Università degli Studi di Bologna

Facoltà di Scienze

Corso di studi in Informatica per il Management

Relazione
di
BOSTARTER
(progetto di Basi Dati)

Nome e Cognome: Zhiyuan Xie

Matricola: 0001088879

Docente del corso: Prof. Marco Di Felice

Tutor del corso: Dr. Leonardo Ciabattini

Anno accademico 2024-2025

RACCOLTA/ANALISI DEI REQUISITI

STEP1. Produrre un documento di specifica seguendo le buone prassi

Ogni **utente** dispone di indirizzo email (univoco), nickname, password, nome, cognome, anno di nascita, luogo di nascita.

La **competenza** dispone di un nome(univoco, di tipo stringa). La lista delle competenze è disponibile a tutti gli utenti, i quali possono associare a ciascuna competenza un livello(numero intero tra 0 e 5).

Alcuni utenti -*ma non tutti*- possono appartenere a due sotto-categorie:

- Un **amministratore** dispone di un codice di sicurezza ed è l'unico autorizzato a popolare la lista delle competenze.
- Un **creatore** dispone di #nr_progetti (ridondanza concettuale) ed affidabilità e solo lui può inserire uno o più progetti.

Ogni **progetto** ha un nome univoco, una descrizione, una data di inserimento, una o più foto, un budget da raggiungere per l'avvio, una data limite entro cui raggiungere tale budget e uno stato (campo enum: aperto/chiuso). Ogni progetto è associato da un solo utente creatore.
Inoltre, ogni progetto prevede una lista di **reward**: una reward dispone di un codice univoco, una descrizione, una foto.

I progetti appartengono *esclusivamente* a due categorie:

- **I progetti hardware** dispongono di una lista delle componenti necessarie: ogni **componente** ha un nome univoco, una descrizione, un prezzo, una quantità (>0).
- **I progetti software** dispongono di una lista dei profili necessari per lo sviluppo. Ogni **profilo** dispone di un nome (es. "Esperto AI") e possiede una lista di competenze, ciascuna con un livello (numero intero tra 0 e 5).

Ogni utente della piattaforma(comprende creatore e amministratore) può finanziare un progetto: ogni **finanziamento** dispone di un importo ed una data. Un utente potrebbe inserire più finanziamenti per lo stesso progetto, ma in date diverse.
Nel momento in cui la somma totale degli importi dei finanziamenti supera il budget del progetto, oppure il progetto resta in stato aperto oltre la data limite, lo stato di tale progetto diventa pari a chiuso: un progetto chiuso non accetta ulteriori finanziamenti.
Ad ogni finanziamento è associata una *sola* reward, tra quelle previste per il progetto finanziato.

Un utente può inserire commenti relativi ad un progetto. Ogni **commento** dispone di un id(univoco), una data ed un campo testo.

Il creatore può inserire massimo una **risposta** per ogni singolo commento(è sottointeso che sia l'utente creatore di quel progetto). Una risposta dispone di un id(univoco), una data ed un campo testo.

Ogni utente della piattaforma può inviare una **candidatura** a un numero qualsiasi di profili, a patto che, per ciascun profilo, l'utente disponga, per ciascuna competenza, di un livello superiore o uguale al valore richiesto.
Un progetto software può ricevere un numero qualsiasi di candidature per un certo profilo.
Il creatore può accettare o rifiutare la candidatura.

STEP2. Costruire un glossario dei termini

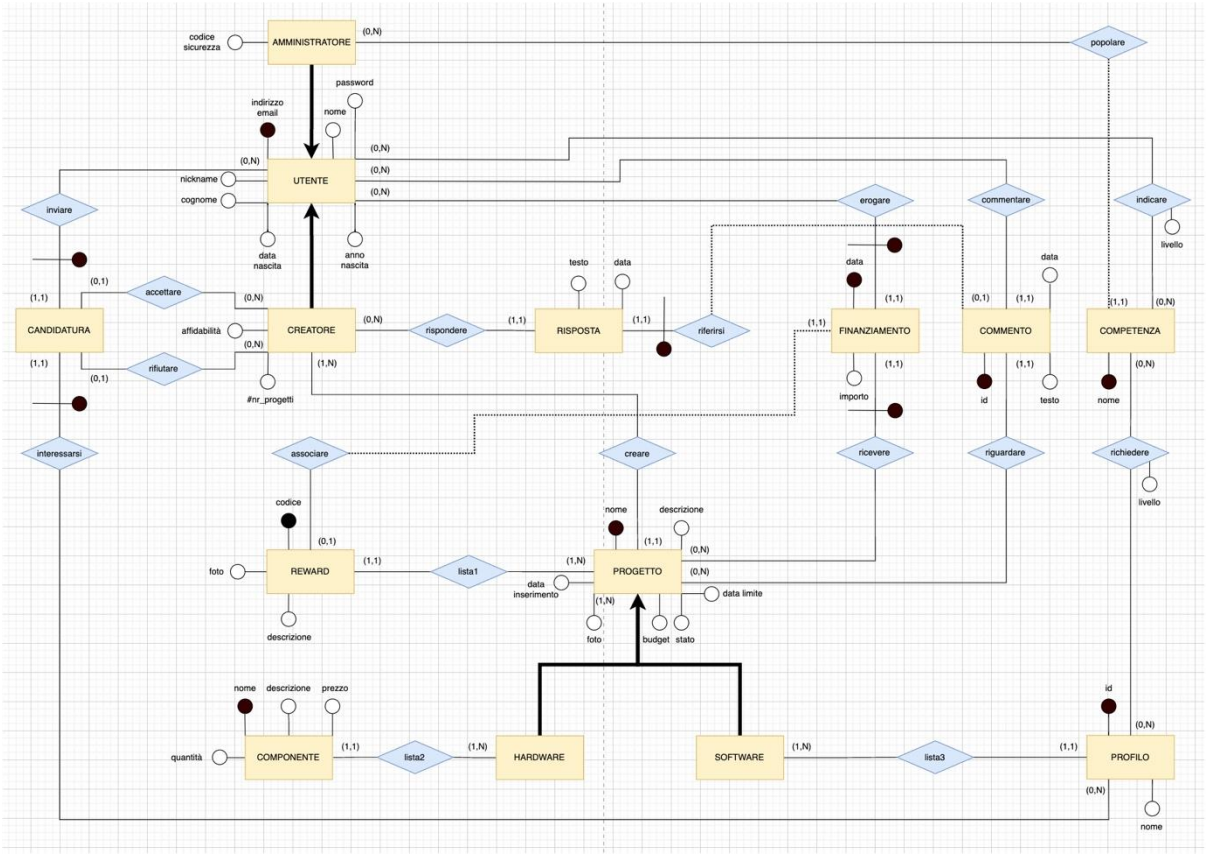
Termine	Descrizione	Sinonimi	Collegamenti
Utente	Persona che utilizza la piattaforma		Competenza, Finanziamento, Commento, Candidatura, Amministratore, Creatore
Competenza	Abilità o skill di un utente		Utente, Profilo, Amministratore
Amministratore	Utente autorizzato a popolare la lista delle competenze.		Utente, Competenza
Creatore	Utente autorizzato a inserire progetti e rispondere ai commenti.		Candidatura, Risposta, Utente, Progetto
Progetto	Iniziativa di un creatore		Creatore, Reward, Prog. Hardware, Prog. Software, Finanziamento, Commento
Reward	Premio di un progetto		Progetto
Prog. Hardware	Tipologia di progetto		Progetto, Componente
Componente	Elementi fisici necessari per un prog. hardware		Prog. Hardware
Prog. Software	Tipologia di progetto		Progetto, Profilo
Profilo	Figura richiesta da un prog. software		Prog. Software, Competenza, Candidatura
Finanziamento	Contributo economico effettuato da un utente su un progetto		Utente, Progetto, Reward
Commento	Feedback di un utente su un progetto		Risposta, Utente, Progetto
Risposta	Replica del creatore a un commento		Creatore, Commento
Candidatura	Proposta lavorativa di un utente per un profilo		Utente, Creatore, Profilo

STEP3. Definire le operazioni sui dati

- Inserire/modificare/visualizzare/eliminare un *utente della piattaforma*(utente generale, creatore o amministratore);
- Inserire ed associare una nuova competenza assegnare una competenza ad un utente della piattaforma specificando il livello (numero intero da 0 a 5);
- Modificare/visualizzare/eliminare una competenza;
- Inserire ed associare un nuovo progetto(hardware o software) ad un creatore;
- Modificare/visualizzare/eliminare un progetto;
- Inserire ed associare una nuova reward ad un progetto;
- Modificare/visualizzare/eliminare una reward;
- Inserire ed associare un nuovo componente ad un progetto hardware;
- Modificare/visualizzare/eliminare una componente;
- Inserire ed associare un nuovo profilo ad un progetto software;
- Modificare/visualizzare/eliminare un profilo;
- Inserire un finanziamento ed associarlo ad un utente della piattaforma e ad un progetto;
- Modificare/visualizzare/eliminare un finanziamento;
- Inserire un commento ed associarlo ad un utente della piattaforma e ad un progetto;
- Inserire una risposta ed associarla ad un creatore e ad un commento;
- Modificare/visualizzare/eliminare un commento/risposta;
- Inserire una candidatura ed associarla ad un utente della piattaforma e ad un profilo;
- Modificare/accettare/rifiutare/visualizzare/eliminare una candidatura.

