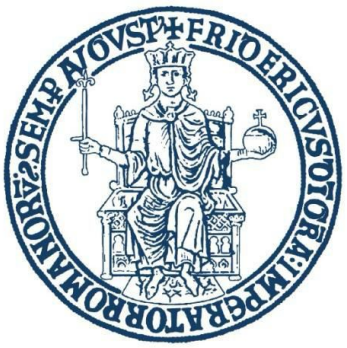


UNIVERSITÀ DEGLI STUDI
DI NAPOLI FEDERICO II

Documentazione Basi di Dati

Lavoro svolto da:

Guido Christian Petrone

Matricola: N86005370

Giulio Pacella

Matricola: N86005257

Anno Accademico 2024/25

Indice

1 Introduzione

Il seguente elaborato ha lo scopo di documentare la progettazione e lo sviluppo di una base di dati relazionale del DBMS PostgreSQL, ad opera degli studenti Petrone Guido Christian e Pacella Giulio del CdL in Informatica presso l'Università degli Studi di Napoli "Federico II". Il database nasce come progetto a scopi valutativi per il corso di Basi di Dati, ed implementa un sistema di gestione delle attività personali da svolgere.

1.1 Descrizione del Problema

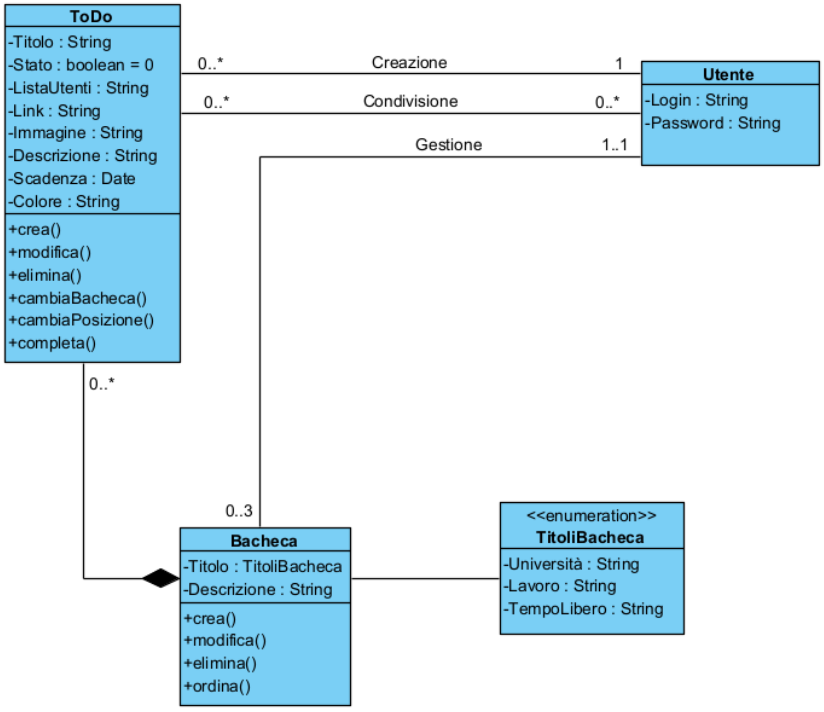
All'interno dell'elaborato verranno riportate progettazione e sviluppo di una base di dati relazionale, che implementi un sistema informativo che permetta di tenere traccia delle attività personali da svolgere, chiamate "ToDo".

Tale sistema consente a ciascun utente di organizzare e gestire in modo efficiente i propri ToDo, i quali sono suddivisi in tre bacheche ognuna con un proprio titolo, ovvero Università, Lavoro e Tempo Libero, e possono essere condivisi con altri utenti.
Inoltre, il sistema permette all'utente di ottenere l'elenco dei ToDo in scadenza nella data odierna o in un certo giorno che egli specifica, e la ricerca per nome o titolo dei ToDo.

2 Progettazione Concettuale

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

2.1 Class Diagram



2.2 Ristrutturazione del Class Diagram

La Ristrutturazione è un processo che punta a migliorare diversi aspetti di un progetto, tra cui l'efficienza e la leggibilità del nostro Class Diagram. Essa si sviluppa secondo i seguenti punti:

- Analisi delle chiavi
- Analisi degli attributi derivati
- Analisi delle ridondanze
- Analisi degli attributi strutturati
- Analisi degli attributi multivalore
- Analisi delle gerarchie di specializzazione

2.2.1 Analisi delle chiavi

Per gestire al meglio le entità che vanno a formare il nostro sistema, sono state adottate delle chiavi primarie per ognuna di esse.
L'entità ToDo utilizza una chiave surrogata, ovvero "id", che risulta essere la scelta più adatta in quanto permette un'identificazione estremamente semplice e rapida.
L'entità Bacheca utilizza anch'essa una chiave surrogata, ovvero "id", poichè risulta essere nuovamente la scelta migliore in quanto i suoi attributi non sono adatti a ricoprire il ruolo di chiave primaria.
Infine, l'entità Utente ha come chiave primaria l'attributo "Login", in quanto è unico per ognuno di essi.

