UNIVERSITÀ DEGLI STUDI DI NAPOLI

FEDERICO II

DOCUMENTAZIONE PER PROGETTO
BASE DI DATI E OBJECT-ORIENTED

CdL Triennale in Informatica
Corso di Basi di Dati e Object-Oriented
PIETRO PELLEGRINO
N86004722
EMANUELE MILANO
N86004210

Anno Accademico: 2024/2025

INDICE

URL a GITHUB

Codice SQL e GUI Java: `https://github.com/ShishRobot2000/ProjectBDD-OO-2025`

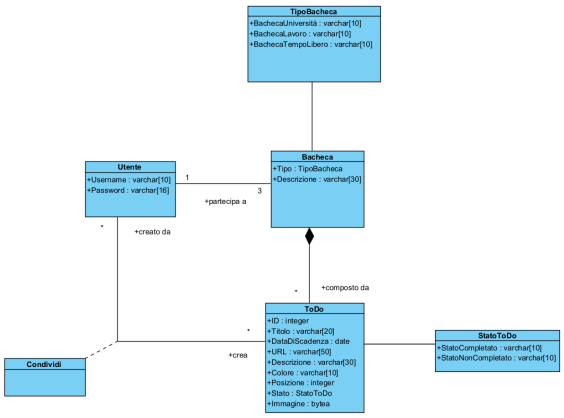
1 Introduzione

1.1 Descrizione del problema

Il sistema ToDo consente a ciascun utente autenticato di gestire bacheche personali suddivise per categoria (Università, Lavoro, Tempo Libero), contenenti attività ("ToDo") classificabili per stato, colore, descrizione e data. È inoltre possibile condividere i ToDo con altri utenti, assegnando loro la possibilità di accettare o rifiutare la collaborazione.

2 Progettazione Concettuale

2.1 Class Diagram



2.2 Ristrutturazione del Class Diagram

2.2.1 Analisi delle chiavi

Per garantire una gestione efficiente dei dati, sono state adottate chiavi primarie ben definite per ciascuna entità del sistema. L'entità `ToDo` utilizza una chiave surrogata (`id`), generata automaticamente, per assicurare un'identificazione univoca semplice e performante.
L'entità `Bacheca` adotta una chiave primaria composta da `(proprietario, tipo)`, che riflette il vincolo secondo cui ciascun utente può avere al massimo una bacheca per tipo. Questo consente una gestione più naturale delle relazioni con i ToDo.
Per l'entità `Condivisione`, è stata scelta una chiave composta `(username_utente, id_todo)`, utile a rappresentare correttamente la relazione N:N tra utenti e task condivisi.
Infine, l'entità `Utente` è identificata univocamente dallo `username`, che funge da chiave primaria.

2.2.2 Analisi degli attributi secondari

In questa fase si analizzano eventuali attributi derivati, ovvero ottenibili a partire da altri già presenti nel sistema. Nel nostro progetto non sono presenti attributi calcolabili in modo efficiente a runtime tali da giustificare una rimozione dalla base di dati.

2.2.3 Analisi delle ridondanze

Analizzando lo schema logico non emergono vere e proprie ridondanze strutturali tra le entità principali.
Un possibile caso di ambiguità logica potrebbe derivare dalla possibilità di rappresentare un legame tra un utente e un `ToDo` in due modi diversi: come proprietario (nella tabella `todo`) e come destinatario di una condivisione (nella tabella `condivisione`). Se non opportunamente gestito, ciò potrebbe portare a rappresentare il proprietario anche come destinatario di una condivisione del proprio ToDo.
Tuttavia, questa situazione è prevenuta tramite vincoli a livello applicativo, che impediscono la condivisione con se stessi. Pertanto, non si riscontrano ridondanze tali da compromettere la coerenza del modello.

2.2.4 Analisi degli attributi strutturati

In questa fase viene effettuata un'analisi degli attributi per verificare l'eventuale presenza di attributi strutturati, ovvero composti da sotto-attributi non rappresentabili direttamente all'interno di un DBMS relazionale.
Nel nostro schema logico non sono presenti attributi strutturati: tutti gli attributi sono atomici e quindi compatibili con il modello relazionale. Non si è quindi reso necessario alcun intervento di normalizzazione o di scomposizione per adattare la struttura degli attributi al DBMS.

