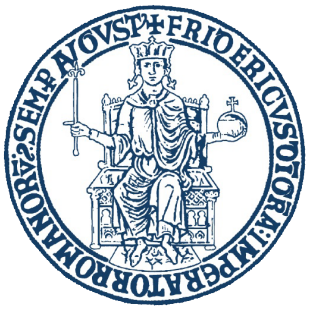


UNIVERSITÀ DEGLI STUDI
DI NAPOLI FEDERICO II

PROGETTAZIONE E IMPLEMENTAZIONE DI UNA BASE DI DATI RELAZIONALE PER LA GESTIONE DI UNA WIKI

Nani Aurelio & Ranavolo Davide

Anno Accademico 2023/2024

Indice

1 Proggettazione Concettuale

1.1 Analisi dei Requisiti

Alla lettura inizale della traccia si possono giá individuare le prime entitá:

> *"Una pagina di una wiki ha un titolo e un testo. Ogni pagina è creata da un determinato autore. Il testo è composto di una sequenza di frasi. Il sistema mantiene traccia anche del giorno e ora nel quale la pagina è stata creata. Una frase può contenere anche un collegamento. Ogni collegamento è caratterizzato da una frase da cui scaturisce il collegamento e da un'altra pagina destinazione del collegamento."*

La *pagina* é l'entitá in cui vengono salvati i titoli delle pagine create dagli autori. Ogni qualvolta un autore crea una nuova pagina viene anche salvato il quando all'interno di una variabile (*dataOraCreazione*). Una *pagina* ha un *testo*, il quale é composto da *frasi*, ogni frase ha una posizione all'interno del testo, la quale viene salvata (*numerazione*). Una frase senza un testo non puó esistere, per questo é un'entitá debole che deve essere collegata con *pagina* per essere identificata univocamente. Una frase puó avere un *collegamento* ad un altra pagina. Poiché il numero di frasi che hanno un collegamento é minore del numero di quelle che non lo hanno, abbiamo reso la classe associativa *collegamento* un'entitá, cosí da non avere un numero elevato di valori a NULL.

> *"Il testo può essere modificato da un altro utente del sistema, che seleziona una frase, scrive la sua versione alternativa (modifica) e prova a proporla. La modifica proposta verrà notificata all'autore del testo originale la prossima volta che utilizzerà il sistema."*

Ogni utente avrá i propri dati generici, *login* e *password* questi ultimi necessari per poter accedere al proprio account. Per poter gestire le varie azioni che possono effettuare all'interno della Wiki, gli *utenti* si distinguono in due tipi: *l'utente generico*, che accede, e puó visualizzare pagine, proporre modifiche e creare testi (dopo aver creato una pagina diventa autore); *l'autore*, che ha tutte le funzionalitá dell'utente, ma in piú puó valutare modifiche, proposte appunto da un altro *utente* del sistema.
Avremo quindi una *frase originale* e varie *frasi proposte* di modifica, sulla frase originale. Ogni qualvolta avviene una proposta di modifica, verrá inviata una *notifica* all'autore del testo, nella quale verrá segnata l'ora, la data di invio e il titolo della pagina di cui viene proposta la modifica.

"L'autore potrà vedere la sua versione originale e la modifica proposta. Egli potrà accettare la modifica (in quel caso la pagina originale diventerà ora quella con la modifica apportata), rifiutare la modifica (la pagina originale rimarrà invariata). La modifica proposta dall'autore verrà memorizzata nel sistema e diventerà subito parte della versione corrente del testo. Il sistema mantiene memoria delle modifiche proposte e anche delle decisioni dell'autore (accettazione o rifiuto). Nel caso in cui si fossero accumulate più modifiche da rivedere, l'autore dovrà accettarle o rifiutarle tutte nell'ordine in ordine dalla più antica alla più recente.
Ad esempio, il testo originale del 3 novembre ore 10 del Prof. Tramontana potrebbe essere "traccia del progetto di OO", mentre il 3 novembre alle ore 11 il Prof. Barra potrà modificarlo in "traccia del progetto di OO e BD" e il Prof. Tramontana potrà accettare la modifica alle ore 12. A quel punto la versione corrente sarà quella proposta dal Prof. Barra. In alternativa il Prof. Tramontana potrebbe, alle ore 12 rifiutare la modifica del Prof. Barra e attuare invece la modifica "traccia del progetto di OO e BD gruppo 2" che diventerà subito parte della versione corrente (essendo la modifica stata disposta dall'autore della pagina)."

Per poter tener traccia delle scelte dell'autore riguardo le modifiche proposte, alle modifiche é stato aggiunto un attributo *stato* che a seconda del suo valore indica se la modifica ancora non é stata visonata, se é stata acettata o se é stata rifiutata. Inoltre si terrá anche traccia della data e dell'ora in cui l'autore fará la sua scelta con lo scopo di ricostruire il testo piú recente. Se la modifica é proposta dall'autore del testo, lo stato verrá impostato direttamente sul valore di accettazione.

"Gli utenti generici del sistema potranno cercare una pagina e il sistema mostrerà la versione corrente del testo e i collegamenti. Gli autori dovranno prima autenticarsi fornendo la propria login e password. Tutti gli autori potranno vedere tutta la storia di tutti i testi dei quali sono autori e di tutti quelli nei quali hanno proposto una modifica."

Con l'aggiunta degli attributi nome utente e password, gli utenti accederanno al loro account cosi da poter proporre modifiche e creare pagine. Inoltre gli autori potranno visionare tutte le versioni dei loro testi e le modifiche proposte ai propri testi e a quelli di altri, visualizzando anche lo stato della modifica.

