Documetazione Data Base

Sistema di gestione di una pagina wiki

Raffaele Raia e Florindo Zecconi
Università di Napoli Federico II
23 febbraio 2024

Indice

1 Progettazione Concettuale

1.1 Analisi dei Requisiti

1.1.1 Traccia

"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. La pagina può contenere anche dei collegamenti. Ogni collegamento è caratterizzato da una frase da cui scaturisce il collegamento e da un'altra pagina destinazione del collegamento.

Il testo può essere modificato da un altro utente del sistema, che seleziona una o più delle frasi, 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.

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 dalla più antica alla più recente."

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.

1.1.2 Preambolo

La prima fase della progettazione consiste nell'analisi dei requisiti. Qui verranno identificate le informazioni principali che porteranno allo sviluppo della struttura del data base per la Wiki. Durante l'analisi saranno individuate le diverse entità, relazioni, alcuni vincoli (ci sarà una sezione apposita per i vincoli) e comportamenti del data base.

1.1.3 Ipotesi

La nostra base di dati immagazzinerà i dati per la gestione di una wiki. Prima di analizzare il problema faremo alcune ipotesi, su alcuni punti non specificati dalla Traccia:

- una pagina può essere creata solo da un autore, se qualcun' altro vorrà contribuire manderà una richiesta di inserimento o modifica al autore della pagina.
- un utente oltre a proporre una modifica può anche richiedere un inserimento.
- un autore oltre ad avere la possibilità di modificare ha anche la possibilità di inserire.

1.1.4 Componenti trovati

- **Pagina**: conterrà il titolo, le generalità dell'autore e la data del ultima modifica.

- **Autore**: Possiamo pensare intuitivamente che non tutti gli utenti della *wiki* siano autori. Ci potranno essere degli utenti che fruiscono soltanto, quest'ultimi possono anche decidere di non creare mai nessuna *wiki* e di conseguenza non essere classificati come un autore. Generalizzando possiamo individuare una *Superclasse(padre)* di nome *Utente* e una *Sottoclasse(figlio)* di nome *Autore*. Un utente non deve essere obbligatoriamente un autore, ma se decidesse di creare una *wiki* allora dovrà essere visto come autore *(Parziale/disgiunta)*. La *Superclasse(padre)* conterrà *nome, cognome, email, password e genere.*

- **Frase**: Un insieme di frasi formerà una *pagina*. Ogni frase avrà un testo e una posizione (per capire chi verrà prima di un'altra). Notiamo che un *link* è una frase che fa riferimento a una *pagina*. Possiamo generalizzare individuando una *Superclasse(padre)* di nome *Frase* e una *Sottoclasse(figlio)* di nome *link*. Una frase può non essere un link, ma se si riferisce a una pagina allora diventerà link*(Parziale/disgiunta)*.

- **Operazione di utente**: Entità che conterrà i dati necessari per costruire un LOG composto dalle azioni effettuate dall'utente. È una *Superclasse(padre)* di nome *operazione utente* con due *Sottoclassi(figlio) "inserimento utente" e "modifica utente". Superclasse(padre)* avrà come attributi *dataR(data Richiesta), dataA (data Accettazione o di rifiuto, dipende dal caso in cui ci troviamo), testo, visionata, link, Riferimento alla pagina e accettata.* La *Sottoclasse(figlio) "inserimento utente"* ha come attributo

aggiuntivo *posizione*(per capire dove verrà inserita questa frase).

- **Operazione di autore**: Questa generalizzazione è simile a *Operazione di utente*, e conterrà i dati necessari per costruire un LOG composto dalle azioni dell'autore. Le differenze sono: il numero di attributi inferiore rispetto a *Operazione utente* (essendo che le modifiche dell'autore hanno valenza immediata) e la mancanza di un'associazione (Notifica).

Alcune considerazioni su Operazione. quest'ultima si divide in *Operazione Utente* e *Operazione autore*, perché le due differiscono fra di loro. l'utente deve *proporre* un inserimento/modifica a differenza del autore che apporta le modifiche o inserimenti *immediatamente*. un altra differenza sono gli attributi e relazioni (mancanza dell'associazione Notifica in Operazione Autore).

1.1.5 Relazioni tra i componenti

- **Una pagina** viene *creata* da un *autore*, la classe-associazione *crea* ricorderà quando è stata creata la *pagina*.

- **Un autore** può inserire nuove frasi e modificarle. Queste *Sottoclassi* di nome *"inserimento autore" e "modifica autore"* avranno rispettivamente *la data di inserimento/modifica e il testo della nuova frase*. Un vincolo è alla creazione della pagina, quest'ultima deve avere almeno un testo iniziale composto da almeno una o più frasi. L'autore quindi alla creazione è obbligato a inserire almeno una frase.

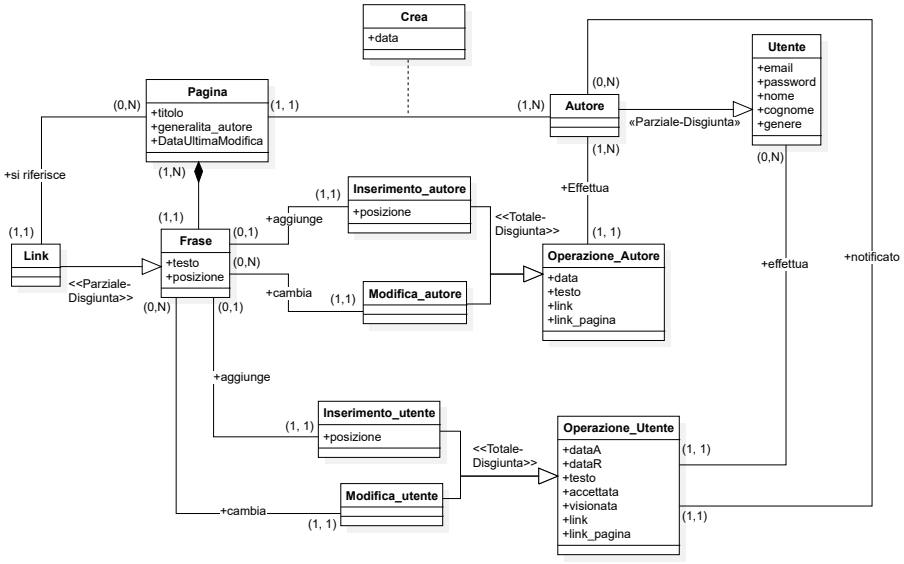
- **Un utente** può effettuare delle *modifiche* alle frasi, quest'ultime saranno contenute in *"modifiche utente"*. l'utente può modificare la *"proposta di modifica"* se quest'ultima non è stata ancora visionata dall'autore. In caso di modifica alla *proposta di modifica* dataR sarà aggiornata alla data dell'ultima modifica apportata a quella *"proposta di modifica"*

- **Un utente** può effettuare degli *inserimenti*, quest'ultimi avranno lo stesso funzionamento e concetto della modifica.

- **Una Operazione di utente** viene *notificata/visualizzata a/da un autore*, ovviamente verranno notificate soltanto quelle non visualizzate.

2 Schema Concettuale

Schema concettuale del database ottenuto dalle informazioni ottenute durante l'analisi dei requisiti.



2.1 Dizionario delle entità

Entità	Descrizione	Attributi
Utente	*Utente* generico del sistema che può essere anche un autore di una pagina wiki	**Nome**(String): Nome dell'utente **Cognome**(String): Cognome dell'utente **Email**(string): Email dell'utente **Password**(string): Password dell'utente **Genere**(Char): genere dell'utente
Autore	*Specializzazione di Utente*. Questa entità sta ad indicare il fatto che un *utente* può anche creare delle *pagine wiki*	
Pagina	L'entità pagina si riferisce alla *pagina wiki* creata da un *utente-autore*	**Generalità autore**(String): Stringa composta dalle generalità dell'autore della pagina **Titolo**(String): Indica il titolo della pagina **Data ultima modifica**(DateTime): Indica la data in cui è stata effettuata l'ultima modifica accettata sulla pagina
Frase	Frase è un elemento essenziale della pagina, perché una o più frasi formano il testo della pagina	**Testo**(String): Indica il contenuto testuale della frase **Posizione**(Int): Indica la posizione numerica all'interno della pagina
Link	Link è una specializzazione della frase, ovvero una frase può essere un link e riferirsi ad un altra pagina	

Entità	Descrizione	Attributi
Operazione utente	Operazione utente è un entità *Superclasse* che tiene traccia di tutte gli inserimenti e modifiche apportate, alle frasi di una pagina, dagli utenti, che esse siano accettate oppure no dall'autore della pagina	**DataA**(DateTime): Data in cui è stata accattata o rifiutata la modifica o inserimento dall'autore **DataR**(DateTime): Data in cui è stata proposta la modifica o inserimento **Testo**(String): Contiene il testo inserito dall'utente **Accettata**(Boolean): Indica se l'operazione proposta è stata accettata (1), rifiutata (0) dall'autore **Visionata**(Boolean): Questo attributo indica se l'autore ha visionato l'operazione
Inserimento utente	Inserimento è una *specializzazione* dell'entità *operazione utente*, ovvero un utente ha la possibilità di proporre una nuova frase	**Posizione**(Int): Posizione numerica specificata dall'utente al momento dell'inserimento di una nuova frase
Modifica utente	Modifica utente è una *specializzazione* dell'entità *operazione utente*, ovvero un utente ha la possibilità di modificare una frase	
Operazione autore	Operazione autore è un entità *Superclasse* che tiene traccia di tutte gli inserimenti e modifiche apportate, alle frasi di una pagina da parte dell'autore	**Data**(DateTime): Data in cui è stata apportata la modifica o inserimento **Testo**(String): Contiene il testo inserito dall'autore
Inserimento autore	Inserimento è una *specializzazione* dell'entità *operazione autore*, ovvero l'autore ha la possibilità di inserire direttamente una nuova frase	**Posizione**(Int): Posizione numerica specificata dall'autore al momento dell'inserimento di una nuova frase
Modifica autore	Modifica autore è una *specializzazione* dell'entità *operazione autore*, ovvero l'autore ha la possibilità di modificare direttamente una frase	

2.2 Dizionario delle associazioni

Associazioni	**Descrizione**	**Attributi**
Crea	Associazione **Molti-a-Uno** tra *Pagina* e *Autore*. Questa classe-associazione è utilizzata per memorizzare dati utili inerenti alla creazione della pagina.	**Data**(DateTime): Questo attributo si riferisce alla data di creazione della pagina wiki
Aggiunge (utente)	Associazione **Uno-a-Uno** tra *Frase* e *Inserimento utente*. Una farse può avere uno o zero inserimenti da parte del utente. Un' inserimento aggiunge una sola frase.	
Cambia (utente)	Associazione **Molti-a-Uno** tra *Frase* e *Modifica utente*. Una frase può essere modificata zero o più volte. Una modifica può modificare solo una frase.	
Aggiunge (autore)	Associazione **Uno-a-Uno** tra *Frase* e *Inserimento autore*. Una farse può avere uno o zero inserimenti da parte dell'autore. Un' inserimento aggiunge una sola frase.	
Cambia (autore)	Associazione **Molti-a-Uno** tra *Frase* e *Modifica autore*. Una frase può essere modificata zero o più volte. Una modifica può modificare solo una frase.	
Si riferisce	Associazione **Molti-a-Uno** tra *Link* e *Pagina*. Questa associazione si riferisce al fatto che una frase (link) è collegata/contiene il collegamento ad un altra pagina wiki.	
Notificato	Associazione **Molti-a-Uno** tra *Operazione* e *Autore*. Questa associazione si riferisce al fatto che un'autore visualizza le modifiche proposte alla sua pagina wiki. Una modifica verrà quindi visualizzata da un autore, ma l'autore visualizzerà tutte le modifiche inerenti alla sua pagina.	