PROGETTAZIONE CONCETTUALE

DIAGRAMMA E-R



Link: [Diagramma-ER-Bostarter]

BUSINESS RULES

Regole di Vincolo

• COMPETENZA.nome è una stringa.
• INDICARE.livello è un numero intero tra 0 e 5.
• RICHIEDERE.livello è un numero intero tra 0 e 5.
• PROGETTO.stato è un campo di tipo enum (aperto/chiuso).
• COMPONENTE.quantità deve essere maggiore di 0.
• Un utente può inserire più finanziamenti per lo stesso progetto, purché in date diverse.
• Un utente inserire una candidatura su un profilo SOLO se, per ogni competenza richiesta da un profilo, dispone di un livello superiore o uguale al valore richiesto.
• Il creatore può decidere di accettare o rifiutare una candidatura(mutua esclusione).

Regole di Derivazione

• Quando la somma totale degli importi dei finanziamenti supera il budget del progetto oppure il progetto resta in stato aperto oltre la data limite, lo stato del progetto viene automaticamente impostato su "chiuso" e il progetto non accetta ulteriori finanziamenti.

DIZIONARIO DELLE ENTITA'

Entità	Descrizione	Attributi	Identificatore
UTENTE	Persona che utilizza la piattaforma	indirizzo email, nickname, password, nome, cognome, anno nascita, luogo nascita	indirizzo email
COMPETENZA	Abilità o skill di un utente	nome	nome
AMMINISTRATORE	Utente autorizzato a popolare la lista delle competenze.	indirizzo email, nickname, password, nome, cognome, anno nascita, luogo nascita, codice sicurezza	indirizzo email
CREATORE	Utente autorizzato a inserire progetti e rispondere ai commenti.	indirizzo email, nickname, password, nome, cognome, anno nascita, luogo nascita, affidabilità, #nr_progetti	indirizzo email
PROGETTO	Iniziativa di un creatore	nome, descrizione, data inserimento, foto, data limite, budget, stato	nome
REWARD	Premio di un progetto	codice, descrizione, foto	codice
PROG. HARDWARE	Tipologia di progetto	nome, descrizione, data inserimento, foto, data limite, budget, stato	nome
COMPONENTE	Elementi fisici necessari per un prog. hardware	nome, descrizione, prezzo, quantità	nome
PROG. SOFTWARE	Tipologia di progetto	nome, descrizione, data inserimento, foto, data limite, budget, stato	nome
PROFILO	Figura richiesta da un prog. software	id, nome	id
FINANZIAMENTO	Contributo economico effettuato da un utente su un progetto	indirizzoEmailUtente, nomeProgetto, data, importo	indirizzoEmailUtente, nomeProgetto, data
COMMENTO	Feedback di un utente su un progetto	id, data, testo	id
RISPOSTA	Replica del creatore a un commento	id, data, testo	id
CANDIDATURA	Proposta lavorativa di un utente per un profilo	indirizzoEmailUtente, idProfilo	indirizzoEmailUtente, idProfilo

DIZIONARIO DELLE RELAZIONI

Relazione	Descrizione	Componenti	Attributi
popolare	Un amministratore popola la lista delle competenze	AMMINISTRATORE, COMPETENZA	
inviare	Un utente della piattaforma invia una candidatura	UTENTE, CANDIDATURA	
erogare	Un utente della piattaforma può erogare un finanziamento	UTENTE, FINANZIAMENTO	
commentare	Un utente della piattaforma può scrivere un commento	UTENTE, COMMENTO	
indicare	Un utente della piattaforma può indicare il livello delle competenze che possiede	UTENTE, COMPETENZA	livello
accettare	Un creatore può accettare una candidatura	CREATORE, CANDIDATURA	
rifiutare	Un creatore può rifiutare una candidatura	CREATORE, CANDIDATURA	
interessarsi	Una candidatura interessa un profilo	CANDIDATURA, PROFILO	
rispondere	Un creatore può rispondere ai commenti	CREATORE, RISPOSTA	
riferirsi	Una risposta si riferisce ad un commento specifico	RISPOSTA, COMMENTO	
associare	Un finanziamento viene associato ad una reward	FINANZIAMENTO, REWARD	
creare	Un creatore può creare progetti	CREATORE, PROGETTO	
ricevere	Un progetto riceve dei finanziamenti	PROGETTO, FINANZIAMENTO	
riguardare	Un commento riguarda un progetto specifico	COMMENTO, PROGETTO	
richiedere	Un profilo richiede un livello minimo a delle competenze	PROFILO, COMPETENZA	livello
lista1	La lista delle reward di un progetto	PROGETTO, REWARD	
lista2	La lista delle componenti di un progetto hardware	PROG. HARWARE, COMPONENTE	
lista3	La lista dei profili necessari di un progetto software	PROG. SOFTWARE, PROFILI	

PROGETTAZIONE LOGICA

RISTRUTTURAZIONE DELLO SCHEMA CONCETTUALE

- Eliminazione delle generalizzazioni:

  L'entità utente è una generalizzazione parziale→

  SOL3: Sostituzione delle generalizzazione con relazioni tra entità genitore ed entità figlie(vincolo: un'occorrenza del padre non può partecipare in contemporanea a tutte le sue figlie).

  L'entità progetto è una generalizzazione totale→

  SOL1: Accorpamento delle entità figlie nell'entità genitore(si sottintende il fatto che se un progetto ha componenti, allora è una progetto hardware. Se ha profili, allora è un progetto software. Non è possibile avere componenti e profili allo stesso tempo).