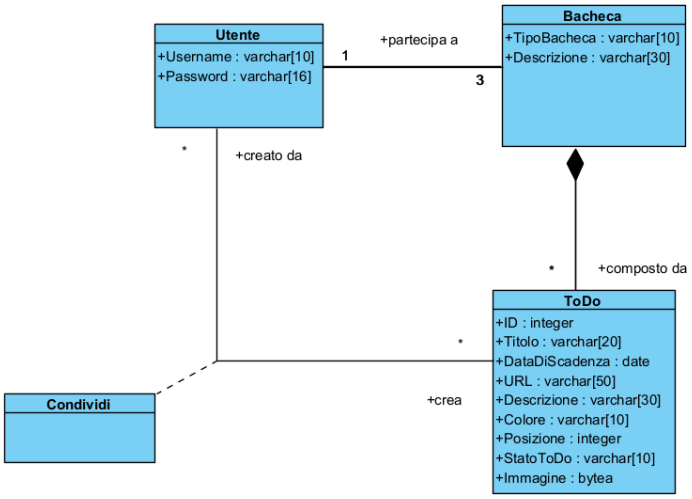
2.2.5 Analisi degli attributi a valore multiplo

Verifichiamo ora la presenza di eventuali attributi a valore multiplo, non logicamente rappresentabili nel modello relazionale e quindi da trasformare.
Nel nostro progetto, il concetto di *utenti con cui un ToDo è stato condiviso* rappresenta un attributo a valore multiplo: un singolo ToDo può essere condiviso con più utenti. Per rendere questo concetto compatibile con il modello relazionale, è stata introdotta l'entità `Condivisione`, che associa ogni ToDo a ciascun utente destinatario, con uno stato.
Tutti gli altri attributi presenti nel modello sono a valore singolo, e non necessitano di ulteriori trasformazioni.

2.2.6 Analisi delle gerarchie di specializzazione

Analizziamo infine la presenza di eventuali gerarchie di specializzazione nel modello concettuale, che non risultano direttamente rappresentabili in un database relazionale e richiedono una ristrutturazione.
Nel progetto non sono presenti gerarchie di specializzazione tra entità, ovvero non esistono entità generiche specializzate in sottotipi disgiunti o sovrapposti. Tutte le entità identificate nel modello sono autonome, e non derivano da entità più generali.
Non è quindi necessaria alcuna trasformazione o ristrutturazione per quanto riguarda questo aspetto.

2.3 Class Diagram Ristrutturato



2.4 Dizionario delle Classi

Classe	Descrizione	Attributi
Utente	Rappresenta un utente registrato nel sistema. Ogni utente può creare bacheche personali e ricevere ToDo condivisi.	– **username** (varchar) : Identificatore univoco dell'utente. – password (varchar) : Password di accesso al sistema.
Bacheca	Rappresenta una bacheca personale appartenente a un utente. Ogni bacheca è di un tipo predefinito e contiene ToDo.	– **proprietario** (varchar) : Username dell'utente proprietario (FK). – **tipo_bacheca** (varchar) : Tipo della bacheca (Università, Lavoro, Tempo Libero). – descrizione (text) : Testo descrittivo della bacheca.
ToDo	Rappresenta un'attività o nota gestita all'interno di una bacheca. Può essere condivisa con altri utenti. Solo il titolo è necessario, il resto è opzionale.	– **id** (integer) : Identificatore univoco del ToDo (generato da sequenza). – titolo (varchar) : Titolo breve dell'attività. – descrizione (text) : Dettagli estesi del ToDo. – data_scadenza (date) : Data di scadenza dell'attività. – colore (varchar) : Codice colore esadecimale (es. FF0000). – stato (varchar) : Stato del ToDo ('COMPLETATO', 'NON_COMPLETATO'). – url (text) : Collegamento esterno. – immagine (bytea) : Immagine associata. – posizione (integer) : Ordine nella bacheca. – proprietario (varchar) : Utente proprietario della bacheca (FK). – tipo_bacheca (varchar) : Tipo della bacheca di riferimento (FK).
Condivisione	Rappresenta una richiesta di condivisione di un ToDo tra utenti. Gestisce l'invio, l'accettazione o la rimozione.	– **username_utente** (varchar) : Utente destinatario della condivisione (FK). – **id_todo** (integer) : Identificativo del ToDo condiviso (FK). – stato (varchar) : Stato della richiesta ('PENDING', 'ACCEPTED').