2.2.2 Analisi degli attributi derivati

Gli attributi derivati sono attributi calcolabili a partire da altri presenti nel sistema, e che possono quindi essere rimossi per ottimizzarel'utilizzo delle risorse di calcolo. Analizzando il sistema, possiamo evincere che non siano presenti attributi derivati.
Tutti gli attributi delle classi (ToDo, Utente, Bacheca) sono indipendenti, ovvero non vengono calcolati a partire da altri dati del sistema, ma rappresentano valori autonomi impostati direttamente dall'utente o dal sistema.

2.2.3 Analisi delle ridondanze

Andando ad analizzare il sistema, possiamo notare un'unica ridondanza vera e propria: l'attributo ListaUtenti risulta ridondante poichè l'associazione "Condivisione" tra ToDo e Utente permette già di ottenere la lista degli utenti.
Per risolvere questa ridondanza, è sufficiente rimuovere l'attributo ListaUtenti dalla classe ToDo, facendo affidamento esclusivamente sulla relazione con la classe Utente.

2.2.4 Analisi degli attributi strutturati

La prossima fase consiste nell'analizzare e correggere eventuali attributi strutturati presenti nelle entità, in quanto essi non sono logicamente rappresentabili all'interno di un DBMS, e vanno quindi eliminati e codificati in altro modo.
All'interno del nostro sistema essi non risultano presenti, quindi non è necessario alcun intervento sul nostro schema concettuale.

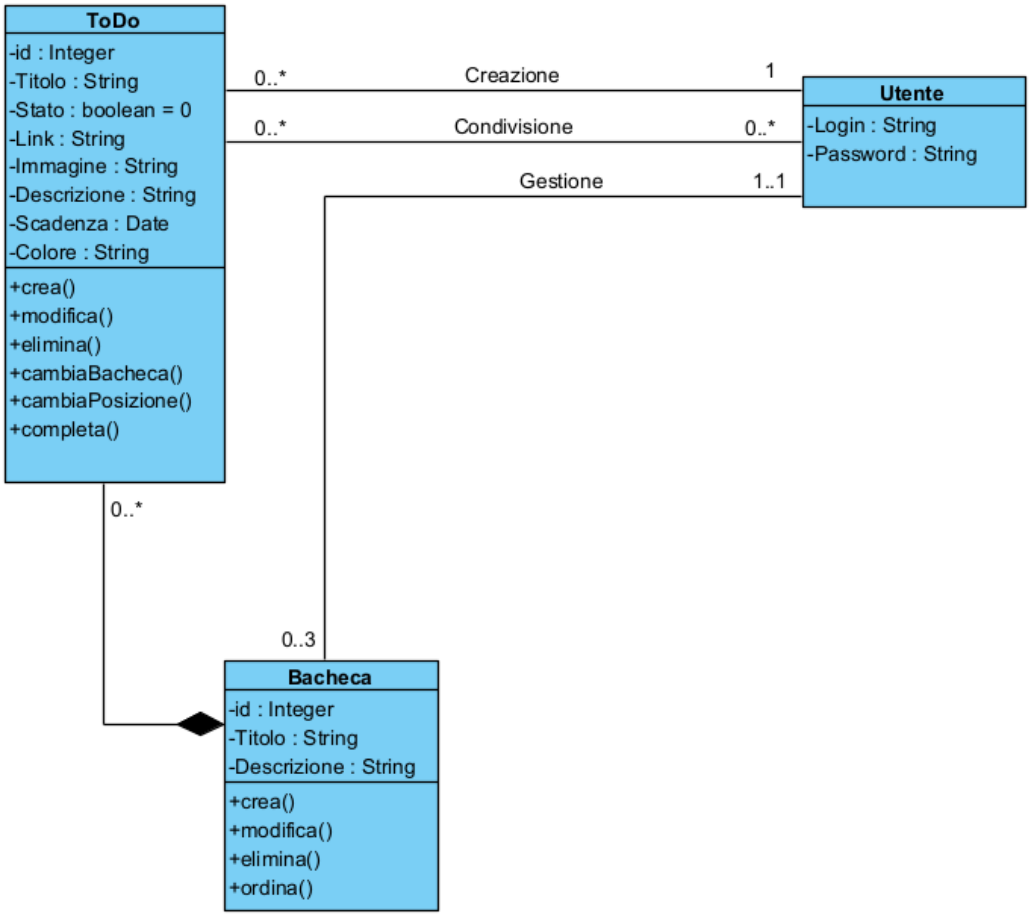
2.2.5 Analisi degli attributi multivalore