1.2 Schema Concettuale

1.2.1 Modello Entity/Relationship non ristrutturato

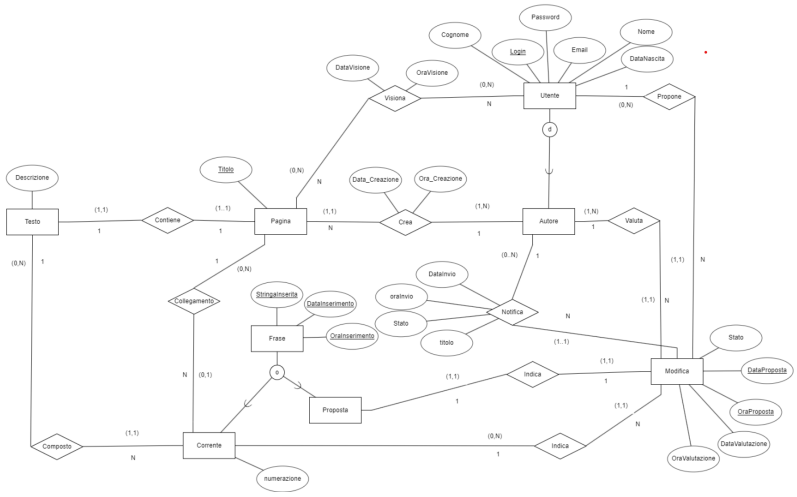


Figure 1: Modello E/R

1.2.2 Modello UML non ristrutturato

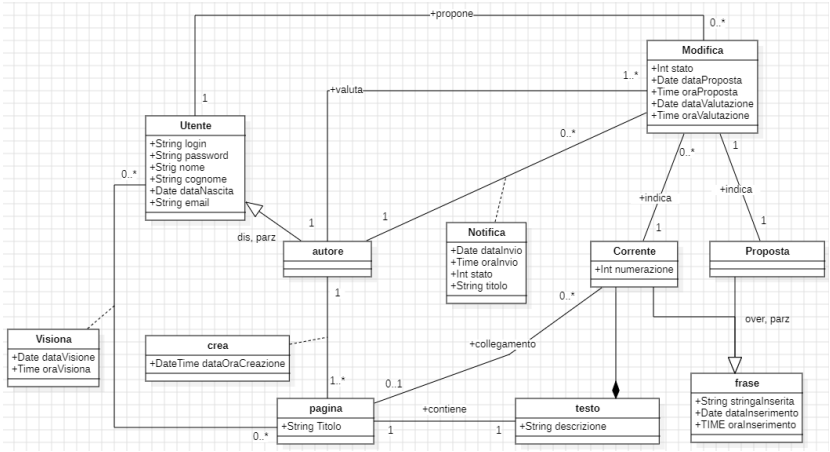


Figure 2: Modello UML

1.3 Dizionario delle entità

Entità	Descrizione	Attributi
Pagina	Rappresenta una determinata pagina della wiki.	titolo(String): intestazione della pagina

Utente	Rappresenta l'utente registrato che puo visionare le pagine di una wiki, e proporre modifiche ai testi	nome(String): nome dell'autore;
		cognome(String): cognome dell' autore;
		dataNascita(Date): indica la data in cui è nato l'autore;
		login(String): è l'identificativo da dover indicare per poter accedere come autore (username);
		password(String): stringa di caratteri necessaria per l'identificazione univoca da parte del sistema a cui si chiede l'accesso;
		email(String): inirizzo di posta elettronica dell'autore.

Autore	Specializzazione di Utente che puo creare pagine wiki e valutare proposte di altri autori o utenti	
Testo	Rappresenta l'insieme di frasi che compongono la pagina	descrizione(String): breve introduzione del testo
Frase	Rappresenta l'espressione scritta dall'autore	stringaInserita(String): contenuta tra una qualsiasi lettera maiuscola o minuscola e tra un segno di punteggiatura(./,/;/?/!); dataInserimento(Date): indica la data in cui viene inserita la frase; oraInserimento(Time): indica l'ora in cui viene inserita la frase;
Corrente	Specializzazione di Frase. Rappresenta la frase presente all'interno della versione originale del testo	numerazione(int): indica la posizione della frase all'interno del testo.
Proposta	Specializzazione di Frase. Rappresenta la frase proposta come modifica a una frase presente in un testo.	

Entità	Descrizione	Attributi
Modifica Proposta	Rappresenta la modifica proposta da un utente, essa potrà essere accettata o rifiutata dall'autore della pagina wiki	dataProposta(Date): indica la data in cui viene inviata la proposta di modifica; oraProposta(Time): indica l'ora in cui viene inviata la proposta di modifica; dataValutazione(Date): indica la data in cui la modifica viene valutata dall'autore; oraValutazione(Time): indica l'ora in cui la modifica viene valutata dall'autore; Stato(int): indica se la modifica è rifiutata (-1), in attesa (0) o accettata (1).

1.4 Dizionario delle associazioni

Associazioni	Descrizione
Crea	Associazione uno-a-molti, tra *Autore* e *Pagina*. Un autore può creare una o più pagine, mentre una pagina può essere creata da un solo autore.
Collegamento	Associazione uno-a-molti, tra *Pagina* e *Corrente*. Una frase può essere collegata a una sola pagina, mentre una pagina può avere più frasi che hanno un collegamento ad essa.
Contiene	Associazione uno-a-uno, tra *Pagina* e *Testo*. Una pagina al suo interno ha un unico testo e un testo è contenuto in un unica pagina.
Propone	Associazione uno-a-molti, tra *Autore* e *Modifica*. Un utente potrebbe proporre una o più modifiche di un testo, mentre una proposta di modifica può essere effettuata da un solo utente
Visiona	Associazione molti-a-molti, tra *Utente* e *Pagina*. Un utente può visionare una o più pagine, una pagina può essere vista da uno o più utenti
Valuta	Associazione uno-a-molti, tra *Autore* e *Modifica*. Un autore può valutare una o più modifiche di un testo, mentre una modifica può essere valutata da un solo autore
Notifica	Associazione uno-a-molti, tra *Autore* e *Modifica*. Un autore può essere notificato su più modifiche di testo, mentre una modifica può essere notificata ad un solo autore

Associazioni	Descrizione
Indica	Associazione uno-a-molti, tra *Corrente* e *Modifica*. Una frase corrente può avere più modifiche, mentre una modifica deve indicare un'unica frase corrente
Indica	Associazione uno-a-uno, tra *Modifica* e *Proposta*. Una Modifica indica una sola frase proposta, una proposta di modifica appartiene ad un unica modifica.
Composizione	Associazione uno-a-molti, tra *Testo* e *Corrente*. Corrente é un'entitá debole. Un testo é composto da piú Corrente, mentre una Corrente é collegata ad un solo testo.

2 Ristrutturazione del modello concettuale

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

2.1 Analisi delle ridondanze

L'entità **Notifica**, che si viene a creare dall'associazione tra *Utente* e *ModificaProposta*, presenta attributi che possono essere derivati direttamente dalle entità *ModificaProposta* e *Pagina*: l'attributo *data* non è altro che dataProposta come anche *ora* che è oraProposta, anche l'attributo *titolo* può essere ricavato da un semplice join tra modificaProposta e Pagina, e infine lo *stato*, può essere recuperato anch'esso dall'entità *ModificaProposta*.
Quindi, tenendo in considerazione le seguenti osservazioni, è stato deciso di eliminare l'entità **Notifica**.