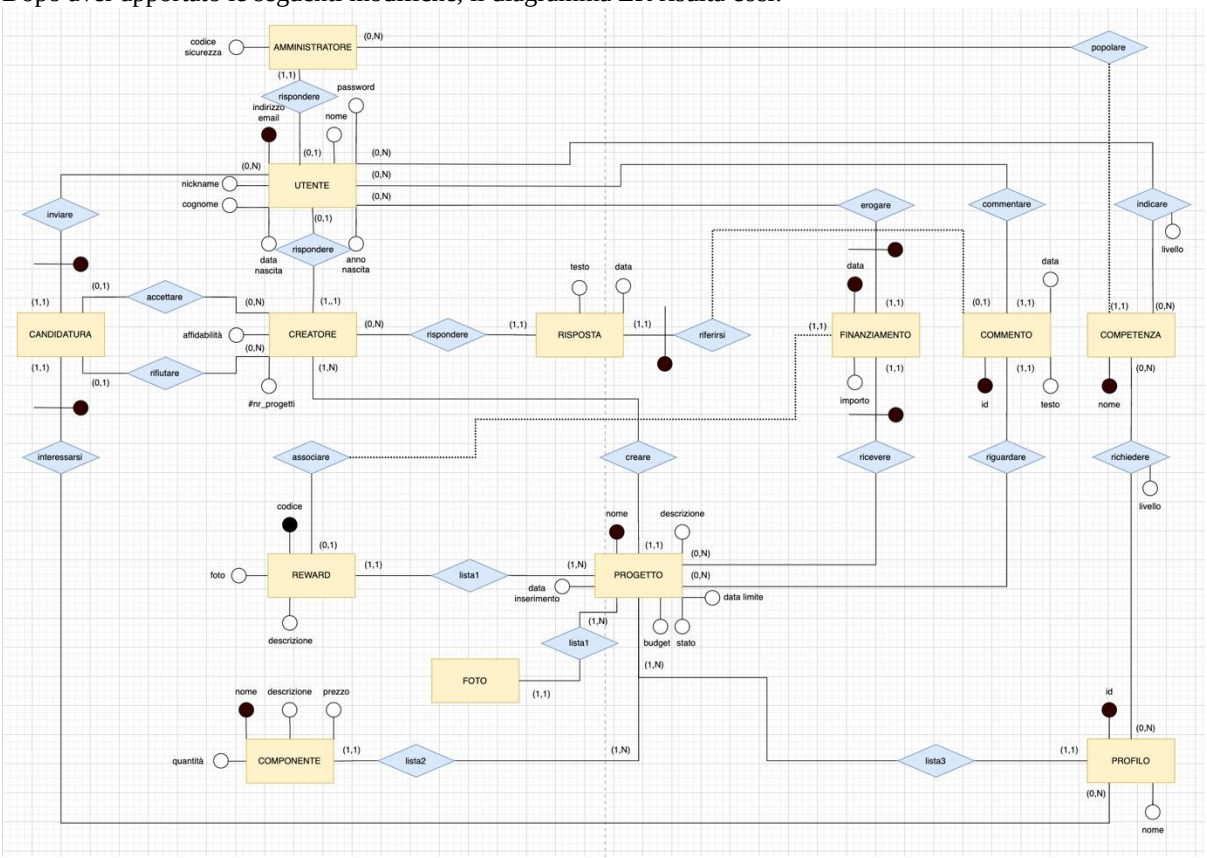
- Eliminazione degli attributi multivalore:

  PROGETTO.foto → foto diventa un'entità collegata a PROGETTO da una cardinalità(1,n).

- Partizionamento/accorpamento di concetti:

  Nessuna operazione necessaria, poiché è necessario avere una stima sul volume dei dati per un'indicazione se/come partizionare/accorpare entità.

Dopo aver apportato le seguenti modifiche, il diagramma ER risulta così:



Link: Diagramma-ER-2.0

ANALISI DELLA RIDONDANZA #nr_progetti (attributo di creatore)

Tabelle dei volumi:

1) Aggiungere(write) un nuovo progetto ad un utente creatore esistente (1 volte/mese, interattiva)
2) Visualizzare(read) tutti i progetti e tutti i finanziamenti (1 volta/mese, batch)
3) Contare(read) il numero di progetti associati ad uno specifico utente (3 volte/mese, batch)

Coefficienti per l'analisi:
wI = 1
wB = 0,5
a = 2

Tabella dei volumi:
10 progetti
3 finanziamenti per progetto
5 utenti
2 progetti per Utente

	CON	SENZA
1)	1*1*(2*3)=6	1*1*(2*2)=4
2)	1*0,5*(5*2 + 10*3)= 20	20
3)	3*0,5*(1)=1,5	3*0,5*(2)=3
	TOTALE = 27,5	TOTALE = 27

LISTA DELLE TABELLE CON VINCOLI DI CHIAVI

UTENTE(**indirizzoEmail**, nickname, password, nome, cognome, anno nascita, luogo nascita)

INDICARE(**indirizzoEmailUtente, nomeCompetenza,** livello)

COMPETENZA(**nome**, indirizzoEmailAmministratore)

AMMINISTRATORE(**indirizzoEmailUtente**, codice sicurezza)

CREATORE(**indirizzoEmailUtente**, affidabilità, #nr_progetti)

PROGETTO(**nome**, descrizione, data inserimento, data limite, budget, stato, indirizzoEmailCreatore)

FOTO(**nomeProgetto**, immagine)

COMPONENTE(**nome**, descrizione, prezzo, quantità, nomeProgetto)

REWARD(**codice**, descrizione, foto, nomeProgetto)

PROFILO(**id**, nome, nomeProgetto)

RICHIEDERE(**idProfilo, nomeCompetenza**, livello)

FINANZIAMENTO(**indirizzoEmailUtente, nomeProgetto, data**, importo, codiceReward)

COMMENTO(**id**, data, testo, indirizzoEmailUtente, nomeProgetto)

RISPOSTA(**idCommento**, data, testo, indirizzoEmailCreatore)

CANDIDATURA(**indirizzoEmailUtente, idProfilo**)

ACCETTARE(**indirizzoEmailUtente, idProfilo, indirizzoEmailCreatore)**

RIFIUTARE(**indirizzoEmailUtente, idProfilo, indirizzoEmailCreatore)**

LISTA DEI VINCOLI INTER-RELAZIONALI

INDICARE.indirizzoEmailUtente	---------->	UTENTE.indirizzoEmail
INDICARE.nomeCompetenza	---------->	COMPETENZA.nome
COMPETENZA.indirizzoEmailAmministratore	---------->	AMMINISTRATORE.indirizzoEmailUtente
AMMINISTRATORE.indirizzoEmailUtente	---------->	UTENTE.indirizzoEmail
CREATORE.indirizzoEmailUtente	---------->	UTENTE.indirizzoEmail
PROGETTO.indirizzoEmailCreatore	---------->	CREATORE.indirizzoEmailUtente
FOTO.nomeProgetto	---------->	PROGETTO.nome
COMPONENTE.nomeProgetto	---------->	PROGETTO.nome
REWARD.nomeProgetto	---------->	PROGETTO.nome
PROFILO.nomeProgetto	---------->	PROGETTO.nome
RICHIEDERE.idProfilo	---------->	PROFILO.id
RICHIEDERE.nomeCompetenza	---------->	COMPETENZA.nome
FINANZIAMENTO.indirizzoEmailUtente	---------->	UTENTE.indirizzoEmail
FINANZIAMENTO.nomeProgetto	---------->	PROGETTO.nome
FINANZIAMENTO.codiceReward	---------->	REWARD.codice
COMMENTO.indirizzoEmailUtente	---------->	UTENTE.indirizzoEmail
COMMENTO.nomeProgetto	---------->	PROGETTO.nome
RISPOSTA.indirizzoEmailCreatore	---------->	CREATORE.indirizzoEmailUtente
RISPOSTA.idCommento	---------->	COMMENTO.id
CANDIDATURA.indirizzoEmailUtente	---------->	UTENTE.indirizzoEmail
CANDIDATURA.idProfilo	---------->	PROFILO.id
ACCETTARE.indirizzoEmailUtente	---------->	UTENTE.indirizzoEmail
ACCETTARE.idProfilo	---------->	PROFILO.id
ACCETTARE.indirizzoEmailCreatore	---------->	CREATORE.indirizzoEmailUtente
RIFIUTARE.indirizzoEmailUtente	---------->	UTENTE.indirizzoEmail
RIFIUTARE.idProfilo	---------->	PROFILO.id
RIFIUTARE.indirizzoEmailCreatore	---------->	CREATORE.indirizzoEmailUtente

NORMALIZZAZIONE

L'obiettivo della normalizzazione è garantire che lo schema relazionale sia privo di anomalie di aggiornamento, inserimento e cancellazione. La normalizzazione si articola in due aspetti fondamentali:

1. L'eliminazione delle ridondanze fisiche (duplicazione di dati nelle tabelle)
2. L'identificazione e gestione controllata delle ridondanze concettuali (attributi derivabili)

STEP1.

Il primo passo consiste nella verifica formale che tutte le relazioni soddisfino almeno la Terza Forma Normale.

Prima Forma Normale (1NF)

Una relazione è in 1NF se e solo se tutti gli attributi contengono solo valori atomici e non sono complessi(più tipi).

Tutte le relazioni del nostro schema soddisfano la 1NF poiché:

- Ogni attributo contiene valori atomici
- Non sono presenti attributi multivalore o gruppi ripetuti

Si nota che l'attributo potenzialmente multivalore è *luogo di nascita* di UTENTE, che per semplicità abbiamo lasciato così, ma si potrebbe sostituire con *paese, provincia, città e numero civico*.