Associazioni	**Descrizione**	**Attributi**
Effettua	Associazione **Multi-a-Uno** tra *Operazione* e *Utente*. Questa associazione si riferisce al fatto che un utente ha la possibilità di proporre una frase alla pagina wiki. Un utente quindi può proporre nessuna o tante operazioni su una pagina, mentre un'operazione specifica di un utente è effettuata da un solo utente	

3 Ristrutturazione del modello concettuale

3.1 Analisi delle Ridondanze

In questo fase analizzeremo le varie ridondanze e vedremo come gestirle

3.1.1 Data ultima modifica

data ultima modifica è un attributo di pagina e potrebbe essere *derivato*. Però essendo un informazione che deve essere reperita ogni volta che apriamo una pagina è conveniente averla subito disponibile e non derivarla ogni volta. Infatti se analizziamo la quantità di accessi ci converrà in grande scala non derivare la data di ultimo accesso. altrimenti dovremmo accedere ogni volta ad operazione utente e autore per capire chi è stato l'ultimo a fare una modifica.

3.1.2 Generalità_Autore

Questo attributo potrebbe essere derivato Notiamo che; le Generalità(Nome è cognome) del autore non cambiano molto spesso, quindi conviene per una questione di accessi creare un attributo apposito.

3.2 Eliminazione delle Generalizzazioni

In questa fase analizzeremo il tipo della generalizzazioni e come ristrutturarle

3.2.1 Operazione utente e operazione autore

Operazione utente è una generalizzazione *totale disgiunta* poiché un operazione è per forza al più una modifica o un inserimento. Per una questione di accessi è molto meglio accorpare i figli nel padre. Questo perché in una situazione con un numero elevato di pagine potremmo avere di conseguenza tante richieste per la visione della storicità, modifiche proposte o proposte di inserimento. avremo in alcuni casi il doppio dei accessi o in altri casi il quadruplo dei accessi in più. Tutto questo può essere risparmiato accorpando i figli nel padre. Come contro avremo solo un *attributo a null*, quest'ultimo è accettabile (attributo posizione potrà essere null) poiché aggiungendo questo valore a null ci risparmia molti accessi.
Operazione autore si elimina per lo stesso ragionamento e nello stesso metodo.

3.2.2 Link

link è una generalizzazione *parziale/disgiunta* poiché una frase può non essere un link ma una frase non può essere altro oltre al link. Sempre per una questione di accessi conviene accorpare il *padre* nei *figli*. Questo perché ogni volta che carichiamo una wiki dobbiamo controllare anche se ci sono dei link. Accorpando il figlio nel padre avremo meno accessi (uno in meno per controllare se una frase è un link e una in meno anche per navigare tra il link e la frase) velocizzando

la risposta del database alla richiesta di visualizzazione di pagina. A discapito avremo un valore a *null* se la frase non è un link. Anche se in un testo ci sono *più frasi* che *non sono un link* quindi i valori a null saranno *consistenti*, Questo non ci interessa poiché è un male affrontabile a noi interessa la *velocità* con cui carichiamo una wiki.

3.2.3 Autore

Un *utente* è una generalizzazione *parziale/disgiunta* poiché un utente può non essere un autore ma una utente non può essere oltre che un autore. Sempre per una questione di accessi preferiamo inserire il figlio nel padre, questo ci permetterà di dimezzare gli accessi per capire chi è un autore. A discapito avremo un attributo che ci indicherà se un utente è un autore o meno. Ci saranno sicuramente più utenti che autori quindi potenzialmente ci potrebbero essere tanti valori a *False(0)* ma il costo di questo attributo è di solo 1 bit. Otterremo sia meno accessi ma utilizzeremo più memoria rispetto a una diversa ristrutturazione ma la quantità di memoria utilizzata per salvare l'informazione sarà minima poiché l'attributo occupa solo un bit. Quindi se 300 utenti accedono avrò solo 300 accessi. Non avremo 600 accessi come nel caso di una diversa ristrutturazione; trasformazione della generalizzazione in relazioni. La ristrutturazione padre nel figlio a costo di accessi sarebbe stata uguale me non avrebbe avuto molto senso a livello logico.

3.3 Eliminazione degli attributi multivalore

Non abbiamo valori multi valore

3.4 Eliminazione degli attributi composti

Come valori composti abbiamo le date ma in questo caso non ci servirà dividerle in ora e data. Per questo motivo le matteremo tutto nello stesso attributo

3.5 Partizionamento/accorpamento di entità/associazioni

Nessun accorpamento da fare

3.6 Identificazione chiavi primarie

Utente $\longmapsto$ Abbiamo scelto come chiave primaria *Email*.

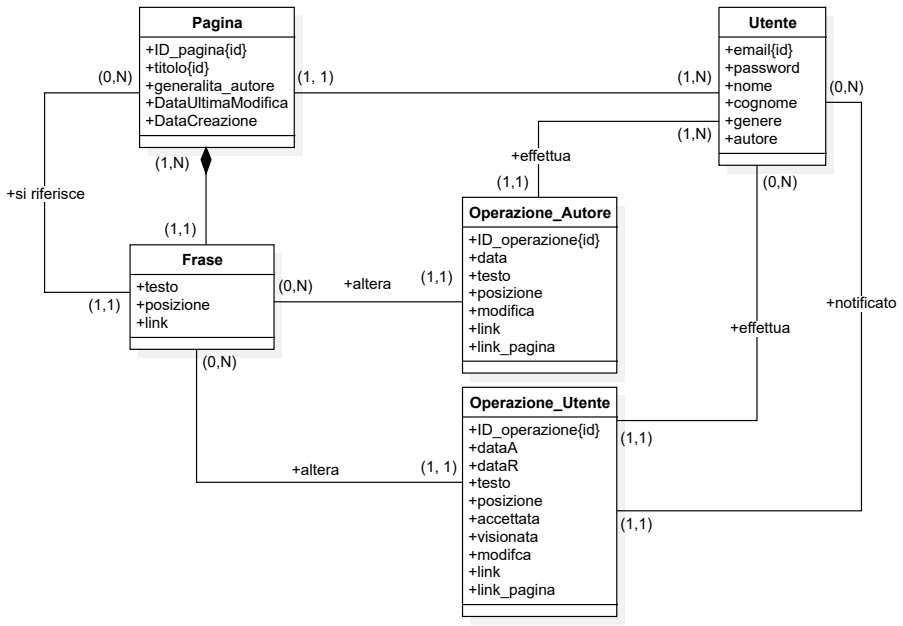
Frase $\longmapsto$ Non abbiamo nessuna chiave candidata però abbiamo una chiave parziale "*posizione*". quest'ultimo insieme alla chiave esterna di *Pagina* possiamo identificare univocamente una frase. questo perché in una pagina ci potrà essere solo una frase con la stessa posizione. La chiave primaria quindi è composta da *chiave esterna* e *posizione*

Pagina $\longmapsto$ il *titolo* è una chiave candidata però per un questione di facilita di identificazione della pagina nella frase e anche per il link e meglio utilizzare una chiave auto incrementata di nome *ID_Pagina*

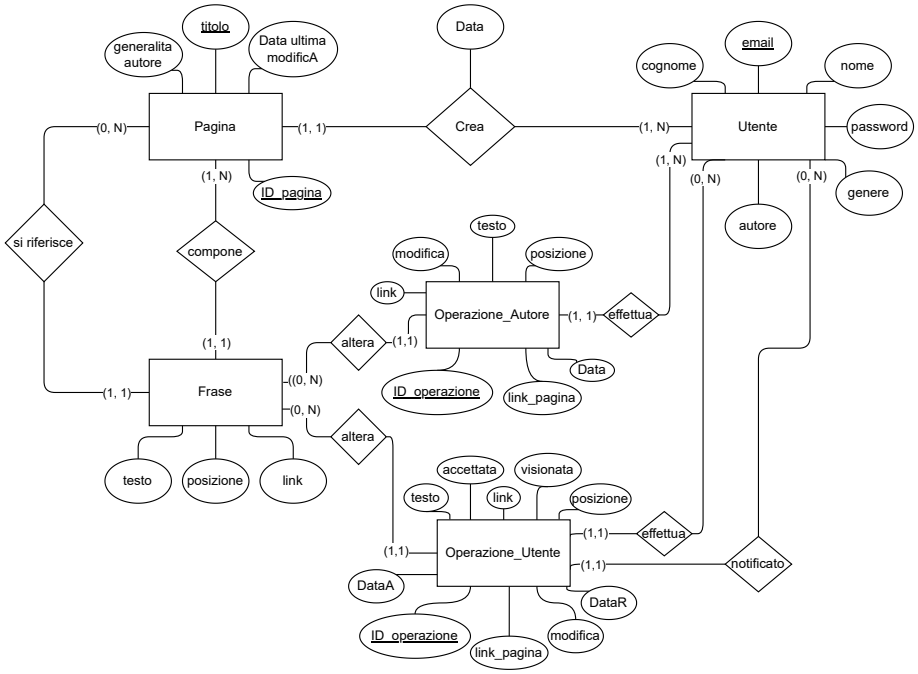
Operazione_Autore $\longmapsto$ dipende sia da *utente* che da *frase* quindi essendo che non ci sono chiavi candidate e nemmeno chiave parziali invece di creare una chiave primaria molto grande raggruppando tutti gli attributi creeremo un codice di nome *ID_operazione* che si auto incrementa. Questo per facilitare l'accesso