2.5 Dizionario delle Associazioni

Relazione	**Descrizione**
Utente ⟶ Bacheca	Ogni utente possiede da **1 a 3 bacheche**. Associazione 1:N tra `Utente.username` e `Bacheca.proprietario`.
Bacheca ⟶ ToDo	Ogni bacheca contiene uno o più ToDo. Associazione 1:N tra `Bacheca(proprietario, tipo_bacheca)` e `ToDo(proprietario, tipo_bacheca)`.
ToDo ⟶ Utente (Condivisione)	Ogni ToDo può essere condiviso con più utenti, e ogni utente può ricevere più ToDo condivisi. Associazione N:N tramite la tabella `Condivisione(username_utente, id_todo)`.
ToDo ⟶ StatoToDo	Ogni ToDo ha un solo stato: `COMPLETATO` oppure `NON_COMPLETATO`. Associazione N:1 tra `ToDo.stato` e `StatoToDo.nome`.
ToDo ⟶ TipoBacheca	Ogni ToDo appartiene a un solo tipo di bacheca: `UNIVERSITA`, `LAVORO`, o `TEMPO_LIBERO`. Associazione N:1 implicita via `ToDo.tipo_bacheca`.

2.6 Dizionario dei Vincoli

Vincolo	Descrizione
Chiavi Primarie (PK)	`Utente.username`, `ToDo.id`, `Bacheca.tipo`, `StatoToDo.nome`, `Condivisione(username_utente, id_todo)`.
Vincolo di Unicità	Ogni utente può avere al massimo una bacheca per ciascun tipo: vincolo di unicità su `(proprietario, tipo)` nella tabella `Bacheca`.
Formato Colore	Il campo `colore` di `ToDo` deve essere una stringa esadecimale valida a 6 cifre, senza il simbolo `#`. Controllato via trigger `check_colore_todo()`.
Posizione unica	La posizione di ogni ToDo in una bacheca deve essere unica rispetto alla coppia `(proprietario, tipo_bacheca)`. (Gestita a livello applicativo).
Stati ammessi	Il campo `stato` in `ToDo` accetta solo due valori: `COMPLETATO`, `NON_COMPLETATO`. Gestito con vincolo `CHECK` e tabella `StatoToDo`.
Tipi di Bacheca ammessi	Il campo `tipo` di una bacheca deve essere uno tra: `UNIVERSITA`, `LAVORO`, `TEMPO_LIBERO`. Vincolo applicato tramite trigger `check_bacheche_standard()`.

3 Progettazione Logica

In questa fase della progettazione si passa da una rappresentazione concettuale a una più vicina all'implementazione.

3.1 Schema Logico

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

- Utente (username, password)
- Bacheca (proprietario, tipo, descrizione)
  - proprietario → Utente.username
- ToDo (id, titolo, descrizione, data_scadenza, colore, stato, url, immagine, posizione, proprietario, tipo_bacheca)
  - (proprietario, tipo_bacheca) → Bacheca(proprietario, tipo)
- Condivisione (username_utente, id_todo, stato)
  - username_utente → Utente.username
  - id_todo → ToDo.id

4 Progettazione Fisica

In questo capitolo viene riportata l'implementazione fisica dello schema logico all'interno del DBMS PostgreSQL. Verranno presentate le definizioni delle tabelle, l'implementazione dei vincoli (inclusi trigger e funzioni) e infine alcune automazioni utili all'interazione con il sistema.