Seconda Forma Normale (2NF)

Una relazione è in 2NF se e solo se è in 1NF e ogni attributo non-primo dipende funzionalmente in modo completo dalla chiave primaria(ovvero che non ci sono dipendenze parziali).

Le tabelle con chiavi composte richiedono particolare attenzione:

1. **INDICARE(indirizzoEmailUtente, nomeCompetenza, livello)**:
   - Dipendenza funzionale: {indirizzoEmailUtente, nomeCompetenza} → {livello}
   - Non esistono dipendenze parziali del tipo {indirizzoEmailUtente} → {livello} o {nomeCompetenza} → {livello}
2. **RICHIEDERE(idProfilo, nomeCompetenza, livello)**:
   - Dipendenza funzionale: {idProfilo, nomeCompetenza} → {livello}
   - Non esistono dipendenze parziali
3. **FINANZIAMENTO(indirizzoEmailUtente, nomeProgetto, data, importo, codiceReward)**:
   - Dipendenza funzionale: {indirizzoEmailUtente, nomeProgetto, data} → {importo, codiceReward}
   - Non esistono dipendenze parziali
4. **CANDIDATURA(indirizzoEmailUtente, idProfilo)**, **ACCETTARE** e **RIFIUTARE**:
   - Non contengono attributi non-primi
   - Risultano automaticamente in 2NF

Queste tabelle e le restanti con chiave semplice sono dunque in 2NF.

Terza Forma Normale (3NF)

Una relazione è in 3NF se e solo se è in 2NF e per ogni dipendenza funzionale non banale X → A, almeno una delle seguenti condizioni è verificata:

- X è una superchiave della relazione
- A è un attributo primo
- A appartiene a X

Analizziamo tabella per tabella, risulta che in tutte le dipendenze funzionali non banali X → A, X è sempre una superchiave, soddisfacendo così la prima condizione della 3NF.

Forma Normale di Boyce-Codd (FNBC)

Una relazione è in BCNF se e solo se, per ogni dipendenza funzionale non banale X → Y, X è una superchiave della relazione.

Nel nostro schema, in ogni tabella:

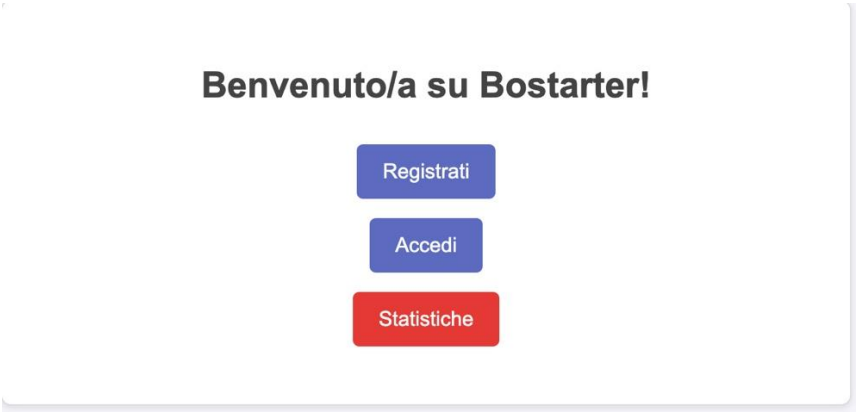
- Le uniche dipendenze funzionali sono della forma "chiave → attributi"
- Non esistono determinanti che non siano superchiavi
- Tutte le tabelle risultano in FNBC

STEP2.

Dopo aver verificato l'assenza di ridondanze fisiche mediante la normalizzazione, procediamo all'identificazione delle ridondanze concettuali. Sono attributi derivabili da altri attributi o relazioni presenti nel database. Nel nostro schema sono presenti i seguenti casi:

1. **Attributo #nr_progetti in CREATORE**:
    - Definizione: conteggio del numero di progetti inseriti dall'utente creatore
    - Derivabilità: potrebbe essere calcolato dinamicamente con una query di conteggio sulla tabella PROGETTO
    - Giustificazione: l'analisi dei volumi ha dimostrato la convenienza di mantenere questo valore pre-calcolato
    - Controllo: viene aggiornato tramite trigger ogni volta che un creatore inserisce un nuovo progetto
2. **Attributo affidabilità in CREATORE**:
    - Definizione: percentuale di progetti che hanno ottenuto almeno un finanziamento
    - Derivabilità: calcolabile mediante query complesse che coinvolgono le tabelle PROGETTO e FINANZIAMENTO
    - Giustificazione: necessario per operazioni frequenti (come visualizzare la classifica degli utenti)
    - Controllo: aggiornato tramite trigger all'inserimento di nuovi progetti (denominatore) e nuovi finanziamenti (numeratore)
3. **Attributo stato in PROGETTO**:
    - Definizione: indicatore binario (aperto/chiuso) dello stato del progetto
    - Derivabilità: determinabile confrontando la somma dei finanziamenti con il budget o la data corrente con la data limite
    - Giustificazione: semplifica le query di selezione dei progetti disponibili per finanziamento
    - Controllo: gestito tramite trigger quando la somma dei finanziamenti supera il budget e tramite evento MySQL per la verifica della data limite

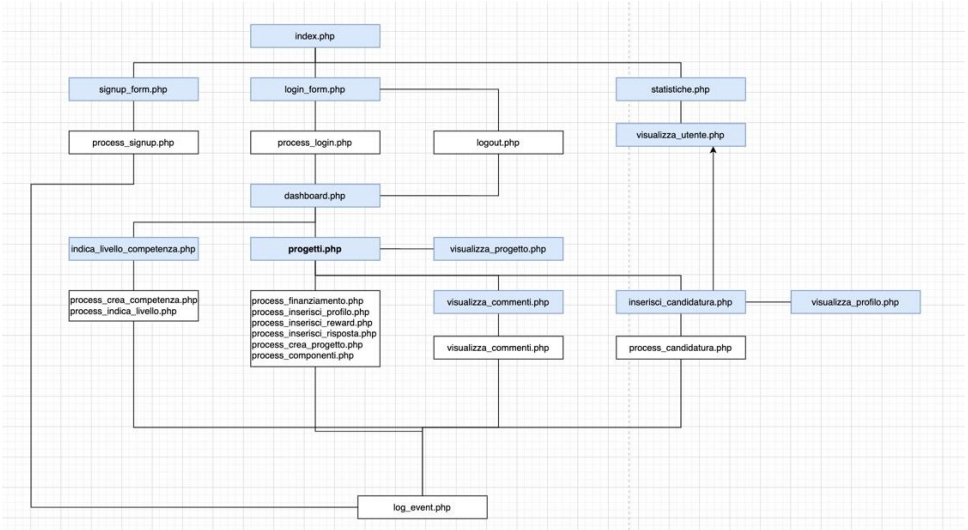
DESCRIZIONE DELL'APP WEB



Ecco un'overview delle feature aggiuntive e delle scelte architetturali che ho introdotto oltre alle operazioni e statistiche richieste:

- **Gestione ruoli in sessione e filtraggio UI**: ho centralizzato il ruolo dell'utente ($_SESSION['ruolo']) in fase di login, e nel front-end mostro/nascondo dinamicamente pulsanti e form in base a questo valore.
- **Gestione eccezioni**: ho cercato di gestire tutte le eccezioni possibili seguendo una validazione a più livello, ovvero sia a livello di database(stored procedures) sia a livello applicativo(script .php).
- **Modal dinamici per input contestuale**: per ogni operazione di inserimento ho utilizzato dei modali JavaScript/CSS anziché pagine separate, migliorando l'usabilità senza ricaricare l'intera pagina.
- **Logout "vero" e prevenzione back-button**: alla pressione di "Esci" la sessione viene distrutta (session_destroy()), e grazie a header di cache-control/social headers impedisco il ritorno alla dashboard tramite il tasto "Indietro" del browser, simulando un ambiente di produzione più sicuro.
- **Modulo di logging esterno**: hai isolato tutta la logica di scrittura su MongoDB in un file log_event.php (incluso con require_once) che espone solo la funzione logEvent(). Così gli script rimangono inalterati: basta invocare logEvent("message")garantendo un tracciamento centralizzato e manutenibile.
- **Vincolo del creatore**: i creatori possono eseguire le operazione aggiuntive solo su progetti creati da lui stesso.
- **Operazione inserisciComponente**: ho aggiunto la possibilità per i creatori di aggiungere componenti necessari per un progetto.
- **Visualizzare il profilo candidato**: quando il creatore apre la lista di candidature, può vedere per ogni candidatura il profilo del candidato.