Operazione_Utente $\longmapsto$ dipende sia da *utente* che da *frase* quindi essendo che non ci sono chiave candidate e nemmeno chiave parziali invece di creare una chiave primaria molto grande raggruppando tutti gli attributi creeremo un codice di nome *ID_operazione* che si auto incrementa. Questo per facilitare l'accesso

3.7 Schema Ristrutturato UML



3.8 Schema Ristrutturato ER



3.9 Dizionario delle Entità

Entità	Descrizione	Attributi
Utente	*Utente* generico del sistema che può essere anche un autore di una pagina wiki	**Nome**(String): Nome dell'utente **Cognome**(String): Cognome dell'utente **Email**(string): Email dell'utente **Password**(string): Password dell'utente **Genere**(Char): genere dell'utente **Autore**(Bit): Indica se l'utente è un autore (1) oppure un utente (0)
Pagina	L'entità pagina si riferisce alla *pagina wiki* creata da un *utente-autore*	**ID_Pagina**(Int): Numero intero auto incrementato che identifica univocamente una pagina dalle altre **Generalità autore**(String): Stringa composta dalle generalità dell'autore della pagina **Titolo**(String): Indica il titolo della pagina, inoltre il titolo identifica univocamente una pagina dalle altre **Data ultima modifica**(Timestamp): Indica la data in cui è stata effettuata l'ultima modifica accettata sulla pagina **Data Creazione**(Timestamp): Indica la data in cui è stata effettuata la creazione della pagina

Entità	**Descrizione**	**Attributi**
Frase	Frase è un elemento essenziale della pagina, perché una o più frasi formano il testo della pagina	**Testo**(String): Indica il contenuto testuale della frase **Posizione**(Int): Indica la posizione numerica all'interno della pagina **link**(Bit): questo attributo specifica se la frase è un collegamento ad un altra pagina se il valore è 1, se invece è 0 allora vuol dire che Frase non è un link

Entità	**Descrizione**	**Attributi**
Operazione utente	Operazione utente è un entità *Superclasse* che tiene traccia di tutte gli inserimenti e modifiche apportate, alle frasi di una pagina, dagli utenti, che esse siano accettate oppure no dall'autore della pagina	**ID_operazione**(String): Attributo auto incrementato che identifica l'operazione univocamente dalle altre **DataA**(Timestamp): Data in cui è stata accattata o rifiutata la modifica o inserimento dall'autore **DataR**(Timestamp): Data in cui è stata proposta la modifica o inserimento **Testo**(String): Contiene il testo inserito dall'utente **Accettata**(Bit): Indica se l'operazione proposta è stata accettata (1), rifiutata (0), il valore è inizializzato a 0 quando la proposta non è stata ancora accettata **Visionata**(Bit): Questo attributo indica se l'autore ha visionato (1) l'operazione oppure no (0) **Posizione**(Int): Posizione numerica specificata dall'utente al momento dell'inserimento di una nuova frase **Modifica**(Bit): Questo attributo indica se l'operazione è una modifica (1) oppure no (0) **link**(Bit): Indica se l'operazione è effettuata su un link (1) oppure no (0) **link_pagina**(Int): Se l'operazione è effettuata su un link, questo attributo specifica a quale pagina la frase si riferisce

Entità	**Descrizione**	**Attributi**
Operazione autore	Operazione autore è un entità *Superclasse* che tiene traccia di tutte gli inserimenti e modifiche apportate, alle frasi di una pagina da parte dell'autore	**ID_operazione**(String): Attributo auto incrementato che identifica l'operazione univocamente dalle altre **Data**(Timestamp): Data in cui è stata apportata la modifica o inserimento **Testo**(String): Contiene il testo inserito dall'autore **Posizione**(Int): Posizione numerica specificata dall'autore al momento dell'inserimento di una nuova frase **Modifica**(Bit): Questo attributo indica se l'operazione è una modifica (1) oppure no (0) **link**(Bit): Indica se l'operazione è effettuata su un link (1) oppure no (0) **link_pagina**(Int): Se l'operazione è effettuata su un link, questo attributo specifica a quale pagina la frase si riferisce

3.10 Dizionario delle associazioni

Associazioni	**Descrizione**	**Attributi**
Crea	Associazione **Molti-a-Uno** tra *Pagina* e *Autore*. Questa classe-associazione è utilizzata per memorizzare dati utili inerenti alla creazione della pagina.	**Data**(Timestamp): Questo attributo si riferisce alla data di creazione della pagina wiki.
Altera	Associazione **Molti-a-Uno** tra *Operazione Autore* e *Frase*. Questa associazione si riferisce al fatto che una frase può subire modifiche oppure essere inserita in una pagina dall'autore	
Altera	Associazione **Molti-a-UNo** tra *Operazione Utente* e *Frase*. Questa associazione si riferisce al fatto che una frase può subire modifiche oppure essere inserita in una pagina dall'utente, dopo l'approvazione dell'autore	
Si riferisce	Associazione **Uno-a-Molti** tra *Frase* e *Pagina*. Questa associazione si riferisce al fatto che una frase (link) è collegata/contiene il collegamento ad un altra pagina wiki.	
Notificato	Associazione **Molti-a-Uno** tra *Operazione Utente* e *Autore*. Questa associazione si riferisce al fatto che un'autore visualizza le modifiche proposte alla sua pagina wiki. Una modifica verrà quindi visualizzata da un autore, ma l'autore visualizzerà tutte le modifiche inerenti alla sua pagina.	
Effettua	Associazione **Molti-a-Uno** tra *Operazione Utente* e *Utente*. Questa associazione si riferisce al fatto che un utente ha la possibilità di proporre una frase alla pagina wiki. Un utente quindi può proporre nessuna o tante operazioni su una pagina	

Associazioni	**Descrizione**	**Attributi**
Effettua	Associazione **Molti-a-Uno** tra *Operazione Autore* e *Utente*. Questa associazione si riferisce al fatto che un autore ha la possibilità la pagina wiki attraverso le operazioni sulle frasi. Un autore quindi può proporre nessuna o tante operazioni su una pagina ma effettuerà come minimo un'inserimento	

4 Traduzione al Modello Logico

4.1 Mapping delle Associazioni

4.1.1 Associazioni 1-N o N-1

- **Autore - Crea - Pagina**, Inseriamo la chiave di Autore come chiave esterna di pagina

- **Operazione Autore - Altera - Frase**, aggiungiamo la chiave di frase come chiave estera in Operazione Autore

- **Operazione Utente - Altera - Frase**, aggiungiamo la chiave di Frase come chiave estera in Operazione Utente

- **Frase - Si riferisce - Pagina**, inseriamo la chiave di pagina come chiave sterna di frase

- **Operazione Utente - Notifica - Autore**, inseriamo la chiave di Autore come chiave esterna di Operazione Utente

- **Utente - Effettua - Operazione utente**, inseriamo la chiave di Utente come chiave esterna di Operazione Utente

- **Autore - Effettua - Operazione Autore**, inseriamo la chiave di Autore come chiave esterna di Operazione Autore

4.1.2 Associazioni N-N

Non sono presenti relazioni N-N

4.2 Modello logico

Gli attributi sottolineati sono chiavi primarie.
Gli Attributi in **MAIUSCOLO** sono chiavi esterne.
Gli Attributi con * possono essere NULL

Utente(Email, Nome, Password, Genere, Autore)

Pagina(ID_Pagina, Titolo, Nome_autore, DataUltimaModifica, Data Creazione, **AUTORE**)
Pagina.AUTORE $\longmapsto$ *Utente.Email*

Frase(Posizione, **PAGINA**, Testo, Link, **LINKPAGINA***)
Frase.PAGINA $\longmapsto$ *pagina.ID_pagina*
Frase.LINKPAGINA $\longmapsto$ *pagina.ID_pagina*

Operazione Autore(ID_Operazione, Data, Testo, PosizioneIns*, Modifica, link, link_pagina*, **FRASE**, **AUTORE**)
Operazione Utente.FRASE $\longmapsto$ *Frase.(Posizione,PAGINA)*
Operazione Autore.AUTORE $\longmapsto$ *Utente.Email*

Operazione Utente(ID_Operazione, DataA*, DataR, Testo, PosizioneIns*, link, link_pagina*, Accettata, Visionata, Modifica, **FRASE**, **UTENTE**, **AUTORE_NOTIFICATO**)
Operazione Utente.FRASE $\longmapsto$ *Frase.(Posizione*,PAGINA)*
Operazione Utente.UTENTE $\longmapsto$ *Utente.Email*
Operazione Utente.AUTORE_NOTIFICATO $\longmapsto$ *Utente.Email*

