UNIVERSITÀ DEGLI STUDI DI NAPOLI

FEDERICO II

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

Elaborato di Gruppo (Traccia 1)

Nappo Luigi - N86005118
Sauzullo Sarah - N86005140
Vajro Giulia - N86004997

Sommario

1. Introduzione

Il seguente elaborato descrive la progettazione concettuale di un database relazionale finalizzato alla gestione dei ToDo. Un ToDo è un'attività personale da svolgere in un periodo di tempo preciso, per cui si è considerato di sviluppare una base di dati in modo da gestire e organizzare i propri ToDo nel modo più efficiente possibile.

1.1. Descrizione del problema

L'obiettivo del progetto è tradurre tutti i requisiti di un programma che gestisca i ToDo in uno schema di database logico e ottimizzato, questo deve essere in grado di gestire svariate situazioni, quali:

- **Gestione degli Utenti:** un utente per poter accedere al programma deve potersi registrare con un username e una password univoche.
- **Gestione delle Bacheche**: l'entità Bacheca deve poter assumere Titoli specializzati (Università, Lavoro, Tempo Libero), con una descrizione opzionale. Il database deve inoltre permettere operazioni di modifica, creazione ed eliminazione, quindi deve essere flessibile.
- **Gestione del ToDo**: l'entità ToDo deve contenere tutte le informazioni necessarie, e deve essere collegata ad una specifica Bacheca. Esattamente come per le Bacheche, anche i ToDo possono essere modificati, creati ed eliminati nel database.
- **Condivisione dei ToDo**: il sistema di condivisione permette ad un ToDo di essere condiviso tra vari utenti, mantenendo quindi le stesse informazioni, bacheca, e posizione tra ognuno degli utenti possessori.
- **Valutazione della Scadenza ToDo**: il sistema deve poter fornire l'elenco dei ToDo in scadenza in giornata odierna, o in una specificata, inoltre deve essere possibile anche la ricerca secondo il titolo dei ToDo.
- **Attività del ToDo**: un'entità ToDo può anche possedere una Checklist di attività da svolgere per considerare quel medesimo ToDo completato, il sistema deve fornire un'analisi delle attività della Checklist, per aggiornare eventualmente lo stato del ToDo a "completato".

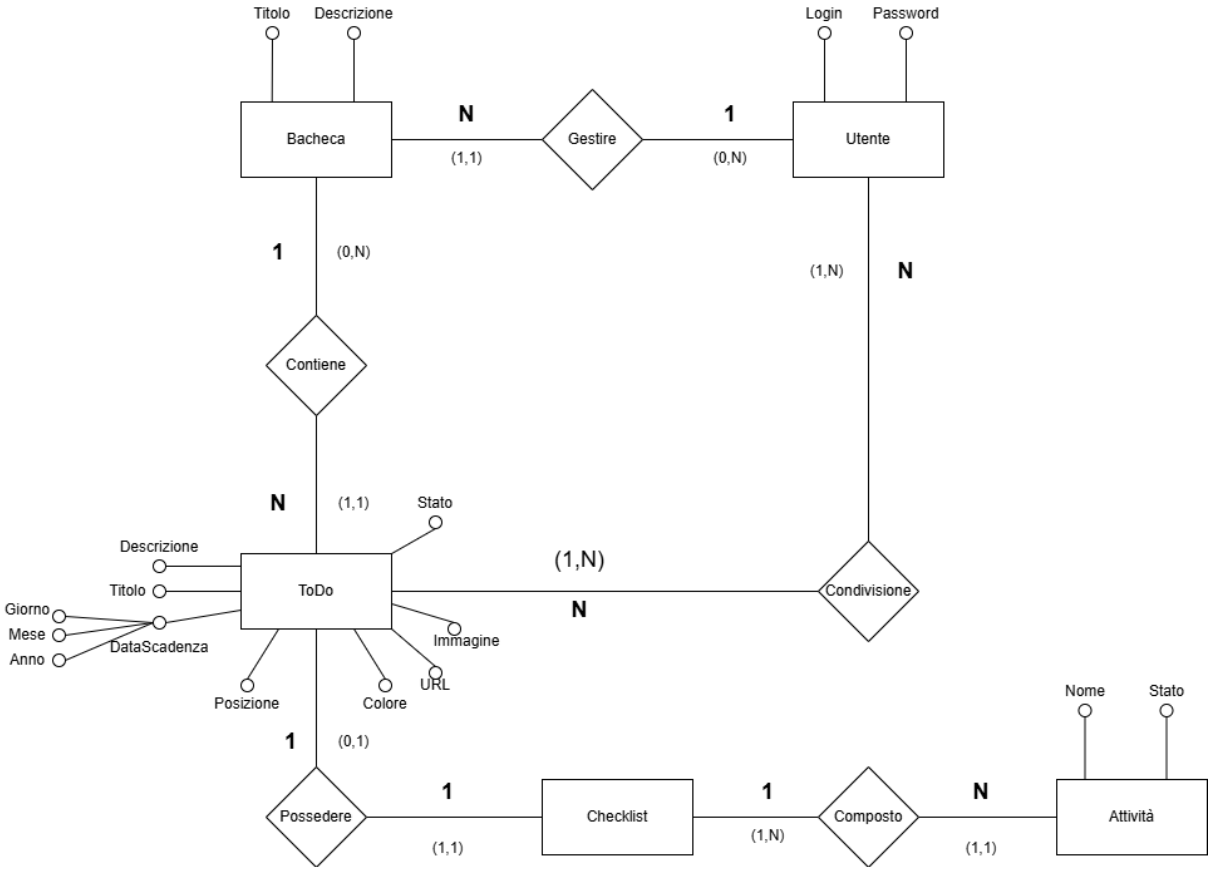
In breve, il nostro obiettivo è quello di produrre un modello di database atto a fungere da base per un sistema di gestione dei ToDo, garantendo il salvataggio di informazioni in modo semplice e facilmente accessibile.

2. Progettazione Concettuale

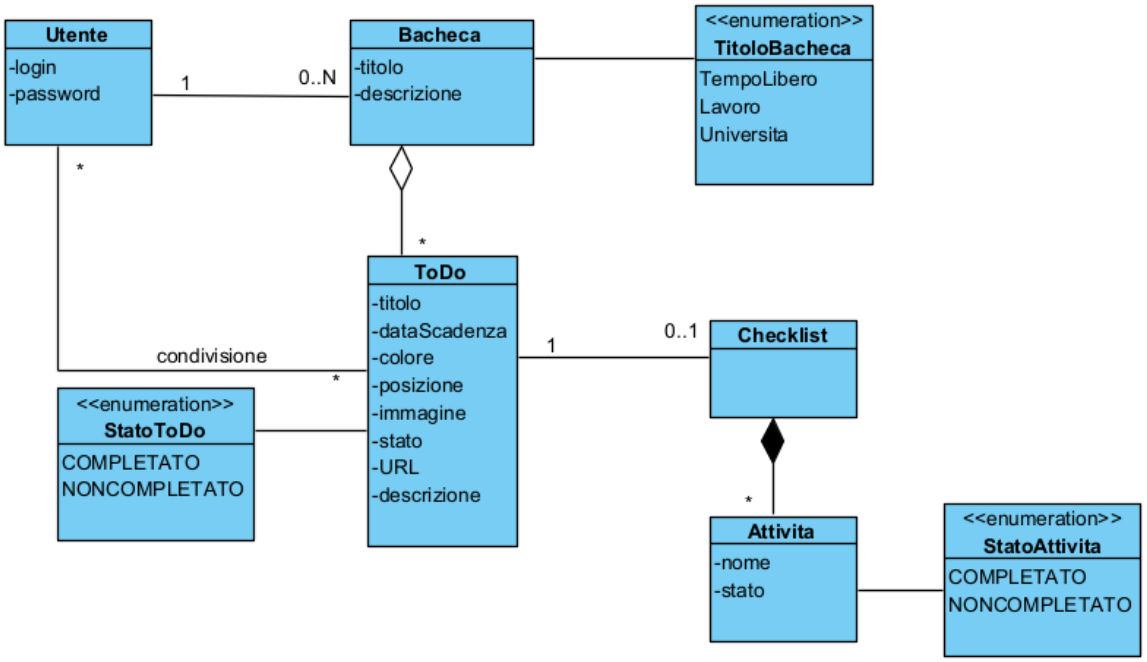
Questa sezione è dedicata alla progettazione concettuale del database che gestirà l'applicazione basata sui ToDo. L'obiettivo è quello di prendere tutti i requisiti che

abbiamo descritto nella traccia e tradurli in uno schema concettuale, che sia valido indipendentemente da come verrà implementato in seguito. Il risultato è quindi il Diagramma Entità-Relazione (ER), una mappa che mostra le entità principali del sistema, cosa le caratterizza, cioè gli attributi, e come sono collegate tra loro, cioè le relazioni. Questo schema farà da base per le successive fasi di progettazione (logica e fisica), e permette di controllare che la correttezza della struttura dei dati prima di passare all'implementazione effettiva della base di dati.