Infine ho creato un diagramma di flusso semplificato degli script .php per tenere traccia delle interazioni tra le pagine:



CODICE SQL

Queste sone le **tabelle:**

```sql
-- 1. UTENTE
CREATE TABLE UTENTE (
    indirizzoEmail VARCHAR(255) PRIMARY KEY,
    nickname       VARCHAR(255) NOT NULL UNIQUE,
    password_      VARCHAR(255) NOT NULL,
    nome           VARCHAR(255) NOT NULL,
    cognome        VARCHAR(255) NOT NULL,
    anno_nascita   INT NOT NULL,
    luogo_nascita  VARCHAR(255) NOT NULL
);

-- 2. AMMINISTRATORE
CREATE TABLE AMMINISTRATORE (
    indirizzoEmailUtente VARCHAR(255) PRIMARY KEY,
    codice_sicurezza     VARCHAR(255) NOT NULL,
    FOREIGN KEY (indirizzoEmailUtente) REFERENCES UTENTE(indirizzoEmail)
    ON DELETE CASCADE
);

-- 3. CREATORE
CREATE TABLE CREATORE (
    indirizzoEmailUtente VARCHAR(255) PRIMARY KEY,
    affidabilita         DECIMAL(5,2), #percentuale
    nr_progetti          INT default 0,
    FOREIGN KEY (indirizzoEmailUtente) REFERENCES UTENTE(indirizzoEmail)
    ON DELETE CASCADE,
    CHECK (affidabilita BETWEEN 0 AND 100),
    CHECK (nr_progetti >= 0)
);
-- 4. COMPETENZA
CREATE TABLE COMPETENZA (
    nome VARCHAR(255) PRIMARY KEY,
    indirizzoEmailAmministratore VARCHAR(255) NOT NULL,
    FOREIGN KEY (indirizzoEmailAmministratore) REFERENCES AMMINISTRATORE(indirizzoEmailUtente)
    ON DELETE CASCADE
);

-- 5. PROGETTO
CREATE TABLE PROGETTO (
    nome VARCHAR(255) PRIMARY KEY,
    descrizione TEXT,
    data_inserimento DATE NOT NULL,
    data_limite DATE NOT NULL,
    budget DECIMAL(10,2) NOT NULL comment 'valuta: EUR',
    stato ENUM('aperto','chiuso') NOT NULL,
    indirizzoEmailCreatore VARCHAR(255) NOT NULL,
    FOREIGN KEY (indirizzoEmailCreatore) REFERENCES CREATORE(indirizzoEmailUtente)
    ON DELETE CASCADE
);

-- 6. FOTO
CREATE TABLE FOTO (
    nomeProgetto VARCHAR(255) PRIMARY KEY,
    immagine     VARCHAR(255) NOT NULL,
    FOREIGN KEY (nomeProgetto) REFERENCES PROGETTO(nome)
    ON DELETE CASCADE
);
```

```sql
-- 7. COMPONENTE
CREATE TABLE COMPONENTE (
    nome         VARCHAR(255) PRIMARY KEY,
    descrizione  TEXT NOT NULL,
    prezzo       DECIMAL(10,2) NOT NULL,
    quantita     INT NOT NULL,
    nomeProgetto VARCHAR(255) NOT NULL,
    FOREIGN KEY (nomeProgetto) REFERENCES PROGETTO(nome)
    ON DELETE CASCADE,
    CHECK (quantita > 0)
);

-- 8. REWARD
CREATE TABLE REWARD (
    codice       VARCHAR(255) PRIMARY KEY,
    descrizione  TEXT,
    foto         VARCHAR(255) NOT NULL,
    nomeProgetto VARCHAR(255) NOT NULL,
    FOREIGN KEY (nomeProgetto) REFERENCES PROGETTO(nome)
    ON DELETE CASCADE
);

-- 9. PROFILO
CREATE TABLE PROFILO (
    id           INT PRIMARY KEY auto_increment,
    nome         VARCHAR(255) NOT NULL,
    nomeProgetto VARCHAR(255) NOT NULL,
    FOREIGN KEY (nomeProgetto) REFERENCES PROGETTO(nome)
      ON DELETE CASCADE
);

-- 10. INDICARE(skill di curriculum)
CREATE TABLE INDICARE (
    indirizzoEmailUtente VARCHAR(255),
    nomeCompetenza       VARCHAR(255),
    livello              INT NOT NULL,
    PRIMARY KEY (indirizzoEmailUtente, nomeCompetenza),
    FOREIGN KEY (indirizzoEmailUtente) REFERENCES UTENTE(indirizzoEmail)
    ON DELETE CASCADE,
    FOREIGN KEY (nomeCompetenza) REFERENCES COMPETENZA(nome)
    ON DELETE CASCADE,
    CHECK (livello BETWEEN 0 AND 5)
);

-- 11. RICHIEDERE(skill richiesti dal profilo)
CREATE TABLE RICHIEDERE (
    idProfilo      INT,
    nomeCompetenza VARCHAR(255),
    livello        INT NOT NULL,
    PRIMARY KEY (idProfilo, nomeCompetenza),
    FOREIGN KEY (idProfilo) REFERENCES PROFILO(id)
    ON DELETE CASCADE,
    FOREIGN KEY (nomeCompetenza) REFERENCES COMPETENZA(nome)
    ON DELETE CASCADE,
    CHECK (livello BETWEEN 0 AND 5)
);
```

```sql
-- 12. FINANZIAMENTO
CREATE TABLE FINANZIAMENTO (
    indirizzoEmailUtente VARCHAR(255),
    nomeProgetto         VARCHAR(255),
    data_                DATE NOT NULL,
    importo              DECIMAL(10,2) NOT NULL comment 'valuta: EUR',
    codiceReward         VARCHAR(255) NOT NULL,
    PRIMARY KEY (indirizzoEmailUtente, nomeProgetto, data_),
    FOREIGN KEY (indirizzoEmailUtente) REFERENCES UTENTE(indirizzoEmail)
    ON DELETE CASCADE,
    FOREIGN KEY (nomeProgetto) REFERENCES PROGETTO(nome)
    ON DELETE CASCADE,
    FOREIGN KEY (codiceReward) REFERENCES REWARD(codice)
    ON DELETE CASCADE
);

-- 13. COMMENTO
CREATE TABLE COMMENTO (
    id                   INT PRIMARY KEY AUTO_INCREMENT,
    data_                DATE NOT NULL,
    testo                TEXT NOT NULL,
    indirizzoEmailUtente VARCHAR(255) NOT NULL,
    nomeProgetto         VARCHAR(255) NOT NULL,
    FOREIGN KEY (indirizzoEmailUtente) REFERENCES UTENTE(indirizzoEmail)
    ON DELETE CASCADE,
    FOREIGN KEY (nomeProgetto) REFERENCES PROGETTO(nome)
    ON DELETE CASCADE
);

-- 14. RISPOSTA
CREATE TABLE RISPOSTA (
    idCommento           INT PRIMARY KEY,
    data_                DATE NOT NULL,
    testo                TEXT NOT NULL,
    indirizzoEmailCreatore VARCHAR(255) NOT NULL,
    FOREIGN KEY (indirizzoEmailCreatore) REFERENCES CREATORE(indirizzoEmailUtente)
    ON DELETE CASCADE,
    FOREIGN KEY (idCommento) REFERENCES COMMENTO(id)
    ON DELETE CASCADE
);

-- 15. CANDIDATURA
CREATE TABLE CANDIDATURA (
    indirizzoEmailUtente VARCHAR(255),
    idProfilo            INT,
    PRIMARY KEY (indirizzoEmailUtente, idProfilo),
    FOREIGN KEY (indirizzoEmailUtente) REFERENCES UTENTE(indirizzoEmail)
    ON DELETE CASCADE,
    FOREIGN KEY (idProfilo) REFERENCES PROFILO(id)
    ON DELETE CASCADE
);

-- 16. ACCETTARE(lista delle candidature accettate)
CREATE TABLE ACCETTARE (
    indirizzoEmailUtente   VARCHAR(255),
    idProfilo              INT,
    indirizzoEmailCreatore VARCHAR(255),
    PRIMARY KEY (indirizzoEmailUtente, idProfilo, indirizzoEmailCreatore),
    FOREIGN KEY (indirizzoEmailUtente) REFERENCES UTENTE(indirizzoEmail)
    ON DELETE CASCADE,
    FOREIGN KEY (idProfilo) REFERENCES PROFILO(id)
    ON DELETE CASCADE,
    FOREIGN KEY (indirizzoEmailCreatore) REFERENCES CREATORE(indirizzoEmailUtente)
    ON DELETE CASCADE
);

-- 17. RIFIUTARE(lista delle candidature rifiutate)
CREATE TABLE RIFIUTARE (
    indirizzoEmailUtente   VARCHAR(255),
    idProfilo              INT,
    indirizzoEmailCreatore VARCHAR(255),
    PRIMARY KEY (indirizzoEmailUtente, idProfilo, indirizzoEmailCreatore),
    FOREIGN KEY (indirizzoEmailUtente) REFERENCES UTENTE(indirizzoEmail)
    ON DELETE CASCADE,
    FOREIGN KEY (idProfilo) REFERENCES PROFILO(id)
    ON DELETE CASCADE,
    FOREIGN KEY (indirizzoEmailCreatore) REFERENCES CREATORE(indirizzoEmailUtente)
    ON DELETE CASCADE
);
```