Gli attributi a valore multiplo, come quelli strutturati, non sono logicamente rappresentabili e quindi vanno eliminati dal nostro schema concettuale.
L'unico attributo multivalore presente nel sistema era ListaUtenti nella classe ToDo, il quale è stato precedentemente rimosso in quanto ridondante rispetto all'associazione già esistente tra ToDo e Utente

2.2.6 Analisi delle gerarchie di specializzazione

Infine, analizziamo la presenza di eventuali gerarchie di specializzazione, che non possono essere direttamente rappresentate all'interno dei sistemi per la gestione di basi di dati e vanno quindi tradotte utilizzando altri costrutti della modellazione concettuale.
Nel nostro schema non sono presenti gerarchie di generalizzazione, in quanto le classi rappresentano concetti distinti e indipendenti tra loro. Di conseguenza, non è necessario alcun intervento in merito alla generalizzazione.

2.3 Class Diagram Ristrutturato



2.4 Dizionario delle Classi

Classe	Descrizione	Attributi
ToDo	Definisce un'attività personale che un Utente deve svolgere, gestita all'interno di una Bacheca. Può essere condivisa con altri utenti.	• **id**(Integer): Codice identificativo univoco del ToDo (chiave surrogata). • **Titolo**(String): Titolo dell'attività. • **Stato**(boolean): Stato del ToDo: 0 corrisponde a "Non Completato", 1 corrisponde a "Completato". • **Link**(String): Collegamento al ToDo. • **Immagine**(String): Immagine associata al ToDo. • **Descrizione**(String): Descrizione dei dettagli del ToDo. • **Scadenza**(Date): Data in cui scade il ToDo. • **Colore**(String): Codice del colore del ToDo. • **Autore**(String): Chiave esterna che fa riferimento all'attributo Login della classe Utente. • **idB**(String): Chiave esterna che fa riferimento all'attributo id della classe Bacheca.
Bacheca	Definisce uno spazio personale in cui vengono categorizzati e mostrati i ToDo di ogni Utente.	• **id**(Integer): Codice identificativo univoco della Bacheca (chiave surrogata). • **Titolo**(String): Titolo della Bacheca, può essere uno tra Università, Lavoro, Tempo Libero.

		• **Descrizione**(String): descrizione dei dettagli della Bacheca. • **Proprietario**(String): Chiave esterna che fa riferimento all'attributo Login della classe Utente.
Utente	Definisce l'utilizzatore del sistema, che quindi può creare e ricevere ToDo e può creare Bacheche	• **Login**(String): nome che identifica l'utente, univoco. • **Password**(String): codice personale che serve ad accedere al sistema.

2.5 Dizionario delle Associazioni

Relazione	Descrizione
ToDo ↔ **Utente** (Condivisione)	Ogni Utente può ricevere più ToDo, e ogni ToDo può essere condiviso da più utenti. È una Associazione N:N.
ToDo ↔ **Utente** (Creazione)	Ogni Utente può creare più ToDo. È una Associazione 1:N.
ToDo ↔ **Bacheca**	Ogni Bacheca può contenere più ToDo, mentre i ToDo non possono esistere senza far parte di una Bacheca. È una Composizione.
Utente ↔ **Bacheca**	Ogni Utente può gestire un massimo di 3 Bacheche. È una Associazione 1:N.

2.6 Dizionario dei Vincoli

Vincolo	Descrizione
Chiavi Primarie	Le chiavi primarie all'interno del sistema sono: • Utente(login) • Bacheca(id) • ToDo(id) • Condivisione(ricevente, idT)
Chiavi Esterne	Le chiavi esterne all'interno del sistema sono: • Bacheca.Proprietario → Utente.Login • ToDo.Autore → Utente.Login • ToDo.idB → Bacheca.id • Condivisione.idT → ToDo.id • Condivisione.Login → Utente.Login
Vincoli di Unicità	UNIQUE (proprietario, numero) nella tabella bacheca. Indica che ogni utente può avere una sola bacheca per ciascun tipo, dove ogni tipo è rappresentato da un numero (1=Università, 2=Lavoro, 3=Tempo Libero).
Vincoli di Check	• CHECK (numero BETWEEN 1 AND 3) in bacheca: Impedisce di inserire in bacheca un tipo, rappresentato con un numero, che non sia 1,2 o 3. • `CHECK (colore ~ '^#([0-9a-fA-F]{6})$')` in todo: Controlla che il valore nel campo colore sia un colore esadecimale valido