2.1. Schema concettuale ER



2.2. Schema concettuale UML



3. Ristrutturazione dello Schema Concettuale

In questa fase, si procede alla ristrutturazione dello schema concettuale elaborato, con l'obiettivo di ottimizzare il modello per renderlo più efficiente e idoneo alla successiva traduzione in uno schema relazionale. La ristrutturazione è stata condotta attraverso un'analisi sistematica dei seguenti punti chiave.

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

3.1. Analisi delle ridondanze

Il processo di analisi delle ridondanze si è concentrato sull'assicurare che le informazioni non fossero duplicate all'interno dello schema. Si conferma che il modello concettuale è stato progettato per essere intrinsecamente privo di ridondanze

strutturali non necessarie. I dati sono centralizzati nelle rispettive entità, e le associazioni stabiliscono le relazioni tra gli oggetti, evitando la duplicazione di attributi tra tabelle.

3.2. Analisi delle gerarchie di specializzazione

Non è stata identificata alcuna gerarchia di specializzazione (generalizzazione/specializzazione) all'interno del modello concettuale, che in un sistema di gestione ToDo non è un requisito previsto. Di conseguenza, non è stata necessaria alcuna operazione di risoluzione o trasformazione di gerarchie per lo schema.

3.3. Analisi degli attributi a valore multipli

Non sono stati identificati attributi a valore multiplo nel modello concettuale. Tutti gli attributi sono stati definiti come a singolo valore. Questa scelta progettuale garantisce la coerenza dei dati e facilita la conversione in uno schema relazionale.

3.4. Analisi degli attributi derivati

A seguito di un'attenta valutazione, si è concluso che il modello è privo di attributi derivati. Non è stato riscontrato alcun attributo che possa essere calcolato a partire da altri dati già presenti nello schema, mantenendo così la struttura snella e prevenendo potenziali incoerenze.

3.5. Analisi degli attributi strutturati

Non è stata rilevata la presenza di attributi strutturati (composti) che richiedessero una scomposizione per l'implementazione in un DBMS relazionale. Tutti gli attributi sono atomici per natura o sono stati definiti come tali, rendendo i dati immediatamente gestibili e ricercabili.

3.6. Analisi delle chiavi

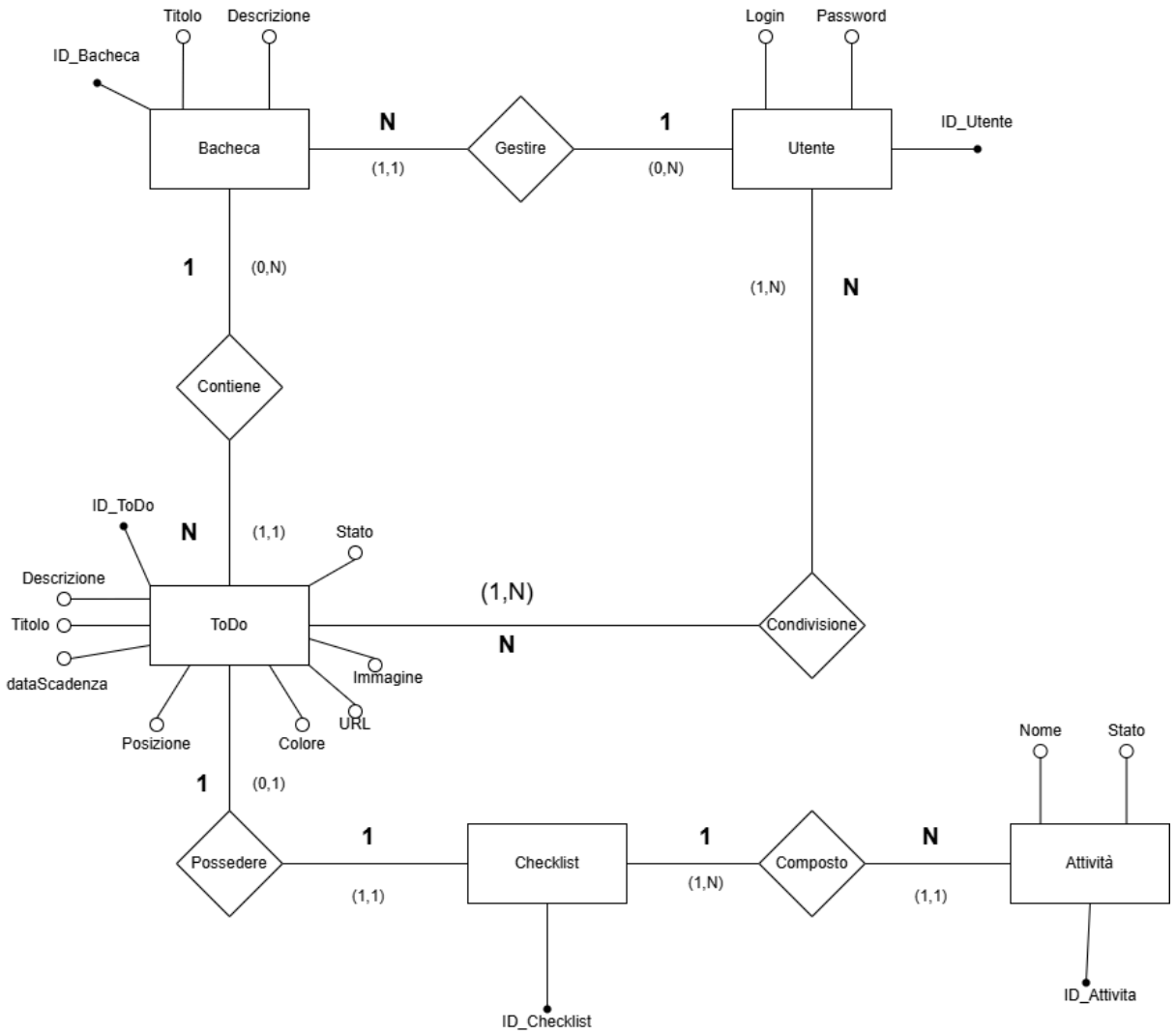
Per assicurare un' identificazione efficiente e non ambigua di ciascuna istanza nel sistema, è stata adottata la strategia di introdurre chiavi primarie surrogate di tipo intero. Questo approccio migliora l' efficienza computazionale nell' identificazione delle entità.

Nello specifico, sono stati definiti i seguenti identificativi univoci:

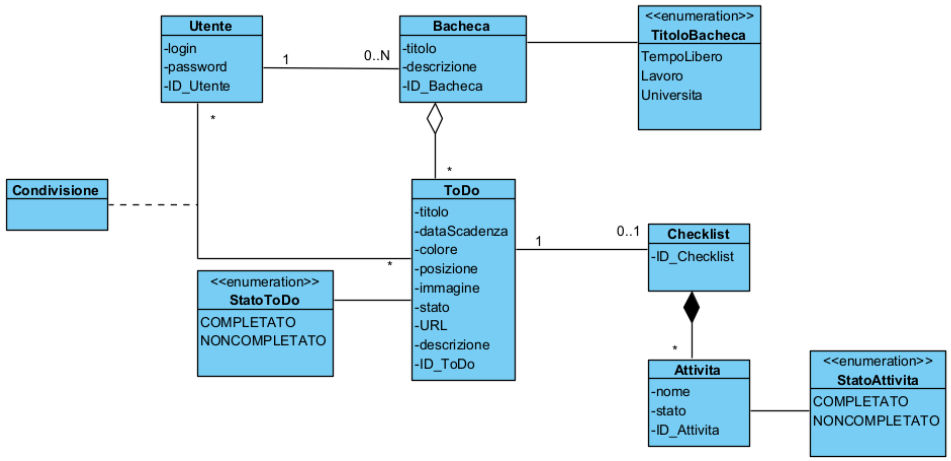
- **Entità principali:**
    - ID_Utente per l'entità Utente.
    - ID_ToDo per l'entità ToDo.
    - ID_Bacheca per l'entità Bacheca.
    - ID_Checklist per l'entità Checklist.
    - ID_Attività per l'entità Attività.
- **Associazioni con attributi (o relazioni M:N):**
    - Per l'entità-relazione Condivisione, è stata stabilita una chiave primaria composta formata da (ID_Utente, ID_ToDo). Questa composizione garantisce che un utente possa condividere un ToDo con un altro utente una sola volta, assicurando l'unicità dell'associazione di condivisione.

Questo metodo centralizzato garantisce che l'identificazione di ogni istanza sia rapida e inequivocabile, ponendo le basi per un modello logico robusto.