2.2 Analisi degli identificativi

In questa fase andremo a scegliere un attributo per identificare univocamente le varie entitá presenti nello schema precedente, in particolare:

1. Per l'entità **Pagina** il solo titolo non basta per identificare univocamente una tupla, per questo viene agiunto un attributo *idPagina*;

2. Nel caso dell'entità **Utente**, è stato scelto di inserire un ulteriore attributo *idUtente*, anche se presenta al suo interno un attributo *login* che è chiave candidata;

3. Sarebbe possibile identificare una **ModificaProposta** tramite un insieme piuttosto ampio di attributi, ma per semplificare l'accesso all'indice, è stato aggiunto un attributo *IdModifica*;

4. **FraseCorrente** viene identificata, essendo entità debole, tramite gli attributi *stringainserita*, *numerazione* e *idPagina* (chiave primaria di Pagina)

2.3 Rimozione degli attributi multivalore

Non sono presenti attributi multivalore

2.4 Rimozione degli attributi composti

Non sono presenti attributi composti

2.5 Rimozione delle gerarchie

In questo diagramma sono presenti 2 generalizzazione e 3 relative specializzazioni. In particolare:

Per la generalizzazione **Utente**, si é scelto di accorpare l'entità figlia nell'entità padre, ottenendo come risultato:

- Un unica entità **Utente**, nella quale è stato aggiunto un attributo *ruolo* per indicare se la persona che accede è utente o autore.

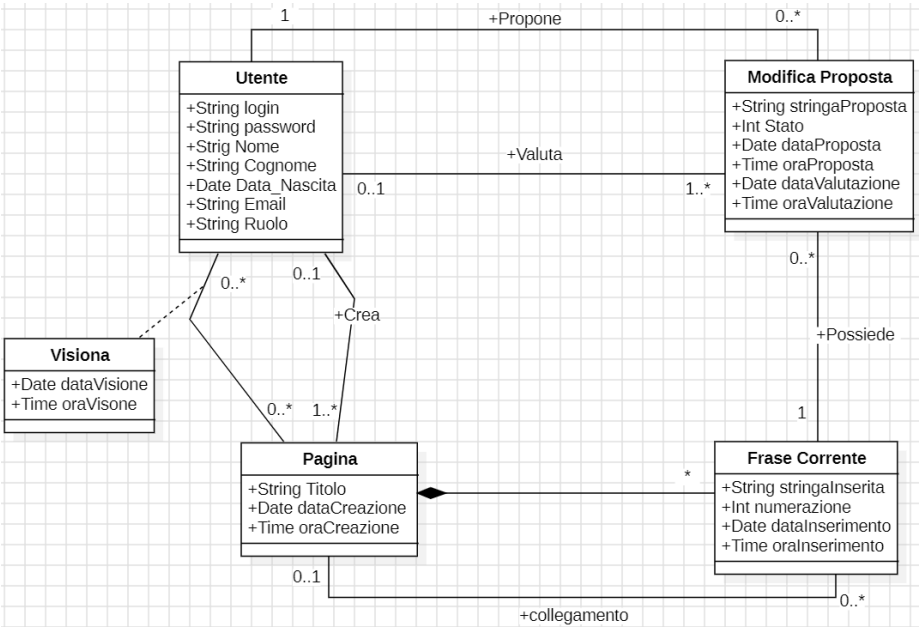
Per la generalizzazione **Frase**, si é scelto di accorpare l'entità padre nelle entità figlie, ottenendo come risultato:

- Un entità **FraseCorrente** avente tutti gli attributi di **Frase** più gli attributi della precedente entità **Corrente**
- Analogamente, l'entità **FraseProposta** avrà come attributi, quelli di **Frase**.

2.6 Partizione/Accorpamento delle Associazioni

In questo Class Diagram, non ristrutturato, sono presenti due associazioni 1:1, **Pagina - Testo** e **Modifica - FraseProposta**. Abbiamo deciso di accorparle entrambe perchè in entrambi i casi le operazioni accedono a dati presenti su entrambe le entità.

2.7 UML Diagram Ristrutturato



2.8 Dizionario delle Classi

Entità	Attributi	Descrizione
Utente	idUtente nome cognome login password email dataNascita ruolo	classe che mantiene i dati inseriti dall'utente quando si registra.
Pagina	idPagina titolo dataOraCreazione	classe che mantiene le informazioni riguardo le pagine
Visiona	dataVisione oraVisiona	classe che mette in associazione Utente e Pagina, memorizzando quando una pagina viene vista da un determinato utente
Crea		classe che mette in relazione Utente e Pagina

Entità	**Attributi**	**Descrizione**
FraseCorrente	stringaInserita dataInserimento oraInserimento numerazione idPagina	classe che mantiene le informazioni riguardo le frasi originali della pagina
Collegamento		mette in relazione l'entità Pagina e FraseCorrente
Composizione		è una composizione, un pagina è formata da un inseme di frasi, e una frase senza pagina non può essere riconosciuta univocamente
Possiede		classe che mette in relazione fraseCorrente e ModificaProposta
ModificaProposta	idModifica stringaProposta dataproposta oraProposta dataValutazione oraValutazione stringaInserita numerazione idPagina	classe che mantiene le informazioni delle proposte di modifica fatte su una frase
propone		classe che mette in realazione Utente e ModificaProposta
valuta		classe che mette in relazione Utente e ModificaProposta

2.9 Dizionario delle Associazioni

Associazione	**Classe coinvolte**	**Descrizione**
Crea	Utente e Pagina	Un utente crea una pagina, un utentnte può creare una o più pagine, una pagina può essere creata da un solo utente. [1,0..*]
Visiona	Utente e Pagina	Un utente puo o non puo visonare una o piu pagine, e una pagina puo o non puo essere visionata da uno o piu utenti. [0..*,0..*]
Collegamento	Pagina e FraseCorrente	Una pagina puo avere come non avere dei collegamenti , e una frase puo collegarsi o non collegarsi a una pagina. [0..1,0..*]
Possiede	FraseCorrente e ModificaProposta	Una FraseCorrente puo avere una o piu modifiche proposte, una ModificaProposta appartiene a una e una sola frase. [1,0..*]
Propone	Utente e ModificaProposta	Un utente può proporre delle modifiche, una Modifica può essere proposta da un solo utente. [1,0..*]
Valuta	Utente e ModificaProposta	Un Utente può valutare una o più ModificheProposte, una ModificaProposta può essere valutata da un solo Utente. [0..1,1..*]
Composizione	Pagina e FraseCorrente	Indica la composizione di una o più Frasi in una Pagina.[1,0..*]

