Dcumentazione database

Sistema di gestione del ciclo di vita di una pagina wiki

Angelo Paolella Alfredo Volpe

Dicembre 2023

Indice

1 Progettazione Concettuale

1.1 Analisi dei Requisiti

Nella fase di **analisi dei requisiti** mediante lo studio della richiesta verranno identificate le informazioni fondamentali per la progettazione del database. Verranno individuate tutte le entità con i loro attributi e le associazioni che ci sono tra di loro. Iniziando con la lettura del primo paragrafo si individuano facilmente le classi principali :

"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* rappresenta la pagina web contenente *Testo*, titolo, data e ora di creazione. L'*Autore* rappresenta l'utente che è in grado di creare *Pagine*. Il *Testo* è contenuto in una *Pagina* e a sua volta contiene delle *Frasi*. La *Frase* rappresenta le frasi del testo contenuto in una *Pagina* e può contenere un collegamento ad esse.

"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. "

L'*Utente* può proporre una modifica di una *Pagina*, proponendo una *Frase*. Una *Modifica* è una frase proposta da un *Utente*.

"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 in ordine dalla più antica alla più recente. "

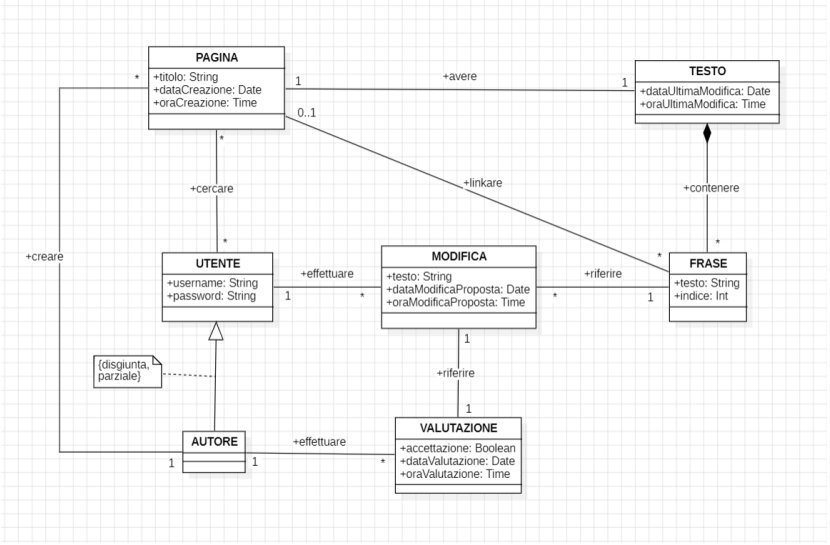
La *Modifica* che valuta l'*Autore* in una determinata dataModificaProposta e oraModificaProposta (poiché si deve tenere conto dell'ordine dalla proposta più antica a quella più recente), potrà essere accettata o rifiutata da esso, mediante l'attributo booleano accettazione, aggiornando rispettivamente dataValutazione e oraValutazione. Accettazione sarà automaticamente TRUE quando la modifica sarà effettuata direttamente dall'*Autore* stesso, invece dataModificaProposta

e oraModificaProposta saranno settate adeguatamente, Nel caso in cui l'*Autore* accetti la modifica proposta dall'*Utente* la *Frase* verrà modificata, in caso contrario rimarrà invariata.

"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."

L'*Utente* potrà cercare delle *Pagine*, trovando il *Testo* sempre aggiornato grazie all'utilizzo degli attributi dataUltimaModifica e oraUltimaModifica. Sia gli *Autori* che gli *Utenti* avranno un username e una password per accedere al sistema.

1.2 Schema Concettuale



1.3 Dizionario delle classi

Classe	Descrizione	Attributi
Pagina	Rappresenta una pagina web della wiki.	**titolo**(String): il titolo della pagina **dataCreazione**(Date): il giorno in cui l'autore ha creato la pagina **oraCreazione**(Time): l'ora in cui l'autore ha creato la pagina
Utente	Rappresenta gli utenti che interagiscono con le pagine, possono essere anche autori	**username**(String): username scelto dall'utente per accedere al sistema **password**(String): password scelta dall'utente per accedere al sistema
Modifica	Rappresenta una possibile modifica di una frase della pagina.	**testo**(String): testo all'interno di ogni frase **dataModificaProposta**(Date): data in cui è stata proposta la modifica **oraModificaProposta**(Time): ora in cui è stata proposta la modifica
Valutazione	Rappresenta la valutazione dell'*Autore* di una modifica.	**accettazione**(Boolean): l'accettazione o il rifiuto da parte di un autore per la modifica di una frase (in caso di accettazione=TRUE la frase proposta diventerà quella principale, in caso di FALSE la frase rimarrà invariata). **dataValutazione**(Date): data in cui è stata valutata la modifica **oraValutazione**(Time): ora in cui è stata valutata la modifica

Classe	**Descrizione**	**Attributi**
Autore	Rappresenta un utente che può creare pagine, è una generalizzazione disgiunta parziale di *Utente*	
Testo	Racchiude la totalità del testo in una pagina ed è composto da frasi	**dataUltimaModifica**(Date): data in cui è stata effettuata l'ultima modifica al testo **oraUltimaModifica**(Time): ora in cui è stata effettuata l'ultima modifica al testo
Frase	Rappresenta una frase contenuta in un testo	**testo**(String): testo all'interno di ogni frase **indice**(Serial): posizione in cui si trova la frase nel testo, si autoincrementa ogni volta che viene aggiunta una frase

1.4 Dizionario delle associazioni

Associazione	Descrizione
creare	Associazione tra *Pagina* e *Autore* **uno-a-molti** poiché una pagina è creata da un autore mentre un autore può creare più pagine (un autore da poco registrato può non aver ancora creato una pagina).
effettuare	Associazione tra *Modifica* e *Utente* **uno-a-molti** poiché una modifica viene effettuata da un utente, mentre un utente può effettuare più modifiche.
riferire	Associazione tra *Modifica* e *Frase* **uno-a-molti** poiché una modifica si riferisce a una frase, mentre una frase può essere selezionata da più modifiche.
effettuare	Associazione tra *Valutazione* e *Autore* **uno-a-molti** poiché una valutazione viene effettuata da un autore, mentre un autore può effettuare più valutazioni.
riferire	Associazione tra *Valutazione* e *Modifica* **uno-a-uno** poiché una valutazione si riferisce a una modifica e una modifica è riferita ad una valutazione.
cercare	Associazione tra *Pagina* e *Utente* **molti-a-molti** poiché una pagina può essere cercata da più utenti e un utente può cercare più pagine.