3.7. Schema Concettuale Ristrutturato ER



3.8. Schema Concettuale Ristrutturato UML



4. Dizionari

4.1. Dizionario delle classi

Classe	*Descrizione*	*Attributi*
Utente	Rappresenta l'entità principale del sistema, ovvero un utente che gestisce le proprie bacheche contenenti i ToDo.	• **ID_Utente** (int): identificativo univoco per un utente (chiave surrogata) • **login** (varchar(30)): username dell'utente • **password** (varchar(30)): password per accedere all'applicazione
ToDo	Rappresenta una singola attività personale con un periodo di tempo preciso per il suo svolgimento.	• **ID_ToDo** (int): identificativo univoco per un ToDo (Chiave surrigata) • **titolo** varchar(30): titolo del ToDo • **descrizione** (varchar(100)): descrizione del ToDo • **dataScadenza** (date): data in cui scade il todo • **colore** (varchar(30)): colore di sfondo del ToDo • **posizione** (int): posizione all'interno della bacheca • **immagine** (varchar(100)): path dell' immagine da inserire nel ToDo • **stato** varchar(30) : stato di completamento del ToDO [Completato, Non Completato]

		• **URL** (varchar(100)): url associato al ToDo
Bacheca	Contenitore che serve ad organizzare e raggruppare i ToDo in base a titoli specifici (es. Università, Lavoro, Tempo Libero).	• **ID_Bacheca** (int): identificativo • **titolo** varchar (30): titolo della bacheca [TempoLibero, Lavoro, Università] • **descrizione** (varchar(100)): descrizione della bacheca
Checklist	Rappresenta un contenitore,collegato a un ToDo, che contiene attività secondarie necessarie al suo completamento.	• **ID_Checklist** (int): identificativo univoco checklist (chiave surrogata).
Attività	Un singolo compito all'interno di una Checklist.	• **ID_Attività** (int): identificativo univoco dell'attività • **descrizione** (varchar(100): descrizione testuale dell'attività. • **stato** varchar(30): stato di completamento dell'attività [Completato,Non Completato]

4.2. Dizionario delle associazioni

Nome	***Descrizione***	***Classi Coinvolte***
Gestire	Definisce la relazione di proprietà, e quindi la libera gestione, delle Bacheche da parte di Utente	• **Utente[1]:** indica l'entità base. • **Bacheca[0..N]:** indica le istanze che può avere un utente, dell'entità Bacheca.
Contenere	Definisce l'appartenenza di un ToDo ad una Bacheca	• **Bacheca[1]:** ruolo contenitore, un ToDo appartiene ad una sola Bacheca. • **ToDo[1..N]:** ruolo componente, una Bacheca è composta da uno o più ToDo.
Condivisione	Definisce la possibilità di condividere dei ToDo tra diversi Utenti.	• **Utente[1..N]:** ruolo possessore, uno o più utenti possono possedere lo stesso ToDo, e quindi condividerlo. • **ToDo[1..N]:** ruolo appartenente, un ToDo può appartenere ad uno o più Utenti.
Possedere	Definisce la possibilità di un ToDo di possedere una CheckList.	• **ToDo[1]:** ruolo possessore, un ToDo può possedere o meno una Checklist. • **CheckList[0..1]:** ruolo posseduto, una CheckList può esistere, e quindi appartenere ad un ToDo, o non esistere.
Composto	Definisce il concetto di una CheckList composta da un insieme di Attività.	• **CheckList[1]:** ruolo composto, una Checklist può essere composta da una o più Attività. • **Attività[0..N]:** ruolo componente, un'Attività può esistere, e quindi

		appartenere ad una CheckList, o non esistere.

4.3 Dizionario dei vincoli

Vincolo	*Descrizione*
CHIAVE Utente.ID_Utente	Il valore di Utente.ID_Utente non può essere NULL, e deve rispettare l'integrità di Entità.
DOMINIO E VALIDITÀ ToDo.dataScadenza	Deve essere nel formato GG-MM-AAAA
DOMINIO ToDo.colore	Deve essere tra i colori prestabiliti
DOMINIO E UNICITÀ ToDo.posizione	La posizione deve essere un numero intero diverso per ogni ToDo, non ci possono essere due ToDo nella stessa posizione.
DOMINIO Bacheca.titolo	Deve essere un valore tra 'TempoLibero', 'Lavoro', 'Università'.
CARDINALITÀ MINIMA Checklist.ID_Checklist	Una Checklist deve esistere solo se include almeno un'attività ed è associata a uno specifico ToDo.
CHIAVE Bacheca.ID_Bacheca	Il valore di Bacheca.ID_Bacheca non può essere NULL, e deve rispettare l'integrità di Entità.
CHIAVE ToDo.ID_ToDo	Il valore di ToDo.ID_ToDo non può essere NULL, e deve rispettare l'integrità di Entità.
CHIAVE CheckList.ID_CheckList	Il valore di CheckList.ID_CheckList non può essere NULL, e deve rispettare l'integrità di Entità.
CHIAVE Attivita.ID_Attivita	Il valore di Attivita.ID_Attivita non può essere NULL, e deve rispettare l'integrità di Entità.

Vincolo	*Descrizione*
CHIAVE E INTEGRITÀ REFERENZIALE Condivisione.ID_Condivisione	Il valore di Condivisione.(ID_ToDo,ID_Utente) non possono essere NULL, e deve rispettare l'integrità di Entità e Referenziale (devono fare riferimento ad una tuola esistente).
INTEGRITÀ REFERENZIALE Bacheca.ID_Utente	Il valore di Bacheca.ID_Utente non può essere NULL, e deve fare riferimento ad una tupla esistente, rispettando quindi l'integrità referenziale per la tabella Utente.
INTEGRITÀ REFERENZIALE ToDo.ID_Bacheca	Il valore di ToDo.ID_Bacheca non può essere NULL, e deve fare riferimento ad una tupla esistente, rispettando quindi l'integrità referenziale per la tabella Bacheca.
INTEGRITÀ REFERENZIALE ToDo.ID_Utente	Il valore di ToDo.ID_Utente non può essere NULL, e deve fare riferimento ad una tupla esistente, rispettando quindi l'integrità referenziale per la tabella Utente.
INTEGRITÀ REFERENZIALE Checklist.ID_ToDo	Il valore di Checklist.ID_ToDo non può essere NULL, e deve fare riferimento ad una tupla esistente, rispettando quindi l'integrità referenziale per la tabella ToDo.
INTEGRITÀ REFERENZIALE Attivita.ID_Checklist	Il valore di Attivita.ID_Checklist non può essere NULL, e deve fare riferimento ad una tupla esistente, rispettando quindi l'integrità referenziale per la tabella Checklist.
UNICITÀ Utente.login	Il valore di Utente.login non può ripetersi nella tabella, in quanto un utente può iscriversi una sola volta e deve avere un nome diverso da qualsiasi altro utente.
UNICITÀ Utente.password	Il valore di Utente.password non può ripetersi nella tabella, in quanto un utente non può avere la stessa password di un altro utente nella tabella.
DOMINIO E UNICITÀ Bacheca	Il titolo di una bacheca deve essere scelto tra 'TempoLibero', 'Lavoro' o 'Università'. Un utente può possedere più bacheche con lo stesso titolo, a patto che abbiano una descrizione differente.

5. Progettazione Logica

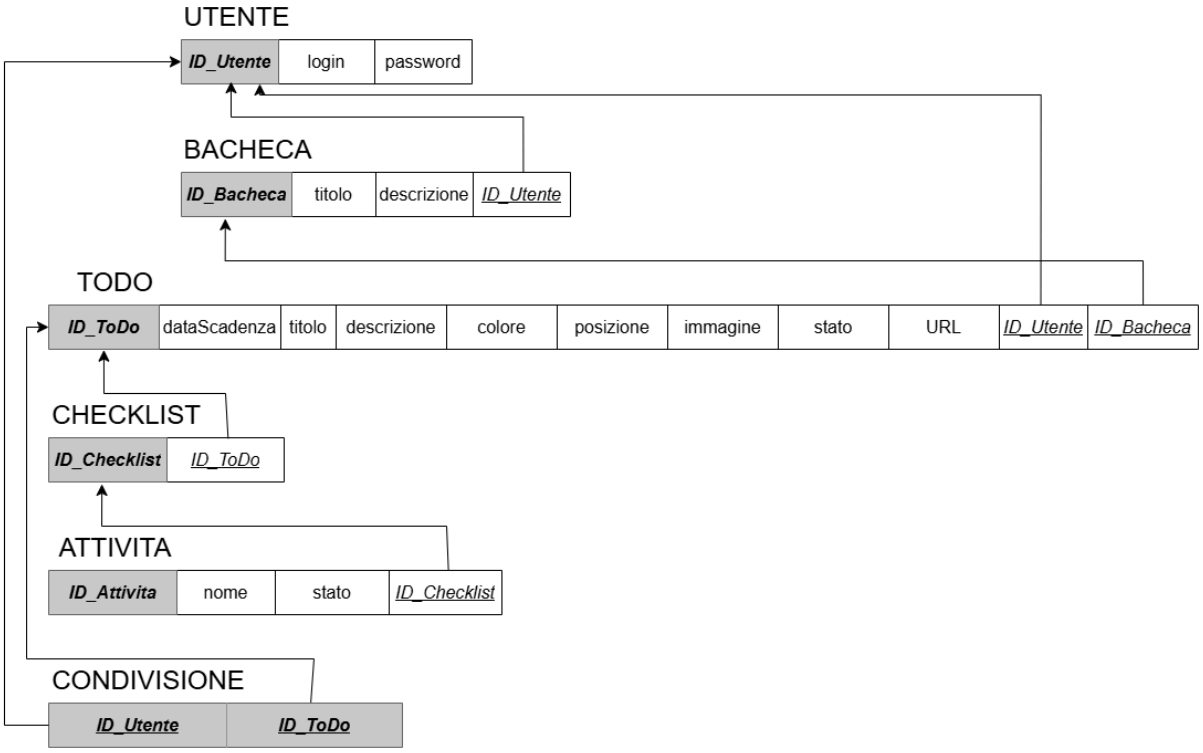
Questo segmento progettuale è dedicato alla traslazione formale del modello, rappresentando la discesa verso il livello di astrazione logica. Lo schema concettuale, validato e normalizzato attraverso la fase di ristrutturazione, viene qui mappato in uno Schema Logico Relazionale. Questa fase è cruciale in quanto introduce la dipendenza dal *Data Model* prescelto, in questo caso, la struttura relazionale pura.

