Wikipedia

Indice

1 Progettazione

1.1 Analisi dei Requisiti

Il seguente documento fornisce una base solida per la progettazione del sistema Wikipedia, consentendo la gestione degli utenti, delle pagine, delle proposte di modifica.

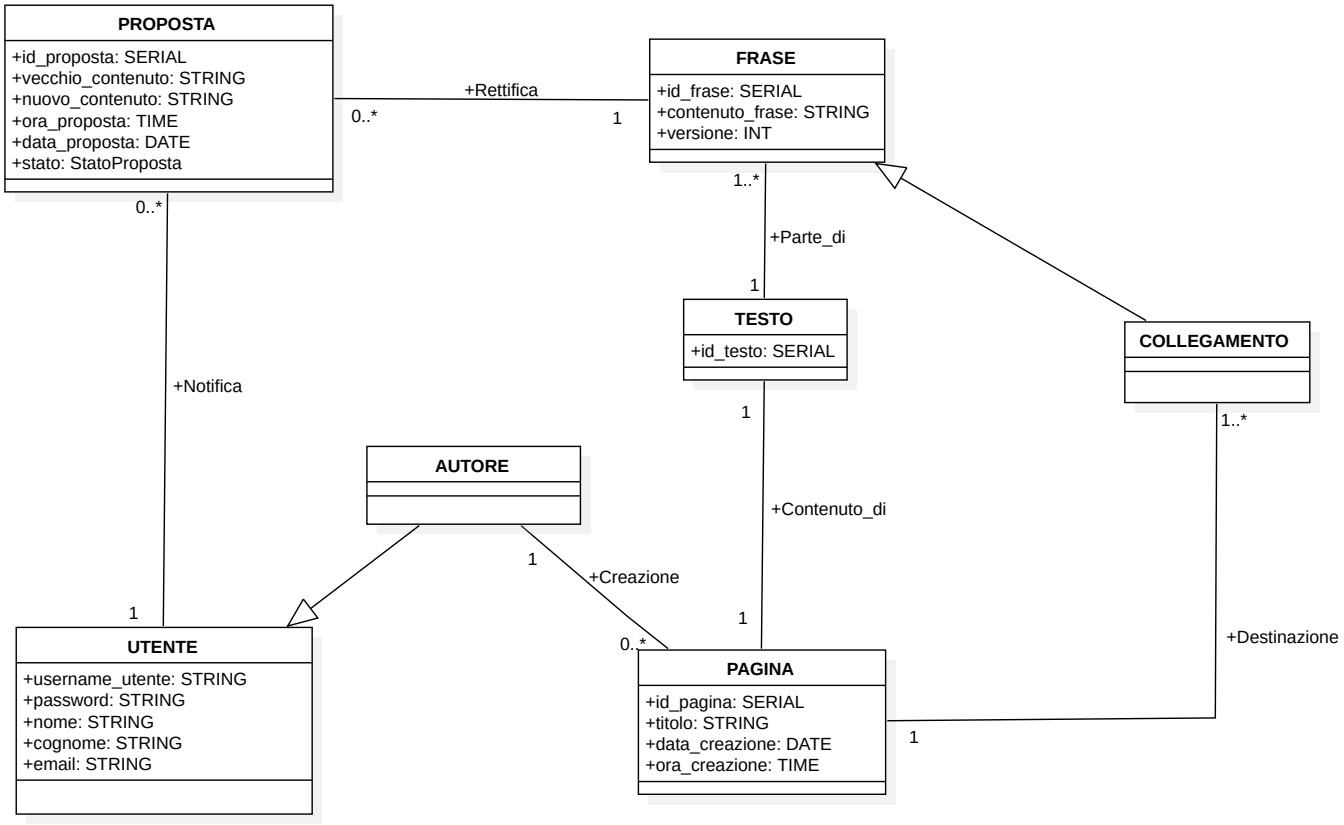
Gestione della pagina
La funzionalità di ricerca sulla pagina deve consentire agli utenti di verificare la presenza di titoli specifici e di trovare rapidamente un determinato articolo. Gli utenti con il ruolo di autore devono avere la capacità di creare nuovi titoli, associando a ciascun articolo appena creato una data e un'ora di creazione.

Gestione dell' utente
Nel sistema di gestione degli utenti, ciascun utente è dotato di un username univoco, una password sicura e un indirizzo email valido. In fase di registrazione, l'utente è tenuto a fornire anche il proprio nome e cognome, oltre a selezionare un ruolo che può essere tra "utente visitatore" e "autore".