Vincoli applicativi: Vincoli implementati a livello procedurale, tramite trigger o funzioni.	• check_bacheche_standard + trg_check_bacheche: impedisce all'utente di creare due bachece dello stesso tipo e di creare bacheche con un valore "numero" diverso da 1,2 o 3. • crea_bacheche_trigger + crea_bacheche(): garantisce che ogni nuovo utente abbia automaticamente 3 bacheche iniziali. • set_posizione_todo() + trg_set_posizione_todo: garatisce che se non viene specificata manualmente la posizione di un nuovo ToDo, questa venga assegnata come l'ultima disponibile nella bacheca. • sposta_todo_tra_bacheche(): impesce lo spostamento se l'utente non è autorizzato e gestisce l'ordine dei ToDo nei vari spostamenti tra bacheche. • condividi_todo(): garantisce che solo l'autore del ToDo possa condividerlo, che non si possa condividere un ToDo con sè stessi e che una condivisione viene eliminata solo se effettivamente esistente. • elimina_bacheca(): garantisce che solo il proprietario di una bacheca possa eliminarla ed elimina tutti i ToDo associati alla bacheca e le condivisioni associate ad essi. • elimina_todo(): garantisce che solo il proprietario della bacheca o l'autore del ToDo possa eliminarli, elimina le varie condivisioni associate ed aggiorna le posizioni dei ToDo rimanenti in quella bacheca. • check_duplicati_login() + trg_check_login: impedisce che un utente si registri utilizzando un username già esistente. Non è strettamente necessario dato che l'attributo "login" è PK di Utente, ma è stato inserito per personalizzare l'errore con un messaggio più chiaro.

3 Progettazione Logica

In questa fase della progettazione andremo a tradurre il nostro schema concettuale in uno schema logico, scendendo ad un livello di astrazione più basso rispetto al precedente.

3.1 Schema Logico

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

- Utente(Login, Password)
- Bacheca(id, Titolo, Descrizione, Proprietario)
  - Bacheca.Proprietario → Utente.Login
- ToDo(id, Titolo, Stato, Link, Immagine, Descrizione, Scadenza, Colore, idB, Autore)
  - ToDo.Autore → Utente.Login
  - ToDo.idB → Bacheca.id
- Condivisione(idT, Login)
  - Condivisione.idT → ToDo.id
  - Condivisione.Login → Utente.Login

4 Progettazione Fisica

In questo capitolo verrà riportata l'implementazione dello schema logico precedentemente descritto all'interno del DBMS PostgreSQL.

4.1 Definizione tabelle

In questa sezione sono riportate le definizioni delle tabelle, dei loro vincoli intrarelazionali e di eventuali strutture per la loro gestione.

4.1.1 Definizione della tabella UTENTE

```
CREATE TABLE IF NOT EXISTS utente (
    login VARCHAR(100) PRIMARY KEY,
    password VARCHAR(100) NOT NULL
);


CREATE OR REPLACE FUNCTION check_duplicati_login()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1 FROM utente WHERE login = NEW.login
    ) THEN
        RAISE EXCEPTION 'Username già esistente: %, inserirne
        uno diverso', NEW.login;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS trg_check_login ON utente;

CREATE TRIGGER trg_check_login
BEFORE INSERT ON utente
FOR EACH ROW
EXECUTE FUNCTION check duplicati login();
```

4.1.2 Definizione della tabella BACHECA

```
CREATE TABLE IF NOT EXISTS bacheca(
id INTEGER PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    titolo VARCHAR(100) NOT NULL,
    descrizione TEXT NOT NULL,
    numero SMALLINT NOT NULL CHECK (numero BETWEEN 1 AND 3),
    proprietario VARCHAR(100) NOT NULL,
    FOREIGN KEY (proprietario) REFERENCES utente(login),
    UNIQUE (proprietario, numero)
);


CREATE OR REPLACE FUNCTION crea_bacheche()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO bacheca (titolo, descrizione, numero, proprietario)
    VALUES
    ('Università', 'Bacheca per la Università', 1, NEW.login),
    ('Lavoro', 'Bacheca per il Lavoro', 2, NEW.login),
    ('Tempo Libero', 'Bacheca per il Tempo Libero', 3, NEW.login);

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS crea_bacheche_trigger ON utente;

CREATE TRIGGER crea_bacheche_trigger
AFTER INSERT ON utente
FOR EACH ROW
EXECUTE FUNCTION crea_bacheche();
```