Associazione	**Descrizione**
Linkare	Associazione tra *Frase* e *Pagina* **uno-a-molti** poiché una frase può linkare a una pagina, mentre una pagina può essere linkata da più frasi (nella maggior parte dei casi abbiamo che una frase non linka a nessuna pagina).
Contenere	Composizione tra *Frase* e *Testo* **uno-a-molti** poiché una frase non può esistere se non appartiene ad un testo, mentre un testo può essere aggregante di più frasi (possiamo avere il caso in cui il testo è stato appena creato quindi non ha nessuna frase cioè è vuoto).
Avere	Associazione tra *Pagina* e *Testo* **uno-a-uno** poiché una pagina può avere un solo testo, e un testo può esistere in una sola pagina.

2 Ristrutturazione del modello concettuale

2.1 Analisi delle ridondanze

Non sono presenti ridondanze.

2.2 Eliminazione attributi multivalore

Non sono presenti attributi multivalore.

2.3 Eliminazione attributi strutturati

Non sono presenti attributi strutturati.

2.4 Analisi delle generalizzazioni

Procediamo con l'eliminazione dell'unica generalizzazione: accorpiamo *Autore* in *Utente* poiché, le operazioni che svolgono, non fanno molta distinzione tra le occorrenze; in più l'*Autore* non ha attributi, perciò non ne avremo nessuno a NULL. Così facendo avremo anche un minor numero di accessi. Per quanto riguarda la differenza tra autore e utente, capiamo che un utente diventa autore nel momento in cui crea una pagina, così sarà in grado anche di valutare le proposte alla sua pagina (in questo modo l'utente potrà fare anche le due operazioni in più che prima poteva fare l'autore), Le associazioni che aveva *Autore* ora le avrà l'*Utente*, rispettivamente: **creare**(tra *Autore* e *Pagina*) e **effettuare**(tra *Autore* e *Valutazione*).

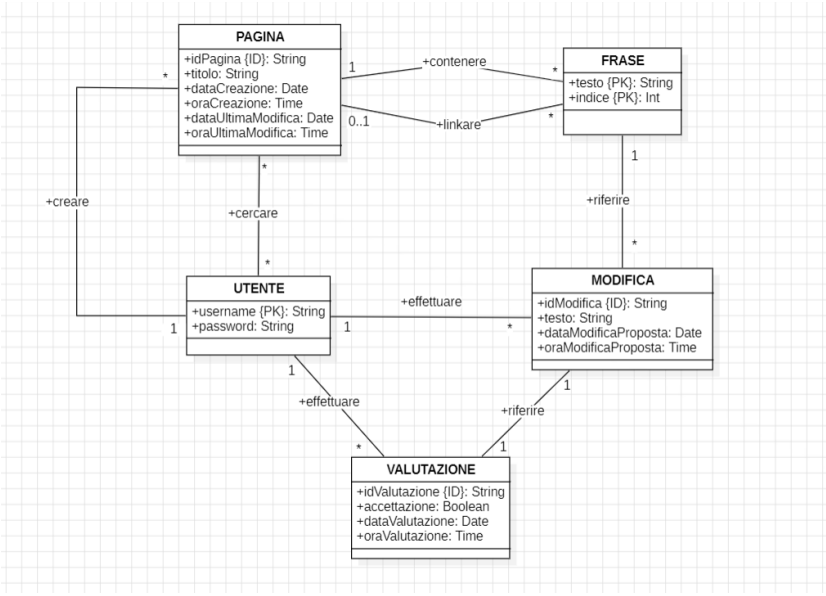
2.5 Partizionamento/Accorpamento di classi e associazioni

Procediamo con l'accorpamento delle classi *Pagina* e *Testo*, poiché sono collegate da un'associazione **uno-a-uno**. Portiamo gli attributi di *Testo* in *Pagina* e con essi anche la composizione con *Frase*. Tale composizione tra *Pagina* e *Frase* verrà trattata come associazione **1-N** con *Frase* come entità debole.
Per l'associazione **1-1** tra *Modifica* e *Valutazione* abbiamo deciso di non procedere con l'accorpamento, per ridurre la presenza di valori a NULL, poiché, accorpando le due classi, avremmo che gli attributi di *Valutazione* saranno a NULL fino a quando non ci sarà la valutazione effettiva, per questo motivo abbiamo deciso di rimanere le due classi separate.

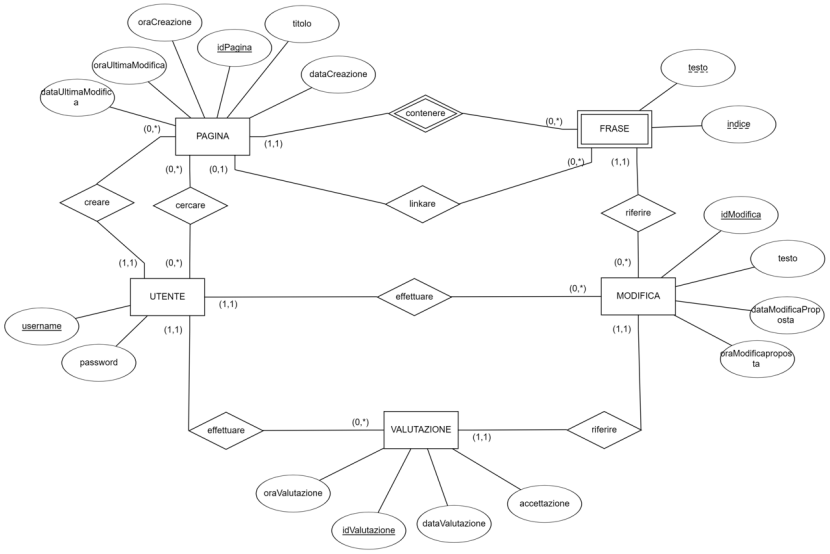
2.6 Identificazione delle chiavi primarie

- *Pagina* è identificata da un nuovo attributo **idPagina**, di tipo serial.
- In *Frase* abbiamo una chiave parziale formata da **testo** e **indice**: rispettivamente, sono il testo della frase formato da massimo 100 caratteri, e l'indice, cioè un intero, che indica la posizione della frase nel testo della pagina.
- *Modifica* è identificata da un nuovo attributo **idModifica**, di tipo serial.
- *Valutazione* è identificata da un nuovo attributo **idValutazione**, di tipo serial.
- *I*n Utente abbiamo l'attributo **username**, una stringa formata da massimo 15 caratteri.