3 Traduzione al Modello Logico

3.1 Mapping associazioni

3.1.1 Assocoiazioni 1-N

- **CREA → Utente - Pagina:** inserimento chiave di *Utente* in *Pagina* come chiave esterna
- **COLLEGAMENTO→ Pagina - FraseCorrente:** essendo una parteciopazione parziale, é stato scelto di creare una tabella intermedia, inserendo al suo interno le chiavi di *Pagina* e *FraseCorrente* come chiavi esterne. La loro composizione forma la chiave primaria.
- **POSSIEDE → FraseCorrente - ModificaProposta:** inserimento chiave di *FraseCorrente* in *ModificaProposta* come chiave esterna
- **COMPOSIZIONE → Pagina - FraseCorrente:** essendo *FraseCorrente* un'entitá debole si inserisce la chiave di *Pagina* in *FraseCorrente* come chiave esterna, diventando poi parte della chiave.
- **PROPONE → Utente - ModificaProposta:** inserimento chiave di *Utente* in *ModificaProposta* come chiave esterna
- **VALUTA → Utente - ModificaProposta:** inserimento chiave di *Utente* in *ModificaProposta* come chiave esterna

3.1.2 Assocoiazioni N-N

Per ognuna di queste relazioni, si inseriscono le chiavi delle due associazioni come chiavi esterne.

Associazione	**Relazione**	**Associazione**
Utente	Visiona	Pagina

3.2 Modello logico

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

Utente (idUtente, nome, cognome, login, password, email, dataNascita, ruolo)

Pagina (idPagina, Titolo, dataOraCreazione, idUtente*)
idUtente → Utente.idUtente

Visiona (dataVisione, oraVisione, idUtente*, idPagina*)
idUtente → Utente.idUtente
idPagina → Pagina.idPagina

FraseCorrente (StringaInserita, numerazione, idPagina*, dataInserimento, oraInserimento)
idPagina → Pagina.idPagina

ModificaProposta (idModifica, stringaProposta, dataProposta, oraProposta, dataValutazione
oraValutazione, stato, StringaInserita*, numerazione*, idPagina*,
utenteP*, autoreV*)
StringaInserita → FraseCorrente.StringaInserita
numerazione → FraseCorrente.numerazione
idPagina → FraseCorrente.idPagina
utenteP → Utente.idUtente
autoreV → Utente.idUtente

Collegamento (StringaInserita*, numerazione*, idPagina*, paginaCollegata*)
StringaInserita → FraseCorrente.StringaInserita
numerazione → FraseCorrente.numerazione
idPagina → FraseCorrente.idPagina
paginaCollegata → Pagina.idPagina

4 Proggettazione Fisica

4.1 Definizioni delle tabelle

4.1.1 UTENTE

```
CREATE TABLE UTENTE
(
    idUtente SERIAL,
    login varchar(25) UNIQUE,
    password varchar(25),
    nome varchar(30),
    cognome varchar(30),
    dataNascita Date,
    email varchar(50) check(email LIKE '%@%.%'),
    ruolo char(6) DEFAULT 'Utente',
    CONSTRAINT pk_utente PRIMARY KEY(idUtente)
)
```

Tabella 1: Utente

4.1.2 PAGINA

```
CREATE TABLE Pagina
(
    idPagina SERIAL,
    titolo varchar(80),
    dataOraCreazione TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    idAutore chiavi_primarie NOT NULL,
    constraint pk_pagina PRIMARY KEY(idPagina),
    constraint Fk_utente FOREIGN KEY(idAutore) REFERENCES utente(
        idUtente)
)
```

Tabella 2: Pagina

4.1.3 VISIONA

```
CREATE TABLE Visiona
(
    idUtente chiavi_primarie,
    idPagina chiavi_primarie,
    dataVisone DATE DEFAULT CURRENT_DATE,
    oraVisione TIME DEFAULT CURRENT_TIME,
    CONSTRAINT Fk_Utente FOREIGN KEY(idUtente) REFERENCES UTENTE(
        idUtente),
    CONSTRAINT Fk_pagina FOREIGN KEY(idPagina) REFERENCES PAGINA(
        idpagina)
)
```

Tabella 3: Visiona

4.1.4 FRASECORRENTE

```
CREATE TABLE fraseCorrente
(
    StringaInserita varchar(1000),
    numerazione int,
    dataInserimento DATE DEFAULT CURRENT_DATE,
    oraInserimento TIME DEFAULT CURRENT_TIME,
    idPagina chiavi_primarie,
    CONSTRAINT pk_frase PRIMARY KEY(StringaInserita, numerazione,
        idPagina),
    CONSTRAINT fk_pagina FOREIGN KEY(idPagina) REFERENCES PAGINA(
        idPagina)
)
```

Tabella 4: FraseCorrente

4.1.5 MODIFICAPROPOSTA

```
CREATE TABLE ModificaProposta
(
    idModifica Serial,
    stringaProposta varchar(1000),
    stato int DEFAULT 0,
    dataProposta DATE DEFAULT CURRENT_DATE,
    oraProposta TIME DEFAULT CURRENT_TIME,
    dataValutazione DATE,
    oraValutazione TIME,
    UtenteP chiavi_primarie NOT NULL,
    AutoreV chiavi_primarie NOT NULL,
    StringaInserita varchar(1000),
    numerazione int,
    idPagina chiavi_primarie,
    CONSTRAINT pk_modifica PRIMARY KEY(idModifica),
    CONSTRAINT fk_stringaInserita FOREIGN KEY(StringaInserita,
        numerazione, idPagina) REFERENCES fraseCorrente(
        stringaInserita, numerazione, idPagina) DELETE ON CASCADE,
    CONSTRAINT fk_AutoreV FOREIGN KEY(AutoreV) REFERENCES UTENTE(
        idUtente),
    CONSTRAINT fk_UtenteP FOREIGN KEY(utenteP) REFERENCES UTENTE(
        idUtente)
)
```

Tabella 5: ModificaProposta

4.1.6 COLLEGAMENTO

```
    CREATE TABLE COLLEGAMENTO
(
  stringaInserita varchar(1000),
  numerazione INTEGER,
  idPagina chiavi_primarie,
  paginaCollegata chiavi_primarie,
  PRIMARY KEY(StringaInserita, numerazione, idPagina,
      paginaCollegata),
  CONSTRAINT pk_fraseInserita FOREIGN KEY(StringaInserita,
      numerazione, idPagina) REFERENCES fraseCorrente(
      stringaInserita, numerazione, idPagina)
)
```