5.1. Schema Logico

Di seguito è esposto l'assetto logico-relazionale risultante. Conformemente alla notazione convenzionale, le chiavi primarie (PK) sono indicate in **grassetto** mentre i vincoli di chiave esterna (FK) sono denotati con una sottolineatura.

- Utente **(ID_Utente**, login, password)
- Bacheca (**ID_Bacheca**, titolo, descrizione, ID_Utente)
  ID_Utente → Utente.ID_Utente
- ToDo **(ID_ToDo,** dataScadenza, titolo, colore, posizione, immagine, stato, URL, descrizione, ID_Utente, ID_Bacheca)
  ID_Utente → Utente.ID_Utente; ID_Bacheca → Bacheca.ID_Bacheca
- Checklist (**ID_Checklist**, ID_ToDo)
  ID_ToDo → ToDo.ID_ToDo
- Attività (**ID_Attività**, nome, stato, ID_Checklist)
  ID_Checklist → Checklist.ID_Checklist
- Condivisione (**ID_Utente**, **ID_ToDo)**
  ID_Utente → Utente.ID_Utente; ID_ToDo → ToDo.ID_ToDo

5.2. Risultato Schema Logico



6. Progettazione Fisica

Di seguito sarà implementata la progettazione dello schema logico nel DBMS PostgreSQL.

6.1. Definizione delle Tabelle

In questa parte saranno mostrate le definizioni delle tabelle, dei vincoli intrarelazionali, e se necessarie, di eventuali strutture per la loro gestione.

6.1.1. Definizione della tabella Utente

```
CREATE TABLE Utente(
    ID_Utente SERIAL PRIMARY KEY ,
    login VARCHAR (30) NOT NULL,
    password VARCHAR (30) NOT NULL,

    CONSTRAINT unique_login UNIQUE(login),
    CONSTRAINT unique_password UNIQUE(password)
);
```

6.1.2. Definizione della tabella Bacheca

```
CREATE TABLE Bacheca (
    ID_Bacheca SERIAL PRIMARY KEY,
    titolo VARCHAR (30) NOT NULL,
    descrizione VARCHAR (100) NOT NULL,
    ID_Utente INT NOT NULL,

    FOREIGN KEY (ID_Utente) REFERENCES Utente (ID_Utente) ON DELETE CASCADE,
    CONSTRAINT unique_bacheca_desc UNIQUE(ID_Utente, titolo, descrizione),
    CONSTRAINT check_titolo CHECK(titolo IN ('TempoLibero', 'Universita', 'Lavoro'))
);
```

6.1.3. Definizione della tabella ToDo

```
CREATE TABLE ToDo (
    ID_ToDo SERIAL PRIMARY KEY,
    descrizione VARCHAR (1000),
    dataScadenza DATE NOT NULL,
    titolo VARCHAR (30) NOT NULL,
    colore VARCHAR (30) NOT NULL,
    posizione INT NOT NULL,
    immagine VARCHAR (100),
    stato VARCHAR (30) NOT NULL DEFAULT 'NonCompletato',
    url VARCHAR (500),
    ID_Utente INT NOT NULL,
    ID_Bacheca INT NOT NULL,

    FOREIGN KEY (ID_Utente) REFERENCES Utente (ID_Utente) ON DELETE CASCADE,
    FOREIGN KEY (ID_Bacheca) REFERENCES Bacheca (ID_Bacheca) ON DELETE CASCADE,

    CONSTRAINT check_statoToDo CHECK (stato IN ('Completato', 'NonCompletato')),
    CONSTRAINT check_colori CHECK (colore IN ('rosso', 'giallo', 'blu', 'verde', 'arancione', 'rosa', 'viola', 'celeste', 'marrone'))
);
```

6.1.4. Definizione della tabella Checklist

```
CREATE TABLE CHECKLIST(
    ID_Checklist SERIAL PRIMARY KEY,
    ID_ToDo INT NOT NULL,

    FOREIGN KEY (ID_ToDo) REFERENCES ToDo (ID_ToDo) ON DELETE CASCADE,
    CONSTRAINT unique_checklist_todo UNIQUE (ID_Todo) -- Un ToDo può avere al massimo una checklist
);
```

6.1.5. Definizione della tabella Attivita

```
CREATE TABLE Attivita(
    ID_Attivita SERIAL PRIMARY KEY,
    nome VARCHAR (30) NOT NULL,
    stato VARCHAR (30) NOT NULL,
    ID_Checklist INT NOT NULL,
    FOREIGN KEY (ID_Checklist) REFERENCES Checklist (ID_Checklist) ON DELETE CASCADE,

    CONSTRAINT check_statoAttivita CHECK (stato IN ('Completato', 'NonCompletato'))
);
```

6.1.6. Definizione della tabella Condivisione

```
CREATE TABLE Condivisione (
    ID_Utente INT NOT NULL, --Utente che riceve la condivisione
    ID_ToDo INT NOT NULL,   --ToDo condiviso

    PRIMARY KEY (ID_Utente, ID_ToDo), -- evita duplicati
    FOREIGN KEY (ID_Utente) REFERENCES Utente (ID_Utente) ON DELETE CASCADE,
    FOREIGN KEY (ID_ToDo) REFERENCES ToDo (ID_ToDo) ON DELETE CASCADE
);
```

7. Trigger

Nel seguente capitolo verranno implementati e spiegati dei trigger per la gestione ottimale del database, inoltre sarà mostrata anche un'ulteriore vista per la visualizzazione unificata dei dati per un utente.

7.1. Dizionario dei Trigger

Nome	***Descrizione***
trg_aggiorna_todo	Il trigger serve a gestire il settaggio in automatico dello stato di un ToDo a "Completato" se tutte le

Nome	*Descrizione*
	attività all'interno della rispettiva Checklist sono complete.
trg_check_condivisione	Garantisce la coerenza visiva delle condivisioni: se l'utente destinatario non possiede una bacheca con lo stesso titolo del ToDo condiviso, il sistema ne crea automaticamente una copia per lui.
trg_prevenire_auto_condivisione	Impedisce di condividere un ToDo con se stessi.
trg_set_posizione_todo	Quando si crea un ToDo, lo inserisce automaticamente in fondo alla lista delle posizioni disponibili.
trg_riordina_dopo_delete	Quando si cancella un ToDo, bisogna fare una sorta di defrag, e quindi scalare/aggiustare le posizioni degli altri ToDo, in modo da poter sfruttare al meglio le posizioni in un secondo momento.
Vista_Bacheca_Unificata	Serve a raggruppare tutti i ToDo che un utente può visualizzare, in modo da avere tutte le informazioni in un'unica tabella.

7.2. Implementazione dei Trigger

7.2.1. Trigger trg_aggiorna_todo

```
CREATE OR REPLACE FUNCTION aggiorna_stato_todo()
RETURNS TRIGGER AS $$
DECLARE
    v_id_checklist INT;
    v_id_todo INT;
    totale_attivita INT;
    attivita_completate INT;
BEGIN
    -- Capisco su quale checklist stiamo lavorando (insert, update o delete)
    IF (TG_OP = 'DELETE') THEN
        v_id_checklist := OLD.ID_Checklist;
    ELSE
        v_id_checklist := NEW.ID_Checklist;
    END IF;
    -- Risalgo al ToDo padre
    SELECT ID_ToDo INTO v_id_todo
    FROM Checklist
    WHERE ID_Checklist = v_id_checklist;

    IF v_id_todo IS NULL THEN
        RETURN NULL;
    END IF;

    -- Conto le attività in totale
    SELECT COUNT(*) INTO totale_attivita
    FROM Attivita
    WHERE ID_Checklist = v_id_checklist;

    --Conto quelle già fatte
    SELECT COUNT(*) INTO attivita_completate
    FROM Attivita
    WHERE ID_Checklist = v_id_checklist AND stato = 'Completato';

    -- Se il totale > 0 e sono tutte fatte allora il ToDo è completo
    IF (totale_attivita > 0 AND totale_attivita = attivita_completate) THEN
        UPDATE ToDo
        SET stato = 'Completato'
        WHERE ID_ToDo = v_id_todo AND stato <> 'Completato';
    ELSE -- altrimenti non è completo
        UPDATE ToDo
        SET stato = 'NonCompletato'
        WHERE ID_ToDo = v_id_todo AND stato <> 'NonCompletato';
    END IF;

    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_aggiorna_todo
AFTER INSERT OR UPDATE OR DELETE ON Attivita
FOR EACH ROW
EXECUTE FUNCTION aggiorna_stato_todo();
```