4.1 Definizione Tabelle

4.1.1 Definizione della tabella UTENTE

```
CREATE TABLE utente (
    username VARCHAR(100) PRIMARY KEY,
    password VARCHAR(100) NOT NULL
);

CREATE FUNCTION crea_bacheche_standard()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO bacheca (tipo, descrizione, proprietario)
    VALUES
        ('UNIVERSITA', 'Bacheca Universit ', NEW.username),
        ('LAVORO', 'Bacheca Lavoro', NEW.username),
        ('TEMPO_LIBERO', 'Bacheca Tempo Libero', NEW.username);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_crea_bacheche
AFTER INSERT ON utente
FOR EACH ROW
EXECUTE FUNCTION crea_bacheche_standard();
```

4.1.2 Definizione della tabella BACHECA

```
CREATE TABLE bacheca (
    tipo VARCHAR(30) NOT NULL,
    descrizione TEXT,
    proprietario VARCHAR(100) NOT NULL,
    PRIMARY KEY (proprietario, tipo),
    FOREIGN KEY (proprietario) REFERENCES utente(username)
);

CREATE FUNCTION check_bacheche_standard()
RETURNS TRIGGER AS $$
DECLARE
    n_bacheche INTEGER;
BEGIN
    -- Verifica che il tipo sia tra quelli previsti
    IF NEW.tipo NOT IN ('UNIVERSITA', 'LAVORO', 'TEMPO_LIBERO') THEN
```

```
        RAISE EXCEPTION 'Tipo bacheca non valido: % (consentiti:
            UNIVERSITA, LAVORO, TEMPO_LIBERO)', NEW.tipo;
    END IF;

    -- Controlla se l'utente ha gi  una bacheca di quel tipo
    SELECT COUNT(*) INTO n_bacheche
    FROM bacheca
    WHERE proprietario = NEW.proprietario AND tipo = NEW.tipo;

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

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

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

4.1.3 Definizione della Tabella TODO

```
-- Definizione tabella
CREATE TABLE TODO (
  id INTEGER PRIMARY KEY DEFAULT nextval('todo_id_seq'),
  titolo VARCHAR(100) NOT NULL,
  descrizione TEXT,
  data_scadenza TEXT,
  colore VARCHAR(7),
  stato VARCHAR(30) NOT NULL,
  url TEXT,
  immagine BYTEA,
  posizione INTEGER,
  proprietario VARCHAR(100) NOT NULL,
  tipo_bacheca VARCHAR(30) NOT NULL,
  CONSTRAINT stato_todo_check CHECK (stato IN ('COMPLETATO', '
      NON_COMPLETATO')),
  FOREIGN KEY (proprietario, tipo_bacheca) REFERENCES bacheca(
      proprietario, tipo)
);

-- Sequenza per ID ToDo
CREATE SEQUENCE todo_id_seq
  START WITH 1
  INCREMENT BY 1
  NO MINVALUE
  NO MAXVALUE;

-- Funzione per impostare ID automatico
```

```
CREATE OR REPLACE FUNCTION Funzione_Sequenza_Todo()
RETURNS TRIGGER AS $$
BEGIN
  IF (NEW.id IS NULL) THEN
    NEW.id := NEXTVAL('todo_id_seq');
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Trigger per ID automatico
CREATE TRIGGER Trigger_Sequenza_Todo
BEFORE INSERT ON TODO
FOR EACH ROW
EXECUTE PROCEDURE Funzione_Sequenza_Todo();

-- Funzione per default stato
CREATE OR REPLACE FUNCTION default_stato_todo()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW.stato IS NULL THEN
    NEW.stato := 'NON_COMPLETATO';
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Trigger per default stato
CREATE TRIGGER trg_default_stato
BEFORE INSERT ON TODO
FOR EACH ROW
EXECUTE PROCEDURE default_stato_todo();

-- Funzione per validazione colore
CREATE OR REPLACE FUNCTION check_colore_todo()
RETURNS TRIGGER AS $$
BEGIN
  IF LEFT(NEW.colore, 1) = '#' THEN
    RAISE EXCEPTION 'Il colore non pu  iniziare con #. Valore ricevuto:
        %', NEW.colore;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Trigger per validazione colore
CREATE TRIGGER trigger_check_colore_todo
BEFORE INSERT OR UPDATE ON TODO
FOR EACH ROW
EXECUTE PROCEDURE check_colore_todo();
```

4.1.4 Definizione della Tabella CONDIVISIONE