5 Progettazione Fisica

5.1 Creazione delle tabelle

```
CREATE TABLE Utente(
        Email d_Email,
        Nome VARCHAR(20) NOT NULL,
    Cognome VARCHAR(20) NOT NULL,
        Password_Utente VARCHAR(50) NOT NULL,
        Genere CHAR(1) NOT NULL,
        Autore BIT(1) NOT NULL,
        CONSTRAINT PK_Utente PRIMARY KEY(Email),
        CONSTRAINT CheckGenere CHECK(Genere LIKE 'F' OR Genere LIKE 'M')
);

CREATE TABLE Pagina(
    ID_Pagina SERIAL,
    Titolo VARCHAR(255) NOT NULL,
    Generalita_Autore VARCHAR(41) NOT NULL,
    DataUltimaModifica timestamp NOT NULL,
    DataCreazione timestamp NOT NULL,
    emailAutore d_email NOT NULL,
    CONSTRAINT PK_PAGINA PRIMARY KEY(ID_Pagina),
    CONSTRAINT UNIQUE_TITOLO UNIQUE(Titolo),
    CONSTRAINT FK_PAGINA_AUTORE FOREIGN KEY(emailAutore) REFERENCES
        Utente(email) ON UPDATE CASCADE
);

CREATE TABLE Frase(
    Pagina INTEGER,
    Posizione INTEGER,
    Testo TEXT NOT NULL,
    Link BIT(1) NOT NULL,
    LinkPagina INTEGER,
    CONSTRAINT PK_FRASE PRIMARY KEY(Pagina,Posizione),
    CONSTRAINT FK_FRASE_PAGINA_REF FOREIGN KEY(LinkPagina) REFERENCES
        Pagina(ID_Pagina) ON DELETE SET NULL ON UPDATE CASCADE,
    CONSTRAINT FK_FRASE_PAGINA_LINK FOREIGN KEY(Pagina) REFERENCES
        Pagina(ID_Pagina)  ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE Operazione_Utente(
    ID_Operazione SERIAL,
    DataA timestamp,
    DataR timestamp NOT NULL,
    Testo TEXT  NOT NULL,
    Accettata BIT(1) NOT NULL,
    Visionata BIT(1) NOT NULL,
    PosizioneIns INTEGER,
```

```
43      modifica BIT(1) NOT NULL,
44      Link BIT(1) NOT NULL,
45      link_pagina INTEGER,
46      Posizione_Frase INTEGER,
47      Pagina_Frase INTEGER NOT NULL,
48      Utente d_Email NOT NULL,
49      Autore_Notificato d_Email NOT NULL,
50      CONSTRAINT CheckLink CHECK((link = 1::BIT(1) AND link_pagina IS NOT
            NULL) OR (link_pagina IS NULL AND link = 0::BIT(1))),
51      CONSTRAINT PK_Operazione_U PRIMARY KEY(ID_Operazione),
52      CONSTRAINT FK_Operazione_U_Frase_Posizione_Pagina FOREIGN KEY(
            Posizione_Frase,Pagina_Frase) REFERENCES frase(Posizione,pagina)
             ON DELETE CASCADE ON UPDATE CASCADE ,
53      CONSTRAINT FK_Operazione_U_Utente  FOREIGN KEY(Utente) REFERENCES
            Utente(Email) ON UPDATE CASCADE,
54      CONSTRAINT FK_Operazione_U_AuotrePagina  FOREIGN KEY(
            Autore_Notificato) REFERENCES Utente(Email) ON UPDATE CASCADE
55  );
56
57  CREATE TABLE Operazione_Autore(
58      ID_Operazione SERIAL,
59      Data timestamp NOT NULL,
60      Testo TEXT  NOT NULL,
61      PosizioneIns INTEGER ,
62      modifica BIT(1) NOT NULL,
63      Posizione_Frase INTEGER NOT NULL,
64      Pagina_Frase INTEGER NOT NULL,
65      Autore d_email NOT NULL,
66      CONSTRAINT PK_Operazione_A PRIMARY KEY(ID_Operazione),
67      CONSTRAINT FK_Operazione_A_Frase_Posizione_Pagina FOREIGN KEY(
            Posizione_Frase,Pagina_Frase) REFERENCES frase(Posizione,pagina)
             ON DELETE CASCADE ON UPDATE CASCADE,
68      CONSTRAINT FK_Operazione_A_Autore FOREIGN KEY(Autore) REFERENCES
            Utente(Email) ON UPDATE CASCADE);
```

5.2 Vincoli

Identifichiamo in questa sezione i vincoli legati al nostro dominio

5.2.1 Vincoli di Domino

- **d_Email** è un dominio che controlla se un email è valida o meno. Questo dominio non è totale, ci potrebbero essere altri problemi come; doppia chioccola o doppio punto. L'Email dovrà essere comunque controllata, tutta via questo check eviterà lavoro superfluo al trigger che avrà il compito di controllare Email al momento del inserimento

```
CREATE DOMAIN d_Email as VARCHAR(255)
CHECK(VALUE LIKE '_%@_%._%')
```

- **CheckGenere** Limita il campo dei VARCHAR a due solo valori 'F' o 'M', creando un dominio formato solo da 'F' o 'M'

5.2.2 Vincoli sui valori NULL e valori predefiniti

- Tutti i valori non possono essere NULL escludendo:
  - **dataA** data Accettazione o Rifiuto dell'operazione
  - **LINKPAGINA** cioè il riferimento a una pagina tramite un link
  - **posizione_frase** è il riferimento alla frase in operazione utente. quest'ultimo può essere null poiché quando faremo una richiesta di inserimento la frase non esiste
  - **posizioneIns** è la posizione di inserimento della frase, se L'operazione è una modifica questo valore sarà NULL
- Non ci sono valori Default

5.2.3 Identificazione vincoli intrarelazionali

Vincoli di Chiave:

- Sono tutte le chiavi primarie nelle rispettive relazioni.

Vincoli di Tupla:

- se l'attributo *"link"* in *"Operazione Utente"* e in *"Operazione Autore"* è impostato a *True* allora *"link_pagina"* non dovrà essere NULL (costrain di nome **CheckLink**)

5.2.4 Identificazione vincoli interrelazionali

Vincoli di integrità referenziale:

- Se la Chiave primaria di una *"pagina"* viene modificata allora anche i riferimenti devono essere aggiornati
- Se la Chiave primaria di un'*"utente"* viene modificata allora anche i riferimenti devono essere aggiornati
- Se un *"link"* fa riferimento ad una *"pagina eliminata"*, allora il link non deve contenere nessun riferimento (NULL)
- Quando viene eliminata una *"pagina"*, deve essere eliminato tutto il suo contenuto (Frasi), le sue occorrenze e le occorrenze del contenuto (Operazioni Utente e Autore)

Vincoli di integrità semantica:

- Se l'attributo*"visionata"* in *"Operazione Utente"* è impostato a *True* allora l'utente non potrà modificare quella proposta. **Clica qui**
- Un *"Utente"* non può essere cancellato. **Clica qui**
- Un *"Utente"* Diventa autore se crea una pagina. **Clica qui**
- Un *"Autore"* Diventa Utente se non ha più nessuna pagina. **Clica qui**
- In *"Pagina"* deve essere aggiornato il valore di Generalità_autore in caso di modifica delle generalità dell'autore. **Clica qui**
- La *"DataR"* deve combaciare sempre con l'ultima Data di modifica del testo. **Clica qui**
- Una *"Pagina"* non può essere vuota. **Clica qui**
- *"DataUltimaModifica"* in "Pagina" deve essere aggiornata quando viene aggiunta o modificata una frase nella pagina. **Clica qui**
- Le azioni dell'autore sono registrate in *Operazione_autore*. **Clica qui** e **Clica qui**
- Le azioni dell'utente sono registrate in *Operazione_utente*. **Clica qui**

5.3 Trigger

5.3.1 Modifica_proposta

Vincolo: Se l'attributo*"visionata"* in *"Operazione Utente"* è impostato a *True* allora l'utente non potrà modificare quella proposta

Descrizione: è un *trigger* che al momento della modifica del *testo* in *operazione_utente*, prima di inserire la tupla controlla se quella "proposta di modifica" è stata già visualizzata. Se visionata è *True(1)* allora impedisce l'Inserimento con un Excption.

```
CREATE OR REPLACE FUNCTION Modifica_Proposta()
RETURNS TRIGGER
AS $Modifica_Proposta$
BEGIN
    RAISE EXCEPTION 'E07: La modifica e gia stata visionata';
    RETURN NEW;
END;
$Modifica_Proposta$ LANGUAGE plpgsql;

CREATE TRIGGER modifica_proposta
BEFORE UPDATE OF testo ON operazione_utente
FOR EACH ROW
WHEN (old.visionata = 1::bit(1))
EXECUTE FUNCTION modifica_proposta();
```

5.3.2 Utente_No_Delete

Vincolo: Un *"utente"* non può essere cancellato

Descrizione: Lancia un'eccezione prima che una tupla nella tabella *utente* venga cancellata.

```
CREATE OR REPLACE FUNCTION Utente_No_Delete()
RETURNS TRIGGER
AS $Utente_No_Delete$
BEGIN
        RAISE EXCEPTION 'E06: Un utente non puo essere cancellato';
        RETURN NEW;
END;
$Utente_No_Delete$ LANGUAGE plpgsql;

CREATE TRIGGER Utente_No_delete
BEFORE DELETE ON Utente
FOR EACH ROW
EXECUTE FUNCTION Utente_No_Delete();
```

5.3.3 ModificaDataMod

Vincolo: La *DataR* deve combaciare sempre con l'ultima Data di modifica del testo

Descrizione: Il *"set time zone 'Europe/Rome';"* serve per impostare l'ora italiana. Questo trigger si occuperà di Modificare DataR nella data in cui è stato fatto l'Ultimo cambiamento al testo.

```
1  set time zone 'Europe/Rome';
2
3  CREATE OR REPLACE FUNCTION AggiornamentoDataMod() RETURNS TRIGGER
4  AS $Aggiornamento_data_mod$
5  BEGIN
6      new.DataR = localtimestamp(0);
7      RETURN NEW;
8  END;
9  $Aggiornamento_data_mod$
10 LANGUAGE plpgsql;
11
12 CREATE OR REPLACE TRIGGER AggiornamentoDataMod
13 BEFORE UPDATE OF testo ON operazione_utente
14 FOR EACH ROW
15 EXECUTE FUNCTION AggiornamentoDataMod();
```

5.3.4 Aggiornamento_Generalita_Pagina

Vincolo: In *"pagina"* deve essere aggiornato il valore di Generalità_autore in caso di modifica delle generalità dell'autore.