Tabella 6: Collegamento

4.2 Definizione dei trigger

4.2.1 diventaAutore

Quando un utente crea per la prima volta una pagina all'interno della Wiki, il suo ruolo passa da quello di utente, ruolo impostato di default, al ruolo di autore, che gli permetterà poi di visualizzare tutte le informazioni riguardo alla pagina da lui scritta.

```
CREATE OR REPLACE FUNCTION diventaAutore() RETURNS TRIGGER
AS $$
DECLARE
    ruoloUtente Utente.ruolo%TYPE;
BEGIN
   SELECT ruolo INTO ruoloUtente
   FROM UTENTE
   WHERE idUtente = new.idAutore;

   IF ruoloUtente != 'Autore' THEN
            UPDATE Utente SET ruolo = 'Autore' WHERE idutente = new.
                idautore;
        RETURN new;
    ELSE
        RETURN NULL;
   END IF;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER diventaAutore
AFTER INSERT ON pagina
FOR EACH ROW
    EXECUTE FUNCTION diventaAutore();
```

Trigger 7: diventaAutore

4.2.2 ModificaAutore

Quando la modifica a una frase di una pagina viene fatta direttamente dall'autore della stessa, la modifica viene accettata senza doverla valutare in un secondo momento. Il trigger quindi modifica l'attributo stato, dandogli il valore 1, cioè accettato.

```
CREATE OR REPLACE FUNCTION ModificaAutore ()
RETURNS TRIGGER
AS $$
BEGIN
    IF new.utentep = new.autorev THEN
        UPDATE modificaproposta SET stato = 1 WHERE idmodifica =
            new.idmodifica;
        RETURN new;
    ELSE
        RETURN NULL;
    END IF ;
END;
$$
language plpgsql;

CREATE OR REPLACE TRIGGER ModificaAutore
AFTER INSERT ON modificaproposta
FOR EACH ROW
EXECUTE FUNCTION ModificaAutore()
```

Trigger 8: ModificaAutore

4.2.3 inserimentofrase

Le frasi all'interno di ogni pagina sono in sequenza, che viene mantenuta tramite l'attributo numerazione che salva la posizione della frase nella pagina. Il trigger non fa altro che gestire la sequenza delle frasi, valorizzando l'attributo numerazione ogni volta che una frase viene inserita, verificando il valore dell'ultima frase nella pagina e inserendo all'interno dell'attributo della frase inserita il valore verificato più 1.

```
CREATE OR REPLACE FUNCTION inserimento_frase() RETURNS TRIGGER
AS $$
DECLARE
valore INTEGER = -1;
BEGIN
    SELECT numerazione INTO valore
    FROM frasecorrente
    WHERE new.idpagina = idpagina
    ORDER BY numerazione DESC LIMIT 1;
    IF valore != -1 THEN
            new.numerazione = valore+1;
    ELSE
        new.numerazione = 0;
    END IF;
```

```
15      RETURN new;
16  END;
17  $$
18  language plpgsql;
19
20  CREATE OR REPLACE TRIGGER inserimento_frase
21  BEFORE INSERT ON frasecorrente
22  FOR EACH ROW
23  EXECUTE FUNCTION inserimento_frase()
```

Trigger 9: inserimento$_frase$

4.2.4 settaggioDataOraValutazione

Ogni volta che un autore valuta una proposta di modifica, accettandola o rifiutandola, viene salvata la data e l'ora della valutazione. Quindi il trigger non fa altro che valorizzare gli attributi *dataValutazione* e *oraValutazione* della tabella *ModificaProposta* ogni qualvolta l'attributo *stato* passa da 0, modifica non valutata, a 1, modifica accettata, o a -1, modifica rifiutata.

```
1  CREATE OR REPLACE settaggioDataOraValutazione() RETURNS TRIGGER
2  AS $$
3  BEGIN
4    UPDATE modificaProposta SET dataValutazione = NOW() WHERE
         idModifica = old.idModifica;
5    UPDATE modificaProposta SET oraValutazione = NOW() WHERE
         idModifica = old.idModifica;
6    RETURN new;
7  END;
8  $$ LANGUAGE plpgsql;
9
10 CREATE OR REPLACE TRIGGER settaggioDataOraValutazione
11 AFTER UPDATE modificaProposta
12 FOR EACH ROW
13 WHEN (old.stato = 0)
14   EXECUTE FUNCTION settaggioDataOraValutazione();
```

Trigger 10: settaggioDataOraValutazione

4.3 Definizione delle procedure

4.3.1 crea_pagina

Inserendo nei parametri della procedura il titolo scelto per la pagina da creare, il nome utente dell'utente che vuole creare la pagina e il testo scritto, la procedura crea una nuova tupla all'interno della tabella Pagina e divide il testo inserito in frasi (ogni qualvolta trova un . / , / ; / ! / ? viene creata una sottostringa) e inserendo ogni frase all'interno della tabella fraseCorrente.

```
CREATE OR REPLACE PROCEDURE crea_pagina(titoloScelto IN TEXT,
    nomeUtente IN TEXT, testoPagina IN TEXT)
AS $$
DECLARE
  frase varchar(100000) := '';
  pos_prec INTEGER := 1;
  pagina pagina.idPagina%TYPE;
  autore utente.idUtente%TYPE;
  pos INTEGER := 1;
  num INTEGER := 0;
  H RECORD;
  J RECORD;
    k RECORD;
BEGIN
  SELECT idUtente INTO autore FROM utente WHERE LOWER(login) =
      LOWER(nomeUtente) LIMIT 1;

  INSERT INTO PAGINA(titolo, idAutore) VALUES (titoloScelto, autore
      );
  SELECT idPagina INTO pagina FROM PAGINA WHERE PAGINA.titolo =
      titoloScelto AND PAGINA.idAutore = autore AND PAGINA.
      dataOraCreazione = now();
  LOOP
    EXIT WHEN pos > LENGTH(testoPagina);
    IF SUBSTRING(testoPagina FROM pos FOR 1) IN ('.', ',', ';', '!'
        , '?') THEN
            frase := REGEXP_REPLACE(SUBSTRING(testoPagina FROM
                pos_prec FOR (pos - pos_prec)+1), '^\s+', '');
    pos_prec := pos + 1;
    CALL inserimento_frase(frase, pagina);
    RAISE NOTICE '%', frase;
    END IF;
    pos := pos + 1;
  END LOOP;
  IF pos < LENGTH(testoPagina) THEN
        frase := REGEXP_REPLACE(SUBSTRING(testoPagina FROM pos_prec
            FOR LENGTH(testoPagina)), '^\s+', '');
    frase := frase || '.';
    CALL inserimento_frase(frase, pagina);
    RAISE NOTICE '%', frase;
  END IF;
END;
```

