorrenti_hackathon();
```

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: Calcolo media voti di un team

Funzione che restituisce la media dei voti ottenuti da un team.

```
CREATE FUNCTION media_voti_team(nome TEXT, titolo TEXT)
RETURNS NUMERIC
AS $$
DECLARE
    media NUMERIC;
BEGIN
    SELECT AVG(valore)
    INTO media
    FROM Voto
    WHERE nometeam = nome AND titolohackathon = titolo;
    RETURN media;
END;
$$ LANGUAGE plpgsql;
```

4.3.2 Funzione: Calcolo classifica di un Hackathon

Funzione che restituisce la classifica dell'Hackathon selezionato.

```
CREATE FUNCTION classifica_hackathon(titolo TEXT)
RETURNS TEXT
AS $$
DECLARE
    risultato TEXT := '';
    r RECORD;
BEGIN
    FOR r IN (
        SELECT nometeam, AVG(valore) AS media
        FROM Voto
        WHERE titolohackathon = titolo
        GROUP BY nometeam
        ORDER BY media DESC
  )
    LOOP
        risultato := risultato || r.nometeam || '␣' || r.media || E
           '\n';
    END LOOP;
    risultato := RTRIM(risultato, ',');
    RETURN risultato;
END;
$$ LANGUAGE plpgsql;
```

4.3.3 Funzione: Lista partecipanti di un Hackathon

Funzione che restituisce una stringa con i dati dei partecipanti di un determinato Hackathon

```
CREATE FUNCTION partecipanti_hackathon(titolo TEXT)
RETURNS TEXT
AS $$
DECLARE
    risultato TEXT := '';
    part RECORD;
BEGIN
    FOR part IN
        SELECT c.nome, c.cognome, ct.emailconcorrente
        FROM concorrente_team ct
        INNER JOIN concorrente c ON c.email = ct.emailconcorrente
        WHERE ct.titolohackathon = titolo
    LOOP
        risultato := risultato || part.nome || ' ' || part.cognome
           || ' ' || part.emailconcorrente || E'\n';
    END LOOP;
    risultato:= RTRIM(risultato, ', ');
    RETURN risultato;
END;
$$ LANGUAGE plpgsql;
```

4.3.4 Funzione: Lista documenti di un Team

Funzione che restituisce una stringa di percorsi file dei documenti caricati da un team di un Hackathon e ordinati in base alla data d'inserimento

```
CREATE FUNCTION documenti_team(nomeT TEXT, titolo TEXT)
RETURNS TEXT
AS $$
DECLARE
    risultato TEXT := '';
    doc RECORD;
BEGIN
    FOR doc IN (
        SELECT d.percorso
        FROM documento d
        WHERE d.nometeam = nomeT AND d.titolohackathon = titolo
        ORDER BY d.datainserimento
    )
    LOOP
        risultato := risultato || doc.percorso || E'\n';
    END LOOP;
    risultato := RTRIM(risultato, ', ');
    RETURN risultato;
END;
$$ LANGUAGE plpgsql;
```

4.3.5 Funzione: Incremento numero partecipanti di un Hackathon

Funzione che incrementa il numero di partecipanti di un Hackathon, ogni volta che vi si iscrive un nuovo concorrente tramite un team (nuovo o già esistente).

```
CREATE FUNCTION add_partecipante_hackathon()
RETURNS TRIGGER
AS $$
BEGIN
    UPDATE hackathon
    SET numiscritti = numiscritti + 1
    WHERE titolo = NEW.titolohackathon;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_add_partecipante_hackathon
AFTER INSERT ON concorrente_team
FOR EACH ROW
EXECUTE FUNCTION add_partecipante_hackathon();
```

4.3.6 Funzione: Incremento numero voti assegnati in un Hackathon

Funzione che incrementa il numero di voti assegnati in un Hackathon, ogni volta che un giudice assegna un voto ad un team.

```
CREATE FUNCTION add_voto_assegnato_hackathon()
RETURNS TRIGGER
AS $$
BEGIN
    UPDATE hackathon
    SET numvotiassegnati = numvotiassegnati + 1
    WHERE titolo = NEW.titolohackathon;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_add_voto_assegnato_hackathon
AFTER INSERT ON voto
FOR EACH ROW
EXECUTE FUNCTION add_voto_assegnato_hackathon();
```