Descrizione: Questo trigger prenderà tutte le pagine con l'email dell'autore a cui sono state aggiornate le generalità. Dopodiché prende il nome e il cognome aggiornati e li separa da un "**;**" poiché un nome o un cognome potrebbero avere spazzi. Per ogni pagina dell'autore aggiorneremo le generalità con le informazioni aggiornate.

```
1  CREATE OR REPLACE FUNCTION Aggiornamento_Generalita_Pagina()
2  RETURNS TRIGGER
3  AS $Aggiornamento_Generalita_Pagina$
4  DECLARE
5      generalita VARCHAR (41);
6      cur CURSOR FOR SELECT id_pagina FROM PAGINA WHERE emailautore=NEW.
           email;
7          idPagina INTEGER;
8  BEGIN
9
10     generalita := NEW.Nome || ';' || NEW.Cognome;
11     OPEN cur;
12         LOOP
13                 FETCH cur INTO idPagina;
14         EXIT WHEN NOT FOUND;
15         UPDATE PAGINA SET generalita_autore=generalita WHERE id_pagina =
               idPagina;
16
17     END LOOP;
18     CLOSE cur;
19
20     RETURN NEW;
21 END;
22 $Aggiornamento_Generalita_Pagina$ LANGUAGE plpgsql;
23
24 CREATE TRIGGER Aggiornamento_Generalita_Pagina
25 AFTER UPDATE OF nome, cognome ON Utente
26 FOR EACH ROW
27 WHEN (NEW.autore=1::BIT(1))
28 EXECUTE FUNCTION Aggiornamento_Generalita_Pagina();
```

5.3.5 SwitchAutore

Vincolo: Un *"Utente"* Diventa autore se crea una pagina

Descrizione: Dopo aver inserito una pagina controlliamo se l'utente che ha creato la pagana è già un autore. Nel caso non lo fosse allora aggiorniamo il parametro autore a *True*.

```
CREATE OR REPLACE FUNCTION SwitchAutore() RETURNS trigger
as $SwitchAutore$
begin
        IF NOT EXISTS(SELECT email FROM utente where new.emailautore =
            email and autore = 1::BIT(1)) THEN
                UPDATE utente set autore = 1::BIT(1) where new.
                    emailautore = email;
        END IF;
        RETURN NEW;
end;
$SwitchAutore$  LANGUAGE plpgsql;

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

5.3.6 SwitchUtente

Vincolo: Un *"Autore"* Diventa Utente se non ha più nessuna pagina

Descrizione: dopo la cancellazione di una pagina controlliamo se l'autore della pagina cancellata ha altre pagine. se la query restituisce qualcosa allora non faremo nulla altrimenti l'autore diventerà Utente.

```
CREATE OR REPLACE FUNCTION SwitchUtente() RETURNS trigger as
    $SwitchUtente$
begin
        IF NOT EXISTS(SELECT p1.Id_pagina FROM (pagina as p1 join pagina
            on old.emailautore = p1.emailautore)) THEN
                UPDATE utente set autore = 0::BIT(1) where old.
                    emailautore = email;
        END IF;
        RETURN NEW;
end;
$SwitchUtente$  LANGUAGE plpgsql;

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

5.3.7 DataModficaPagina

Vincolo: *"DataUltimaModifica"* in *"Pagina"* deve essere aggiornata quando viene aggiunta o modificata una frase nella pagina.

Descrizione: Quando inserisco o modifico in **frase** significa che ho modificato la pagina. Quindi aggiorno la *"DataUltimaModifica"* della pagina.

```
CREATE OR REPLACE FUNCTION DataModficaPagina() RETURNS TRIGGER
AS $DataModficaPagina$
BEGIN
        UPDATE pagina set dataultimamodifica = localtimestamp(0) where
            id_pagina = new.pagina;
        RETURN NEW;
END;
$DataModficaPagina$ LANGUAGE PLPGSQL;

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

5.3.8 OperazioneAutoreIntegritaIns

Vincolo: Le azioni dell'autore sono registrate in *Operazione_autore*

Descrizione: Quando un autore inserisce una frase l'operazione verrà registrata nei **LOG** dell'autore.

```
CREATE OR REPLACE FUNCTION OperazioneAutoreIntegritaIns() RETURNS TRIGGER
    AS $OperazioneAutoreIntegritaIns$
BEGIN
IF NOT EXISTS(SELECT id_operazione from operazione_utente where new.
    pagina = pagina_frase and posizioneins = new.posizione) THEN
        INSERT INTO operazione_autore(data,testo,posizioneins,modifica,
            link,link_pagina,posizione_frase,pagina_frase,autore) VALUES(
            localtimestamp(0),new.testo,new.posizione,0::BIT(1),new.link,
            new.linkpagina,new.posizione,new.pagina,(select emailautore
            from pagina where new.pagina = id_pagina));
END IF;
RETURN NEW;
END;
$OperazioneAutoreIntegritaIns$ LANGUAGE PLPGSQL;

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

5.3.9 OperazioneAutoreIntegritaMod

Vincolo: Le azioni dell'autore sono registrate in *Operazione_autore*

Descrizione: Quando un autore modifica una frase l'operazione verrà registrata nei **LOG** dell'autore.

```
1  CREATE OR REPLACE FUNCTION OperazioneAutoreIntegritaMod() RETURNS TRIGGER
       AS
2  $OperazioneAutoreIntegritaMod$
3  BEGIN
4  IF NOT EXISTS(SELECT id_operazione from operazione_utente where new.
       pagina = pagina_frase and posizione_frase = new.posizione) THEN
5          INSERT INTO operazione_autore(data,testo,posizioneins,modifica,
               link,link_pagina,posizione_frase,pagina_frase,autore) VALUES(
               localtimestamp(0),new.testo,NULL,0::BIT(1),new.link,new.
               linkpagina,new.posizione,new.pagina,(select emailautore from
               pagina where new.pagina = id_pagina));
6  END IF;
7  RETURN NEW;
8  END;
9  $OperazioneAutoreIntegritaMod$ LANGUAGE PLPGSQL;
10
11 CREATE OR REPLACE TRIGGER OperazioneAutoreIntegritaMod
12 AFTER UPDATE OF testo ON Frase
13 FOR EACH ROW
14 EXECUTE FUNCTION OperazioneAutoreIntegritaMod();
```

5.3.10 PositionControll

Scopo: Permette l'inserimento e ottimizza l'inserimento. Si creano dei intervalli tra le parole per permettere di inserire nel mezzo senza spostare tutte le frasi.

Descrizione: Questo trigger viene attivato ogni volta che viene inserita una frase in una pagina. Esso controlla la posizione massima (La posizione dell'ultima frase), se esiste l'immagazzina in una variabile(max_pos) altrimenti imposta la variabile a zero (max_pos). se la posizione massima (max_pos) è più piccola della posizione di inserimento (new.posizione) allora creo un intervallo tra max_pos e la nuova posizione della parola, questo perché se voglio inserire nel mezzo non troverò difficoltà. Nel caso opposto (max_pos maggiore uguale a new.posizione) controlliamo prima se la nuova posizione è *"1"* poiché è un caso particolare (ovvero se si inserisce una frase in testa, allora devo spostare tutte le frasi). Altrimenti imposto la nuova posizione (new.posizione) a meta dell'intervallo (tra il più piccolo dei successori e il più grande dei predecessori). Se questa posizione già esiste (caso in cui lo spazio tra due frasi è terminato) allora chiamo **UpadatePosition** (che aggiunge spazzino nel'intervallo). In fine *new.posizione* sarà impostata a meta del nuovo intervallo.

```
1  CREATE OR REPLACE FUNCTION PositionControll() RETURNS TRIGGER AS
2  $PositionControll$
3  DECLARE
4          max_pos integer;
5          min_pos integer;
6          max_int_pos integer;
7
8  BEGIN
9
10         SELECT MAX(posizione) into max_pos from frase where pagina = new.
               pagina;
11         IF NOT FOUND THEN
12                 max_pos = 0;
13         END IF;
14
15         IF max_pos < new.posizione THEN
16                 new.posizione = max_pos + 200;
17
18         ELSEIF max_pos >= new.posizione then
19
20                 IF new.posizione = 1 THEN
21                         CALL UpadatePosition(new.posizione,new.pagina,1::
                               BIT(1));
22                         RETURN NEW;
23                 END IF;
24
```

```
25                  SELECT MIN(posizione) into min_pos  from frase where
                        pagina = new.pagina and new.posizione <= posizione;
26                  SELECT MAX(posizione) into max_int_pos  from frase where
                        pagina = new.pagina and new.posizione > posizione;
27                  new.posizione := (min_pos + max_int_pos)/2;
28                  IF EXISTS(SELECT posizione from frase where new.pagina =
                        pagina and new.posizione = posizione) THEN
29
30                          CALL UpadatePosition(new.posizione,new.pagina,0::
                                BIT(1));
31                          new.posizione := (min_pos + 100);
32                  END IF;
33          END IF;
34
35          RETURN NEW;
36  END;
37  $PositionControll$ LANGUAGE PLPGSQL;
38
39  CREATE OR REPLACE TRIGGER PositionControll
40  BEFORE INSERT ON Frase
41  FOR EACH ROW
42  EXECUTE FUNCTION PositionControll();
```

5.3.11 CheckAggiornamentoPagina

Scopo: Inserisce/modifica le frasi accettate dall'autore nelle pagine corrispondenti.

```
CREATE OR REPLACE FUNCTION CheckAggiornamentoPagina()
RETURNS TRIGGER
AS $CheckAggiornamentoPagina$
BEGIN
        IF NEW.accettata = 1::bit(1) THEN
                IF NEW.modifica = 1::bit(1) THEN
                        UPDATE frase SET testo = NEW.testo, link = NEW.
                            link, linkpagina = NEW.link_pagina WHERE
                            pagina = NEW.pagina_frase AND posizione = NEW
                            .posizione_frase;
                ELSE
                        INSERT INTO frase VALUES (NEW.pagina_frase, NEW.
                            posizioneins, NEW.testo, NEW.link, NEW.
                            link_pagina);
                END IF;
        END IF;

    RETURN NEW;
END;
$CheckAggiornamentoPagina$
LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER CheckAggiornamentoPagina
AFTER UPDATE OF Dataa ON operazione_utente
FOR EACH ROW
EXECUTE FUNCTION CheckAggiornamentoPagina();
```

5.3.12 DeleteFraseOpUtente

Implementazione di: *Quando viene eliminata una "pagina", deve essere eliminato tutto il suo contenuto (Frasi), le sue occorrenze e le occorrenze del contenuto (Operazioni Utente e Autore).*

Descrizione: Questo trigger serve per eliminare le occorrenze di proposte di inserimento quando una pagina viene eliminata.

```
CREATE OR REPLACE FUNCTION DeleteFraseOpUtente() RETURNS TRIGGER
AS $DeleteFraseOpUtente$
BEGIN

        DELETE FROM operazione_utente WHERE pagina_frase=OLD.id_pagina
            AND posizione_frase is NULL;
```

```
6
7           RETURN NEW;
8   END;
9   $DeleteFraseOpUtente$ LANGUAGE PLPGSQL;
10
11  CREATE OR REPLACE TRIGGER DeleteFraseOpUtente
12  AFTER DELETE ON Pagina
13  FOR EACH ROW
14  EXECUTE FUNCTION DeleteFraseOpUtente();
```