7.2.2. Trigger trg_check_condivisione

```
-- Trigger 2: gestione della condivisione
--Crea una copia della bacheca dell'utente che condivide il todo,
--e l'aggiunge tra quelle dell'utente con cui ha condiviso il todo
CREATE OR REPLACE FUNCTION check_creazione_bacheca_condivisione()
RETURNS TRIGGER AS $$
DECLARE
    v_titolo_orig VARCHAR(30);
    v_desc_orig VARCHAR(100);
    v_esiste INT;
BEGIN
    -- Recupero sia il titolo che la descrizione dalla bacheca originale
    SELECT B.titolo, B.descrizione
    INTO v_titolo_orig, v_desc_orig
    FROM ToDo T
    JOIN Bacheca B ON T.ID_Bacheca = B.ID_Bacheca
    WHERE T.ID_ToDo = NEW.ID_ToDo;

    --Controllo se esiste una bacheca con lo stesso titolo
    SELECT ID_Bacheca INTO v_esiste
    FROM Bacheca
    WHERE ID_Utente = NEW.ID_Utente AND titolo = v_titolo_orig
    LIMIT 1;--basta una
    -- Se NON esiste nessuna bacheca con quel titolo, allora la creo
    IF v_esiste IS NULL THEN
        INSERT INTO Bacheca (titolo, descrizione, ID_Utente)
        VALUES (v_titolo_orig, v_desc_orig, NEW.ID_Utente);
        RAISE NOTICE 'Creata nuova bacheca % per utente %', v_titolo_orig, NEW.ID_Utente;
    END IF;

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

7.2.3. Trigger trg_prevenire_auto_condivisione

```
CREATE OR REPLACE FUNCTION prevenire_auto_condivisione()
RETURNS TRIGGER AS $$
DECLARE
    v_id_autore INT;
BEGIN
    --trovo chi ha creato il todo
    SELECT ID_Utente INTO v_id_autore
    FROM ToDo
    WHERE ID_ToDo = NEW.ID_ToDo;

    --se chi riceve è lo stesso che ha creato allora errore
    IF NEW.ID_Utente = v_id_autore THEN
        RAISE EXCEPTION 'Non puoi condividere un ToDo con te stesso (sei già l''autore).';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_prevenire_auto_condivisione
BEFORE INSERT ON Condivisione
FOR EACH ROW
EXECUTE FUNCTION prevenire_auto_condivisione();
```

7.2.4. Trigger trg_set_posizione_todo

```
CREATE OR REPLACE FUNCTION return_max_posizione()
RETURNS TRIGGER AS $$
DECLARE
    max_pos INT;
BEGIN
    --calcolo il massimo numero di posizione ed aggiungo 1
    SELECT MAX(Posizione) INTO max_pos
    FROM ToDo
    WHERE ID_Bacheca = NEW.ID_Bacheca;

    NEW.Posizione := COALESCE(max_pos, 0) + 1;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_set_posizione_todo
BEFORE INSERT OR UPDATE OF ID_Bacheca ON ToDo
FOR EACH ROW
EXECUTE FUNCTION return_max_posizione();
```

7.2.5. Trigger trg_riordina_dopo_delete

```
--
CREATE OR REPLACE FUNCTION riordina_posizioni_delete()
RETURNS TRIGGER AS $$
BEGIN
    -- prende tutti i todo che stavano dopo quello cancellato
    -- e scala la loro posizione di -1
    UPDATE ToDo
    SET posizione = posizione - 1
    WHERE ID_Bacheca = OLD.ID_Bacheca
      AND posizione > OLD.posizione;

    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_riordina_dopo_delete
AFTER DELETE ON ToDo
FOR EACH ROW
EXECUTE FUNCTION riordina_posizioni_delete();
```

7.2.6. Vista Vista_Bacheca_Unificata

```sql
-- Questa vista unifica i ToDo creati dall'utente e quelli condivisi con esso.
CREATE VIEW Vista_Bacheca_Unificata AS
-- ToDo propri
SELECT
    T.ID_ToDo,
    T.titolo AS Nome_ToDo,
    T.descrizione,
    T.dataScadenza,
    T.stato,
    T.colore,
    T.immagine,
    T.url,
    T.posizione,
    T.ID_Utente AS ID_Autore,
    B.ID_Utente AS ID_Utente_Visualizzatore,
    B.ID_Bacheca AS ID_Bacheca_Visualizzazione,
    B.titolo AS Titolo_Bacheca,
    B.descrizione AS Descrizione_Bacheca
FROM ToDo T
JOIN Bacheca B ON T.ID_Bacheca = B.ID_Bacheca

UNION ALL

--ToDo condivisi
SELECT
    T.ID_ToDo,
    T.titolo AS Nome_ToDo,
    T.descrizione,
    T.dataScadenza,
    T.stato,
    T.colore,
    T.immagine,
    T.url,
    T.posizione,
    T.ID_Utente AS ID_Autore,
    C.ID_Utente AS ID_Utente_Visualizzatore,
    B_Clone.ID_Bacheca AS ID_Bacheca_Visualizzazione,
    B_Clone.titolo AS Categoria_Bacheca,
    B_Clone.descrizione AS Descrizione_Bacheca
FROM Condivisione C
JOIN ToDo T ON C.ID_ToDo = T.ID_ToDo
JOIN Bacheca B_Originale ON T.ID_Bacheca = B_Originale.ID_Bacheca
JOIN Bacheca B_Clone ON
    B_Clone.ID_Utente = C.ID_Utente
    AND B_Clone.titolo = B_Originale.titolo
```

8. Funzioni

In questo capitolo verranno implementati e spiegati le funzioni per la gestione e l'utilizzo ottimale del database.

8.1. Dizionario delle funzioni

Nome	***Descrizione***
registra_utente	La funzione consente la registrazione di un nuovo utente nel sistema. Richiede l'inserimento del login e password, verificando che non siano già presenti nel database (vincolo di unicità). Restituisce l'ID del nuovo utente in caso di successo.
login_utente	La funzione permette di restituire l'ID dell'utente nel caso che l'utente sia già presente nella tabella, mentre restituisce -1 nel caso che le credenziali siano errate.
crea_bacheca_utente	Consente a un utente di creare una nuova bacheca specificando titolo e descrizione. La funzione verifica che l'utente non possieda già una bacheca con lo stesso titolo e la stessa descrizione.
modifica_bacheca_utente	Permette all'utente di aggiornare i dettagli di una bacheca esistente, in particolare la descrizione o il titolo, garantendo che le modifiche rispettino i vincoli di integrità del sistema.
elimina_bacheca_utente	Rimuove definitivamente una bacheca specificata dall'utente. Grazie ai vincoli di integrità referenziale (ON DELETE CASCADE), l'eliminazione comporta la rimozione automatica di tutti i ToDo contenuti in quella bacheca.
visualizza_bacheche_utente	Questa funzione permette di cercare e visualizzare le bacheche di un utente.
crea_todo_utente	Inserisce un nuovo ToDo all'interno di una specifica bacheca. La funzione assegna automaticamente la posizione corretta al ToDo all'interno della lista e

	associa i dati inseriti (titolo, scadenza, ecc..) all'utente creatore.
modifica_todo_utente	Consente di aggiornare le informazioni di un ToDo esistente, come il suo id, la descrizione, la data di scadenza, il colore o lo stato di completamento, salvando le modifiche nel database.
elimina_todo_utente	Cancella un ToDo dal sistema. L'operazione rimuove anche le eventuali checklist, attività e condivisioni associate a quel ToDo, mantenendo pulito il database.
sposta_todo_posizione	La funzione che cambia la posizione di un ToDo nella stessa bacheca, inoltre gestisce lo spostamento di tutti gli altri ToDo per fare spazio.
cambia_bacheca_todo	La funzione serve a cambiare la bacheca di riferimento di un ToDo, l'utente passerà come parametri id del ToDo, il suo login, il titolo e la descrizione della bacheca dove vuole spostarlo.
visualizza_todo_ordinati	La funzione serve a cambiare l'ordine di visualizzazione dei ToDo in una bacheca, l'utente può scegliere se ordinarli per Posizione, Titolo, DataScadenza o StatoCompletamento. L'utente passerà come parametri l'ID della bacheca, e il criterio di ordinamento.
filtra_todo_bacheca	Questa funzione serve per cercare i ToDo in base a tre criteri: per titolo, per data di scadenza, e quelli in scadenza nella data odierna. Se nesssun filtro viene applicato mostra tutti i ToDo in quella bacheca.
aggiungi_attivita_checklist	Questa funzione permette di aggiungere attività alla checklist
visualizza_checklist_todo	Questa funzione serve a recuperare la lista delle attività salvate , quindi la checklist.
modifica_stato_attivita	La funzione serve a modificare lo stato di un'attività di un ToDo, inoltre controlla se tutte sono aggiornate, per aggiornare

	anche lo stato del ToDo. Il trigger trg_aggiorna_stato_todo controlla e aggiorna.
modifica_stato_todo	La funzione permette di modificare lo stato di un ToDo, ignorando nel caso la Checklist. Come parametri ho id del ToDo, possessore e statoToDo.
elimina_attivita	Questa funzione permette di eliminare le attività dalla checklist
aggiungi_condivisione_todo	La funzione serve a condividere un ToDo tra l'utente possessore e un altro utente. Si passa come parametri l'id del ToDo, il possessore e il destinatario.
modifica_condivisione_todo	La funzione serve a modificare la condivisione di un ToDo da un vecchio utente ad uno nuovo, come parametri ha gli stessi della creazione, ma con uno nuovo, ovvero il nuovo destinatario.
elimina_condivisione_todo	La funzione serve a cancellare la condivisione di un ToDo con un utente. Come parametri ha id del ToDo, il possessore, e l'utente con cui togliere la condivisione.
visualizza_condivisioni_todo	Restituisce l'elenco degli utenti (tramite login) con cui un determinato ToDo è stato condiviso. È utile all'autore per monitorare chi ha accesso all'attività.

8.2. Implementazione delle Funzioni

8.2.1. Registra_utente