Gestione della proposta
Nel sistema di gestione delle proposte, un utente ha la possibilità di richiedere la modifica di una frase. L'autore, responsabile della frase in questione, è incaricato di confermare o rifiutare la proposta avanzata dall'utente. Durante questo processo, è essenziale tenere traccia sia del contenuto originale ("vecchio") che di quello proposto in modifica ("nuovo").

1.2 Schema UML

WIKIPEDIA::Main



2 Ristrutturazione del modello concettuale

2.1 Analisi della ristrutturazione del class diagram

Durante questa fase, verranno apportate alcune modifiche al diagramma delle classi al fine di renederlo più adatto per una traduzione al modello logico

2.2 Analisi delle ridondanze

Non sono presenti ridondanze.

2.3 Analisi degli identificativi

In questa fase si sceglie un attributo per identificare univocamente le varie entità presenti nello schema precedente, in particolare:

Proposta:
Presenta l'attributo *id_proposta* che garantisce l'accesso e il riconoscimento immediato di ogni singola proposta che dovrà gestire l'autore.

Pagina:
Possiede l'attributo identificativo *id_pagina* utile per identificare una pagina specifica, senza dover consultare un insieme ampio di attributi

Testo:
Possiede l'attributo *id_testo* che serve per poter gestire facilmente i vari tipi di testo, soprattutto dopo una modifica.

Frase: Presenta l'attributo *id _frase* che permette la gestione delle frasi in modo semplice e senza ambiguità, gestione molto utile durante la modifica.

2.4 Rimozione degli attributi multivalore

Non sono presenti attributi multivalore.

2.5 Rimozione degli attributi composti

Non sono presenti attributi composti.

2.6 Rimozione delle gerarchie

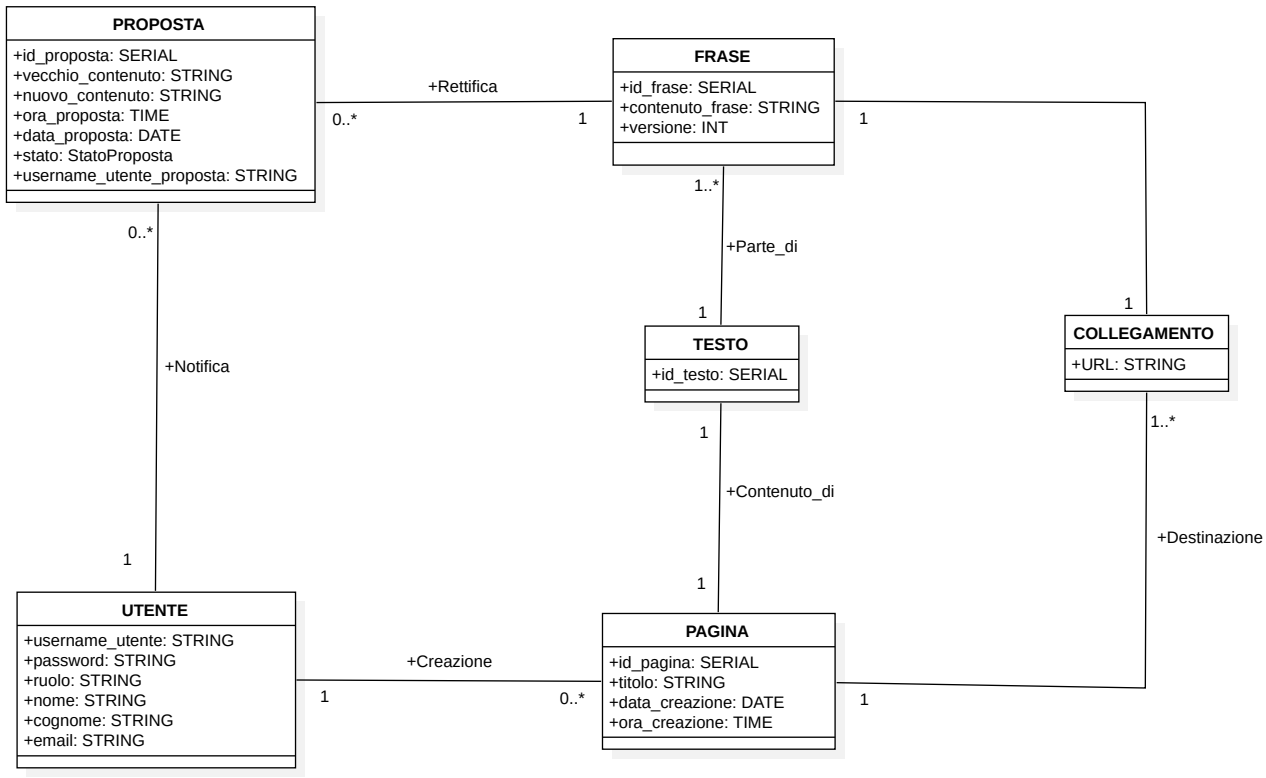
In questo diagramma sono presenti due generalizzazioni una **Frase composta** e una **Autore**, si è scelto di accorpare l'entità Autore figlia nell'entità Utente padre, ottendendo come risultato:

- **Utente**

avente tutti gli attributi della precedente entità con l'aggiunta dell'attributo
ruolo per gestire la differenza tra autore e utente generico.
Inoltre si è scelto di rendere associazione il legame che c'è tra Frase e Frase composta, ottenendo **Collegamento**

2.7 Schema UML Ristrutturato

WIKIPEDIA::Main



2.8 Dizionario delle entità e delle associazioni

Entità	Descrizione	Attributi
Utente	Se l'utente riveste il ruolo di autore, è responsabile della creazione della pagina; in caso contrario, si occupa esclusivamente delle modifiche	username_utente,password,nome,cognome, email,ruolo
Pagina	Fornisce dettagliate informazioni su qualsiasi argomento attraverso un testo descrittivo, offrendo al lettore una risorsa informativa completa e accessibile.	id_pagina,titolo,data_creazione,ora_creazione
Proposta	L'insieme di una o più frasi proposte dall'utente per modificare il testo originale sarà considerato e implementato nel caso in cui l'autore accetti tali modifiche.	id_proposta,vecchio_contenuto, nuovo_contenuto,ora_proposta, data_proposta,stato,username_utente_proposta
Frase	Essa costituisce la componente essenziale per la composizione del testo della pagina.	id_frase,contenuto_frase,versione
Testo	La funzione del testo è quella di memorizzare il contenuto principale di una pagina. Uno dei suoi compiti è quello di rappresentare una suddivisione ad esempio in frasi, soprattutto per facilitare le operazioni di modifica.	id_testo
Collegamento	Questa entità stabilisce una connessione tra una pagina e qualsiasi altra pagina avente lo stesso nome del collegamento.	URL

2.9 Dizionario delle associazioni

Associazioni	Descrizione
Notifica	Associazione uno-a-molti tra utente e proposta. Permette all'utente di proporre una modifica del testo originale che in caso venga accettata dall'autore sostituisce il vecchio contenuto con quello nuovo, specificando anche l'ora e la data in cui avviene la modifica. Un utente può modificare nessuna o più proposte, e nessuna o più proposte possono essere modificate da un utente.
Destinazione	Associazione uno-a-molti tra collegamento e pagina. In questa associazione teniamo conto dei collegamenti all'interno di una pagina che consentono l'accesso ad un'altra pagina che contiene un argomento inerente alla principale. Un collegamento può essere destinate ad una pagina, e una pagina può avere uno o più collegamenti.
Contenuto_di	Associazione uno-a-uno tra pagina e testo. Il testo rappresenta un frammento o una parte del contenuto di una specifica pagina. Una pagina contiene un testo e un testo contiene una pagina.
Creazione	Associazione uno-a-molti tra pagina e autore. L'autore crea da una a n pagine,la pagina è creata da un solo autore
Parte_di	Associazione uno-a-molti tra frase e testo. Una frase è contenuta in un solo testo,un testo contiene n frasi.
Rettifica	Associazione uno-a-molti tra proposta e frase. La proposta riguarda la modifica di una sola frase, la frase da modificare è contenuta in una sola proposta.

2.10 Dizionario dei vincoli

Nome vincolo	Descrizione
uk_email	Vincolo che garantisce che ogni indirizzo email sia unico.
chk_ruolo	Questo vincolo impedisce l'inserimento di valori diversi da "Autore" o "Utente" nella colonna 'ruolo', assicurando che solo ruoli specifici siano assegnati agli utenti.
chk_data_creazione	Il vincolo limita la data di creazione delle pagine in modo che non possano essere create con una data futura rispetto all'attuale data di sistema (CURRENT_DATE).
chk_vecchio_nuovo_contenuto_different	Questo vincolo impedisce che una proposta abbia il vecchio contenuto uguale al nuovo contenuto, garantendo che vi sia un cambiamento effettivo tra i due campi.
uk_titolo	Vincolo che impone l'unicità del titolo

3 Traduzione al Modello Logico

3.1 Mapping delle associazioni

Associazioni 1-N

Utente-Pagina: inserimento chiave di Utente in Pagina come chiave esterna.
Utente-Proposta:Inserimento chiave di Utente in Proposta come chiave estrena.
Aggiunta di una tabella modifica per lo storico delle proposte effettuate.
Frase-Proposta: inserimento chiave Frase in Proposta come chiave esterna.
Testo-Frase: Inserimento chiave Testo in Frase come chiave esterna.
Pagina-Collegamento: Inserimento chiave Pagina in Collegamento come chiave esterna.