5.4 Funzioni e Procedure

5.4.1 CreazionePagina

Vincolo: Una *"Pagina"* non può essere vuota

Descrizione: Si potrà creare una pagina solo tramite questa Procedura. Inoltre la procedura garantirà che subito dopo alla creazione di una pagina ci sarà un inserimento di una frase. Essendo tutta la Procedura in un blocco **BEGIN/END** nel caso ci fosse un errore sarà fatto il **ROLLBACK** cosi mantenendo la *consistenza nel DB*.

```
1  CREATE OR REPLACE PROCEDURE CreazionePagina(Titolo_in VARCHAR(255),
       Emailautore VARCHAR(255), Testo TEXT, link BIT(1), titolo_pagina_link
        varchar(255), posizione INTEGER)
2  as $CreazionePagina$
3  DECLARE
4          Nome VARCHAR(20);
5          Cognome VARCHAR(20);
6          Generalita VARCHAR(41);
7          id_page integer;
8  BEGIN
9  IF EXISTS(SELECT email FROM Utente where Emailautore = utente.email) THEN

10                 SELECT u.cognome , u.nome into Cognome, nome FROM Utente
                       as u where Emailautore = u.email;
11                 Generalita := nome || ';' || cognome;
12                 INSERT INTO Pagina (Titolo, Generalita_Autore,
                       DataUltimaModifica, DataCreazione, emailAutore)
                       VALUES(Titolo_in, Generalita, localtimestamp(0),
                       localtimestamp(0), Emailautore);
13                 SELECT id_pagina into id_page FROM pagina where Titolo_in
                        = pagina.titolo;
14                 CALL InsertFraseAutore(id_page, Emailautore , Testo, link
                        , titolo_pagina_link , posizione);
15 else
16         RAISE EXCEPTION 'E05: Utente non trovato';
17         ROLLBACK;
18 END IF;
19 END;
20 $CreazionePagina$ LANGUAGE plpgsql;
```

5.4.2 InsertFraseAutore

Scopo: Inserire una frase

Descrizione: La Procedura controlla che l'utente inserito esiste, controlla se la frase che stiamo inserendo deve diventare un link e controlla se la pagina a cui fa riferimento il link esiste. Dopodiché inseriamo la frase. Se la frase non è link inseriamo senza mettere riferimenti ad altre pagine.

```
1  CREATE OR REPLACE PROCEDURE InsertFraseAutore(Id_pagina_in INTEGER,
       Emailautore VARCHAR(255), Testo TEXT, link BIT(1), titolo_pagina_link
        varchar(255), posizione INTEGER)
2  as $InsertFraseAutore$
3  DECLARE
4          idpagina_link INTEGER;
5  BEGIN
6          CALL CheckAutore(EmailAutore, id_pagina_in);
7          IF(link = 1::BIT(1)) then
8                  SELECT id_pagina INTO idpagina_link FROM pagina where
                       titolo_pagina_link = pagina.titolo;
9                  IF FOUND THEN
10                         INSERT INTO frase VALUES (Id_pagina_in, posizione
                               ,Testo,link,idpagina_link);
11
12                 else
13                         RAISE EXCEPTION 'E04: La pagina non esiste(link
                               non valido)';
14                         ROLLBACK;
15                 END IF;
16         ELSE
17                 INSERT INTO frase VALUES(Id_pagina_in, posizione,Testo,
                       link,null);
18         END IF;
19
20 END;
21 $InsertFraseAutore$ LANGUAGE plpgsql;
```

5.4.3 ModificaFraseAutore

Scopo: Premete la modifica di una frase da parte dell'autore della pagina

Descrizione: La Procedura controlla che l'utente inserito esiste, se la frase che stiamo modificando deve diventare un link e se la pagina a cui fa riferimento il link esiste. Dopodiché modifichiamo la frase. se la frase non è link modifichiamo solo il testo.

```
1  CREATE OR REPLACE PROCEDURE ModificaFraseAutore(Id_pagina_in INTEGER,
       Emailautore VARCHAR(255), Testo_in TEXT, link_in BIT(1),
       titolo_pagina_link varchar(255), posizione_in INTEGER)
2  AS $ModificaFraseAutore$
3  DECLARE
4          idpagina_link INTEGER;
5
6  BEGIN
7
8          CALL CheckAutore(EmailAutore, id_pagina_in);
9          IF(link_in = 1::BIT(1)) then
10                 SELECT id_pagina INTO idpagina_link FROM pagina where
                       titolo_pagina_link = pagina.titolo;
11                 IF FOUND THEN
12                         UPDATE frase SET testo = Testo_in , link =
                               link_in, linkpagina = idpagina_link  where
                               Id_pagina_in = pagina and posizione_in =
                               posizione;
13                 else
14                         RAISE EXCEPTION 'E10: La pagina non esiste(link
                               non valido)';
15                         ROLLBACK;
16                 END IF;
17         ELSE
18                 UPDATE frase SET testo = Testo_in , link = link_in,
                       linkpagina = NULL where Id_pagina_in = pagina and
                       posizione_in = posizione;
19         END IF;
20
21 END;
22 $ModificaFraseAutore$ LANGUAGE PLPGSQL;
```

5.4.4 CheckAutore

Scopo: Funzione che controlla se le operazioni sono concrete e svolte dall'autore della pagina.

Descrizione: Questa procedura permette di controllare le seguiteti condizioni: se la pagina inserita esiste, se l'utente inserito esiste ed è autore e se la pagina su cui vogliamo applicare la modifica è effettivamente la nostra. Se una di queste condizioni non è vera allora verrà lanciato un **EXCETION**.

```
1  CREATE OR REPLACE PROCEDURE CheckAutore(EmailAutore_in VARCHAR(255),
       id_pagina_in INTEGER) AS
2  $CheckAutore$
3  BEGIN
4  IF EXISTS(SELECT id_pagina FROM pagina where id_pagina_in = pagina.
       id_pagina) THEN
5          IF EXISTS(SELECT email FROM Utente where EmailAutore_in = utente.
               email and Utente.autore = 1::BIT(1)) THEN
6                  IF EXISTS(SELECT email from utente join pagina on pagina.
                       emailautore = utente.email where EmailAutore_in like
                       utente.email and id_pagina_in = pagina.id_pagina)
                       THEN
7                          RETURN;
8                  ELSE
9                          RAISE EXCEPTION 'E02: Questo utente non e autore
                               della pagina';
10                         ROLLBACK;
11                 END IF;
12         ELSE
13                 RAISE EXCEPTION 'E03: Utente non trovato o utente non e
                       autore';
14                 ROLLBACK;
15         END IF;
16 ELSE
17         RAISE EXCEPTION 'E09: La pagina non esiste';
18         ROLLBACK;
19 END IF;
20 END;
21 $CheckAutore$ LANGUAGE PLPGSQL;
```

5.4.5 OperazioneUtenteRichiesta

Vincolo: Le azioni dell'utente sono registrate in *Operazione_utente*

Descrizione: Questa funzione permette al utente di richiedere un inserimento o una modifica. in base ai parametri inseriti andremo nel caso del inserimento o della modifica. Sempre tramite parametri possiamo capire se vogliamo che quella frase diventi/sia un link

```
1  CREATE OR REPLACE PROCEDURE OperazioneUtenteRichiesta(utente_in varchar
       (255),Testo_in Text,posizioneins_in integer, modifica_in BIT(1),
       pagina_in integer,posizione_frase_in integer, link_in BIT(1),
       link_pagina integer)
2  AS $OperazioneUtenteRichiesta$
3  BEGIN
4          IF EXISTS(SELECT id_pagina FROM pagina where pagina_in = pagina.
               id_pagina) THEN
5               IF link_in = 1::BIT(1) and EXISTS(SELECT id_pagina FROM
                    pagina where link_pagina = pagina.id_pagina) THEN
6                   IF EXISTS(select email from utente where
                        utente_in like email) THEN
7                       IF modifica_in = 0::BIT(1) THEN
8                           INSERT INTO operazione_utente(
                                dataA,dataR,testo,accettata,
                                visionata,posizioneins,
                                modifica,link,link_pagina,
                                posizione_frase,pagina_frase,
                                utente,autore_notificato)
9                           VALUES(NULL,LOCALTIMESTAMP(0),
                                Testo_in,0::BIT(1),0::BIT(1),
                                posizioneins_in,modifica_in,
                                link_in,link_pagina,NULL,
                                pagina_in,utente_in,(Select
                                emailautore from pagina where
                                 pagina_in = id_pagina));
10
11                      ELSEIF EXISTS(select posizione from frase
                            where pagina = pagina_in and
                           posizione_frase_in = frase.posizione)
                            THEN
12                          INSERT INTO operazione_utente(
                                dataA,dataR,testo,accettata,
                                visionata,posizioneins,
                                modifica,link,link_pagina,
                                posizione_frase,pagina_frase,
                                utente,autore_notificato)
13                          VALUES(NULL,LOCALTIMESTAMP(0),
                                Testo_in,0::BIT(1),0::BIT(1),
                                posizioneins_in,modifica_in,
```

```
                                        link_in,link_pagina,
                                        posizione_frase_in,pagina_in,
                                        utente_in,(Select emailautore
                                         from pagina where pagina_in
                                        = id_pagina));
14                              ELSE
15                                      RAISE EXCEPTION 'E08: La frase
                                            non esiste';
16
17                              END IF;
18                      ELSE
19                              RAISE EXCEPTION 'E10: Utente non corretto
                                    ';
20                      END IF;
21              ELSEIF link_in = 0::BIT(1) THEN
22                      IF EXISTS(select email from utente where
                            utente_in like email) THEN
23                              IF modifica_in = 0::BIT(1) THEN
24                                      INSERT INTO operazione_utente(
                                            dataA,dataR,testo,accettata,
                                            visionata,posizioneins,
                                            modifica,link,link_pagina,
                                            posizione_frase,pagina_frase,
                                            utente,autore_notificato)
25                                      VALUES(NULL,LOCALTIMESTAMP(0),
                                            Testo_in,0::BIT(1),0::BIT(1),
                                            posizioneins_in,modifica_in,
                                            link_in,NULL,NULL,pagina_in,
                                            utente_in,(Select emailautore
                                             from pagina where pagina_in
                                            = id_pagina));
26
27                              ELSEIF EXISTS(select posizione from frase
                                     where pagina = pagina_in and
                                    posizione = posizione_frase_in) THEN
28                                      INSERT INTO operazione_utente(
                                            dataA,dataR,testo,accettata,
                                            visionata,posizioneins,
                                            modifica,link,link_pagina,
                                            posizione_frase,pagina_frase,
                                            utente,autore_notificato)
29                                      VALUES(NULL,LOCALTIMESTAMP(0),
                                            Testo_in,0::BIT(1),0::BIT(1),
                                            posizioneins_in,modifica_in,
                                            link_in,NULL,
                                            posizione_frase_in,pagina_in,
                                            utente_in,(Select emailautore
                                             from pagina where pagina_in
                                            = id_pagina));
30                              ELSE
```

```
31                                          RAISE EXCEPTION 'E08: La frase
                                                non esiste';
32
33                                  END IF;
34              ELSE
35                  RAISE EXCEPTION 'E10: Utente non corretto';
36                          END IF;
37                  ELSE
38                          RAISE EXCEPTION 'E04: La pagina non esiste(per il
                                riferimento del link)';
39                  END IF;
40          ELSE
41                  RAISE EXCEPTION 'E09: La pagina non esiste';
42          END IF;
43  END;
44  $OperazioneUtenteRichiesta$ LANGUAGE PLPGSQL;
```

5.4.6 GetUserModPage

Scopo: Lista di tutti gli utenti che hanno almeno una volta proposto un operazione su almeno una pagina dell'autore in input oppure una Lista di utenti che hanno almeno una volta proposto un operazione su una pagina in input e su un autore specifico.