```
-- Crea l'utente e mi ridà l'ID.
CREATE OR REPLACE FUNCTION registra_utente(
    p_login VARCHAR (30),
    p_password VARCHAR (30)
)
RETURNS INT AS $$
DECLARE
    v_id_utente INT;
BEGIN
    INSERT INTO Utente (login, password)
    VALUES (p_login, p_password)
    RETURNING ID_Utente INTO v_id_utente;

    RETURN v_id_utente;
EXCEPTION
    WHEN unique_violation THEN
        RETURN -1; -- Ritorna -1 se il login esiste già
END;
$$ LANGUAGE plpgsql;
```

8.2.2 Login_utente

```
--Controlla se login e passwword coincidono e mi restituisce l'id
CREATE OR REPLACE FUNCTION login_utente(
    p_login VARCHAR(30),
    p_password VARCHAR(30)
)
RETURNS INT AS $$
DECLARE
    v_id_utente INT;
BEGIN
    SELECT ID_Utente INTO v_id_utente
    FROM Utente
    WHERE login = p_login AND password = p_password;

    IF FOUND THEN
        RETURN v_id_utente;--corrispondono
    ELSE
        RETURN -1;--fallito
    END IF;
END;
$$ LANGUAGE plpgsql;
```

8.2.3 Crea_bacheca_utente

```
-- Creazione bacheca.
CREATE OR REPLACE FUNCTION crea_bacheca_utente(
    p_titolo VARCHAR (30),
    p_descrizione VARCHAR (100),
    p_id_utente INT
)
RETURNS INT AS $$
DECLARE
    v_esiste INT;
    v_nuovo_id INT;
BEGIN
    -- Verifica se esiste già una bacheca con lo stesso titolo per questo utente
    SELECT COUNT (*) INTO v_esiste
    FROM Bacheca
    WHERE ID_Utente = p_id_utente AND titolo = p_titolo AND descrizione = p_descrizione;

    IF v_esiste > 0 THEN
        RETURN -1; -- Errore: Bacheca duplicata
    END IF;
    --creo la bacheca
    INSERT INTO Bacheca (titolo, descrizione, ID_Utente)
    VALUES(p_titolo, p_descrizione, p_id_utente)
    RETURNING ID_Bacheca INTO v_nuovo_id;

    RETURN v_nuovo_id;
EXCEPTION
    WHEN check_violation THEN
        RAISE NOTICE 'Errore: Il titolo deve essere Universita, Lavoro o TempoLibero';
        RETURN -2;
END;
$$ LANGUAGE plpgsql;
```

8.2.4 Modifica_bacheca_utente

```
--Modifica della bacheca: nome, descrizione
CREATE OR REPLACE FUNCTION modifica_bacheca_utente(
    p_id_bacheca INT,
    p_nuovo_titolo VARCHAR(30),
    p_nuova_descrizione VARCHAR(100)
)
RETURNS BOOLEAN AS $$
DECLARE
    v_id_utente INT;
    v_count INT;
BEGIN
    -- recupera l'utente proprietario della bacheca
    SELECT ID_Utente INTO v_id_utente
    FROM Bacheca
    WHERE ID_Bacheca = p_id_bacheca;

    IF v_id_utente IS NULL THEN
        RETURN FALSE; -- Bacheca non trovata
    END IF;

    -- Verifica che non esistano altre bacheche dello stesso utente col nuovo nome
    SELECT COUNT(*) INTO v_count
    FROM Bacheca
    WHERE ID_Utente = v_id_utente
        AND titolo = p_nuovo_titolo
        AND descrizione = p_descrizione
        AND ID_Bacheca <> p_id_bacheca;

    IF v_count > 0 THEN
        RETURN FALSE; -- Nome e descrizione già in uso
    END IF;
    --modifica
    UPDATE Bacheca
    SET titolo = p_nuovo_titolo,
        descrizione = p_nuova_descrizione
    WHERE ID_Bacheca = p_id_bacheca;

    RETURN TRUE;

EXCEPTION
    WHEN check_violation THEN-- deve essere tra quelli di (Universita, Lavoro,TempoLibero)
        RAISE NOTICE 'Errore: Titolo non valido.';
        RETURN FALSE;
END;
$$ LANGUAGE plpgsql;
```

8.2.5 Elimina_bacheca_utente

```
--Elimino la bacheca (e a cascata i ToDo)
CREATE OR REPLACE FUNCTION elimina_bacheca_utente(
    p_id_bacheca INT
)
RETURNS BOOLEAN AS $$
BEGIN
    DELETE FROM Bacheca
    WHERE ID_Bacheca = p_id_bacheca;

    IF FOUND THEN
        RETURN TRUE;
    ELSE
        RETURN FALSE;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

8.2.6 Visualizza_bacheca_utente

```
--Elenco bacheche di un utente
CREATE OR REPLACE FUNCTION visualizza_bacheche_utente(
    p_id_utente INT
)
RETURNS TABLE (
    ID_Bacheca INT,
    Titolo VARCHAR,
    Descrizione VARCHAR
) AS $$
BEGIN
    RETURN QUERY
    SELECT
        B.ID_Bacheca,
        B.titolo,
        B.descrizione
    FROM Bacheca B
    WHERE B.ID_Utente = p_id_utente
    ORDER BY B.ID_Bacheca ASC;
END;
$$ LANGUAGE plpgsql;
```

8.2.7 Crea_todo_utente

```
--Creazione di un ToDo
CREATE OR REPLACE FUNCTION crea_todo_utente(
    p_titolo VARCHAR(30),
    p_descrizione VARCHAR(1000),
    p_data_scadenza DATE,
    p_colore VARCHAR(30),
    p_immagine VARCHAR(100),
    p_url VARCHAR(500),
    p_id_utente INT,
    p_id_bacheca INT
)
RETURNS INT AS $$
DECLARE
    v_new_id INT;
BEGIN
    INSERT INTO ToDo (
        titolo,
        descrizione,
        dataScadenza,
        colore,
        immagine,
        url,
        ID_Utente,
        ID_Bacheca,
        posizione -- posizione calcolata dal trigger
    )
    VALUES (
        p_titolo,
        p_descrizione,
        p_data_scadenza,
        p_colore,
        p_immagine,
        p_url,
        p_id_utente,
        p_id_bacheca,
        0 --il trigger lo sovrascriverà
    )
    RETURNING ID_ToDo INTO v_new_id;
    RETURN v_new_id;

EXCEPTION
    WHEN check_violation THEN
        RAISE NOTICE 'Errore: Colore non valido o vincolo violato.';
        RETURN -1;
    WHEN foreign_key_violation THEN
        RAISE NOTICE 'Errore: Bacheca o Utente non esistenti.';
        RETURN -2;
END;
$$ LANGUAGE plpgsql;
```

8.2.8 Modifica_todo_utente

```
--Modifica i dati principali di un ToDo
CREATE OR REPLACE FUNCTION modifica_todo_utente(
    p_id_todo INT,
    p_titolo VARCHAR(30),
    p_descrizione VARCHAR(1000),
    p_data_scadenza DATE,
    p_colore VARCHAR(30),
    p_immagine VARCHAR(100),
    p_url VARCHAR(500),
    p_stato VARCHAR(30)
)
RETURNS BOOLEAN AS $$
BEGIN
    UPDATE ToDo
    SET titolo = p_titolo,
        descrizione = p_descrizione,
        dataScadenza = p_data_scadenza,
        colore = p_colore,
        immagine = p_immagine,
        url = p_url,
        stato = p_stato
    WHERE ID_ToDo = p_id_todo;

    IF FOUND THEN
        RETURN TRUE;
    ELSE
        RETURN FALSE;
    END IF;

EXCEPTION
    WHEN check_violation THEN
        RAISE NOTICE 'Errore: Colore o Stato non validi.';
        RETURN FALSE;
END;
$$ LANGUAGE plpgsql;
```

8.2.9 Elimina_todo_utente

```
--Cancello un ToDo( e in cascata la sua checklist e condivisioni)
CREATE OR REPLACE FUNCTION elimina_todo_utente(
    p_id_todo INT
)
RETURNS BOOLEAN AS $$
BEGIN
    DELETE FROM ToDo
    WHERE ID_ToDo = p_id_todo;

    IF FOUND THEN
        RETURN TRUE;
    ELSE
        RETURN FALSE;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

8.2.10 Sposta_todo_posizione