Queste sono le **stored procedures:**

```
CREATE PROCEDURE signUpUtente(
    IN p_indirizzoEmail VARCHAR(255),
    IN p_nickname VARCHAR(255),
    IN p_password VARCHAR(255),
    IN p_nome VARCHAR(255),
    IN p_cognome VARCHAR(255),
    IN p_anno_nascita INT,
    IN p_luogo_nascita VARCHAR(255)
)
BEGIN
    -- Controlla se l'indirizzo email è già registrato
    IF EXISTS (
        SELECT 1 FROM UTENTE
        WHERE indirizzoEmail = p_indirizzoEmail
    ) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Registrazione fallita: indirizzo email già esistente';
    END IF;

    -- Controlla se il nickname è già utilizzato
    IF EXISTS (
        SELECT 1 FROM UTENTE
        WHERE nickname = p_nickname
    ) THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Registrazione fallita: nickname già esistente';
    END IF;

    -- Inserisce il nuovo utente
    INSERT INTO UTENTE (indirizzoEmail, nickname, password_, nome, cognome, anno_nascita, luogo_nascita)
    VALUES (p_indirizzoEmail, p_nickname, p_password, p_nome, p_cognome, p_anno_nascita, p_luogo_nascita);

    SELECT 'Registrazione effettuata con successo' AS Messaggio;
END //
```

```
DELIMITER //
CREATE PROCEDURE signUpCreatore(IN p_indirizzoEmail VARCHAR(255))
BEGIN
    DECLARE v_userCount INT;
    DECLARE v_creatoreCount INT;

    -- Controlla se l'email corrisponde a un Utente registrato
    SELECT COUNT(*) INTO v_userCount
    FROM UTENTE
    WHERE indirizzoEmail = p_indirizzoEmail;

    IF v_userCount = 0 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Per essere creatori, bisogna essere prima registrati come utenti';
    END IF;

    -- Controlla se esiste già un creatore con lo stesso indirizzo email
    SELECT COUNT(*) INTO v_creatoreCount
    FROM CREATORE
    WHERE indirizzoEmailUtente = p_indirizzoEmail;

    IF v_creatoreCount > 0 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Non ci possono essere due creatori con lo stesso indirizzo email';
    END IF;

    -- Inserisce il nuovo creatore
    INSERT INTO CREATORE(indirizzoEmailUtente, affidabilita, nr_progetti) VALUES (p_indirizzoEmail, 0, 0);
END //
DELIMITER ;
```

```
DELIMITER //
CREATE PROCEDURE signUpAmministratore(
    IN p_indirizzoEmail VARCHAR(255),
    IN p_codice VARCHAR(50)
)
BEGIN
    DECLARE v_userCount INT;
    DECLARE v_adminCount INT;
    -- Verifica che il codice di sicurezza sia corretto
    IF p_codice <> 'admin123' THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Codice di sicurezza non valido';
    END IF;
    -- Verifica che l'email corrisponda a un utente registrato
    SELECT COUNT(*) INTO v_userCount
    FROM UTENTE
    WHERE indirizzoEmail = p_indirizzoEmail;
    IF v_userCount = 0 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Per essere amministratori, bisogna essere prima registrati come utenti';
    END IF;
    -- Verifica che non esista già un amministratore con lo stesso indirizzo email
    SELECT COUNT(*) INTO v_adminCount
    FROM AMMINISTRATORE
    WHERE indirizzoEmailUtente = p_indirizzoEmail;
    IF v_adminCount > 0 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Non ci possono essere due amministratori con lo stesso indirizzo email';
    END IF;
    -- Inserisce il nuovo amministratore
    INSERT INTO AMMINISTRATORE(indirizzoEmailUtente, codice_sicurezza) VALUES (p_indirizzoEmail, p_codice);
END //
DELIMITER ;
```

```
DELIMITER //
CREATE PROCEDURE loginUtente(
    IN p_nickname VARCHAR(255),
    IN p_password VARCHAR(255)
)
BEGIN
    IF EXISTS (
        SELECT 1 FROM UTENTE
        WHERE nickname = p_nickname
          AND password_ = p_password
    ) THEN
        SELECT 'Login utente effettuato con successo' AS Messaggio;
    ELSE
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Credenziali utente non valide';
    END IF;
END //
DELIMITER ;
```

```
DELIMITER //
CREATE PROCEDURE loginCreatore(
    IN p_nickname VARCHAR(255),
    IN p_password VARCHAR(255)
)
BEGIN
    IF EXISTS (
        SELECT 1
        FROM UTENTE U
        INNER JOIN CREATORE C ON U.indirizzoEmail = C.indirizzoEmailUtente
        WHERE U.nickname = p_nickname
          AND U.password_ = p_password
    ) THEN
        SELECT 'Login creatore effettuato con successo' AS Messaggio;
    ELSE
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Credenziali creatore non valide';
    END IF;
END //
DELIMITER ;
```

```
DELIMITER //
CREATE PROCEDURE loginAmministratore(
    IN p_nickname VARCHAR(255),
    IN p_password VARCHAR(255),
    IN p_codice VARCHAR(50)
)
BEGIN
    -- Controlla il codice di sicurezza
    IF p_codice <> 'admin123' THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Codice di sicurezza non valido';
    END IF;

    IF EXISTS (
        SELECT 1
        FROM UTENTE U
        INNER JOIN AMMINISTRATORE A ON U.indirizzoEmail = A.indirizzoEmailUtente
        WHERE U.nickname = p_nickname
          AND U.password_ = p_password
    ) THEN
        SELECT 'Login amministratore effettuato con successo' AS Messaggio;
    ELSE
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Credenziali amministratore non valide';
    END IF;
END //
DELIMITER ;
```

```
DELIMITER //
CREATE PROCEDURE indicaLivello(
    IN p_indirizzoEmail VARCHAR(255),
    IN p_nomeCompetenza VARCHAR(255),
    IN p_livello INT
)
BEGIN
    -- Controlla che il livello sia compreso tra 0 e 5
    IF p_livello < 0 OR p_livello > 5 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Livello non valido. Deve essere un numero intero compreso tra 0 e 5';
    END IF;

    -- Se l'utente ha già indicato un livello per quella competenza, cancella il record esistente
    DELETE FROM INDICARE
    WHERE indirizzoEmailUtente = p_indirizzoEmail
      AND nomeCompetenza = p_nomeCompetenza;

    -- Inserisce il nuovo livello nella tabella INDICARE
    INSERT INTO INDICARE(indirizzoEmailUtente, nomeCompetenza, livello)
    VALUES (p_indirizzoEmail, p_nomeCompetenza, p_livello);

    SELECT 'Livello assegnato correttamente' AS Messaggio;
END //
DELIMITER ;
```

```
CREATE PROCEDURE creaFinanziamento(IN p_indirizzoEmail VARCHAR(255),IN p_nomeProgetto VARCHAR(255),
    IN p_data DATE, IN p_importo DECIMAL(10,2), IN p_codiceReward VARCHAR(50)
)
BEGIN
    DECLARE stato_progetto VARCHAR(50);
    DECLARE cnt INT;
    -- Recupera lo stato attuale del progetto
    SELECT stato INTO stato_progetto
    FROM PROGETTO
    WHERE nome = p_nomeProgetto;
    -- Se il progetto non esiste o non è aperto, interrompi con errore
    IF stato_progetto IS NULL THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Progetto non trovato';
    ELSEIF stato_progetto <> 'aperto' THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Il progetto non è disponibile per il finanziamento';
    END IF;
    -- Controlla se esiste già un finanziamento per lo stesso utente, progetto e data
    SELECT COUNT(*) INTO cnt
    FROM FINANZIAMENTO
    WHERE indirizzoEmailUtente = p_indirizzoEmail
      AND nomeProgetto = p_nomeProgetto
      AND DATE(data_) = p_data;
    IF cnt > 0 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Hai già finanziato questo progetto oggi';
    END IF;
    -- Inserisce il finanziamento solo se il controllo duplicato non rileva operazioni precedenti
    INSERT INTO FINANZIAMENTO(indirizzoEmailUtente, nomeProgetto, data_, importo, codiceReward)
    VALUES (p_indirizzoEmail, p_nomeProgetto, p_data, p_importo, p_codiceReward);
    SELECT 'Finanziamento inserito correttamente' AS Messaggio;
END //
DELIMITER ;
```

```
DELIMITER //
CREATE PROCEDURE inserisciCandidatura(
    IN p_indirizzoEmail VARCHAR(255),
    IN p_idProfilo INT
)
BEGIN
    DECLARE cnt INT;

    -- Verifica se esiste già una candidatura per quell'indirizzo email e idProfilo
    SELECT COUNT(*) INTO cnt
    FROM CANDIDATURA
    WHERE indirizzoEmailUtente = p_indirizzoEmail
      AND idProfilo = p_idProfilo;

    IF cnt > 0 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Candidatura già esistente';
    END IF;

    -- Inserisce la nuova candidatura
    INSERT INTO CANDIDATURA(indirizzoEmailUtente, idProfilo)
    VALUES (p_indirizzoEmail, p_idProfilo);

    SELECT 'Candidatura inserita correttamente' AS Messaggio;
END //
DELIMITER ;
```

```
DELIMITER //
CREATE PROCEDURE creaCompetenza(
    IN p_nomeCompetenza VARCHAR(255),
    IN p_indirizzoEmailUtente VARCHAR(255)
)
BEGIN
    DECLARE cnt INT;

    -- Controlla se la competenza esiste già
    SELECT COUNT(*) INTO cnt
    FROM COMPETENZA
    WHERE nome = p_nomeCompetenza;

    IF cnt > 0 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Competenza già esistente';
    ELSE
        INSERT INTO COMPETENZA(nome, indirizzoEmailAmministratore)
        VALUES(p_nomeCompetenza, p_indirizzoEmailUtente);
        SELECT 'Competenza creata correttamente' AS Messaggio;
    END IF;
END //
DELIMITER ;
```

```
DELIMITER //
CREATE PROCEDURE inserisciCommento(
    IN p_indirizzoEmail VARCHAR(255),
    IN p_nomeProgetto VARCHAR(255),
    IN p_data DATE,
    IN p_testo TEXT
)
BEGIN
    INSERT INTO COMMENTO(data_, testo, indirizzoEmailUtente, nomeProgetto)
    VALUES(p_data, p_testo, p_indirizzoEmail, p_nomeProgetto);

    SELECT 'Commento inserito correttamente' AS Messaggio;
END //
DELIMITER ;
```

```
DELIMITER //
CREATE PROCEDURE creaProgetto(
    IN p_nome VARCHAR(255),
    IN p_descrizione TEXT,
    IN p_data_inserimento DATE,
    IN p_data_limite DATE,
    IN p_budget DECIMAL(10,2),
    IN p_stato VARCHAR(50),
    IN p_indirizzoEmailCreatore VARCHAR(255)
)
BEGIN
    DECLARE cnt INT;

    -- Controlla se esiste già un progetto con lo stesso nome
    SELECT COUNT(*) INTO cnt
    FROM PROGETTO
    WHERE nome = p_nome;

    IF cnt > 0 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Progetto già esistente';
    ELSE
        INSERT INTO PROGETTO(nome, descrizione, data_inserimento, data_limite, budget, stato, indirizzoEmailCreatore)
        VALUES(p_nome, p_descrizione, p_data_inserimento, p_data_limite, p_budget, p_stato, p_indirizzoEmailCreatore);
        SELECT 'Progetto creato correttamente' AS Messaggio;
    END IF;
END //
DELIMITER ;
```

```
DELIMITER //
CREATE PROCEDURE inserisciRisposta(
    IN p_idCommento INT,
    IN p_data DATE,
    IN p_testo TEXT,
    IN p_indirizzoEmailCreatore VARCHAR(255)
)
BEGIN
    INSERT INTO RISPOSTA(idCommento, data_, testo, indirizzoEmailCreatore)
    VALUES(p_idCommento, p_data, p_testo, p_indirizzoEmailCreatore);

    SELECT 'Risposta inserita correttamente' AS Messaggio;
END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE inserisciProfilo(
    IN p_nomeProfilo VARCHAR(255),
    IN p_nomeProgetto VARCHAR(255)
)
BEGIN
    INSERT INTO PROFILO(nome, nomeProgetto)
    VALUES (p_nomeProfilo, p_nomeProgetto);

    SELECT 'Profilo inserito correttamente' AS Messaggio;
END //
DELIMITER ;
```

```
DELIMITER //
CREATE PROCEDURE accettaCandidatura(
    IN p_idProfilo INT,
    IN p_indirizzoEmailUtente VARCHAR(255),
    IN p_indirizzoEmailCreatore VARCHAR(255)
)
BEGIN
    -- Elimina la candidatura dalla tabella CANDIDATURA
    DELETE FROM CANDIDATURA
    WHERE idProfilo = p_idProfilo;
    -- Elimina il profilo dalla tabella CANDIDATURA
    -- Inserisce il record nella tabella ACCETTARE
    INSERT INTO ACCETTARE(indirizzoEmailUtente, idProfilo, indirizzoEmailCreatore
    VALUES (p_indirizzoEmailUtente, p_idProfilo, p_indirizzoEmailCreatore);

    SELECT 'Candidatura accettata correttamente' AS Messaggio;
END //
DELIMITER ;
```

```
DELIMITER //
CREATE PROCEDURE inserisciReward(
    IN p_codice VARCHAR(50),
    IN p_descrizione TEXT,
    IN p_foto VARCHAR(255),
    IN p_nomeProgetto VARCHAR(255)
)
BEGIN
    DECLARE cnt INT;

    -- Controlla se esiste già una reward con lo stesso codice per il progetto indicato
    SELECT COUNT(*) INTO cnt
    FROM REWARD
    WHERE codice = p_codice AND nomeProgetto = p_nomeProgetto;

    IF cnt > 0 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Reward già esistente per questo progetto';
    ELSE
        INSERT INTO REWARD(codice, descrizione, foto, nomeProgetto)
        VALUES(p_codice, p_descrizione, p_foto, p_nomeProgetto);
        SELECT 'Reward inserita correttamente' AS Messaggio;
    END IF;
END //
DELIMITER ;
```

```
DELIMITER //
CREATE PROCEDURE rifiutaCandidatura(
    IN p_idProfilo INT,
    IN p_indirizzoEmailUtente VARCHAR(255),
    IN p_indirizzoEmailCreatore VARCHAR(255)
)
BEGIN
    -- Elimina la candidatura dalla tabella CANDIDATURA
    DELETE FROM CANDIDATURA
    WHERE idProfilo = p_idProfilo
      AND indirizzoEmailUtente = p_indirizzoEmailUtente;

    -- Inserisce il record nella tabella RIFIUTARE
    INSERT INTO RIFIUTARE(indirizzoEmailUtente, idProfilo, indirizzoEmailCreatore)
    VALUES (p_indirizzoEmailUtente, p_idProfilo, p_indirizzoEmailCreatore);

    SELECT 'Candidatura rifiutata correttamente' AS Messaggio;
END //
DELIMITER ;

DELIMITER //
```

```
DELIMITER //

CREATE PROCEDURE inserisciComponente (
    IN p_nome          VARCHAR(100),
    IN p_descrizione   TEXT,
    IN p_prezzo        DECIMAL(10,2),
    IN p_quantita      INT,
    IN p_nomeProgetto  VARCHAR(100)
)
BEGIN
    /* Validazioni */
    IF p_prezzo <= 0 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Il prezzo deve essere maggiore di 0';
    END IF;

    IF p_quantita <= 0 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'La quantità deve essere maggiore di 0';
    END IF;

    /* Inserimento */
    INSERT INTO COMPONENTE
        (nome, descrizione, prezzo, quantita, nomeProgetto)
    VALUES
        (p_nome, p_descrizione, p_prezzo, p_quantita, p_nomeProgetto);

    /* Messaggio di feedback per il chiamante */
    SELECT 'Componente inserita correttamente.' AS Messaggio;
END;
//
DELIMITER ;
```

```
DELIMITER //
CREATE PROCEDURE inserisciProfilo(
    IN p_nomeProfilo VARCHAR(255),
    IN p_nomeProgetto VARCHAR(255)
)
BEGIN
    INSERT INTO PROFILO(nome, nomeProgetto)
    VALUES (p_nomeProfilo, p_nomeProgetto);

    SELECT 'Profilo inserito correttamente' AS Messaggio;
END //
DELIMITER ;
```