Descrizione: Utilizzeremo SQL dinamico per creare la query. Se l'input della pagina (paginaIn) sarà *'0'* Allora la funzione ritorna tutte gli utenti che hanno Richiesto un Operazione su almeno una pagina dell'autore che richiama la funzione. Quindi tramite un cursore scorro tutte le pagine dell'autore e le aggiungo alla condizione della query. Altrimenti aggiungo solo la pagina interessata. La funzione restituisce una stringa dove ogni utente e diviso tramite un *';'*

```
1  CREATE OR REPLACE FUNCTION GetUserModPage(emailIn IN VARCHAR(255),
       paginaIn IN INTEGER)
2  RETURNS TEXT
3  AS $GetUserModPage$
4  DECLARE
5      ListaUtenti TEXT := '';
6          Flag bit(1) := 0::bit(1);
7      sql_instr TEXT := 'select distinct utente from storicita_totale WHERE
           utente NOT LIKE ''' ||emailIn || ''' AND (id_pagina = ';
8          pagina_rec pagina%ROWTYPE;
9          user_rec storicita_totale.utente%TYPE;
10 BEGIN
11 IF paginaIn = 0 THEN
12     FOR pagina_rec IN SELECT id_pagina FROM pagina WHERE emailautore =
           emailIn LOOP
13           IF flag = 1::bit(1) THEN sql_instr := sql_instr || ' OR
               id_pagina = ' || pagina_rec.id_pagina;
14                   ELSE sql_instr := sql_instr || pagina_rec.id_pagina;
                         Flag := 1::bit(1); END IF;
15     END LOOP;
16 ELSE
17         CALL CheckAutore(emailIn, paginaIn);
18     sql_instr := sql_instr || paginaIn;
19 END IF;
20
21 sql_instr := sql_instr || ')';
22
23 FOR user_rec IN EXECUTE sql_instr LOOP
24     ListaUtenti := ListaUtenti || user_rec || ';';
25 END LOOP;
26
27 RETURN ListaUtenti;
28
29 END;
30 $GetUserModPage$ LANGUAGE plpgsql;
```

5.4.7 UpadatePosition

Scopo: Aggiorna la posizione da un certo valore. Aggiorna i valori dal più grande al più piccolo

Descrizione: Questa funzione dato la pagina (pagina_in) la posizione (posizione_in) e il caso aggiornerà tutte le posizioni che sono maggiori della posizione_in nel caso diverso da 1 (caso != 1) altrimenti quelli maggiori uguali. Questo perché 1 è un caso particolare e deve essere gestito singolarmente.

```
1  CREATE OR REPLACE PROCEDURE UpadatePosition(posizione_in integer ,
       pagina_in integer, caso BIT(1)) AS
2  $UpadatePosition$
3  DECLARE
4          f frase%ROWTYPE;
5
6  BEGIN
7  IF caso = 1::BIT THEN
8          FOR f IN SELECT pagina,posizione FROM frase where pagina_in =
               pagina and posizione >= posizione_in order by posizione DESC
               LOOP
9                   UPDATE frase set posizione = posizione + 200 where f.
                        pagina = pagina and posizione = f.posizione;
10         END LOOP;
11 ELSE
12         FOR f IN SELECT pagina,posizione FROM frase where pagina_in =
               pagina and posizione > posizione_in order by posizione DESC
               LOOP
13                  UPDATE frase set posizione = posizione + 200 where f.
                        pagina = pagina and posizione = f.posizione;
14         END LOOP;
15 END IF;
16 END;
17 $UpadatePosition$ LANGUAGE PLPGSQL;
```

5.4.8 AccettaProposta

Scopo: Procedura che permetta all'autore di accettare o rifiutare una proposta di modifica da parte di un utente.

Descrizione:La procedura andrà a registrare nel LOG di operazione_utente la data di accettazione/rifiuto e setterà l'attributo accettata a True. L'inserimento o l'aggiornamento delle modifiche di una frase sarà fatto da un trigger.

```
1  CREATE OR REPLACE PROCEDURE AccettaProposta (EmailIn IN d_Email,
       IdOperazione IN INTEGER, AccettataIn IN  bit(1))
2  AS $AccettaProposta$
3  DECLARE
4
5          notifica_rec notifica%ROWTYPE;
6
7  BEGIN
8          select * into notifica_rec from notifica where autore_notificato
               = EmailIn AND id_operazione = IdOperazione;
9          IF FOUND THEN
10                 UPDATE operazione_utente SET accettata = AccettataIn,
                       dataa = LOCALTIMESTAMP(0), visionata = 1::bit(1)
                       WHERE id_operazione = IdOperazione AND
                       autore_notificato = EmailIn;
11         ELSE
12                 RAISE EXCEPTION 'E01: Non esiste una proposta per % con
                       id_operazione %', EmailIn, IdOperazione;
13         END IF;
14
15 END;
16 $AccettaProposta$ LANGUAGE plpgsql;
```

5.4.9 GetLimtUserModPage

Scopo: Ci permette di vedere tutti gli utenti che hanno proposto almeno una volta una modifica/inserimento rispetto a una serie di pagine in input.

Descrizione: La funzione prenderà in input una stringa contente una serie di Id_pagina divisi da un *'+'*. Tramite sql dinamico creiamo una query che selezionerà tutti gli utenti che hanno almeno proposto una modifica/inserimento nelle pagine in input. Dopodiché aver preso tutte l'email distinte fra di loro inseriamo ogni email in una stringa di output, dove ogni email è diviso da un *'+'*.

```
1  CREATE OR REPLACE FUNCTION GetLimtUserModPage(Ids_page VARCHAR(255),
       emailIn VARCHAR(255)) RETURNS VARCHAR AS $GetLimtUserModPage$
2  DECLARE
3          pos INTEGER := 0;
4          oldpos INTEGER := 1;
5          parola varchar(10000) := '';
6          commando varchar(10000) := 'SELECT distinct(utente) FROM
               operazione_utente where utente NOT LIKE ''' || emailIn || '''
                AND (pagina_frase = ';
7          u operazione_utente.utente%TYPE;
8  BEGIN
9          LOOP
10                 pos := POSITION('+' IN SUBSTRING(Ids_page FROM oldpos));
11                 EXIT WHEN pos = 0;
12                 parola := SUBSTRING(Ids_page FROM oldpos FOR pos-1);
13                 oldpos := oldpos + pos;
14                 call CheckAutore(emailin,parola::INTEGER);
15                 commando := commando || parola || ' OR pagina_frase = ';
16         END LOOP;
17         parola := SUBSTRING(Ids_page FROM oldpos);
18         commando := commando || parola || ');';
19         parola := '';
20         FOR u in EXECUTE commando LOOP
21                 parola := parola || '+' || u;
22         END LOOP;
23
24         RETURN parola;
25 END;
26 $GetLimtUserModPage$ LANGUAGE PLPGSQL;
```

5.4.10 VisualizzaPropostaAndConfronta

Scopo: Funzione che permettere di visualizzare i dati di un inserimento e confrontare i dati tra una frase e la sua proposta di modifica.

Descrizione: La funzione verifica che l'id dell'operazione passata esiste, se l'utente che sta visualizzando l'operazione ha diritto a visualizzare la notifica e se è autore della pagina in cui è stata effettuata la proposta. Superati i controlli nel caso dell'inserimento la funzione restituisce una stringa, contente i dati della frase che l'utente vuole inserire, cosi formattata: **'titolo+testo,posizione,link,id_pagina_riferita,titolo_pagina_riferita'**. Nel caso di una modifica verranno confrontati i dati originali con quelli proposti, e la stringa restituita sarà cosi formattata: **'Titolo+testo, posizione,link, id_pagina_riferita, titolo_pagina_riferita; proposta_testo, proposta_posizione, proposta_link, proposta_id_pagina_riferita, proposta_titolo_pagina_riferita'**.

```
1  CREATE OR REPLACE FUNCTION visualizzaPropostaAndConfronta (idOp IN
       INTEGER, emailAutore IN VARCHAR(255) )
2  RETURNS TEXT
3  AS $visualizzaProposta$
4  DECLARE
5
6          confronto TEXT := '';
7          notifica_rec notifica%ROWTYPE;
8          frase_rec frase%ROWTYPE;
9
10         NotificaLinkPagina INTEGER := 0;
11         FraseLinkPagina INTEGER := 0;
12
13         FraseTitolo VARCHAR(255);
14         NotificaRefTitolo VARCHAR(255) := 'null' ;
15         FraseRefTitolo VARCHAR(255) := 'null' ;
16
17 BEGIN
18
19         select * into notifica_rec from notifica where id_operazione =
               idOp; -- se esiste quella notifica
20         raise notice 'notifica rec %', notifica_rec;
21         if found then
22
23                 if emailAutore != (Select autore_notificato from notifica
                       where id_operazione  = idop) then
24                         raise exception 'l''autore inserito non e
                               autorizzato a visualizzare questa notifica';
25                 end if;
26
27                 CALL checkautore(emailAutore, notifica_rec.pagina_frase);
28
```

```
29                  update operazione_utente set visionata = 1::bit(1) where
                        id_operazione = idOp; --aggiorno il visualizzato
30
31                  if notifica_rec.link_pagina is not null then
                        NotificaLinkPagina := notifica_rec.link_pagina;
                        select titolo into NotificaRefTitolo from pagina
                        where id_pagina = notifica_rec.link_pagina; end if;
32
33                  select titolo into FraseTitolo from pagina where
                        id_pagina = notifica_rec.pagina_frase;
34
35                  if notifica_rec.modifica = 1::bit(1) then
36                          select * into frase_rec from frase where pagina =
                                notifica_rec.pagina_frase AND posizione =
                               notifica_rec.posizione_frase;
37                          raise notice '%', frase_rec;
38
39                          if notifica_rec.link_pagina is not null then
                                FraseLinkPagina := frase_rec.linkpagina;
                                select titolo into FraseRefTitolo from pagina
                                 where id_pagina = notifica_rec.link_pagina;
                                end if;
40              raise notice 'FraseRefTitolo: %', FraseRefTitolo;
41                          confronto := FraseTitolo ||'+'||frase_rec.testo
                                ||'-'|| frase_rec.posizione ||'-' ||
                                frase_rec.link || '-'|| FraseLinkPagina || '-
                                ' || FraseRefTitolo ||'|'|| notifica_rec.
                                testo || '-' || notifica_rec.posizioneins ||
                                '-' || notifica_rec.link || '-' ||
                                NotificaLinkPagina || '-' ||
                                NotificaRefTitolo;
42
43                  else
44                          confronto := FraseTitolo ||'+'||notifica_rec.
                                testo || '-' || notifica_rec.posizioneins ||
                                '-' || notifica_rec.link || '-' ||
                                NotificaLinkPagina || '-' ||
                                NotificaRefTitolo;
45                  end if;
46
47          else
48                  raise exception 'non esisete una notifica con l''
                        identificativo proposto';
49          end if;
50          raise notice 'confronto hh: %', confronto;
51
52          RETURN confronto;
53  END;
54  $visualizzaProposta$
55  LANGUAGE plpgsql;
```