Associazioni 1-1

Pagina-Testo: Inserimento chiave di Pagina in Testo come chiave esterna.
Frase-Collegamento: Inserimento chiave di Frase in Collegamento come chiave esterna,

3.2 Modello logico

Gli attributi sottolineati sono chiavi primarie, gli attributi con * alla fine sono chiavi esterne

UTENTE(username_utente, password, ruolo, nome, cognome, email)

PAGINA(id_pagina, titolo, data_crezione, ora_creazione, username_autore*)
username_autore→ utente.username_utente

FRASE(id_frase, contenuto_frase, versione, id_pagina* , id_testo*)

PROPOSTA(id_proposta, vecchio_
contenuto, nuovo_contenuto, stato, ora_proposta,
data_proposta, id_frase*,
username_utente_proposta*)
username_utente _proposta→ utente.username_utente

COLLEGAMENTO(URL, id_pagina*, id_frase*)
TESTO(id_testo, id_pagina*)

MODIFICA(id_proposta*, username _utente*, stato)

4 Progettazione Fisica

L'ultima fase della progettazione di una base di dati è la progettazione fisica
Prima di iniziare la progettazione occorre scegliere un DBMS (DataBase Management System) che implementi il modello dei dati dello schema logico.
Il DBMS in questo caso è **PostgresSQL**

4.1 Definizione delle tabelle

```
CREATE TYPE stato_proposta AS ENUM ('Accettata', 'Rifiutata','In attesa');

CREATE TABLE utente (
    username_utente varchar(15),
    password varchar(15) NOT NULL,
    ruolo varchar(6),
    nome varchar(20),
    cognome varchar(20),
    email varchar(255),
    PRIMARY KEY (username_utente)
);

CREATE TABLE pagina (
    id_pagina SERIAL PRIMARY KEY,
    titolo VARCHAR(100) NOT NULL,
    data_creazione DATE,
    ora_creazione TIME,
    username_autore VARCHAR(15),
    FOREIGN KEY (username_autore) REFERENCES utente(username_utente) ON DELETE CASCADE
);

CREATE TABLE testo (
    id_testo SERIAL  PRIMARY KEY,
    id_pagina Int,
    FOREIGN KEY (id_pagina) REFERENCES pagina(id_pagina) ON DELETE CASCADE
);

CREATE TABLE frase (
    Id_frase SERIAL PRIMARY KEY,
    contenuto_frase VARCHAR,
    versione int,
    id_pagina int,
    id_testo int,
    FOREIGN KEY (id_pagina) REFERENCES pagina(id_pagina) ON DELETE CASCADE,
    FOREIGN KEY (id_testo) REFERENCES testo(id_testo) ON DELETE CASCADE
);

CREATE TABLE proposta (
    id_proposta SERIAL PRIMARY KEY,
    vecchio_contenuto VARCHAR(255),
    nuovo_contenuto VARCHAR(255),
    stato stato_proposta,
    id_frase int,
    ora_proposta TIME,
    data_proposta DATE,
    username_utente_proposta VARCHAR(15),
    FOREIGN KEY (id_frase) REFERENCES frase(id_frase) ON DELETE CASCADE,
    FOREIGN KEY (username_utente_proposta) REFERENCES utente(username_utente) ON DELETE CASCADE
);
```

```
CREATE TABLE modifica (
    id_proposta INT NOT NULL,
    username_utente VARCHAR(15) NOT NULL,
    stato stato_proposta,
    PRIMARY KEY (id_proposta, username_utente,stato),
    FOREIGN KEY (id_proposta) REFERENCES proposta(id_proposta) ON DELETE CASCADE,
    FOREIGN KEY (username_utente) REFERENCES utente(username_utente) ON DELETE CASCADE
);

CREATE TABLE collegamento (
    id_pagina INT NOT NULL,
    id_frase INT NOT NULL,
URL VARCHAR (100) PRIMARY KEY,
    FOREIGN KEY (id_pagina) REFERENCES pagina(id_pagina) ON DELETE CASCADE,
    FOREIGN KEY (id_frase) REFERENCES frase(id_frase) ON DELETE CASCADE
);
```