Infine, abbiamo **view**, **event** e **trigger**:

```
DELIMITER //
CREATE TRIGGER trg_update_affidabilita_after_finanziamento
AFTER INSERT ON FINANZIAMENTO
FOR EACH ROW
BEGIN
    DECLARE creatorEmail VARCHAR(255);
    DECLARE total INT;
    DECLARE funded INT;

    -- Recupera l'indirizzo email del creatore in base al progetto finanziato
    SELECT indirizzoEmailCreatore INTO creatorEmail
    FROM PROGETTO
    WHERE nome = NEW.nomeProgetto;

    -- Recupera il numero totale di progetti creati dal creatore (valore già memorizzato in nr_progetti)
    SELECT nr_progetti INTO total
    FROM CREATORE
    WHERE indirizzoEmailUtente = creatorEmail;

    -- Conta quanti progetti del creatore hanno ricevuto almeno un finanziamento
    SELECT COUNT(DISTINCT f.nomeProgetto) INTO funded
    FROM FINANZIAMENTO f
    INNER JOIN PROGETTO p ON f.nomeProgetto = p.nome
    WHERE p.indirizzoEmailCreatore = creatorEmail;

    -- Aggiorna l'affidabilità del creatore (se total > 0, altrimenti 0)
    UPDATE CREATORE
    SET affidabilita = IF(total > 0, (funded / total) * 100, 0)
    WHERE indirizzoEmailUtente = creatorEmail;
END //

DELIMITER ;
```

```
DELIMITER //

CREATE TRIGGER trg_update_nr_progetto_e_affidabilita
AFTER INSERT ON PROGETTO
FOR EACH ROW
BEGIN
    DECLARE total INT;
    DECLARE funded INT;

    -- Incrementa nr_progetti per il creatore
    UPDATE CREATORE
    SET nr_progetti = nr_progetti + 1
    WHERE indirizzoEmailUtente = NEW.indirizzoEmailCreatore;

    -- Recupera il numero totale di progetti dal creatore (ora aggiornato)
    SELECT nr_progetti INTO total
    FROM CREATORE
    WHERE indirizzoEmailUtente = NEW.indirizzoEmailCreatore;

    -- Conta quanti progetti creati dal creatore hanno ricevuto almeno un finanziamento
    SELECT COUNT(DISTINCT f.nomeProgetto) INTO funded
    FROM FINANZIAMENTO f
    INNER JOIN PROGETTO p ON f.nomeProgetto = p.nome
    WHERE p.indirizzoEmailCreatore = NEW.indirizzoEmailCreatore;

    -- Aggiorna l'affidabilità come percentuale
    UPDATE CREATORE
    SET affidabilita = IF(total > 0, (funded / total) * 100, 0)
    WHERE indirizzoEmailUtente = NEW.indirizzoEmailCreatore;
END //
DELIMITER ;
```