5.4.11 trovaPagina

Scopo: Permette di cercare le pagine wiki.

Descrizione: Una volta passata la stringa di ricerca/input la funzione andrà a cercare tutte le pagine che contengono nel titolo le stesse parole e ritornerà una stringa cosi formattata **'id1-id2-...-idn'** contenente gli id delle pagine che corrispondono alla ricerca. Se nessuna pagina viene trovata allora la funzione ritornerà **'-1'**.

```
1  CREATE OR REPLACE FUNCTION trovaPagine (testoRicerca IN TEXT)
2  RETURNS TEXT AS
3  $trovaPagine$
4  DECLARE
5
6          stringaRicerca TEXT  := '%';--||testoRicerca||'%';
7
8          pos INTEGER;
9          oldPos INTEGER:= 1;
10         parola TEXT;
11         page pagina%ROWTYPE;
12
13         idTrovati TEXT:= '';
14
15 BEGIN
16
17         LOOP
18
19                 pos := POSITION(' ' IN SUBSTRING(testoRicerca FROM oldPos
                       ));
20                 EXIT WHEN pos = 0;
21                 parola := SUBSTRING (testoRicerca FROM oldPos FOR pos-1);
22                 stringaRicerca := stringaRicerca || parola ||'%';
23                 oldPos := oldPos + pos;
24
25         END LOOP;
26
27         parola := SUBSTRING(testoRicerca FROM oldPos);
28         stringaRicerca := stringaRicerca || parola ||'%';
29
30         FOR page IN SELECT id_pagina FROM pagina WHERE titolo LIKE
               stringaRicerca LOOP
31                 idTrovati:=idTrovati||page.id_pagina||'-';
32         END LOOP;
33         idTrovati:=RTRIM(idTrovati, '-');
34
35         IF page IS NULL THEN
36                 idTrovati := '-1';
37         END IF;
38
```

```
39          RETURN idTrovati;
40
41  END;
42  $trovaPagine$
43  LANGUAGE plpgsql;
```

5.4.12 ModificaRichestaProposta

Scopo: Permette di modificare una proposta inviata.

Descrizione: La procedura controlla se esiste la proposta del utente. Se esiste modifica il testo altrimenti lancia un **EXCEPTION**.

```
1   CREATE OR REPLACE PROCEDURE ModificaRichestaProposta(Id_operazione_in
        INTEGER, Email_in VARCHAR(255), Testo_in TEXT) AS
        $ModificaRichestaProposta$
2   BEGIN
3
4           IF EXISTS(SELECT utente from operazione_utente where Email_in =
                utente and Id_operazione_in = id_operazione) THEN
5                   UPDATE operazione_utente SET testo = Testo_in where
                        id_operazione = id_operazione_in;
6
7           ELSE
8                   RAISE EXCEPTION 'E13: utente non ha questa proposta';
9
10          END IF;
11
12  END;
13  $ModificaRichestaProposta$ LANGUAGE PLPGSQL;
```

5.5 Viste

5.5.1 Storicita_totale

Implementazione di: *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.*

```
1 CREATE OR REPLACE VIEW Storicita_totale AS
2 SELECT pagina_frase AS id_pagina, dataa AS Data_Accettazione, datar AS
      Data_Richiesta, posizioneins AS posizione, testo, utente, accettata,
      modifica, link, link_pagina FROM
3     (SELECT pagina_frase, null::TIMESTAMP AS dataa, datar, posizioneins,
          testo, utente, accettata, modifica, link, link_pagina FROM
          operazione_utente WHERE dataa IS NULL)
4     UNION
5     (SELECT pagina_frase, dataa, datar, posizioneins, testo, utente,
          accettata, modifica, link, link_pagina FROM operazione_utente
          WHERE dataa IS NOT NULL)
6     UNION
7     (SELECT pagina_frase, data AS dataa, NULL AS datar, posizioneins,
          testo, autore AS utente, NULL AS accettata, NULL AS modifica,
          link, link_pagina FROM operazione_autore)
8 ORDER BY id_pagina, Data_Accettazione ASC;
```

5.5.2 Notifica

Implementazione di: *La modifica proposta verrà notificata all'autore del testo originale la prossima volta che utilizzerà il sistema.*

Descrizione: Vista che contiene tutte le notifiche di tutti gli autori che hanno una proposta di modifica ancora non accettata/rifiutata. Questa vista è utile per ottenere tutte le informazioni riguardanti le proposte in corso, per questo è preferibile una vista anziché una funzione funzione che faccia la stessa cosa filtrando per autore.

```
1 CREATE OR REPLACE VIEW Notifica as
2 SELECT id_operazione,datar,testo,accettata,visionata,posizioneins,
      modifica,link,link_pagina,posizione_frase,pagina_frase,utente,
      autore_notificato
3 FROM operazione_utente
4 WHERE dataa IS NULL
5 ORDER BY datar, visionata ASC;
```

5.5.3 Log_page

Implementazione di: *Tutti gli autori potranno vedere tutta la storia di tutti i testi dei quali sono autori.*

Descrizione: Questa vista permette di vedere la storicità di tutti i testi ordinati per pagina e data.

```
1  CREATE OR REPLACE VIEW log_page AS
2  SELECT data, pagina_frase,testo,posizione_frase,link,link_pagina  from (
3    SELECT data, pagina_frase, testo, posizione_frase,link,link_pagina
4    FROM Operazione_autore_ordered
5
6    UNION
7
8    SELECT data, pagina_frase, testo, posizione_frase,link,link_pagina
9    FROM Operazione_utente_ordered
10 ) order by pagina_frase, data ASC;
```

6 Dizionari degli Errori

1. 'E01: Non esiste una proposta per <email-autore> con id_operazione <id-operazione>'

   Descrizione: Eccezione lanciata dalla procedura AccettaProposta, quando l'autore tenta di accettare una proposta di operazione che non esiste oppure che non deve essere accettata da quell'autore.

2. 'E02: Questo utente non e autore della pagina'

   Descrizione: Eccezione lanciata dalla procedura CheckAutore, quando un utente-autore non è autore della pagina su cui si sta effettuando delle operazioni.

3. 'E03: Utente non trovato o utente non e autore'

   Descrizione: Eccezione lanciata dalla procedura CheckAutore, quando si sta esaminando l'operazione da effettuare e risulta che l'utente-autore che sta effettuando l'operazione non esiste oppure non è un autore.

4. 'E04: La pagina non esiste (link non valido)'

   Descrizione: Eccezione lanciata dalle procedure:

   - InsertFraseAutore
   - OperazioneUtenteRichiesta

   quando si sta esaminando l'operazione da effettuare e risulta che la pagina a cui il link fa riferimento non esiste.

5. 'E05: Utente non trovato'

   Descrizione: Eccezione lanciata dalla procedura CreazionePagina, quando l'utente che sta creando la pagina non corrisponde ad un utente registrato nel database

6. 'E06: utente non può essere cancellato'

   Descrizione: Eccezione lanciata dal trigger Utente_No_Delete, quando si effettua una qualunque operazione di cancellazione di un utente

7. 'E07: La modifica e gia stata visionata'

   Descrizione: Eccezione lanciata dal trigger Modifica_proposta, quando un utente sta tentando di modificare una proposta di modifica, ma l'autore ha già visionato la proposta.

8. 'E08:La frase non esiste'

   Descrizione: Eccezione lanciata dalla procedura OperazioneUtenteRichiesta quando si tenta di proporre una modifica ad una frase che non esiste.

9. 'E09:La pagina non esiste'

   Descrizione: Eccezione lanciata dalle Procedure:

   - OperazioneUtenteRichiesta
   - CheckAutore

   quando si tenta di inserire/proporre una frase in una pagina che non esiste.

10. 'E10: Utente non corretto'

    Descrizione: Eccezione lanciata dalla procedura OperazioneUtenteRichiesta quando l'utente che sta effettuando l'operazione non esiste.

11. 'E11: l"autore inserito non e autorizzato a visualizzare questa notifica'

    Descrizione: Eccezione lanciata dalla funzione visualizzaPropostaAndConfronta quando l'autore che sta tentando di visualizzare le proposte di modifica non è lo stesso autore della pagina in cui è stata effettuata la proposta.

12. 'E12: non esiste una notifica con l"identificativo proposto'

    Descrizione: Eccezione lanciata dalla funzione visualizzaPropostaAndConfronta quando la posposta non esiste oppure l'operazione è già stata accettata/rifiutata.

13. 'E13: utente non ha questa proposta'

    Descrizione: Eccezione lanciata dalla procedura ModificaRichestaProposta quando si tenta di modificare una proposta di modifica ma la proposta non esiste oppure l'utente che sta tentando di modificare la proposta non è lo stesso utente che ha effettuato la proposta di modifica.