```
CREATE TABLE condivisione (
    username_utente VARCHAR(100) NOT NULL,
    id_todo INTEGER NOT NULL,
    stato VARCHAR(20) NOT NULL DEFAULT 'PENDING',
    PRIMARY KEY (username_utente, id_todo),
    FOREIGN KEY (username_utente) REFERENCES utente(username),
    FOREIGN KEY (id_todo) REFERENCES todo(id) ON DELETE CASCADE,
    CHECK (stato IN ('PENDING', 'ACCEPTED'))
);
```

4.2 Implementazione dei vincoli

4.2.1 Controllo standard delle bacheche

```
CREATE FUNCTION check_bacheche_standard() RETURNS TRIGGER AS $$
BEGIN
    IF NEW.tipo NOT IN ('UNIVERSITA', 'LAVORO', 'TEMPO_LIBERO') THEN
        RAISE EXCEPTION 'Tipo bacheca non valido: %', NEW.tipo;
    END IF;
    IF EXISTS (
        SELECT 1 FROM bacheca
        WHERE proprietario = NEW.proprietario AND tipo = NEW.tipo
    ) THEN
        RAISE EXCEPTION 'Bacheca gi  esistente per utente % e tipo %',
            NEW.proprietario, NEW.tipo;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

4.2.2 Controllo del colore dei ToDo

```
CREATE FUNCTION check_colore_todo() RETURNS TRIGGER AS $$
BEGIN
    -- Non deve iniziare con #
    IF LEFT(NEW.colore, 1) = '#' THEN
        RAISE EXCEPTION 'Il colore non pu  iniziare con #. Valore
            ricevuto: %', NEW.colore;
    END IF;

    -- Deve essere lungo 6 caratteri ed esadecimale
    IF NEW.colore !~ '^[0-9A-Fa-f]{6}$' THEN
        RAISE EXCEPTION 'Colore non valido: deve essere un codice
            esadecimale di 6 cifre. Valore ricevuto: %', NEW.colore;
    END IF;

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

4.2.3 Stato di default per i ToDo

```
CREATE FUNCTION default_stato_todo() RETURNS TRIGGER AS $$
BEGIN
    IF NEW.stato IS NULL THEN
        NEW.stato := 'NON_COMPLETATO';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

4.2.4 Controllo stato della condivisione

Vincolo CHECK direttamente nella tabella:

```
stato character varying(20) DEFAULT 'PENDING' NOT NULL,
CONSTRAINT condivisione_stato_check CHECK (
    stato IN ('PENDING', 'ACCEPTED')
)
```

4.3 Funzioni, Procedure e altre automazioni

4.3.1 Funzione: aggiorna_stato_condivisione

```
CREATE FUNCTION aggiorna_stato_condivisione(p_destinatario text,
    p_id_todo integer, p_nuovo_stato text) RETURNS void AS $$
BEGIN
    IF UPPER(p_nuovo_stato) = 'ACCEPTED' THEN
        UPDATE condivisione
        SET stato = 'ACCEPTED'
        WHERE username_utente = p_destinatario AND id_todo = p_id_todo;
        IF NOT FOUND THEN
            RAISE EXCEPTION 'Nessuna condivisione trovata da aggiornare'
                ;
        END IF;
    ELSIF UPPER(p_nuovo_stato) = 'REJECTED' THEN
        DELETE FROM condivisione
        WHERE username_utente = p_destinatario AND id_todo = p_id_todo
            AND stato = 'PENDING';
        IF NOT FOUND THEN
            RAISE EXCEPTION 'Nessuna richiesta PENDING trovata da
                rimuovere';
        END IF;
    ELSE
        RAISE EXCEPTION 'Stato non valido: usa solo ACCEPTED o REJECTED'
            ;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

4.3.2 Funzione: aggiorna_todo

```
CREATE OR REPLACE FUNCTION aggiorna_todo(
  p_id integer,
  p_titolo text,
  p_descrizione text,
  p_data_scadenza text,
  p_colore text,
  p_stato text,
  p_url text,
  p_immagine bytea
)
RETURNS boolean AS $$
BEGIN
  UPDATE todo
  SET titolo = p_titolo,
      descrizione = p_descrizione,
      data_scadenza = p_data_scadenza,
      colore = p_colore,
      stato = p_stato,
      url = p_url,
```