2.7 Schema ristrutturato UML



2.8 Schema ristrutturato ER



2.9 Dizionario delle classi ristrutturato

Classe	Descrizione	Attributi
Pagina	Rappresenta una pagina web della wiki.	**idPagina**(Serial): Identificativo della pagina che si auto incrementa **titolo**(String): il titolo della pagina **dataCreazione**(Date): il giorno in cui l'autore ha creato la pagina **oraCreazione**(Time): l'ora in cui l'autore ha creato la pagina **dataUltimaModifica**(Date): il giorno in cui è stata effettuata l'ultima modifica **oraUltimaModifica**(Time): l'ora in cui è stata effettuata l'ultima modifica
Modifica	Rappresenta una possibile modifica di una frase della pagina.	**idModifica**(Serial): Identificativo della modifica che si auto incrementa **testo**(String): testo all'interno di ogni frase di massimo 100 caratteri **dataModificaProposta**(Date): data in cui è stata proposta la modifica **oraModificaProposta**(Time): ora in cui è stata proposta la modifica

Classe	**Descrizione**	**Attributi**
Utente	Rappresenta un utente registrato nel sistema.	**username**(String): username univoco composto da massimo 15 caratteri ed è scelto dall'utente per accedere al sistema **password**(String): password scelta dall'utente per accedere al sistema
Valutazione	Rappresenta la valutazione di una frase da parte di un utente.	**idValutazione**(Serial): identificativo della valutazione che si auto incrementa. **accettazione**(Boolean): l'accettazione o il rifiuto da parte di un autore per la modifica della propria pagina (in caso di accettazione = TRUE la frase proposta prenderà il posto di quella modificata, in caso di FALSE la frase originale rimarrà invariata) **dataValutazione**(Date): data in cui è stata valutata la modifica **oraValutazione**(Time): ora in cui è stata valutata la modifica
Frase	Rappresenta una frase contenuta in un testo	**indice**(Int): un numero intero indicativo per la posizione della frase nel testo, si auto incrementa tramite un trigger ogni volta che viene inserita una frase, inoltre avrà come valore default di partenza 0 **testo**(String): testo composto da massimo 100 caratteri all'interno della frase

2.10 Dizionario delle associazioni ristrutturato

Associazione	Descrizione
creare	Associazione tra *Pagina* e *Utente* **uno-a-molti** poiché una pagina è creata da un utente mentre un utente può creare più pagine.
effettuare	Associazione tra *Modifica* e *Utente* **uno-a-molti** poiché una modifica viene effettuata da un utente, mentre un utente può effettuare più modifiche.
riferire	Associazione tra *Modifica* e *Frase* **uno-a-molti** poiché una modifica si riferisce a una frase, mentre una frase può riferirsi a più modifiche.
effettuare	Associazione tra *Valutazione* e *Utente* **uno-a-molti** poiché una valutazione viene effettuata da un utente (l'autore della pagina), mentre un utente può effettuare più valutazioni.
riferire	Associazione tra *Valutazione* e *Modifica* **uno-a-uno** poiché una valutazione si riferisce a una modifica e una modifica viene valutata da una valutazione.
cercare	Associazione tra *Pagina* e *Utente* **molti-a-molti** poiché una pagina può essere cercata da più utenti e un utente può cercare più pagine.
linkare	Associazione tra *Frase* e *Pagina* **uno-a-molti** poiché una frase può linkare a una pagina, mentre una pagina può essere linkata da più frasi (nella maggior parte dei casi abbiamo che una frase non linka a nessuna pagina).
contenere	Associazione tra *Frase* e *Pagina* **uno-a-molti** poiché una frase non può esistere se non appartiene ad una pagina, mentre una pagina può contenere più frasi (possiamo avere il caso in cui la pagina è stata appena creata quindi non ha nessuna frase).

3 Traduzione al modello logico

3.1 Mapping associazioni

3.1.1 Associazioni 1-N

- *Modifica-Effettuare-Utente*: Inserimento chiave di *Utente* in *Modifica* come chiave esterna.
- *Valutazione-Effettuare-Utente*: Inserimento chiave di *Utente* in *Valutazione* come chiave esterna.
- *Pagina-Creare-Utente*: Inserimento chiave di *Utente* in *Pagina* come chiave esterna.
- *Frase-Contenere-Pagina*: Inserimento chiave di *Pagina* in *Frase* come chiave esterna, ma è anche chiave primaria insieme a **testo** e **indice** poiché *Frase* è un'entità debole e tramite questa associazione l'**idPagina** va a completare la chiave di *Frase*.
- *Modifica-Riferire-Frase*: Inserimento chiave di *Frase* in *Modifica* come chiave esterna.
- *Frase-Linkare-Pagina*: Entrambe hanno una partecipazione parziale, si procede come *N-N*, per ridurre la presenza di valori a NULL.

3.1.2 Associazioni N-N

- *Utente-Cercare-Pagina*: Inserimento chiave di *Utente* e *Pagina* in *Cercare* come chiavi esterne.

3.1.3 Associazioni 1-1

- *Modifica-Riferire-Valutazione*: Inserimento chiave di *Modifica* in *Valutazione* come chiave esterna.

3.2 Modello logico

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

Utente (username, password)

Pagina (idPagina, titolo, dataCreazione, oraCreazione, dataUltimaModifica, oraUltimaModifica, usernameAutore*)
usernameAutore →Utente.username

Cercare (idPagina*, username*)
username →Utente.username
idPagina →Pagina.idPagina

Frase (testo, indice, idPagina*)
idPagina →Pagina.idPagina
poiché *Frase* è un'entità debole, infatti ha come chiave parziale **testo** e **indice**, che viene completata da **idPagina**. Questi tre attributi insieme formano la chiave primaria di *Frase*.

Modifica (idModifica, testo, dataModificaProposta, oraModificaProposta, username*, testoFrase*, indice*, idPaginaFrase*)
username →Utente.username
idPaginaFrase →Frase.idPagina
testoFrase →Frase.testo
indice →Frase.indice

Valutazione (idValutazione, accettazione, dataValutazione, oraValutazione, usernameAutore*, idModifica*)
usernameAutore →Utente.username
idModifica →Modifica.idModifica

Linkare (idPaginaLinkata*, idPaginaFrase*, testo*, indice*)
idPaginaLinkata →Pagina.idPagina
idPaginaFrase →Frase.idPagina
testo →Frase.testo
indice →Frase.indice

4 Progettazione fisica

4.1 Creazione del database

- `CREATE DATABASE Progetto;`

4.2 Creazione dello schema

- `CREATE schema wiki AUTHORIZATION postgres;`
- `SET search\_path TO wiki;`

4.3 Creazione domini

- `create domain vartype as varchar(15) NOT NULL;`
- `create domain maxlength as varchar(100) NOT NULL;`