4.1.3 Definizione della tabella TODO

```
CREATE TABLE IF NOT EXISTS todo(
    id INTEGER PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    titolo VARCHAR(100) NOT NULL,
    stato BOOLEAN DEFAULT FALSE,
    link TEXT,
    immagine TEXT,
    descrizione TEXT,
    scadenza DATE,
    colore VARCHAR(7),
    posizione INTEGER,
    autore VARCHAR(100) NOT NULL,
    idB INTEGER NOT NULL,
    FOREIGN KEY (idB) REFERENCES bacheca(id),
    FOREIGN KEY (autore) REFERENCES utente(login),
    CHECK (colore ~ '^#([0-9a-fA-F]{6})$')
);
```

4.1.4 Definizione della tabella CONDIVISIONE

```
CREATE TABLE IF NOT EXISTS condivisione(
   ricevente VARCHAR(100) NOT NULL,
   idT INTEGER NOT NULL,
   PRIMARY KEY (ricevente, idT),
   FOREIGN KEY (ricevente) REFERENCES utente(login),
   FOREIGN KEY (idT) REFERENCES todo(id)
   );
```

4.2 Implementazione dei Vincoli

Di seguito sono riportati tutti i vincoli che non sono stati già mostrati all'interno delle tabelle.

4.2.1 Creazione automatica delle bacheche standard

Quando un nuovo utente si registra nel sistema, gli vengono automaticamente attribuite le tre bacheche standard.

```
  CREATE OR REPLACE FUNCTION check_bacheche_standard()
RETURNS TRIGGER AS $$
DECLARE
    n_bacheche INTEGER;
BEGIN
    IF NEW.numero NOT IN (1, 2, 3) THEN
        RAISE EXCEPTION 'Tipo bacheca non valido: %
        (consentiti:
        1=Università, 2=Lavoro, 3=Tempo Libero)',
        NEW.numero;
    END IF;
    SELECT COUNT(*) INTO n_bacheche
    FROM bacheca
    WHERE proprietario = NEW.proprietario AND numero = NEW.numero;

    IF n_bacheche > 0 THEN
        RAISE EXCEPTION 'L''utente % ha già una bacheca di tipo %',
        NEW.proprietario, NEW.numero;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS trg_check_bacheche ON bacheca;

CREATE TRIGGER trg_check_bacheche
BEFORE INSERT ON bacheca
FOR EACH ROW
EXECUTE FUNCTION check_bacheche_standard();
```

4.2.2 Gestione posizione dei ToDo

Quando un nuovo todo viene creato senza specificarne la posizione, viene collocato automaticamente sul fondo della lista dei ToDo.

```
CREATE OR REPLACE FUNCTION set_posizione_todo()
RETURNS TRIGGER AS $$
DECLARE
    max_pos INTEGER;
BEGIN
    IF NEW.posizione IS NULL THEN
        SELECT COALESCE(MAX(posizione), 0) INTO max_pos
        FROM todo
        WHERE idB = NEW.idB;

        NEW.posizione := max_pos + 1;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS trg_set_posizione_todo ON todo;

CREATE TRIGGER trg_set_posizione_todo
BEFORE INSERT ON todo
FOR EACH ROW
EXECUTE FUNCTION set_posizione_todo();
```

4.3 Funzioni, Procedure ed altre automazioni

4.3.1 Funzione lista_funzioni

```
CREATE OR REPLACE FUNCTION lista_funzioni()
RETURNS TABLE (
    nome TEXT, descrizione TEXT
) AS $$
BEGIN
    RETURN QUERY
    SELECT * FROM (VALUES
        ('modifica_todo', 'Modifica i campi di un ToDo già esistente
         (solo autore può farlo)'),

        ('modifica_bacheca', 'Modifica titolo o descrizione di una bacheca
         (solo proprietario può farlo)'),

        ('elimina_todo', 'Elimina un ToDo se sei autore o proprietario
         della bacheca'),

        ('elimina_bacheca', 'Elimina una bacheca e tutti i suoi ToDo
         (solo proprietario)'),

         ('registra_utente', 'Inserisce un nuovo utente all''interno
          del sistema'),

        ('sposta_todo_tra_bacheche', 'Sposta un ToDo da una
         bacheca a un''altra dello stesso proprietario'),

        ('set_stato_todo', 'Serve ad impostare lo stato del
         ToDo: TRUE = completato (sottinteso), FALSE = non completato'),

        ('condividi_todo', 'Permette all''autore di condividere o rimuovere
         la condivisione di un ToDo'),

        ('lista_condivisioni_todo', 'Mostra tutti gli utenti con cui è
         condiviso quel determinato ToDo'),

        ('todo_in_scadenza', 'Mostra ToDo dell''utente o quelli condivisi
         che sono in scadenza nella data odierna o entro una data a scelta'),

        ('cerca_todo_per_titolo', 'Permette di cercare un ToDo all''interno
        del sistema inserendone il titolo (solo se esistente)')

    ) AS elenco(nome, descrizione);
END;
$$ LANGUAGE plpgsql;
```