```
      immagine = p_immagine
  WHERE id = p_id;

  RETURN FOUND;
END;
$$ LANGUAGE plpgsql;
```

4.3.3 Funzione: aggiorna_todo_parziale

```
CREATE FUNCTION aggiorna_todo_parziale(
  p_id integer, p_titolo text, p_descrizione text, p_data_scadenza text,
  p_colore text, p_stato text, p_url text, p_immagine bytea, p_posizione
      integer
) RETURNS boolean AS $$
DECLARE
    v_sql TEXT := 'UPDATE todo SET ';
    v_sep TEXT := '';
BEGIN
    IF p_titolo IS NOT NULL THEN
        v_sql := v_sql || v_sep || 'titolo = ' || quote_literal(p_titolo
            ); v_sep := ', ';
    END IF;
    IF p_descrizione IS NOT NULL THEN
        v_sql := v_sql || v_sep || 'descrizione = ' || quote_literal(
            p_descrizione); v_sep := ', ';
    END IF;
    IF p_data_scadenza IS NOT NULL THEN
        v_sql := v_sql || v_sep || 'data_scadenza = ' || quote_literal(
            p_data_scadenza); v_sep := ', ';
    END IF;
    IF p_colore IS NOT NULL THEN
        v_sql := v_sql || v_sep || 'colore = ' || quote_literal(p_colore
            ); v_sep := ', ';
    END IF;
    IF p_stato IS NOT NULL THEN
        v_sql := v_sql || v_sep || 'stato = ' || quote_literal(p_stato);
             v_sep := ', ';
    END IF;
    IF p_url IS NOT NULL THEN
        v_sql := v_sql || v_sep || 'url = ' || quote_literal(p_url);
            v_sep := ', ';
    END IF;
    IF p_immagine IS NOT NULL THEN
        v_sql := v_sql || v_sep || 'immagine = ' || quote_literal(encode
            (p_immagine, 'hex')::bytea); v_sep := ', ';
    END IF;
    IF v_sep = '' THEN
        RETURN FALSE;
    END IF;
    v_sql := v_sql || ' WHERE id = ' || p_id;
    EXECUTE v_sql;
    RETURN FOUND;
```

```
END;
$$ LANGUAGE plpgsql;
```

4.3.4 Funzione: elimina_utente

```
CREATE FUNCTION elimina_utente(p_username text) RETURNS boolean AS $$
DECLARE
    id_todo INTEGER;
BEGIN
    DELETE FROM condivisione WHERE username_utente = p_username;

    FOR id_todo IN SELECT id FROM todo WHERE proprietario = p_username
        LOOP
        DELETE FROM condivisione WHERE id_todo = id_todo;
    END LOOP;

    DELETE FROM todo WHERE proprietario = p_username;
    DELETE FROM bacheca WHERE proprietario = p_username;
    DELETE FROM utente WHERE username = p_username;

    RETURN TRUE;
EXCEPTION WHEN OTHERS THEN
    RETURN FALSE;
END;
$$ LANGUAGE plpgsql;
```

4.3.5 Funzione: condividi_todo

```
CREATE OR REPLACE FUNCTION public.condividi_todo(p_destinatario text,
    p_id_todo integer) RETURNS boolean
LANGUAGE plpgsql
AS $$
DECLARE
  v_proprietario text;
BEGIN
  -- Recupera il proprietario del ToDo
  SELECT proprietario INTO v_proprietario
  FROM todo
  WHERE id = p_id_todo;

  -- Se il ToDo non esiste
  IF NOT FOUND THEN
    RETURN FALSE;
  END IF;

  -- Impedisce la condivisione con se stessi
  IF v_proprietario = p_destinatario THEN
    RETURN FALSE;
  END IF;

  -- Inserisce la nuova condivisione
```

```
  INSERT INTO condivisione (username_utente, id_todo, stato)
  VALUES (p_destinatario, p_id_todo, 'PENDING');