```
-- Gestisce lo spostamento di tutti gli altri ToDo per fare spazio.
CREATE OR REPLACE FUNCTION sposta_todo_posizione(
    p_id_todo INT,
    p_nuova_posizione INT
)
RETURNS VOID AS $$
DECLARE
    v_id_bacheca INT;
    v_vecchia_posizione INT;
BEGIN
    -- Recupera posizione attuale e bacheca
    SELECT ID_Bacheca, Posizione
    INTO v_id_bacheca, v_vecchia_posizione
    FROM ToDo
    WHERE ID_ToDo = p_id_todo;

    -- Controlla se il todo esiste
    IF NOT FOUND THEN
        RAISE EXCEPTION 'ToDo con ID % non trovato.', p_id_todo;
    END IF;

    -- Se la posizione non cambia, esci
    IF v_vecchia_posizione = p_nuova_posizione THEN
        RETURN;
    END IF;

    -- Sposta gli altri ToDo per fare spazio
    IF v_vecchia_posizione < p_nuova_posizione THEN
        -- Spostamento verso il BASSO (es. da 2 a 5)
        UPDATE ToDo
        SET Posizione = Posizione - 1
        WHERE ID_Bacheca = v_id_bacheca
          AND Posizione > v_vecchia_posizione
          AND Posizione <= p_nuova_posizione;
    ELSE
        -- Spostamento verso l'ALTO (es. da 5 a 2)
        UPDATE ToDo
        SET Posizione = Posizione + 1
        WHERE ID_Bacheca = v_id_bacheca
          AND Posizione >= p_nuova_posizione
          AND Posizione < v_vecchia_posizione;
    END IF;

    -- Aggiorna il ToDo target alla nuova posizione
    UPDATE ToDo
    SET Posizione = p_nuova_posizione
    WHERE ID_ToDo = p_id_todo;

    RAISE NOTICE 'ToDo spostato dalla posizione % alla %.', v_vecchia_posizione, p_nuova_posizione;
END;
$$ LANGUAGE plpgsql;
```

8.2.11 Visualizza_todo_ordinati

```
--Funzione per ordinare i todo in base a vari criteri
CREATE OR REPLACE FUNCTION visualizza_todo_ordinati(
    p_id_bacheca INT,
    p_criterio VARCHAR -- 'POSIZIONE', 'TITOLO', 'DATA', 'STATO'
)
RETURNS TABLE (
    ID_ToDo INT,
    Titolo VARCHAR,
    Descrizione VARCHAR,
    DataScadenza DATE,
    Stato VARCHAR,
    Colore VARCHAR,
    Immagine VARCHAR,
    Url VARCHAR,
    Posizione INT
) AS $$
BEGIN
    RETURN QUERY
    SELECT
        v.ID_ToDo,
        v.Nome_ToDo,
        v.Descrizione,
        v.DataScadenza,
        v.Stato,
        v.Colore,
        v.Immagine,
        v.Url,
        v.Posizione
    FROM Vista_Bacheca_Unificata v -- comprendo anche i condivisi
    WHERE v.ID_Bacheca_Visualizzazione = p_id_bacheca
    ORDER BY
        CASE WHEN p_criterio = 'POSIZIONE' THEN v.Posizione END ASC,
        CASE WHEN p_criterio = 'TITOLO' THEN v.Nome_ToDo END ASC,
        CASE WHEN p_criterio = 'DATA' THEN v.DataScadenza END ASC,
        CASE WHEN p_criterio = 'STATO' THEN
            CASE v.Stato
                WHEN 'Completato' THEN 1
                WHEN 'NonCompletato' THEN 2
                ELSE 3
            END
        END ASC,
        v.Posizione ASC;
END;
$$ LANGUAGE plpgsql;
```

8.2.12 Cambia_bacheca_todo

```
--Sposto un todo in un altra bacheca
CREATE OR REPLACE FUNCTION cambia_bacheca_todo(
    p_id_todo INT,
    p_login_utente VARCHAR(30),
    p_titolo_nuova_bacheca VARCHAR(30),
    p_descrizione_nuova_bacheca VARCHAR(100)
)
RETURNS VOID AS $$
DECLARE
    v_id_utente INT;
    v_id_nuova_bacheca INT;
    v_id_vecchia_bacheca INT;
    v_vecchia_posizione INT;
BEGIN
    --verifica esistenza utente
    SELECT ID_Utente INTO v_id_utente
    FROM Utente
    WHERE login = p_login_utente;
    IF v_id_utente IS NULL THEN
        RAISE EXCEPTION 'Utente "%" non trovato.', p_login_utente;
    END IF;

    --verfica esistenza Todo
    SELECT ID_Bacheca, Posizione INTO v_id_vecchia_bacheca, v_vecchia_posizione
    FROM ToDo
    WHERE ID_ToDo = p_id_todo AND ID_Utente = v_id_utente;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'ToDo con ID % non trovato o non appartente all''utente %.', p_id_todo, p_login_utente;
    END IF;

    --verifica esistenza Bacheca
    SELECT ID_Bacheca INTO v_id_nuova_bacheca
    FROM Bacheca
    WHERE titolo = p_titolo_nuova_bacheca
      AND descrizione = p_descrizione_nuova_bacheca
      AND ID_Utente = v_id_utente;

    IF v_id_nuova_bacheca IS NULL THEN
        RAISE EXCEPTION 'Nessuna bacheca trovata con titolo "%" e descrizione "%" per l''utente %.',
        p_titolo_nuova_bacheca, p_descrizione_nuova_bacheca, p_login_utente;
    END IF;

    IF v_id_vecchia_bacheca = v_id_nuova_bacheca THEN
        RETURN;
    END IF;

    --Sposto il todo in una nuova bacheca
    UPDATE ToDo
    SET ID_Bacheca = v_id_nuova_bacheca
    WHERE ID_ToDo = p_id_todo;

    --tappo il buco di posizione lasciato nella vecchia bacheca
    UPDATE ToDo
    SET Posizione = Posizione - 1
    WHERE ID_Bacheca = v_id_vecchia_bacheca
      AND Posizione > v_vecchia_posizione;
END;
$$ LANGUAGE plpgsql;
```

8.2.13 Filtra_todo_bacheca

```
--funzione per filtrare e ricercare i todo all'interno della bacheca
CREATE OR REPLACE FUNCTION filtra_todo_bacheca(
    p_id_bacheca INT,
    p_criterio VARCHAR,-- 'TUTTO', 'TITOLO', 'OGGI', 'ENTRO'
    p_filtro_testo VARCHAR DEFAULT NULL,
    p_filtro_data DATE DEFAULT NULL
)
RETURNS TABLE (
    ID_ToDo INT,
    Titolo VARCHAR,
    Descrizione VARCHAR,
    DataScadenza DATE,
    Stato VARCHAR,
    Colore VARCHAR,
    Immagine VARCHAR,
    Url VARCHAR,
    Posizione INT
) AS $$
BEGIN
    RETURN QUERY
    SELECT
        v.ID_ToDo,
        v.Titolo,
        v.Descrizione,
        v.DataScadenza,
        v.Stato,
        v.Colore,
        v.Immagine,
        v.Url,
        v.Posizione
    FROM Vista_Bacheca_Unificata v
    WHERE v.ID_Bacheca_Visualizzazione = p_id_bacheca
    AND (
        (p_criterio = 'TUTTO')-- nessun criterio, mostro tutto
        OR
        (p_criterio = 'TITOLO' AND (--ricerca qualsiasi parola nel titolo o nella descrizione
            v.Titolo ILIKE '%' || COALESCE(p_filtro_testo, '') || '%' OR
            v.Descrizione ILIKE '%' || COALESCE(p_filtro_testo, '') || '%'
        ))
        OR
        (p_criterio = 'OGGI' AND v.DataScadenza = CURRENT_DATE)
        OR
        (p_criterio = 'ENTRO' AND v.DataScadenza <= p_filtro_data)
    )
    ORDER BY v.Posizione ASC;
END;
$$ LANGUAGE plpgsql;
```

8.2.14 Aggiungi_attivita_checklist

```
--aggiunge un'attività alla checklist
CREATE OR REPLACE FUNCTION aggiungi_attivita_checklist(
    p_id_todo INT,
    p_nome_attivita VARCHAR
)
RETURNS INT AS $$
DECLARE
    v_id_checklist INT;
    v_id_attivita INT;
BEGIN
    -- cerca l'ID della Checklist associata a questo ToDo
    SELECT ID_Checklist INTO v_id_checklist
    FROM Checklist
    WHERE ID_ToDo = p_id_todo;

    -- se non esiste (è la prima attività da inserire), crea la riga in checklist
    IF v_id_checklist IS NULL THEN
        INSERT INTO Checklist (ID_ToDo)
        VALUES (p_id_todo)
        RETURNING ID_Checklist INTO v_id_checklist;
    END IF;

    -- inserisce l'attività e restituisce l'ID
    INSERT INTO Attivita (nome, ID_Checklist, stato)
    VALUES (p_nome_attivita, v_id_checklist, 'NonCompletato')
    RETURNING ID_Attivita INTO v_id_attivita;

    RETURN v_id_attivita;
END;
$$ LANGUAGE plpgsql;
```

8.2.15 Modifica_stato_attivita