4.4 Creazione tabelle

- ```
  create table utente
  (
      username vartype,
      password vartype,

      constraint pkUtente primary key(username),

      constraint checkLengthPassword check(length(password) > 5)
  );
  ```

- ```
  create table pagina
  (
      idpagina serial,
      titolo vartype,
      datacreazione date NOT NULL DEFAULT CURRENTDATE,
      oracreazione time NOT NULL DEFAULT CURRENTTIME,
      dataultimamodifica date,
      oraultimamodifica time,
      usernameautore vartype,

      constraint pkPagina primary key(idpagina),
      constraint fkAutore foreign key(usernameautore) references utente(username)
      on update cascade
      on delete set NULL,

      constraint checkTitolo check(length(titolo) > 2),
      constraint checkDate check((dataultimamodifica > datacreazione) or
      (dataultimamodifica = datacreazione and oraultimamodifica >= oracreazione))
  );
  ```

- ```
  create table cercare (
      username vartype,
      idpagina int NOT NULL,

      constraint pkCercare primary key(username, idpagina),
      constraint fkUtente foreign key(username) references utente(username)
      on update cascade
      on delete set null,
      constraint fkPagina foreign key(idpagina) references pagina(idpagina)
      on update cascade
      on delete set null
  );
  ```
  ```

- ```
  create table frase (
      testo maxlength,
      indice int NOT NULL default 0,
      idpagina int NOT NULL,

      constraint pkFrase primary key(testo, indice, idpagina),
      constraint fkPagina foreign key(idpagina) references pagina(idpagina)
      on update cascade
      on delete set null,
      constraint checkTesto check(length(testo)>0),
      constraint uniqueFrase unique(testo,idpagina),
  );
  ```

- ```
  create table modifica (
      idmodifica serial,
      testo maxlength,
      datamodificaproposta date NOT NULL DEFAULT CURRENTDATE,
      oramodificaproposta time NOT NULL DEFAULT CURRENTTIME,
      username vartype,
      testofrase maxlength,
      indice int NOT NULL,
      idpaginafrase int NOT NULL,

      constraint pkModifica primary key(idmodifica),
      constraint fkUtente foreign key(username) references utente(username)
      on update cascade
      on delete set null,
      constraint fkFrase foreign key(testofrase, indice, idpaginafrase)
      references frase(testo, indice, idpagina)

      on update cascade
      on delete set null,
      constraint checkTesto check (length(testo)>0),
      constraint checkFrasiDiverse check(testo<>testofrase)
      constraint uniqueDate unique(dataModificaproposta,oraModificaproposta,idPaginaFrase)
  );
  ```

- ```
  create table valutazione (
      idvalutazione serial,
      accettazione boolean NOT NULL,
      datavalutazione date NOT NULL DEFAULT CURRENTDATE,
      oravalutazione time NOT NULL DEFAULT CURRENTTIME,
      usernameautore vartype,
      idmodifica int NOT NULL,

      constraint pkValutazione primary key(idvalutazione),
      constraint fkAutore foreign key(usernameautore) references utente(username)
      on update cascade
      on delete set null,
      constraint fkModifica foreign key(idmodifica) references modifica(idmodifica)
      on update cascade
      on delete set null
      constraint uniqueIdModifica unique(idModifica)
  );
  ```

- ```
  create table linkare (
      idpaginalinkata int NOT NULL,
      idpaginafrase int NOT NULL,
      testo maxlength,
      indice int NOT NULL,

      constraint pkLinkare primary key(idpaginalinkata,
      idpaginafrase, testo, indice),
      constraint fkPagina foreign key(idpaginalinkata) references pagina(idpagina)
      on update cascade
      on delete set null,
      constraint fkFrase foreign key(idpaginafrase, testo, indice)
      references frase(idpagina, testo, indice)

      on update cascade
      on delete set null,
      constraint checkLinkPagina check(idpaginalinkata<>idpaginafrase)
  );
  ```

4.5 Trigger e Trigger Function

- **Trigger:**

```
create trigger aggiornaUltimaModifica
after insert or update of accettazione on valutazione
for each row
when (NEW.accettazione = true)
    EXECUTE FUNCTION aggiornaUltimaModificaFunction();
```

Trigger Function:

```
CREATE OR REPLACE function aggiornaUltimaModifica() RETURNS TRIGGER AS
$$
DECLARE
    idPaginaAppoggio int;
    testoappoggio  varchar(100);
    indiceappoggio int;
    idPaginaAppoggioFrase  int;
    testoProposto  varchar(100);
BEGIN
    -- Trovo la pagina in cui è stata effettuata la valutazione
    SELECT p.idpagina
    into idPaginaAppoggio
    FROM pagina p, utente u, valutazione v
    where p.usernameAutore = u.username and u.username = v.usernameAutore
    and v.idvalutazione = new.idvalutazione;

    -- Aggiorno l'ultima modifica della pagina poichè la valutazione è
    -- stata accettata
    UPDATE pagina
    SET dataUltimaModifica = new.dataValutazione, oraUltimaModifica =
    new.oraValutazione
    WHERE idPaginaAppoggio = idPagina;

    SELECT f.testo,f.indice,f.idpagina,m.testo as testoprop
    into testoAppoggio,indiceAppoggio,idPaginaAppoggioFrase,testoProposto
    FROM frase f, modifica m, valutazione v
    where f.testo = m.testofrase and f.indice=m.indice and
    f.idpagina=m.idPaginaFrase
    and m.idmodifica=v.idModifica and v.idvalutazione = new.idvalutazione;

    -- Creo la nuova frase con lo stesso indice
    insert into frase (testo,indice,idpagina) values
    (testoProposto,indiceAppoggio,idPaginaAppoggio);

    return NEW;
```

```
end;
$$
LANGUAGE plpgsql;
```

Descrizione: Abbiamo per prima un trigger che viene attivato all'inserimento o all'aggiornamento di una valutazione di una frase, quando l'accettazione=TRUE, esso manda in esecuzione la funzione "aggiornaUltimaModificaFunction()"; Tale funzione che ritorna un trigger per prima cosa vede a quale pagina è riferita la valutazione, aggiornandone poi la data e ora di ultima modifica (essendo cambiata la pagina). Infine va ad aggiungere in frase la frase accettata, non eliminando la precedente, in modo da mantenere uno storico, per chi volesse vedere tutte le modifiche di una frase o di una pagina o addirittura una frase in un determinato indice. Il problema potrebbe essere come si fa a sapere qual'è l'ultima versione di una pagina ? Abbiamo pensato ad una procedura che vedremo nella prossima sezione.

- **Trigger:**

```
create trigger inserimentoPagina
after insert on pagina
for each row
EXECUTE FUNCTION inserimentoPaginaFunction();
```

Trigger Function:

```
CREATE OR REPLACE function inserimentoPaginaFunction() RETURNS TRIGGER AS
$$
BEGIN
    -- Quando la pagina è stata appena creata la data (e ora) di ultima
    modifica viene impostata
    -- alla data (e ora) di creazione
    UPDATE pagina
    SET dataultimamodifica = new.datacreazione, oraultimamodifica =
    new.oracreazione
    where idpagina = new.idpagina;

    return NEW;
end;
$$
LANGUAGE plpgsql;
```

Descrizione: Trigger che viene attivato all'inserimento di una pagina creata da un'utente che manda in esecuzione la funzione "inserimentoPaginaFunction()"; Tale funzione che ritorna un trigger semplicemente appena creata una pagina, setta subito la data e ora di ultima modifica uguali alla data e ora di creazione, per non avere attributi a NULL e per effetuare dopo controlli sulla data e ora di ultima modifica.

- **Trigger:**