  RETURN TRUE;

EXCEPTION WHEN OTHERS THEN
  RETURN FALSE;
END;
$$;
```

4.3.6 Funzione: cancella_immagine_todo

```
CREATE FUNCTION cancella_immagine_todo(p_id integer) RETURNS boolean AS
    $$
DECLARE
  v_check BYTEA;
BEGIN
  SELECT immagine INTO v_check
  FROM todo
  WHERE id = p_id;

  IF NOT FOUND OR v_check IS NULL THEN
    RETURN FALSE;
  END IF;

  UPDATE todo
  SET immagine = NULL
  WHERE id = p_id;

  RETURN TRUE;
END;
$$ LANGUAGE plpgsql;
```

4.3.7 Funzione: elimina_todo

```
CREATE FUNCTION elimina_todo(p_id integer) RETURNS boolean AS $$
BEGIN
    DELETE FROM todo WHERE id = p_id;
    RETURN FOUND;
END;
$$ LANGUAGE plpgsql;
```

4.3.8 Funzione: elimina_condivisioni_collegate

```
CREATE FUNCTION elimina_condivisioni_collegate(p_id_todo integer)
    RETURNS boolean AS $$
BEGIN
    DELETE FROM condivisione WHERE id_todo = p_id_todo;
    RETURN TRUE;
END;
```

```
$$ LANGUAGE plpgsql;
```

4.3.9 Funzione: esiste_condivisione

```
CREATE FUNCTION esiste_condivisione(p_destinatario text, p_id_todo
    integer) RETURNS boolean AS $$
DECLARE
    exists_bool boolean;
BEGIN
    SELECT EXISTS (
        SELECT 1 FROM condivisione
        WHERE username_utente = p_destinatario
          AND id_todo = p_id_todo
    ) INTO exists_bool;

    RETURN exists_bool;
END;
$$ LANGUAGE plpgsql;
```

4.3.10 Funzione: richieste_pendenti_per_utente

```
CREATE FUNCTION richieste_pendenti_per_utente(p_username text) RETURNS
    TABLE(richiedente text, tipo_bacheca text, titolo text) AS $$
BEGIN
    RETURN QUERY
    SELECT
        t.proprietario::text,
        t.tipo_bacheca::text,
        t.titolo::text
    FROM condivisione c
    JOIN todo t ON c.id_todo = t.id
    WHERE c.username_utente = p_username AND c.stato = 'PENDING';
END;
$$ LANGUAGE plpgsql;
```

4.3.11 Funzione: rimuovi_condivisione

```
CREATE FUNCTION rimuovi_condivisione(p_destinatario text, p_id_todo
    integer) RETURNS void AS $$
BEGIN
    DELETE FROM condivisione
    WHERE username_utente = p_destinatario
      AND id_todo = p_id_todo;
END;
$$ LANGUAGE plpgsql;
```

4.3.12 Funzione: salva_todo

```
CREATE FUNCTION salva_todo(p_titolo text, p_descrizione text,
    p_data_scadenza text, p_colore text, p_url text, p_immagine bytea,
    p_proprietario text, p_tipo_bacheca text) RETURNS integer AS $$
DECLARE
    new_id INTEGER;
    colore_finale text;
BEGIN
    colore_finale := COALESCE(p_colore, 'FFFFFF');

    UPDATE todo
    SET posizione = posizione + 1
    WHERE proprietario = p_proprietario AND tipo_bacheca =
        p_tipo_bacheca;

    INSERT INTO todo (
        titolo, descrizione, data_scadenza, colore, stato,
        url, immagine, posizione, proprietario, tipo_bacheca
    )
    VALUES (
        p_titolo, p_descrizione, p_data_scadenza, colore_finale, '
            NON_COMPLETATO',
        p_url, p_immagine, 1, p_proprietario, p_tipo_bacheca
    )
    RETURNING id INTO new_id;