```
-- Cambio stato di una voce checklist (Completato/NonCompletato).
CREATE OR REPLACE FUNCTION modifica_stato_attivita(
    p_id_attivita INT,
    p_nuovo_stato VARCHAR
)
RETURNS VOID AS $$
BEGIN
    UPDATE Attivita
    SET stato = p_nuovo_stato
    WHERE ID_Attivita = p_id_attivita;
END;
$$ LANGUAGE plpgsql;

-- Rimuove un'attività dalla checklist.
CREATE OR REPLACE FUNCTION elimina_attivita(
    p_id_attivita INT
)
RETURNS VOID AS $$
BEGIN
    DELETE FROM Attivita
    WHERE ID_Attivita = p_id_attivita;
END;
$$ LANGUAGE plpgsql;
```

8.2.16 Elimina_Attivita

```
-- Rimuove un'attività dalla checklist.
CREATE OR REPLACE FUNCTION elimina_attivita(
    p_id_attivita INT
)
RETURNS VOID AS $$
BEGIN
    DELETE FROM Attivita
    WHERE ID_Attivita = p_id_attivita;
END;
$$ LANGUAGE plpgsql;
```

8.2.17 Modifica_stato_todo

```
--Funzione che forza lo stato del todo a mano senza passare per la checklist
CREATE OR REPLACE FUNCTION modifica_stato_todo(
    p_id_todo INT,
    p_login_utente VARCHAR(30),
    p_nuovo_stato VARCHAR(30)
)
RETURNS VOID AS $$
DECLARE
    v_id_utente INT;
BEGIN
    -- Controllo se lo stato è accettato
    IF p_nuovo_stato NOT IN ('Completato', 'NonCompletato') THEN
        RAISE EXCEPTION 'Stato non valido. I valori ammessi sono "Completato" o "NonCompletato".';
    END IF;
''
    -- Recupero ID Utente
    SELECT ID_Utente INTO v_id_utente
    FROM Utente
    WHERE login = p_login_utente;

    -- Verifica proprietà e aggiorno stato del ToDo
    UPDATE ToDo
    SET stato = p_nuovo_stato
    WHERE ID_ToDo = p_id_todo AND ID_Utente = v_id_utente;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'ToDo non trovato o non appartenente all''utente.';
    ELSE
        RAISE NOTICE 'ToDo segnato come %.', p_nuovo_stato;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

8.2.18 Visualizza_checklist_todo

```
--Funzione che serve a visualizzare la checklist relativa ad un todo
CREATE OR REPLACE FUNCTION visualizza_checklist_todo(
    p_id_todo INT
)
RETURNS TABLE (
    ID_Attivita INT,
    Nome VARCHAR,
    Stato VARCHAR
) AS $$
BEGIN
    RETURN QUERY
    SELECT A.ID_Attivita, A.nome, A.stato
    FROM Attivita A
    JOIN Checklist C ON A.ID_Checklist = C.ID_Checklist
    WHERE C.ID_ToDo = p_id_todo
    ORDER BY A.ID_Attivita ASC;
END;
$$ LANGUAGE plpgsql;
```

8.2.19 Aggiungi_condivisione_todo

```
--Funzione che permette adun autore di condividere il proprio todo
CREATE OR REPLACE FUNCTION aggiungi_condivisione_todo(
    p_id_todo INT,
    p_login_autore VARCHAR(30),
    p_login_destinatario VARCHAR(30)
)
RETURNS VOID AS $$
DECLARE
    v_id_autore INT;
    v_id_destinatario INT;
BEGIN
    -- recupero ID autore e destinatario
    SELECT ID_Utente INTO v_id_autore FROM Utente WHERE login = p_login_autore;
    SELECT ID_Utente INTO v_id_destinatario FROM Utente WHERE login = p_login_destinatario;

    IF v_id_autore IS NULL OR v_id_destinatario IS NULL THEN
        RAISE EXCEPTION 'Utenti non trovati.';
    END IF;

    IF v_id_autore = v_id_destinatario THEN
        RAISE EXCEPTION 'Non puoi condividere con te stesso.';
    END IF;

    -- controllo che il ToDo sia dell'autore
    PERFORM 1 FROM ToDo WHERE ID_ToDo = p_id_todo AND ID_Utente = v_id_autore;
    IF NOT FOUND THEN
        RAISE EXCEPTION 'ToDo non trovato o non tuo.';
    END IF;

    -- controllo se già condiviso(non tengo conto dei dati trovati)
    PERFORM 1 FROM Condivisione WHERE ID_Utente = v_id_destinatario AND ID_ToDo = p_id_todo;
    IF FOUND THEN
        RAISE NOTICE 'Già condiviso.';
        RETURN;
    END IF;

    -- inserimento (farà scattare il trigger che crea la bacheca copia)
    INSERT INTO Condivisione (ID_Utente, ID_ToDo)
    VALUES (v_id_destinatario, p_id_todo);
END;
$$ LANGUAGE plpgsql;
```

8.2.20 Modifica_condivisione_todo

```
-- Sposto la condivisione da un utente a un altro.
    CREATE OR REPLACE FUNCTION modifica_condivisione_todo(
    p_id_todo INT,
    p_login_autore VARCHAR(30),
    p_login_vecchio_destinatario VARCHAR(30),
    p_login_nuovo_destinatario VARCHAR(30)
)
RETURNS VOID AS $$
DECLARE
    v_id_autore INT;
    v_id_vecchio_dest INT;
    v_id_nuovo_dest INT;
BEGIN
    -- recupero ID autore, vecchio destinatario e nuovo
    SELECT ID_Utente INTO v_id_autore FROM Utente WHERE login = p_login_autore;
    SELECT ID_Utente INTO v_id_vecchio_dest FROM Utente WHERE login = p_login_vecchio_destinatario;
    SELECT ID_Utente INTO v_id_nuovo_dest FROM Utente WHERE login = p_login_nuovo_destinatario;

    IF v_id_autore IS NULL OR v_id_vecchio_dest IS NULL OR v_id_nuovo_dest IS NULL THEN
        RAISE EXCEPTION 'Utenti non trovati.';
    END IF;

    -- controllo delle proprietà
    PERFORM 1 FROM ToDo WHERE ID_ToDo = p_id_todo AND ID_Utente = v_id_autore;
    IF NOT FOUND THEN
        RAISE EXCEPTION 'ToDo non trovato.';
    END IF;

    -- controllo vecchia condivisione
    PERFORM 1 FROM Condivisione WHERE ID_Utente = v_id_vecchio_dest AND ID_ToDo = p_id_todo;
    IF NOT FOUND THEN
        RAISE EXCEPTION 'Non era condiviso con %.', p_login_vecchio_destinatario;
    END IF;

    -- scambio dei destinatari
    DELETE FROM Condivisione WHERE ID_Utente = v_id_vecchio_dest AND ID_ToDo = p_id_todo;

    -- inserisco il nuovo (Il trigger creerà la bacheca clonata per il nuovo utente)
    INSERT INTO Condivisione (ID_Utente, ID_ToDo) VALUES (v_id_nuovo_dest, p_id_todo);

    RAISE NOTICE 'Condivisione spostata.';
END;
$$ LANGUAGE plpgsql;
```

8.2.21 Elimina_condivisione_todo

```
--Funzione che elimina una condivisione
CREATE OR REPLACE FUNCTION elimina_condivisione_todo(
    p_id_todo INT,
    p_login_autore VARCHAR(30),
    p_login_destinatario VARCHAR(30)
)
RETURNS VOID AS $$
DECLARE
    v_id_autore INT;
    v_id_destinatario INT;
BEGIN
    SELECT ID_Utente INTO v_id_autore
    FROM Utente
    WHERE login = p_login_autore;

    SELECT ID_Utente INTO v_id_destinatario
    FROM Utente
    WHERE login = p_login_destinatario;

    IF v_id_autore IS NULL OR v_id_destinatario IS NULL THEN
        RAISE EXCEPTION 'Utente non trovato.';
    END IF;

    PERFORM 1 FROM ToDo WHERE ID_ToDo = p_id_todo AND ID_Utente = v_id_autore;
    IF NOT FOUND THEN
        RAISE EXCEPTION 'ToDo non trovato o non di proprietà dell''autore.';
    END IF;

    DELETE FROM Condivisione
    WHERE ID_Utente = v_id_destinatario AND ID_ToDo = p_id_todo;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Nessuna condivisione trovata.';
    ELSE
        RAISE NOTICE 'Condivisione rimossa.';
    END IF;
END;
$$ LANGUAGE plpgsql;
```

8.2.22 Visualizza_condivisioni_todo

```
--Funzione che restituisce il login degli utenti con i quali
--è stato condviso il todo
CREATE OR REPLACE FUNCTION visualizza_condivisioni_todo(
    p_id_todo INT
)
RETURNS TABLE (
    login_utente VARCHAR
) AS $$
BEGIN
    RETURN QUERY
    SELECT U.login
    FROM Condivisione C
    JOIN Utente U ON C.ID_Utente = U.ID_Utente
    WHERE C.ID_ToDo = p_id_todo;
END;
$$ LANGUAGE plpgsql;
```