Procedura 11: notificaM

4.3.2 inserimento_frase

Indicata una frase e la pagina dove inserire la frase, la procedura si occupa dell'inseriemto della frase sfruttando anche il trigger *inserimentofrase* per la numerazione.

```
CREATE OR REPLACE PROCEDURE inserimento_frase(stringa IN varchar
    (100), pagina IN pagina.idPagina%TYPE)
AS $$
DECLARE
valore INTEGER;
BEGIN
  SELECT numerazione INTO valore
  FROM frasecorrente
  WHERE pagina = idpagina
  ORDER BY numerazione DESC LIMIT 1;
  INSERT INTO frasecorrente (stringainserita, idpagina) VALUES (
      stringa, pagina);
END;
$$
LANGUAGE PLPGSQL;
```

Procedura 12: inserimento_frase

4.3.3 ricerca_testi

Indicando delle parole chiavi o un frase, la procedura cerca e mostra tutte le pagine (titolo e testo) che contengono all'interno del loro titolo la parola chiave o la frase indicata.

```
CREATE OR REPLACE PROCEDURE ricerca_testi(titoloInserito IN varchar
    (100))
AS $$
DECLARE
  frase varchar(100000) := '';
  stringa varchar(1000);
  pagina pagina.idpagina%type;
    num INTEGER;
    dataVal DATE;
    sProposta varchar(1000);
  H RECORD;
  J RECORD;
    k RECORD;
BEGIN
  FOR H IN (SELECT idPagina, titolo FROM PAGINA WHERE LOWER(titolo)
       LIKE '%' || LOWER(titoloInserito) || '%' ORDER BY titolo)
      LOOP
  RAISE INFO '%', 'TITOLO PAGINA:';
  RAISE INFO '%', H.titolo;

  CREATE TABLE tempVersione(
        stringaIns varchar(1000),
        numerazione INTEGER );
```

```
21      FOR k IN (SELECT stringaInserita, numerazione, datainserimento
            FROM frasecorrente WHERE H.idpagina = idpagina ORDER BY
            numerazione ASC) LOOP
22        SELECT numerazione, dataValutazione, stringaProposta INTO num
              , dataVal, sProposta
23        FROM modificaproposta
24        WHERE H.idpagina = idpagina AND k.stringainserita =
              stringainserita AND numerazione = k.numerazione AND stato
               = 1 AND datavalutazione <= K.datainserimento
25        ORDER BY dataValutazione DESC, oraValutazione DESC LIMIT 1;
26        IF num IS NULL THEN
27          INSERT INTO tempVersione VALUES (k.StringaInserita, k.
                numerazione);
28        ELSE
29          INSERT INTO tempVersione VALUES (sProposta, num);
30        END IF;
31      END LOOP;
32      FOR J IN (SELECT stringaIns FROM tempVersione) LOOP
33        stringa := J.stringaIns;
34        frase := frase || ' ' || stringa;
35      END LOOP;
36      RAISE INFO '%', frase;
37      RAISE INFO '%', '';
38      frase := '';
39      DROP TABLE tempVersione;
40    END LOOP;
41
42  END;
43  $$
44  LANGUAGE PLPGSQL;
```

Procedura 13: ricerca_testi

4.3.4 ricostruzione_versione

Indicando una determinata pagina e una determinata data questa procedura ricostruisce la pagina nella versione in cui era nella data passata per parametro.

```
1   CREATE OR REPLACE PROCEDURE ricostruzione_versione(
        paginaSelezionata IN pagina.idpagina%type, dataInsertita IN
        DATE)
2   AS $$
3   DECLARE
4       frase varchar(1000);
5       num INTEGER;
6       dataVal DATE;
7     dataCreazione DATE;
8       sProposta varchar(1000);
9     testo TEXT:= '';
10      k RECORD;
11  BEGIN
12      CREATE TABLE tempVersione
13      (
14          stringa TEXT,
15          numerazione INTEGER
```

```
    );
  SELECT dataoracreazione::DATE INTO dataCreazione FROM pagina
      WHERE paginaSelezionata = idpagina;

  IF dataInsertita < dataCreazione THEN
    RAISE INFO 'data precedente alla creazione del testo';
    RAISE INFO 'TESTO ORIGINALE:';
  END IF;
    FOR k IN (SELECT stringaInserita, numerazione, datainserimento
        FROM frasecorrente WHERE paginaSelezionata = idpagina) LOOP
    SELECT numerazione, dataValutazione, stringaProposta INTO num,
        dataVal, sProposta
        FROM modificaproposta
        WHERE paginaSelezionata = idpagina AND k.stringainserita =
            stringainserita AND numerazione = k.numerazione AND
            stato = 1 AND datavalutazione <= dataInsertita
        ORDER BY dataValutazione DESC, oraValutazione DESC LIMIT 1;
        IF num IS NULL THEN
            INSERT INTO tempVersione VALUES (k.StringaInserita, k.
                numerazione);
        ELSE
            INSERT INTO tempVersione VALUES (sProposta, num);
        END IF;
    END LOOP;

 FOR k IN (SELECT * FROM tempVersione ORDER BY numerazione ASC)
     LOOP
      testo := testo || k.stringa || ' ';
    END LOOP;
    RAISE INFO '%', testo;
    DROP TABLE tempVersione;
END;
$$
LANGUAGE PLPGSQL;
```

Procedura 14: ricostruzione_versione

4.3.5 visiona_notifiche

Dato un idUtente come parametro, questa procedura mostra le notifiche ricevute dall'utente in ordine di data (dalla data meno recente alla più recente).

```
CREATE OR REPLACE PROCEDURE visiona_notifiche(autore IN utente.
    idUtente%TYPE)
AS $$
DECLARE
  notifiche INTEGER := 0;
  fraseProposta varchar(100):= '';
  K RECORD;
  titoloNotifica TEXT;
BEGIN
  notifiche = numero_notifiche(autore);

  RAISE INFO '%', 'Ci sono ' || notifiche || ' notifiche da
      visualizare';
```

```
12    RAISE INFO '%', '';
13    IF notifiche > 0 THEN
14      FOR K IN (SELECT * FROM modificaProposta WHERE stato = 0 AND
            autorev = autore ORDER BY DataProposta ASC, oraProposta ASC
            )
15      LOOP
16        SELECT stringaProposta INTO fraseProposta FROM
              modificaProposta WHERE autorev = autore AND idPagina = K.
              idPagina AND numerazione = K.numerazione AND stato = 1
              AND datavalutazione <= K.dataProposta AND oravalutazione
              < K.oraProposta ORDER BY datavalutazione DESC,
              oravalutazione DESC LIMIT 1;
17
18        RAISE INFO '%', '-----------------------------------------'
              ;
19        SELECT titolo INTO titoloNotifica FROM PAGINA WHERE idPagina
              = K.idpagina;
20        RAISE INFO '%', 'TITOLO PAGINA: ' || titoloNotifica;
21        IF fraseProposta != '' THEN
22          RAISE INFO '%', 'FRASE SELEZIONATA: ' || fraseproposta;
23        ELSE
24          RAISE INFO '%', 'FRASE SELEZIONATA: ' || K.stringaInserita;
25        END IF;
26
27        RAISE INFO '%', 'FRASE PROPOSTA: ' || K.stringaProposta;
28        RAISE INFO '%', '';
29      END LOOP;
30      RAISE INFO '%', '-----------------------------------------';
31    END IF;
32  END;
33  $$
34  LANGUAGE PLPGSQL;
```