4.3.2 Funzione registra_utente

```
CREATE OR REPLACE FUNCTION registra_utente(login TEXT, password TEXT)
RETURNS VOID AS $$
BEGIN
    INSERT INTO utente(login, password) VALUES (login, password);
END;
$$ LANGUAGE plpgsql;
```

4.3.3 Funzione modifica_todo

```
CREATE OR REPLACE FUNCTION modifica_todo(
    id_todo INTEGER,
    username TEXT,
    ntitolo TEXT DEFAULT NULL,
    ndescrizione TEXT DEFAULT NULL,
    nscadenza DATE DEFAULT NULL,
    ncolore TEXT DEFAULT NULL,
    nlink TEXT DEFAULT NULL,
    nimmagine TEXT DEFAULT NULL
)
RETURNS VOID AS $$
BEGIN
    UPDATE todo
    SET
        titolo = COALESCE(ntitolo, titolo),
        descrizione = COALESCE(ndescrizione, descrizione),
        scadenza = COALESCE(nscadenza, scadenza),
        colore = COALESCE(ncolore, colore),
        link = COALESCE(nlink, link),
        immagine = COALESCE(nimmagine, immagine)
    WHERE id = id_todo AND autore = username;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Impossibile modificare poichè il ToDo
        non è stato trovato oppure non si dispone dei permessi';
    END IF;
END;
$$ LANGUAGE plpgsql;
```

4.3.4 Funzione modifica_bacheca

```
CREATE OR REPLACE FUNCTION modifica_bacheca(
    id_bacheca INTEGER,
    username TEXT,
    ntitolo TEXT DEFAULT NULL,
    ndescrizione TEXT DEFAULT NULL
)
RETURNS VOID AS $$
BEGIN
    UPDATE bacheca
    SET
        titolo = COALESCE(ntitolo, titolo),
        descrizione = COALESCE(ndescrizione, descrizione)
    WHERE id = id_bacheca AND proprietario = username;
    IF NOT FOUND THEN
        RAISE EXCEPTION 'Impossibile modificare poichè la bacheca
        non è stata trovata oppure non si dispone dei permessi';
    END IF;
END;
$$ LANGUAGE plpgsql;
```

4.3.5 Funzione elimina_bacheca

```
CREATE OR REPLACE FUNCTION elimina_bacheca(
    p_id_bacheca INTEGER,
    p_login_richiedente VARCHAR
)
RETURNS VOID AS $$
DECLARE
    proprietario_bacheca VARCHAR;
BEGIN
    SELECT proprietario INTO proprietario_bacheca
    FROM bacheca
    WHERE id = p_id_bacheca;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Bacheca con id % non esiste.', p_id_bacheca;
    END IF;

    IF proprietario_bacheca IS DISTINCT FROM p_login_richiedente THEN
        RAISE EXCEPTION 'Accesso negato: solo il proprietario può
        eliminare la bacheca.';
    END IF;

    DELETE FROM condivisione
    WHERE idT IN (
        SELECT id FROM todo WHERE idB = p_id_bacheca
    );

    DELETE FROM todo
    WHERE idB = p_id_bacheca;

    DELETE FROM bacheca
    WHERE id = p_id_bacheca;
END;
$$ LANGUAGE plpgsql;
```