```
create or replace trigger verificaAutore
after insert or update on valutazione
for each row
 execute function verificaAutoreFunction();
```

Trigger Function:

```
CREATE OR REPLACE function verificaAutoreFunction() RETURNS TRIGGER AS
$$
DECLARE

    pagineAutore cursor for
    select p.idpagina
    from utente u, pagina p
    where new.usernameautore=u.username and u.username=p.usernameautore;
    paginaCorrente int;

    paginaAppoggio int;

    controllo int;

BEGIN

    controllo:=0;

    select m.idpaginafrase into paginaAppoggio
    from modifica m
    where new.idmodifica=m.idmodifica;

    open pagineAutore;

    loop

    fetch pagineAutore into paginaCorrente;

    if not found then
     EXIT;
```

```
    end if;

    --Controllo se l'autore delle pagine coincide con quello inserito,
    --prenendolo dalla modifica associata alla valutazione
    if paginaCorrente = paginaAppoggio then
        controllo:=1;
    end if;

    end loop;

    if controllo = 1 then
        RAISE NOTICE 'Perfetto puoi continuare l autore e corretto';
        RETURN NEW;
    else
        RAISE EXCEPTION 'ATTENZIONE L AUTORE CHE HAI INSERITO NON E
        IL CREATORE DELLA PAGINA, RIPROVA!';
    end if;

    close pagineAutore;

end;
$$
LANGUAGE plpgsql;
```

Descrizione: Abbiamo per prima un trigger che viene attivato all'inserimento o all'aggiornamento di una valutazione di una frase, esso manda in esecuzione la funzione "verificaAutoreFunction()"; Tale funzione che ritorna un trigger, va a controllare che quando effettuiamo una valutazione, tale valutazione la faccia l'autore creatore di quella pagina, in caso non si è inserito l'autore corretto l'inserimento viene bloccato.

- **Trigger:**

```
create trigger inserimentoFrase
after insert on frase
for each row
EXECUTE FUNCTION inserimentoFraseFunction();
```

Trigger Function:

```
CREATE OR REPLACE function inserimentoFraseFunction() RETURNS TRIGGER AS
$$
DECLARE
    countAppoggio int;
    maxIndiceAppoggio int;
```

```
    countAppoggioIndice int;
BEGIN

    -- Controllo quante frasi hanno questo indice
    SELECT count(*) into countAppoggioIndice
    FROM frase
    WHERE new.idpagina=idpagina and new.indice=indice;

    -- Se ci sono altre frasi con lo stesso indice allora questa sarà una
    -- modifica
    if countAppoggioIndice > 1 then
        return new;
    end if;

    -- Se non è stato specificato l'indice la frase non è una modifica e il
    -- suo indice sarà di default = 0
    -- Conto quante frasi ci sono nella pagina
    SELECT count(*) into countAppoggio
    FROM frase
    WHERE new.idpagina=idpagina;

    -- Se la frase è l'unica allora il suo indice sarà 1
    if countAppoggio = 1 then
        UPDATE frase
        SET indice = 1
        WHERE new.idpagina=idpagina;
    else -- Altrimenti sarà uguale all'indice massimo + 1
        SELECT MAX(indice) into maxIndiceappoggio
        FROM frase
        WHERE new.idpagina=idpagina;

        UPDATE frase
        SET indice = maxIndiceAppoggio + 1
        WHERE new.idpagina=idpagina and new.testo=testo and
        new.indice=indice;
    end if;

    return NEW;
end;
$$
LANGUAGE plpgsql;
```

Descrizione: Trigger che viene attivato all'inserimento di una frase in una pagina che manda in esecuzione la funzione "inserimentoFraseFunction()"; Tale funzione che ritorna un trigger serve per manutenere al meglio gli indici di una frase (cioè di avere una sequenza

ordinata per ogni pagina). Per prima cosa vedo se quell'indice già è presente nella mia pagina, se si allora si tratterà di una modifica accettata, quindi gli rimango quell'indice che ha, Invece se non ce ne sono altri uguali sto inserendo una frase normalmente, quindi faccio un conteggio di quante frasi ci sono in quella pagina, se non c'è ne nemmeno una allora l'indice della nuova frase va a 1, mentre se ci sono già altre frasi, mi calcolo l'indice massimo nella pagina e alla nuova frase gli assegno l'indice massimo + 1. In questo modo avremmo per ogni pagina indici coerenti ed ordinati.

- **Trigger:**

```
create trigger modificaDellAutore
after insert or update on modifica
for each row
execute function modificaDellAutoreFunction();
```

Trigger Function:

```
create or replace function modificaDellAutoreFunction() returns trigger as
$$
DECLARE
autore varchar(15);
BEGIN
    select p.usernameautore
    into autore
    from frase f, pagina p
    where new.testofrase = f.testo and new.indice = f.indice and
    f.idpagina = new.idpaginafrase and f.idpagina = p.idpagina;

    if(autore <> new.username) then
        return NEW;
    end if;

    -- L'Autore è lo stesso che ha proposto la modifica quindi sarà
    -- accettata in automatico

    insert into valutazione(accettazione, datavalutazione, oravalutazione,
    usernameautore, idmodifica)
    values('true', new.datamodificaproposta, new.oramodificaproposta,
    autore, new.idmodifica);
    RAISE NOTICE 'Modifica accettata in automatico';

    return NEW;

END;
$$
```

```
language plpgsql;
```

Descrizione: Trigger che viene attivato al momento di un inserimento o aggiornamento di una modifica di una frase che manda in esecuzione la funzione "modificaDellAutoreFunction()"; Tale funzione che ritorna un trigger si recupera l'autore della pagina a cui è stata apportata la modifica, va a controllare se è stato proprio l'autore ad effettuare tale modifica, in caso positivo la modifica verrà accettata automaticamente (andando a modificare quindi ultima versione delle pagina con data e ora), in caso contratio verrà inserita la modifica normalmente.

- **Trigger:**

```
create or replace trigger correzioneValutazioni
after insert on valutazione
for each row
 execute function correzioneValutazioniFunction();
```

Trigger Function:

```
CREATE OR REPLACE function correzioneValutazioniFunction() RETURNS TRIGGER AS
$$
DECLARE

    idModificaAppoggio int;

paginaRef int;