Procedura 15: visiona_notifiche

4.3.6 proponi_modifica

Dato un utente, la pagina, la frase da modificare, la frase da proporre e la numerazione della frase, la procedura inserisce all'interno della tabella *ModificaProposta* la proposta effettuata dall'utente.

```
1   CREATE OR REPLACE PROCEDURE proponi_modifica(loginUtente IN Utente.
        login%TYPE, paginaSelezionata IN pagina.idpagina%TYPE,
        fraseDaModificare IN TEXT, fraseDaProporre IN TEXT, numFrase IN
         INTEGER )
2   AS $$
3   DECLARE
4       autore Utente.idUtente%TYPE;
5       utente Utente.idUtente%TYPE;
6       fraseInserita TEXT := '';
7   BEGIN
8       SELECT idUtente INTO utente FROM Utente WHERE login =
            loginUtente;
9       SELECT idAutore INTO autore FROM Pagina WHERE idPagina =
            paginaSelezionata;
```

```
10
11      SELECT stringaInserita INTO fraseInserita FROM FraseCorrente
            WHERE idpagina = paginaSelezionata AND fraseDaModificare =
            stringaInserita AND numerazione = numFrase LIMIT 1;
12
13    IF fraseInserita IS NULL THEN
14          SELECT stringaInserita INTO fraseInserita FROM
                ModificaProposta WHERE idpagina = paginaSelezionata AND
                 fraseDaModificare = stringaProposta AND numerazione =
                numFrase LIMIT 1;
15    END IF;
16
17      INSERT INTO ModificaProposta(stringaProposta, utenteP, autoreV,
             stringaInserita, numerazione, idPagina)
18      VALUES (fraseDaProporre, utente, autore, fraseInserita,
            numFrase, paginaSelezionata);
19
20      RAISE INFO '%', 'Operazione completata con successo';
21  END;
22  $$
23  LANGUAGE PLPGSQL;
```

Procedura 16: proponi_modifica

4.3.7 visiona_testo

Dato il titolo di una pagina e il nomeutente dell'utente che visualizza, la procedura non fa altro che mostrare uno dei testi che ha all'interno del suo titolo la parola chiave/frase passata per parametro nella sua versione più recente. Inoltre aggiunge una tupla all'interno della tabella visiona per tener traccia delle pagine visualizzate dall'utente.

```
1   CREATE OR REPLACE PROCEDURE visiona_testo(titoloInserito IN varchar
        (100), nomeUtente IN Utente.login%TYPE)
2   AS $$
3   DECLARE
4     frase varchar(100000) := '';
5     titolotesto varchar(100);
6     stringa varchar(1000);
7     pagina pagina.idpagina%type;
8       num INTEGER;
9       dataVal DATE;
10      sProposta varchar(1000);
11    visualizzatore utente.idUtente%TYPE;
12      k RECORD;
13  BEGIN
14  CREATE TABLE tempVersione
15      (
16          stringaIns varchar(1000),
17          numerazione INTEGER
18      );
19
20    SELECT idPagina, titolo INTO pagina, titolotesto FROM PAGINA
          WHERE LOWER(titolo) LIKE '%' || LOWER(titoloInserito) || '%'
```

```
      ORDER BY titolo DESC LIMIT 1;
  RAISE INFO '%', titolotesto;

    FOR k IN (SELECT stringaInserita, numerazione, datainserimento
        FROM frasecorrente WHERE pagina = idpagina ORDER BY
        numerazione ASC) LOOP
        SELECT numerazione, dataValutazione, stringaProposta INTO
            num, dataVal, sProposta
        FROM modificaproposta
        WHERE pagina = idpagina AND k.stringainserita =
            stringainserita AND numerazione = k.numerazione AND
            stato = 1 AND datavalutazione <= K.datainserimento
        ORDER BY dataValutazione DESC, oraValutazione DESC LIMIT 1;
        IF num IS NULL THEN
            INSERT INTO tempVersione VALUES (k.StringaInserita, k.
                numerazione);
        ELSE
            INSERT INTO tempVersione VALUES (sProposta, num);
        END IF;
    END LOOP;

  FOR K IN (SELECT stringaIns FROM tempVersione) LOOP
  stringa := K.stringaIns;
  frase := frase || ' ' || stringa;
  END LOOP;

  SELECT idUtente INTO visualizzatore FROM UTENTE WHERE login =
      nomeUtente;
  INSERT INTO VISIONA(idUtente, idPagina) VALUES (visualizzatore,
      pagina);
  RAISE INFO '%', frase;
    DROP TABLE tempVersione;
END;
$$
LANGUAGE PLPGSQL;
```

Procedura 17: visiona_testo

4.3.8 numerazione_frasi

Dato l'id di una pagina la procedura mostra ogni frase del testo affiancato dalla sua posizione, in ordine crescente.

```
CREATE OR REPLACE PROCEDURE numerazione_frasi(paginaSelezionata IN
    pagina.idpagina%type)
AS $$
DECLARE
    frase varchar(1000);
    num INTEGER;
    dataVal DATE;
  dataCreazione DATE;
    sProposta varchar(1000);
    k RECORD;
BEGIN
    CREATE TABLE tempVersione
```

```
12      (
13          stringa TEXT,
14          numerazione INTEGER
15      );
16      FOR k IN (SELECT stringaInserita, numerazione, datainserimento
            FROM frasecorrente WHERE paginaSelezionata = idpagina) LOOP
17      SELECT numerazione, dataValutazione, stringaProposta INTO num,
            dataVal, sProposta
18          FROM modificaproposta
19          WHERE paginaSelezionata = idpagina AND k.stringainserita =
                stringainserita AND numerazione = k.numerazione AND
                stato = 1
20          ORDER BY dataValutazione DESC, oraValutazione DESC LIMIT 1;
21          IF num IS NULL THEN
22              INSERT INTO tempVersione VALUES (k.StringaInserita, k.
                    numerazione);
23          ELSE
24              INSERT INTO tempVersione VALUES (sProposta, num);
25          END IF;
26      END LOOP;
27
28   FOR k IN (SELECT * FROM tempVersione ORDER BY numerazione ASC)
         LOOP
29        RAISE INFO '%', K.numerazione || ' - ' || k.stringa;
30   END LOOP;
31
32      DROP TABLE tempVersione;
33  END;
34  $$
35  LANGUAGE PLPGSQL;
```