4.3.6 Funzione elimina_todo

```
CREATE OR REPLACE FUNCTION elimina_todo(
    p_id_todo INTEGER,
    p_login_richiedente VARCHAR
)
RETURNS VOID AS $$
DECLARE
    v_autore VARCHAR;
    v_idB INTEGER;
    v_posizione INTEGER;
    v_proprietario_bacheca VARCHAR;
BEGIN
    SELECT autore, idB, posizione
    INTO v_autore, v_idB, v_posizione
    FROM todo
    WHERE id = p_id_todo;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'ToDo con id % non esiste.', p_id_todo;
    END IF;

    SELECT proprietario INTO v_proprietario_bacheca
    FROM bacheca
    WHERE id = v_idB;

    IF p_login_richiedente IS DISTINCT FROM v_autore
       AND p_login_richiedente IS DISTINCT FROM v_proprietario_bacheca
       THEN
        RAISE EXCEPTION 'Accesso negato: solo autore o proprietario
        della bacheca possono eliminare il ToDo.';
    END IF;

    DELETE FROM condivisione WHERE idT = p_id_todo;

    DELETE FROM todo WHERE id = p_id_todo;

    UPDATE todo
    SET posizione = posizione - 1
    WHERE idB = v_idB AND posizione > v_posizione;
END;
$$ LANGUAGE plpgsql;
```

4.3.7 Funzione condividi_todo

```
CREATE OR REPLACE FUNCTION condividi_todo(
    p_id_todo INTEGER,
    p_ricevente VARCHAR,
    p_login_richiedente VARCHAR,
    p_azione TEXT  -- 'aggiungi' o 'rimuovi'
)
RETURNS VOID AS $$
DECLARE
    v_autore VARCHAR;
BEGIN
    SELECT autore INTO v_autore
    FROM todo
    WHERE id = p_id_todo;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'ToDo con id % non esiste.', p_id_todo;
    END IF;

    IF v_autore IS DISTINCT FROM p_login_richiedente THEN
        RAISE EXCEPTION 'Solo l''autore del ToDo può modificarne le
                         condivisioni.';
    END IF;

    IF p_ricevente = v_autore THEN
        RAISE EXCEPTION 'Non ha senso condividere un ToDo con sé stessi.';
    END IF;

    IF p_azione = 'aggiungi' THEN
        INSERT INTO condivisione (idT, ricevente)
        VALUES (p_id_todo, p_ricevente)
        ON CONFLICT DO NOTHING;

    ELSIF p_azione = 'rimuovi' THEN
        DELETE FROM condivisione
        WHERE idT = p_id_todo AND ricevente = p_ricevente;

    ELSE
        RAISE EXCEPTION 'Azione non valida: % (consentite:
                         aggiungi, rimuovi)', p_azione;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

4.3.8 Funzione sposta_todo_tra_bacheche

```
    CREATE OR REPLACE FUNCTION sposta_todo_tra_bacheche(
    p_id_todo      INTEGER,
    idB_dest       INTEGER,
    p_login_richiedente VARCHAR
)
RETURNS VOID AS $$
DECLARE
    idB_orig    INTEGER;
    old_posizione    INTEGER;
    new_posizione    INTEGER;
    proprietario_orig TEXT;
BEGIN
    SELECT idB, posizione
    INTO idB_orig, old_posizione
    FROM todo
    WHERE id = p_id_todo;
    IF NOT FOUND THEN RAISE EXCEPTION 'ToDo con id % non esiste.', p_id_todo;
    END IF;

    SELECT proprietario INTO proprietario_orig
    FROM bacheca
    WHERE id = idB_orig;

    IF proprietario_orig IS DISTINCT FROM p_login_richiedente THEN
        RAISE EXCEPTION 'Accesso negato: solo il proprietario
        della bacheca può spostare i ToDo.';
    END IF;

    IF NOT EXISTS (
        SELECT 1 FROM bacheca WHERE id = idB_dest
    ) THEN RAISE EXCEPTION 'Bacheca di destinazione con id %
        non esiste.', idB_dest;
    END IF;

    UPDATE todo
    SET posizione = posizione - 1
    WHERE idB = idB_orig AND posizione > old_posizione;

    SELECT COALESCE(MAX(posizione), 0) + 1
    INTO new_posizione
    FROM todo
    WHERE idB = idB_dest;

    UPDATE todo
    SET idB = idB_dest,
        posizione = new_posizione
    WHERE id = p_id_todo;
END; $$ LANGUAGE plpgsql;
```

4.3.9 Funzione set_stato_todo

```
CREATE OR REPLACE FUNCTION set_stato_todo(
    p_id_todo INTEGER,
    p_login_richiedente VARCHAR,
    p_stato BOOLEAN DEFAULT TRUE
)
RETURNS VOID AS $$
DECLARE
    v_autore VARCHAR;
    v_idB INTEGER;
    v_proprietario_bacheca VARCHAR;
    v_condiviso BOOLEAN;
BEGIN
    SELECT autore, idB INTO v_autore, v_idB
    FROM todo
    WHERE id = p_id_todo;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'ToDo con id % non esiste.', p_id_todo;
    END IF;