r:

```
DELIMITER //

CREATE TRIGGER trg_update_progetto_stato
AFTER INSERT ON FINANZIAMENTO
FOR EACH ROW
BEGIN
    DECLARE total_finanziato DECIMAL(10,2);
    DECLARE target_budget DECIMAL(10,2);

    -- Recupera il budget e la data limite del progetto associato al finanziamento inserito
    SELECT budget
    INTO target_budget
    FROM PROGETTO
    WHERE nome = NEW.nomeProgetto;

    -- Calcola il totale dei finanziamenti effettuati per il progetto
    SELECT IFNULL(SUM(importo),0)
    INTO total_finanziato
    FROM FINANZIAMENTO
    WHERE nomeProgetto = NEW.nomeProgetto;

    -- Se il totale finanziato raggiunge (o supera) il budget oppure se la data limite coincide con la data odierna,
    -- aggiorna lo stato del progetto a "chiuso"
    IF total_finanziato >= target_budget THEN
        UPDATE PROGETTO
        SET stato = 'chiuso'
        WHERE nome = NEW.nomeProgetto;
    END IF;
END //

DELIMITER ;
```

```
DELIMITER //
CREATE TRIGGER verifica_eta_utente BEFORE INSERT ON UTENTE
FOR EACH ROW
BEGIN
    DECLARE eta INT;
    -- Calcola l'età dell'utente
    SET eta = YEAR(CURDATE()) - NEW.anno_nascita;

    -- Verifica il lower bound per l'anno di nascita, ad esempio 1900
    IF NEW.anno_nascita < 1900 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Anno di nascita troppo basso. Registrazione non consentita.';
    END IF;

    -- Verifica che l'utente abbia almeno 18 anni
    IF eta < 18 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'L''utente deve avere almeno 18 anni.';
    END IF;
END//
DELIMITER ;
```

```sql
CREATE VIEW classifica_creatori_affidabili AS
SELECT u.nickname, affidabilita
FROM CREATORE
INNER JOIN UTENTE AS u ON u.indirizzoEmail = indirizzoEmailUtente
ORDER BY affidabilita DESC, nr_progetti DESC;

CREATE VIEW view_progetti_vicinanza AS
SELECT
    p.nome,
    p.descrizione,
    p.data_inserimento,
    p.data_limite,
    p.budget,
    p.stato,
    IFNULL(SUM(f.importo), 0) AS total_funding,
    (p.budget - IFNULL(SUM(f.importo), 0)) AS gap
FROM PROGETTO p
LEFT JOIN FINANZIAMENTO f ON p.nome = f.nomeProgetto
WHERE p.stato = 'aperto'
GROUP BY p.nome, p.descrizione, p.data_inserimento, p.data_limite, p.budget, p.stato
ORDER BY gap ASC;

CREATE VIEW view_classifica_finanziatori AS
SELECT
    u.nickname,
    f.indirizzoEmailUtente,
    SUM(f.importo) AS totale_finanziamenti
FROM FINANZIAMENTO f
INNER JOIN UTENTE u ON f.indirizzoEmailUtente = u.indirizzoEmail
GROUP BY f.indirizzoEmailUtente, u.nickname
ORDER BY totale_finanziamenti DESC;
```

```sql
CREATE EVENT `ev_chiusura_progetti`
ON SCHEDULE EVERY 1 DAY STARTS '2025-04-11 15:00:00.000000'
ON COMPLETION NOT PRESERVE DISABLE
COMMENT '\"Chiude i progetti se la data attuale supera la data limite\"'
DO UPDATE PROGETTO SET stato = 'chiuso' WHERE CURDATE() > data_limite AND stato = 'aperto';
```