    --Le modifiche non ancora valutate
    modificheNonValutate refcursor;

comandosql text:='select m.idmodifica, m.testo, m.dataModificaProposta,
        m.oraModificaProposta, m.username, m.testofrase, m.indice, m.idpaginafrase
    from modifica m
    where m.idpaginafrase = $1 and m.idmodifica not in (select v.idmodifica
                                                        from valutazione v)';

    modificaCorrente modifica%rowtype;

    datamin date;

    oraMin time;

pagAppoggio int;
```

```
BEGIN

select m.idpaginafrase into paginaRef
from modifica m
where new.idmodifica=m.idmodifica;

    OPEN modificheNonValutate FOR EXECUTE comandosql USING paginaRef;

    --Metto in dataMin e oraMin la data e l'ora della modifica a cui è
    --riferita la valutazione
    select m.dataModificaProposta, m.oraModificaProposta, m.idpaginafrase
    into dataMin, oraMin, pagAppoggio
    from modifica m
    where new.idmodifica=m.idmodifica;

    loop

    FETCH modificheNonValutate INTO modificaCorrente;

    if not found then
     EXIT;
    end if;

     if(modificaCorrente.dataModificaProposta < dataMin or
       ((modificaCorrente.dataModificaProposta = dataMin)
       and (modificaCorrente.oraModificaProposta < oraMin))) then

       dataMin:=modificaCorrente.dataModificaProposta;
       oramin:=modificaCorrente.oramodificaProposta;

     end if;

    end loop;

    select m.idmodifica into idModificaAppoggio
    from modifica m
    where m.dataModificaProposta = datamin and m.oraModificaProposta = oramin
    and m.idpaginafrase = pagAppoggio;

    if new.idmodifica = idModificaAppoggio then
        RAISE NOTICE 'Perfetto la data coincide con
        la data minore non ancora valutata';
        RETURN NEW;
    else
        RAISE EXCEPTION 'ATTENZIONE LA DATA NON E LA MINORE, RIPROVA!';
    end if;
```

```
    close modificheNonValutate;

end;
$$
LANGUAGE plpgsql;
```

Descrizione: Trigger che viene attivato al momento di un inserimento di una valutazione di una frase che manda in esecuzione la funzione "correzioneValutazioniFunction()"; Tale funzione che ritorna un trigger va a vedere tra le modifiche non ancora valutate da quell'autore, quale sia la minore, nel caso quella inserita sia la minore allora l'inserimento procederà normalmente, altrimenti il sistema darà errore, dicendo che quella non sia la data minore e quindi la correzione ancora non potrà avvenire.

- **Trigger:**

```
create trigger coerenzaDataValutazione
after insert or update on valutazione
for each row
EXECUTE FUNCTION coerenzaDataValutazioneFunction();
```

Trigger Function:

```
create or replace function coerenzaDataValutazioneFunction() returns
trigger as
$$
DECLARE
dataAppoggiomod date;
oraAppoggiomod time;
BEGIN
    select m.dataModificaproposta , m.oramodificaproposta
    into dataAppoggiomod,oraAppoggiomod
    from valutazione v, modifica m
    where new.idmodifica = m.idmodifica and
    new.idvalutazione=v.idvalutazione;

    if(new.datavalutazione < dataAppoggiomod or (new.datavalutazione =
    dataAppoggiomod and new.oravalutazione < oraAppoggiomod)) then
        RAISE EXCEPTION 'ATTENZIONE DATE NON COERENTI, RIPROVA!';
        --poichè non è possibile che una frase sia valutata prima della
        -- modifica stessa
    end if;

    -- Altrimenti l'inserimento continuerà senza problemi

    RAISE NOTICE 'Date coerenti puoi continuare!';

    return NEW;

END;
$$
language plpgsql;
```

Descrizione: Trigger che viene attivato all'inserimento o all'aggiornamento di una valutazione di una frase che manda in esecuzione la funzione "coerenzaDataValutazioneFunction()"; Tale funzione che ritorna un trigger va a controllare se le date che vengono inserite sono coerenti. Va a controllare che la data valutazione sia maggiore o uguale (in caso sono uguali andrà a controllare che l'ora della valutazione sia maggiore di quella della modifica) della data di modifica proposta (come normale che sia), in caso questa cosa accada uscirà un messaggio d'errore e la valutazione non sarà inserita, mentre in caso positivo l'inserimento procederà normalmente. In questo modo abbiamo che data valutazione sarà anche sempre maggiore della data creazione poiché appena creata una pagina la data ultima modifica viene messa uguale a quella creazione quindi e come se già lo sapessimo.

4.6 Procedure e Funzioni

```
create or replace procedure StampaUltimaVersionePagina(in idpagInput int) as
$$
DECLARE
countAppoggio int;
testoAppoggio varchar(100);
indiceAppoggio int;
idPaginaAppoggio int;
controllo int;
-- Questo cursore ad ogni fetch restituisce il testo, l'indice e
        -- l'idpagina di ogni frase ordinate per l'indice
frasiPagina cursor for
select f.testo,f.indice,f.idpagina
    from frase f left join modifica m on (f.idpagina = idpaginput and f.idpagina = m.idpagin
         left join valutazione v on (v.idmodifica = m.idmodifica and accettazione = true)
    order by f.indice asc, m.datamodificaproposta asc, m.oramodificaproposta asc;
fraseCorrente frase%rowtype;

BEGIN

RAISE NOTICE 'Frasi della pagina: %', idpagInput;

-- Questa tabella serve a tenere traccia degli indici già visitati
execute 'create table IndiciVisitati (indice int)';

open frasiPagina;

loop

controllo:=0;

fetch frasiPagina into fraseCorrente;

IF NOT FOUND THEN
        EXIT;
    END IF;

select count(*) into countAppoggio
from IndiciVisitati i
where i.indice=fraseCorrente.indice;

-- Controllo se ho già visitato questo indice, in tal
           --caso salto il loop corrente
if countAppoggio>0 then
continue;
```

```
end if;

-- Conto quante frasi hanno lo stesso indice
SELECT count(*) into countAppoggio
FROM frase f
WHERE idpagInput=f.idpagina and f.indice=fraseCorrente.indice;

-- Se c'è più di una frase con lo stesso indice allora controllerò
            -- quale sarà l'ultima valutata
if countAppoggio > 1 then
select m.testo, m.indice, m.idpaginafrase into testoAppoggio,indiceAppoggio,idpaginaAppoggio
from frase f join modifica m on (f.idpagina = idpaginput and f.indice = fraseCorrente.indice
                 join valutazione v on (v.idmodifica = m.idmodifica and v.accettazione = tru
            order by f.indice asc, v.datavalutazione desc, v.oravalutazione desc LIMIT 1;

controllo = 1;

end if;

insert into IndiciVisitati (indice) values (fraseCorrente.indice);