Procedura 18: ricostruzione_testo

4.3.9 frase_collegamento

Inserito un id di una pagina, la procedura mostra le frasi che hanno un collegamento ad un'altra pagina.

```
1   CREATE OR REPLACE PROCEDURE frase_collegamento(paginaSelezionata
        Pagina.idPagina%TYPE)
2   AS $$
3   DECLARE
4     titoloPagina TEXT;
5     num INTEGER;
6     dataVal DATE;
7     sProposta TEXT;
8     pCollegata chiavi_primarie;
9     controllo INTEGER := -1;
10    K RECORD;
11
12  BEGIN
13    SELECT titolo INTO titoloPagina FROM PAGINA WHERE idpagina =
          paginaSelezionata;
14    RAISE INFO '%', 'TITOLO PAGINA: ' || titoloPagina;
```

```
15      FOR k IN (SELECT stringaInserita, numerazione, datainserimento
            FROM frasecorrente WHERE paginaSelezionata = idpagina ORDER
             BY numerazione ASC) LOOP
16          SELECT paginacollegata INTO pCollegata FROM COLLEGAMENTO
                WHERE paginaSelezionata = idpagina AND k.
                stringainserita = stringainserita AND numerazione = k.
                numerazione;
17      IF pCollegata IS NOT NULL THEN
18      controllo = 1;
19        SELECT numerazione, dataValutazione, stringaProposta INTO num
              , dataVal, sProposta
20        FROM modificaproposta
21        WHERE paginaSelezionata = idpagina AND k.stringainserita =
              stringainserita AND numerazione = k.numerazione AND stato
               = 1 AND datavalutazione <= K.datainserimento
22        ORDER BY dataValutazione DESC, oraValutazione DESC LIMIT 1;
23        IF num IS NULL THEN
24          SELECT titolo INTO titoloPagina FROM PAGINA WHERE idpagina
                = pcollegata;
25          RAISE INFO '%', k.numerazione || ' - ' || k.stringainserita
                || ' -> ' || titoloPagina;
26        ELSE
27          SELECT titolo INTO titoloPagina FROM PAGINA WHERE idpagina
                = pcollegata;
28          RAISE INFO '%', k.numerazione || ' - ' || k.stringainserita
                || ' -> ' || titoloPagina;
29        END IF;
30      END IF;
31      END LOOP;
32      IF controllo = -1 THEN
33        RAISE INFO '%', 'nessuna frase ha un collegamento ad una
              pagina';
34      END IF;
35  END
36  $$
37  LANGUAGE PLPGSQL;
```

Procedura 19: ricostruzione_testo

4.3.10 visiona_ModificheProposte

Dato un idUtente come parametro, questa procedura mostra le modifiche effettuate dall'utente sui testi in ordine di data (dalla data meno recente alla più recente).

```
1   CREATE OR REPLACE PROCEDURE visiona_ModificheProposte(utente IN
        chiavi_primarie)
2   AS $$
3   DECLARE
4     modifiche INTEGER := 0;
5     fraseProposta varchar(100):= '';
6     K RECORD;
7     titoloNotifica TEXT;
8   BEGIN
9     modifiche = numero_modifiche(utente);
```

```
10
11    RAISE INFO '%', 'Hai proposto ' || modifiche || ' modifiche';
12    RAISE INFO '%', '';
13    IF modifiche > 0 THEN
14      FOR K IN (SELECT * FROM modificaProposta WHERE utentep = utente
            ORDER BY DataProposta ASC, oraProposta ASC)
15      LOOP
16        SELECT stringaProposta INTO fraseProposta FROM
              modificaProposta WHERE utentep = utente AND idPagina = K.
              idPagina AND numerazione = K.numerazione AND stato = 1
              AND datavalutazione <= K.dataProposta AND oravalutazione
              < K.oraProposta ORDER BY datavalutazione DESC,
              oravalutazione DESC LIMIT 1;
17
18        RAISE INFO '%', '----------------------------------------';
19        SELECT titolo INTO titoloNotifica FROM PAGINA WHERE idPagina
              = K.idpagina;
20        RAISE INFO '%', 'TITOLO PAGINA: ' || titoloNotifica;
21        IF fraseProposta != '' THEN
22          RAISE INFO '%', 'FRASE SELEZIONATA: ' || fraseproposta;
23        ELSE
24          RAISE INFO '%', 'FRASE SELEZIONATA: ' || K.stringaInserita;
25        END IF;
26
27        RAISE INFO '%', 'FRASE PROPOSTA: ' || K.stringaProposta;
28
29        IF K.stato = 0 THEN
30          RAISE INFO '%', 'STATO: non valutata';
31        ELSIF K.stato = 1 THEN
32          RAISE INFO '%', 'STATO: accettata';
33        ELSE
34          RAISE INFO '%', 'STATO: rifutata';
35        END IF;
36
37        RAISE INFO '%', '';
38      END LOOP;
39      RAISE INFO '%', '----------------------------------------';
40    END IF;
41  END;
42  $$
43  LANGUAGE PLPGSQL;
```

Procedura 20: visiona_ModificheProposte

4.4 Definizione delle funzioni

4.4.1 numero_notifiche

Questa funzione restituisce il numero di notifiche ancora non visualizzate dall'autore passato per parametro.

```
CREATE OR REPLACE FUNCTION numero_notifiche(autore IN utente.
    idUtente%TYPE) RETURNS INTEGER
AS $$
DECLARE
    notifiche INTEGER := 0;
BEGIN
    SELECT COUNT(*) INTO notifiche
    FROM MODIFICAPROPOSTA
    WHERE stato = 0 AND autorev = autore;

    RETURN notifiche;
END;
$$
LANGUAGE PLPGSQL;
```

Funzione 21: numero _notifiche

4.4.2 numero_modifiche

Questa funzione restituisce il numero di modifiche proposte dall'autore passato per parametro.

```
CREATE OR REPLACE FUNCTION numero_modifiche(utente IN
    chiavi_primarie) RETURNS INTEGER
AS $$
DECLARE
    modifiche INTEGER := 0;
BEGIN
    SELECT COUNT(*) INTO modifiche
    FROM MODIFICAPROPOSTA
    WHERE utentep = utente;

    RETURN modifiche;
END;
$$
LANGUAGE PLPGSQL;
```

Funzione 22: numero _modifiche