4.2 Definizione dei vincoli

```
ALTER TABLE utente
ADD CONSTRAINT uk_email UNIQUE (email);

ALTER TABLE utente
ADD CONSTRAINT chk_ruolo CHECK (ruolo IN ('Autore', 'Utente'));

ALTER TABLE utente
ALTER COLUMN nome SET NOT NULL;

ALTER TABLE pagina
ADD CONSTRAINT chk_data_creazione CHECK (data_creazione <= CURRENT_DATE);
ALTER TABLE proposta
ADD CONSTRAINT chk_vecchio_nuovo_contenuto_different
CHECK (vecchio_contenuto <> nuovo_contenuto);
-- Aggiungi il vincolo di default a stato
ALTER TABLE proposta
ALTER COLUMN stato SET DEFAULT 'In attesa';

ALTER TABLE pagina
ADD CONSTRAINT uk_titolo UNIQUE (titolo);

ALTER TABLE proposta
ALTER COLUMN vecchio_contenuto TYPE VARCHAR(255);

ALTER TABLE proposta
ALTER COLUMN nuovo_contenuto TYPE VARCHAR(255);
```

4.3 Sequenze

Istruzioni SQL per reimpostare il valore iniziale delle sequeneze:

```
SELECT pg_get_serial_sequence('pagina', 'id_pagina');
ALTER SEQUENCE nome_sequenza RESTART WITH 1;

SELECT pg_get_serial_sequence('testo', 'id_testo');
ALTER SEQUENCE public.testo_id_testo_seq RESTART WITH 1;
```

```
SELECT pg_get_serial_sequence('frase', 'id_frase');
ALTER SEQUENCE public.frase_id_frase_seq RESTART WITH 1;

SELECT pg_get_serial_sequence('proposta', 'id_proposta');
ALTER SEQUENCE public.proposta_id_proposta_seq RESTART WITH 1;
```

4.4 Definizione dei trigger e delle funzioni

Registrazioni Utente:

Gestisce l'inserimento di nuovi utenti

```
--Funzione
CREATE OR REPLACE FUNCTION before_insert_utente()RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (SELECT 1 FROM utente WHERE username_utente = NEW.username_utente) THEN
        RAISE EXCEPTION 'L''utente con username % è già registrato', NEW.username_utente;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
--Trigger
CREATE TRIGGER before_insert_utente_trigger
BEFORE INSERT ON utente
FOR EACH ROW
EXECUTE FUNCTION before_insert_utente();
```

Verifica il formato della password

```
-- Funzione
CREATE OR REPLACE FUNCTION public.verifica_formato_password()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
AS $BODY$
BEGIN
    -- Verifica se la password ha almeno 6 caratteri, contiene almeno una lettera maiuscola,
    un numero o un simbolo
    IF LENGTH(NEW.password) >= 6 AND
       (NEW.password ~ '[0-9]' OR NEW.password ~ '[!@#$%^&*()]') AND
       (NEW.password ~ '[A-Z]')
    THEN
        -- Se la password è valida, restituisci la nuova riga per l'inserimento
        RETURN NEW;
    ELSE
        -- Se la password non è valida, lancia un'eccezione
        RAISE EXCEPTION 'La password deve contenere almeno 6 caratteri,
        almeno una lettera maiuscola, un numero o un simbolo speciale';
    END IF;
END;
$BODY$;
-- Trigger
CREATE TRIGGER before_insert_utente
BEFORE INSERT ON utente
FOR EACH ROW
EXECUTE FUNCTION public.verifica_formato_password();
```

Controlla il formato dell'email durante la registrazione dell'utente.

```
-- Funzione
CREATE OR REPLACE FUNCTION public.verifica_formato_email()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
AS $BODY$
BEGIN
    -- Verifica il formato dell'email
    IF NEW.email ~ '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}$' THEN
        -- Se l'email è valida, restituisci la nuova riga per l'inserimento
        RETURN NEW;
    ELSE
        -- Se l'email non è valida, lancia un'eccezione
        RAISE EXCEPTION 'Il formato dell''email non è valido';
    END IF;
END;
$BODY$;
-- Trigger
CREATE TRIGGER before_insert_utente_email
BEFORE INSERT ON utente
FOR EACH ROW
EXECUTE FUNCTION public.verifica_formato_email();
```

Inserimento e Gestione di Contenuti:

Inserisce titoli, testi e frasi in varie tabelle.

```
-- Funzione
CREATE OR REPLACE FUNCTION inserisci_titolo_e_frasi(
    titolo_input VARCHAR(100),
    testo_input TEXT,
    username_autore_input VARCHAR(15))
RETURNS VOID AS $$
DECLARE
    id_pagina_new INT;
    id_frase_new INT;
    id_testo_new INT;
    frase_parti TEXT[];
    frase_parte TEXT;
BEGIN
    -- Inserisci il titolo nella tabella pagina
    INSERT INTO pagina (titolo, data_creazione, ora_creazione, username_autore)
    VALUES (titolo_input, CURRENT_DATE, CURRENT_TIME, username_autore_input)
    RETURNING id_pagina INTO id_pagina_new;

    -- Inserisci il testo associato alla pagina nella tabella testo
    INSERT INTO testo (id_pagina)
    VALUES (id_pagina_new)
    RETURNING id_testo INTO id_testo_new;

    -- Divide il testo in frasi usando sia il punto che il carattere di a capo come delimitatori
    frase_parti := regexp_split_to_array(testo_input, '[.\n]+');

    -- Rimuovi eventuali stringhe vuote dall'array
    frase_parti := array_remove(frase_parti, '');
```

```
    -- Inserisci ogni frase nella tabella frase
    FOREACH frase_parte IN ARRAY frase_parti
    LOOP
        -- Rimuovi eventuali spazi iniziali e finali
        frase_parte := trim(frase_parte);

        -- Inserisci la frase nella tabella frase
        INSERT INTO frase (contenuto_frase, versione, id_pagina, id_testo)
        VALUES (frase_parte, 1, id_pagina_new, id_testo_new)
        RETURNING id_frase INTO id_frase_new;

    END LOOP;

    -- Esempio: Aggiungi un log delle frasi inserite
    RAISE NOTICE 'Inserite % frasi per la pagina %', array_length(frase_parti, 1), titolo_input;

    -- Esempio: Restituisci un messaggio di successo
    RAISE NOTICE 'Pagina "%", titolo "%", inserita con successo.', id_pagina_new, titolo_input;

END;
$$ LANGUAGE plpgsql;
```

Verifica se l'utente è un autore prima di consentirgli di creare pagine.

```
-- Funzione
CREATE OR REPLACE FUNCTION check_autore_per_pagina()
RETURNS TRIGGER AS $$
BEGIN
    IF (SELECT ruolo FROM utente WHERE username_utente = NEW.username_autore) != 'Autore' THEN
        RAISE EXCEPTION 'Solo gli autori possono creare pagine.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
-- Trigger
CREATE TRIGGER trigger_check_autore_per_pagina
BEFORE INSERT ON pagina
FOR EACH ROW
EXECUTE FUNCTION check_autore_per_pagina();
```

Impedisce la creazione di pagine con data e ora future.

```
-- Funzione
CREATE OR REPLACE FUNCTION trigger_prevent_data_ora_futura()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.data_creazione > CURRENT_DATE THEN
        RAISE EXCEPTION 'Impossibile creare una pagina con data di creazione futura.';
    ELSIF NEW.data_creazione = CURRENT_DATE AND NEW.ora_creazione > CURRENT_TIME THEN
        RAISE EXCEPTION 'Impossibile creare una pagina con data e ora di creazione future.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
-- Trigger
CREATE TRIGGER before_insert_prevent_data_ora_futura
BEFORE INSERT ON pagina
FOR EACH ROW
EXECUTE FUNCTION trigger_prevent_data_ora_futura();
```

Verifica se il vecchio contenuto di una frase corrisponde al contenuto attualmente presente nel database.

```
    CREATE OR REPLACE FUNCTION public.check_vecchio_contenuto_exists(
p_id_frase integer,
p_vecchio_contenuto character varying)
    RETURNS boolean
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE PARALLEL UNSAFE
AS $BODY$
DECLARE
    contenuto_attuale VARCHAR;
BEGIN
    -- Ottieni il contenuto attuale della frase
    SELECT contenuto_frase INTO contenuto_attuale
    FROM frase
    WHERE id_frase = p_id_frase;

    -- Restituisci true se il vecchio_contenuto corrisponde al contenuto attuale
    RETURN contenuto_attuale = p_vecchio_contenuto;
END;
$BODY$;
```

Restituisce le frasi associate a una pagina specifica.

```
-- Funzione
    CREATE OR REPLACE FUNCTION visualizza_frasi_per_pagina(
    titolo_input VARCHAR(20))
RETURNS TABLE (
    contenuto_frase VARCHAR,
    versione INT,
    data_creazione DATE,
    ora_creazione TIME,
    autore_frase VARCHAR
) AS $$
BEGIN
    RETURN QUERY
    SELECT
        f.contenuto_frase,
        f.versione,
        p.data_creazione,
        p.ora_creazione,
        p.username_autore AS autore_frase
    FROM
        pagina p
    JOIN
        frase f ON p.id_pagina = f.id_pagina
    WHERE
        p.titolo = titolo_input
    ORDER BY
        f.id_frase; -- Ordina per id_frase invece che per versione
END;
$$ LANGUAGE plpgsql;
```