    SELECT proprietario INTO v_proprietario_bacheca
    FROM bacheca
    WHERE id = v_idB;

    IF p_login_richiedente IS DISTINCT FROM v_autore
       AND p_login_richiedente IS DISTINCT FROM v_proprietario_bacheca THEN

        SELECT EXISTS (
            SELECT 1 FROM condivisione
            WHERE idT = p_id_todo AND ricevente = p_login_richiedente
        ) INTO v_condiviso;

        IF NOT v_condiviso THEN
            RAISE EXCEPTION 'Accesso negato: il ToDo non è visibile da %',
            p_login_richiedente;
        END IF;
    END IF;

    UPDATE todo
    SET stato = p_stato
    WHERE id = p_id_todo;
END;
$$ LANGUAGE plpgsql;
```

4.3.10 Funzione lista_condivisioni_todo

```
CREATE OR REPLACE FUNCTION lista_condivisioni_todo(
    p_id_todo INTEGER,
    p_login_richiedente VARCHAR
)
RETURNS TABLE(ricevente VARCHAR) AS $$
DECLARE
    v_autore VARCHAR;
    v_idB INTEGER;
    v_proprietario VARCHAR;
    v_accesso BOOLEAN;
BEGIN
    SELECT autore, idB INTO v_autore, v_idB
    FROM todo
    WHERE id = p_id_todo;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'ToDo con id % non esiste.', p_id_todo;
    END IF;

    SELECT proprietario INTO v_proprietario
    FROM bacheca
    WHERE id = v_idB;

    v_accesso := (
        p_login_richiedente = v_autore OR
        p_login_richiedente = v_proprietario OR
        EXISTS (
            SELECT 1 FROM condivisione
            WHERE idT = p_id_todo AND ricevente = p_login_richiedente
        )
    );

    IF NOT v_accesso THEN
        RAISE EXCEPTION 'Accesso negato al ToDo %', p_id_todo;
    END IF;

    RETURN QUERY
    SELECT ricevente
    FROM condivisione
    WHERE idT = p_id_todo;
END;
$$ LANGUAGE plpgsql;
```

4.3.11 Funzione todo_in_scadenza

```
CREATE OR REPLACE FUNCTION todo_in_scadenza(
    p_login VARCHAR,
    p_data_fine DATE DEFAULT CURRENT_DATE
)
RETURNS TABLE (
    id INTEGER,
    titolo TEXT,
    descrizione TEXT,
    scadenza DATE,
    autore TEXT,
    condiviso BOOLEAN
) AS $$
BEGIN
    RETURN QUERY
    SELECT
        t.id,
        t.titolo,
        t.descrizione,
        t.scadenza,
        t.autore,
        FALSE AS condiviso
    FROM todo t
    WHERE t.autore = p_login
      AND t.scadenza IS NOT NULL
      AND t.scadenza <= p_data_fine

    UNION

    SELECT
        t.id,
        t.titolo,
        t.descrizione,
        t.scadenza,
        t.autore,
        TRUE AS condiviso
    FROM todo t
    JOIN condivisione c ON t.id = c.idT
    WHERE c.ricevente = p_login
      AND t.scadenza IS NOT NULL
      AND t.scadenza <= p_data_fine;
END;
$$ LANGUAGE plpgsql;
```

4.3.12 Funzione cerca_todo_per_titolo

```
CREATE OR REPLACE FUNCTION cerca_todo_per_titolo(titolo_input TEXT)
RETURNS TABLE (
    id INT,
    titolo TEXT,
    descrizione TEXT,
    autore TEXT,
    scadenza DATE,
    completato BOOLEAN
) AS $$
BEGIN
    RETURN QUERY
    SELECT id, titolo, descrizione, autore, scadenza, completato
    FROM todo
    WHERE titolo ILIKE titolo_input;
END;
$$ LANGUAGE plpgsql;
```

4.3.13 Funzione inverti_posizione_todo

```
CREATE OR REPLACE FUNCTION inverti_posizione_todo(id1 INT, id2 INT)
RETURNS VOID AS $$
DECLARE
    pos1 INT;
    pos2 INT;
BEGIN
    SELECT posizione INTO pos1 FROM todo WHERE id = id1;
    SELECT posizione INTO pos2 FROM todo WHERE id = id2;

    IF pos1 IS NULL OR pos2 IS NULL THEN
        RAISE EXCEPTION 'Almeno uno dei ToDo non esiste';
    END IF;

    UPDATE todo SET posizione = pos2 WHERE id = id1;
    UPDATE todo SET posizione = pos1 WHERE id = id2;
END;
$$ LANGUAGE plpgsql;
```