-- Se il controllo sarà 0 vorrà dire che non ci sono più frasi con lo
        -- stesso indice, altrimenti stampo quella appropriata
if controllo = 0 then
RAISE NOTICE 'testo: %', fraseCorrente.testo;
RAISE NOTICE 'indice: %', fraseCorrente.indice;
RAISE NOTICE 'idpagina: %', fraseCorrente.idpagina;
elsif controllo = 1 then
RAISE NOTICE 'testo: %', testoAppoggio;
RAISE NOTICE 'indice: %', indiceAppoggio;
RAISE NOTICE 'idpagina: %', idPaginaAppoggio;
end if;

end loop;

close frasiPagina;

-- Elimino la tabella temporanea, altrimenti rimarrebbe in memoria fino
        -- alla chiusura della sessione
execute 'DROP TABLE Indicivisitati';

END;
$$
LANGUAGE plpgsql;
```

- **Descrizione:** Come avevamo già anticipato prima, questa procedura ci va a recuperare dato in input un idpagina la sua utlima versione, cioè la versione corrente (nel caso qualcuno volesse visionarla). Tale procedura non ritorna nulla (cioè non ha valori di tipo OUT o IN/OUT), per il suo funzionamento abbiamo usato un cursore, che ci è tornato utile poiché la nostra query non ritornava dei singolo valori, ma di più, quindi abbiamo proceduto "attaccando" il cursore alla nostra query che ci ritornava testo,indice e pagina in cui ci trovavamo, scorrendo man mano il nostro cursore (con la fetch) dopo averlo aperto, appena incontro un indice già lo inserisco in una tabella d'appoggio creata tramite l'SQL DINAMICO , così che dopo non controllerà più quell'indice, se invece quell'indice non l'ho incontrato già vedo se è presente solamente 2 volte, semplicemente vuol dire che uno è l'originale mentre l'altro è la modifica accettata, quindi prendo quest'ultima, mentre nel caso in cui ne sono più di 2, vuol dire che devo fare dei controlli sulle date di valutazione, l'ultima modifica valutata è ovviamente accettata sarà quella da prendere in considerazione in questo caso. Ho un controllo che se nel caso un indice è senza modifiche lo stampo direttamente, mentre se un indice ha almeno una modifica stampo i valori appropriati. Infine chiudo il cursore ed elimino la tabella temporanea cerata in precedenza, altrimenti rimarrebbe in memoria fino alla chiusura della sessione.

```
create or replace procedure StampaUnaVersionePagina(in idpagInput int, in
dataInput date, in oraInput time) as
$$
DECLARE
countAppoggio int;
testoAppoggio varchar(100);
indiceAppoggio int;
idPaginaAppoggio int;
controllo int;
-- Questo cursore ad ogni fetch restituisce il testo, l'indice e
        -- l'idpagina di ogni frase ordinate per l'indice
frasiPagina cursor for
select f.testo,f.indice,f.idpagina
    from frase f left join modifica m on (f.idpagina = idpaginput and f.idpagina = m.idp
         left join valutazione v on (v.idmodifica = m.idmodifica and accettazione = true
    order by f.indice asc, m.datamodificaproposta asc, m.oramodificaproposta asc;
fraseCorrente frase%rowtype;

BEGIN

RAISE NOTICE 'Frasi della pagina: %', idpagInput;

-- Questa tabella serve a tenere traccia degli indici già visitati
execute 'create table IndiciVisitati (indice int)';

open frasiPagina;

loop

controllo:=0;

fetch frasiPagina into fraseCorrente;

IF NOT FOUND THEN
        EXIT;
    END IF;

select count(*) into countAppoggio
from IndiciVisitati i
where i.indice=fraseCorrente.indice;

-- Controllo se ho già visitato questo indice, in tal
           --caso salto il loop corrente
if countAppoggio>0 then
continue;
end if;
```

```
-- Conto quante frasi hanno lo stesso indice
SELECT count(*) into countAppoggio
FROM frase f
WHERE idpagInput=f.idpagina and f.indice=fraseCorrente.indice;

-- Se c'è più di una frase con lo stesso indice allora controllerò
            -- quale sarà l'ultima valutata
if countAppoggio > 1 then
select m.testo, m.indice, m.idpaginafrase into testoAppoggio,indiceAppoggio,idpaginaAppo
from frase f join modifica m on (f.idpagina = idpaginput and f.indice = fraseCorrente.in
                 join valutazione v on (v.idmodifica = m.idmodifica and v.accettazione =
where v.datavalutazione < dataInput or (v.datavalutazione = dataInput and v.oravalutazio
            order by f.indice asc, v.datavalutazione desc, m.oravalutazione desc LIMIT 1

controllo = 1;

end if;

insert into IndiciVisitati (indice) values (fraseCorrente.indice);

-- Se il controllo sarà 0 vorrà dire che non ci sono più frasi con lo
        -- stesso indice, altrimenti stampo quella appropriata
if controllo = 0 then
RAISE NOTICE 'testo: %', fraseCorrente.testo;
RAISE NOTICE 'indice: %', fraseCorrente.indice;
RAISE NOTICE 'idpagina: %', fraseCorrente.idpagina;
elsif controllo = 1 then
RAISE NOTICE 'testo: %', testoAppoggio;
RAISE NOTICE 'indice: %', indiceAppoggio;
RAISE NOTICE 'idpagina: %', idPaginaAppoggio;
end if;

end loop;

close frasiPagina;

-- Elimino la tabella temporanea, altrimenti rimarrebbe in memoria fino
        -- alla chiusura della sessione
execute 'DROP TABLE Indicivisitati';

END;
$$
LANGUAGE plpgsql;
```

- **Descrizione:** Procedura simile a quella vista prima, solo che ora in input alla procedura oltre alla pagina abbiamo anche la data e l'ora in cui vogliamo vedere la versione della pagina, cosa cambia dalla prima procedura ? Per prima cosa la query che restituisce sempre indice,testo e pagina in cui si trova la frase, ma ora andiamo a prenderci anche le frasi che non hanno subito modifiche e a loro volta valutazioni (tramite una left-join) che ci permette di portarci valori a NULL, Infine il cambiamento cruciale è quello che ora andiamo a prenderci la data maggiore tra tutte MA minore o uguale di quella in input, nel caso le date siano uguali, andremo a controllare le ore, così da mantenere sempre la coerenza.