Elimina proposte in attesa tranne quella accettata dopo l'aggiornamento.

```
-- Funzione
    CREATE OR REPLACE FUNCTION public.trigger_elimina_proposte_in_attesa()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
BEGIN
    -- Se la proposta è stata accettata e lo stato precedente era "In attesa"
    IF NEW.stato = 'Accettata' AND COALESCE(OLD.stato, 'In attesa') = 'In attesa' THEN
        -- Qui eliminiamo tutte le proposte in attesa tranne quella accettata
        DELETE FROM proposta
        WHERE id_frase = NEW.id_frase AND stato = 'In attesa' AND id_proposta <> NEW.id_proposta;
    END IF;

    RETURN NEW;
END;
$BODY$;
-- Trigger
CREATE TRIGGER trigger_elimina_proposte_in_attesa
AFTER UPDATE ON proposta
FOR EACH ROW
EXECUTE FUNCTION public.trigger_elimina_proposte_in_attesa();
```

Gestione delle Proposte e Modifiche:

Restituisce le proposte ordinate per data e ora.

```
-- Funzione
    CREATE OR REPLACE FUNCTION public.get_proposte_ordered_by_data_ora(
    autore_destinatario varchar(15)
)
RETURNS TABLE(
    id_proposta integer,
    vecchio_contenuto varchar(255),
    nuovo_contenuto varchar(255),
    stato stato_proposta,
    id_frase integer,
    ora_proposta time without time zone,
    data_proposta date,
    username_utente_proposta varchar(15),
    titolo_pagina varchar(20),
    autore_pagina varchar(15)
)
LANGUAGE 'plpgsql'
COST 100
VOLATILE PARALLEL UNSAFE
ROWS 1000
AS $BODY$
BEGIN
    RETURN QUERY
    SELECT
        p.id_proposta,
        p.vecchio_contenuto,
        p.nuovo_contenuto,
        p.stato,
        p.id_frase,
        p.ora_proposta,
        p.data_proposta,
        p.username_utente_proposta,
        pa.titolo AS titolo_pagina,
        ua.username_utente AS autore_pagina
    FROM
        proposta AS p
    INNER JOIN frase AS f ON p.id_frase = f.id_frase
    INNER JOIN pagina AS pa ON f.id_pagina = pa.id_pagina
    INNER JOIN utente AS ua ON pa.username_autore = ua.username_utente
    -- Rimuovi la condizione di filtro sull'autore
    -- WHERE
    --     p.username_utente_proposta = autore_destinatario
    ORDER BY
        p.data_proposta ASC,
        p.ora_proposta ASC;
END;
$BODY$;
```

Aggiorna il contenuto della frase dopo l'accettazione di una proposta.

```
-- Funzione
    CREATE OR REPLACE FUNCTION after_update_proposta_trigger()
RETURNS TRIGGER AS $$
BEGIN
    -- Se la proposta è accettata, aggiorna il contenuto della frase
    IF NEW.stato = 'Accettata' THEN
        UPDATE frase
        SET contenuto_frase = NEW.nuovo_contenuto
        WHERE id_frase = OLD.id_frase;  -- Utilizza l'ID della frase precedente (OLD)

    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
-- Trigger
CREATE TRIGGER after_update_proposta
AFTER UPDATE ON proposta
FOR EACH ROW
WHEN (OLD.stato IS DISTINCT FROM NEW.stato)
EXECUTE FUNCTION after_update_proposta_trigger();
```

Inserisce una nuova riga nella tabella modifica dopo l'aggiornamento di una proposta.

```
-- Funzione
    CREATE OR REPLACE FUNCTION post_update_proposta_trigger()
RETURNS TRIGGER AS $$
BEGIN
    -- Inserisci una nuova riga nella tabella modifica
    INSERT INTO modifica (id_proposta, username_utente, stato)
    VALUES (NEW.id_proposta, NEW.username_utente_proposta, NEW.stato);

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Trigger dopo l'aggiornamento su proposta
CREATE TRIGGER post_update_proposta
AFTER UPDATE ON proposta
FOR EACH ROW
WHEN (OLD.stato IS DISTINCT FROM NEW.stato)
EXECUTE FUNCTION post_update_proposta_trigger();
```

La funzione è chiamata dopo l'inserimento di una nuova riga nella tabella proposta.

```
-- Funzione
    CREATE OR REPLACE FUNCTION public.after_insert_proposta_trigger()
RETURNS TRIGGER
LANGUAGE plpgsql
AS $$
BEGIN
    -- Chiamiamo la funzione di modifica direttamente dopo l'inserimento della proposta
    PERFORM public.modifica_diretta_frase
    (NEW.id_frase, NEW.nuovo_contenuto, NEW.username_utente_proposta);

    RETURN NEW;
END;
```

```
$$;

-- Crea il trigger AFTER INSERT sulla tabella proposta
CREATE TRIGGER trigger_after_insert_proposta
AFTER INSERT ON proposta
FOR EACH ROW
EXECUTE FUNCTION public.after_insert_proposta_trigger();
```