    RETURN new_id;
END;
$$ LANGUAGE plpgsql;
```

4.3.13 Funzione: salva_utente

```
CREATE FUNCTION salva_utente(p_username text, p_password text) RETURNS
    boolean AS $$
BEGIN
    INSERT INTO utente (username, password)
    VALUES (LOWER(TRIM(p_username)), p_password);
    RETURN TRUE;
EXCEPTION
    WHEN unique_violation THEN
        RAISE NOTICE 'Utente gi  esistente: %', LOWER(TRIM(p_username))
            ;
        RETURN FALSE;
    WHEN OTHERS THEN
        RAISE NOTICE 'Errore generico durante salva_utente: %', SQLERRM;
        RETURN FALSE;
END;
$$ LANGUAGE plpgsql;
```

4.3.14 Funzione: trova_todo_per_bacheca

```
CREATE FUNCTION trova_todo_per_bacheca(p_proprietario text,
    p_tipo_bacheca text)
RETURNS TABLE(id integer, titolo text, descrizione text, data_scadenza
    text, colore text, stato text, url text, immagine bytea, posizione
    integer) AS $$
BEGIN
    RETURN QUERY
    SELECT id, titolo, descrizione, data_scadenza, colore, stato, url,
        immagine, posizione
    FROM todo
    WHERE proprietario = p_proprietario AND tipo_bacheca =
        p_tipo_bacheca
    ORDER BY posizione ASC;
END;
$$ LANGUAGE plpgsql;
```

4.3.15 Funzione: mostra_funzioni

```
CREATE FUNCTION mostra_funzioni()
RETURNS TABLE(nome text, firma text) AS $$
  SELECT p.proname::text AS nome,
         pg_get_function_identity_arguments(p.oid) AS firma
  FROM pg_proc p
  JOIN pg_namespace n ON p.pronamespace = n.oid
  WHERE n.nspname = 'public'
    AND pg_function_is_visible(p.oid);
$$ LANGUAGE sql;
```

4.3.16 Funzione: mostra_view

```
CREATE FUNCTION mostra_view()
RETURNS TABLE(nome_view text, definizione text) AS $$
BEGIN
  RETURN QUERY
  SELECT
    viewname::TEXT,
    definition::TEXT
  FROM pg_views
  WHERE schemaname = 'public';
END;
$$ LANGUAGE plpgsql;
```

4.3.17 Funzione get_todo_completati

```
CREATE OR REPLACE FUNCTION get_todo_completati(p_username TEXT)
RETURNS TABLE (
  id integer,
  titolo text,
  descrizione text,
```

```
  data_scadenza text,
  colore text,
  stato text,
  url text,
  posizione integer,
  proprietario text,
  tipo_bacheca text
) AS $$
BEGIN
  RETURN QUERY
  SELECT
    v.id,
    v.titolo::text,
    v.descrizione::text,
    v.data_scadenza::text,
    v.colore::text,
    v.stato::text,
    v.url::text,
    v.posizione,
    v.proprietario::text,
    v.tipo_bacheca::text
  FROM vista_todo_senza_immagine v
  WHERE v.stato = 'COMPLETATO'
    AND v.proprietario = p_username;
END;
$$ LANGUAGE plpgsql;
```

4.3.18 Funzione get_todo_scaduti

```
CREATE OR REPLACE FUNCTION get_todo_scaduti(p_username text)
RETURNS TABLE (
  id integer,
  titolo text,
  descrizione text,
  data_scadenza text,
  colore text,
  stato text,
  url text,
  posizione integer,
  proprietario text,
  tipo_bacheca text
) AS $$
BEGIN
  RETURN QUERY
  SELECT
    v.id,
    v.titolo::text,
    v.descrizione::text,
    v.data_scadenza::text,
    v.colore::text,
    v.stato::text,
    v.url::text,
```

```
    v.posizione,
    v.proprietario::text,
    v.tipo_bacheca::text
  FROM vista_todo_senza_immagine v
  WHERE v.data_scadenza::date < CURRENT_DATE
    AND v.stato != 'COMPLETATO'
    AND v.proprietario = p_username;
END;
$$ LANGUAGE plpgsql;
```

4.3.19 Vista: vista_todo_senza_immagine

```
CREATE VIEW vista_todo_senza_immagine AS
SELECT id, titolo, descrizione, data_scadenza, colore, stato,
       url, posizione, proprietario, tipo_bacheca
FROM todo;
```