```
CREATE OR REPLACE PROCEDURE StampaUtentiEAutori() AS
$$
DECLARE

    autori cursor for
    SELECT distinct u.username
    FROM utente u,pagina p
    WHERE u.username=p.usernameAutore;
    autoreCorrente varchar(15);

    utenti cursor for
    SELECT u.username
    FROM utente u
    WHERE u.username not in ( SELECT p.usernameautore
                              FROM pagina p);
    utenteCorrente varchar(15);

BEGIN

    open autori;

    RAISE NOTICE 'Gli utenti creatori di almeno una pagina sono : ';

    loop

        fetch autori into autoreCorrente;

        IF NOT FOUND THEN
            EXIT;
        END IF;

        RAISE NOTICE '-%',autoreCorrente;

    end loop;
```

```
    close autori;

    open utenti;

    RAISE NOTICE 'Gli utenti del sistema sono : ';

    loop

        fetch utenti into utenteCorrente;

        IF NOT FOUND THEN
            EXIT;
        END IF;

        RAISE NOTICE '-%',utenteCorrente;

    end loop;

    close utenti;

END
$$
LANGUAGE plpgsql;
```

- **Descrizione:** Procedura che distingue e stampa gli utenti generici e gli utenti che hanno almeno creato una pagina.

```
create or replace function NumeroModificheIndice
(in idPagInput pagina.idpagina%type, in indiceInput int)
returns int as
$$
DECLARE

    numeroIndici int;

BEGIN

    numeroIndici:=0;

    SELECT count(*) into numeroIndici
    FROM frase f
    WHERE idpagInput=f.idpagina and f.indice=indiceInput;
```

```
    RETURN numeroIndici-1;
    --ritorno il numero di indici - 1 poichè ovviamente questa query mi
    --conta anche l'indice corrente, quindi va sottratto

END
$$
language plpgsql;
```

- **Descrizione:** Funzione che dato in input una pagina e un indice, ci restituisce il numero delle modifiche apportate a quell'indice, restituisce -1 nel caso inserisco un indice che non esiste o 0 nel caso quell'indice non ha subito modifiche.

```
create or replace function NumeroIndici(in idPagInput pagina.idpagina%type) returns int
as $$
DECLARE

    numeroIndici int;

    countAppoggio int;

    indici cursor for
    SELECT f.indice
    FROM frase f
    WHERE f.idpagina=idpaginput;
    indiceCorrente int;

BEGIN

    numeroIndici := 0;

    -- Questa tabella serve a tenere traccia degli indici già visitati
    execute 'create table IndiciVisitati (indice int)';

    open indici;

    loop

        fetch indici into indiceCorrente;

        IF NOT FOUND THEN
            EXIT;
        END IF;

        select count(*) into countAppoggio
        from IndiciVisitati i
        where i.indice=indiceCorrente;

        -- Controllo se ho già visitato questo indice, in tal caso
        --vado al prossimo loop
        if countAppoggio>0 then
            continue;
        end if;
```

```
            numeroIndici := numeroIndici + 1;
            --faccio una sorta di count per vedere quanti indici distinti ho

            insert into IndiciVisitati (indice) values (indiceCorrente);
            --ogni volta che visito un'indice lo inserisco nella tabella

        end loop;

        close indici;

        execute 'DROP TABLE Indicivisitati';
        -- Elimino la tabella temporanea, altrimenti rimarrebbe in memoria fino
            -- alla chiusura della sessione

    RETURN numeroIndici;

    END
    $$
    language plpgsql;
```

- **Descrizione:** Funzione che dato in input un idPagina ci restituisce il numero di indici di quella pagina, non contando ovviamente le modifiche ma solo la versione originale. Nwl caso inseriamo un idPagina non esistente la funzione restituirà 0.

4.7 Dizionario dei vincoli

Ora andremo ad affrontare i vincoli intra-relazionali e inter-relazionali. Qual'è la differenza principale tra i 2 ? Che per i primi parliamo di vincoli che "rimangono" nella tabella in cui si trovano, quindi per esempio vincoli come il **check**, **unique**, **NOT NULL** e **primary key** (abbiamo utilizzato tutti questi vincoli). Mentre per gli inter-relazionali, parliamo di vincoli che "attraversano" le tabelle quindi come nel nostro caso abbiamo le **foreign key**.

4.7.1 Vincoli intra-relazionali

In questa sezione verranno elencati i vincoli intra-relazionali implementati nelle tabelle. Tra questi troveremo spesso not null e per non essere ripetitivi verranno descritti qui: all'inserimento nelle tabelle con essi presenti, tali valori dovranno per forza essere inseriti, poiché senza di loro la tupla non sarebbe consistente. Analogamente per i vincoli di primary key, che sono presenti in ogni tabella, vanno ad identificare univocamente ogni istanza di quella tabella.

Vincolo	**Classe**	**Descrizione**
checkLengthPassword	Utente	”check (length(password) > 5)”. Controlla che la lunghezza della password inserita sia maggiore di 5.
checkTitolo	Pagina	”(check)(length(titolo) > 2))”. Controlla che la lunghezza del titolo inserito sia maggiore di 2.
checkDate	Pagina	”check((dataultimamodifica > datacreazione) or (dataultimamodifica = datacreazione and oraultimamodifica >= oracreazione))”. Controlla che la data dell’ultima modifica sia maggiore di quella della creazione o nel caso siano uguali fa un controllo analogo per l’ora.
checkTesto	Frase	”check(length(testo)>0)”. Controlla che la lunghezza della frase inserita sia maggiore di 0, cioè se effettivamente si sta scrivendo qualcosa.
uniqueFrase	Frase	”unique(testo,idpagina)”. Controlla che in ogni pagina non ci siano frasi uguali.
checkFrasiDiverse	Modifica	”check(testo<>testofrase)”. Controlla che il testo della modifica proposta alla frase sia diverso dal testo originale.
checkTesto	Modifica	”check(length(testo)>0)”. Controlla che la lunghezza della frase inserita sia maggiore di 0, cioè se effettivamente si sta scrivendo qualcosa.
checkLinkPagina	Linkare	”check (idpaginalinkata<>idpaginafrase)”. Controlla che una frase non linki alla stessa pagina in cui si trova.

Vincolo	Classe	Descrizione
uniqueIdModifica	Valutazione	”unique(idModifica)”. Indica che una valutazione non può riferirsi più di una volta ad una modifica.
uniqueDate	Modifica	”unique(dataModificaproposta,oraModificaproposta,idPaginaFrase)”. Controlla che per ogni pagina le modifiche vengano proposte ad un tempo unico, per aver miglior distinzione tra loro.

4.7.2 Vincoli inter-relazionali

Nel nostro l'unico vincolo inter-relazionale che troviamo, come anticipato sopra sono le *foreign key*, che servono per mantenere il vincolo di **integrità referenziale**, cioè che la *foreign key* di una tabella si riferisca ad una tupla esistente nella tabella dove la *foreign key* è *primary key*. Di fianco a questo vincolo in ogni tabella abbiamo deciso di far ”valere” le seguenti azioni **on update cascade** e **on delete set NULL**, che rispettivamente stanno a significare che nel caso l'attributo riferito alla *foreign key* viene modificato a CASCATA modificami tutti quelli che si riferivano ad esso, mentre il secondo nel caso tale attributo venga eliminato mette a NULL tutti quelli che si riferivano ad esso.