Se l'utente è l'autore della pagina associata alla frase, la funzione esegue un'operazione di aggiornamento sulla tabella proposta. Imposta lo stato della proposta associata alla frase a 'Accettata', presumibilmente indicando che la modifica proposta è stata accettata.

```
-- Funzione
    CREATE OR REPLACE FUNCTION public.modifica_diretta_frase(
    p_id_frase integer,
    p_nuovo_contenuto character varying,
    p_username_autore character varying)
RETURNS void
LANGUAGE plpgsql
AS $$
BEGIN
    -- Stampa di debug per controllare i valori dei parametri
    RAISE NOTICE 'modifica_diretta_frase - p_id_frase: %, p_nuovo_contenuto:%,
    p_username_autore: %', p_id_frase, p_nuovo_contenuto, p_username_autore;

    -- Verifica se l'utente è l'autore della pagina con quella frase
    IF EXISTS (
        SELECT 1
        FROM pagina AS p
        JOIN frase AS f ON p.id_pagina = f.id_pagina
        WHERE f.id_frase = p_id_frase AND p.username_autore = p_username_autore
    ) THEN
        -- Effettua la modifica dello stato della proposta
        UPDATE proposta
        SET stato = 'Accettata'
        WHERE id_frase = p_id_frase;

        -- Stampa di debug per confermare l'aggiornamento
        RAISE NOTICE 'Stato della proposta associata alla frase con id_frase % aggiornato
        a ''accettata''.', p_id_frase;
    END IF;

    -- Stampa di debug per confermare il completamento della funzione
    RAISE NOTICE 'modifica_diretta_frase completata';

END;
$$;
```

Analisi e Statistiche Utente:

Calcola il rapporto tra modifiche accettate e totale modifiche per un utente.

```
-- Funzione
    CREATE OR REPLACE FUNCTION calcola_ratio_modifiche(username_utente_input VARCHAR(15))
RETURNS DECIMAL(10,2) AS $$
DECLARE
    total_modifiche INT;
    modifiche_accettate INT;
    ratio DECIMAL(10,2);
```

```
BEGIN
    -- Calcola il totale delle modifiche fatte dall'utente
    SELECT COUNT(*) INTO total_modifiche
    FROM modifica
    WHERE username_utente = username_utente_input;

    -- Calcola il totale delle modifiche accettate dall'utente
    SELECT COUNT(*) INTO modifiche_accettate
    FROM modifica
    WHERE username_utente = username_utente_input AND stato = 'Accettata';

    -- Calcola il rapporto tra modifiche accettate e totale modifiche
    IF total_modifiche > 0 THEN
    ratio := modifiche_accettate::DECIMAL(10,2) / total_modifiche;
ELSE
    ratio := 0.0;
END IF;


    RETURN ratio;
END;
$$ LANGUAGE PLpgSQL;
```

Calcola il totale delle pagine create da un autore.

```
-- Funzione
    CREATE OR REPLACE FUNCTION calcola_totale_pagine_autore(username_autore_input VARCHAR(15))
RETURNS INT AS $$
DECLARE
    total_pagine INT;
BEGIN
    -- Calcola il totale delle pagine realizzate dall'autore
    SELECT COUNT(*) INTO total_pagine
    FROM pagina
    WHERE username_autore = username_autore_input;

    RETURN total_pagine;
END;
$$ LANGUAGE plpgsql;
```

Gestione delle Frasi e Versioni:

Trova l'ID di una frase dato il titolo della pagina e il contenuto della frase.

```
-- Funzione
    CREATE OR REPLACE FUNCTION trova_id_frase_con_titolo(
    IN titolo_pagina VARCHAR,
    IN vecchio_contenuto_frase VARCHAR
)
RETURNS INT
AS $$
DECLARE
    frase_id INT;
BEGIN
    SELECT f.id_frase INTO frase_id
    FROM frase f
    JOIN pagina p ON f.id_pagina = p.id_pagina
    WHERE p.titolo = titolo_pagina
        AND f.contenuto_frase = vecchio_contenuto_frase;
```

```
    RETURN frase_id;
END;
$$ LANGUAGE plpgsql;
```

Incrementa la versione di una frase prima dell'aggiornamento.

```
-- Funzione
    CREATE OR REPLACE FUNCTION public.trigger_incrementa_versione()
    RETURNS TRIGGER
    LANGUAGE plpgsql
AS $$
BEGIN
    NEW.versione := COALESCE(OLD.versione, 0) + 1;
    RETURN NEW;
END;
$$;
-- Trigger
CREATE TRIGGER before_update_frase
BEFORE UPDATE ON frase
FOR EACH ROW
EXECUTE FUNCTION public.trigger_incrementa_versione();
```