Corso di Laurea Triennale In Informatica

Documentazione del Database per Progetto Wiki

Relatore:
Prof. Silvio Barra

Laureandi:
Giuliano Savino N86004523
Silvio Stucovitz N86004533

Anno accademico 2023/2024

Indice

1 Traccia **2**

2 Output Attesi Dal Committente **3**
2.1 **Documento Di Design Della Base Di Dati:** 3
2.1.1 Class Diagramm Della Base Di Dati. 3
2.1.2 Ristrutturazione Class Diagramm. 3
2.1.3 Dizionario delle Classi,delle Associazioni, e dei Vincoli. 3
2.1.4 Schema Logico con descrizione di Trigger e Procedure individuate. 3
2.2 **File SQL Contenenti:** 3
2.2.1 Creazione Della Struttura Della Base Di Dati. 3
2.2.2 Popolamento Del DB. 3
2.2.3 (Facoltativo, ma apprezzato) README contenente i commenti all'SQL. 3

3 Capitolo 2 **4**

4 Progettazione **4**
4.1 Analisi Dei Dati 4
4.2 Schema Concettuale 4
4.3 Ristrutturazione Schema Concettuale 4
4.4 Analisi Ridondanze 4
4.5 Rimozione Attributi MultiValore 5
4.6 Rimozione Classi Associazione 5
4.7 Rimozione Generalizzazioni 5
4.8 Accorpamento Di Entità E Associazioni 6
4.9 Identificatori Primari 6
4.10 Progettazione Logica 8
4.11 `Traduzione Delle Classi` 8
4.12 Traduzione Delle Associazioni 8
4.12.1 Traduzione Delle Associazioni Molti A Molti 8
4.12.2 Traduzione Delle Associazioni Uno A Molti 9
4.13 Dizionario Dei Dati 10
4.14 Dizionario Delle Associazioni 12

5 Capitolo 3 **17**
5.1 **Definizione Delle Tabelle** 17
5.1.1 Utente 17
5.1.2 Pagina 18
5.1.3 Frase 18
5.1.4 Tema 18
5.1.5 ModificaTesto 19
5.1.6 CronologiaTesto 19
5.1.7 Pagina_Frase 20

5.1.8 Pagina_Tema 20
5.2 **Definizione Dei Trigger** 21
5.2.1 Check_DeletePagina 21
5.2.2 Check_DeleteUtente 22
5.2.3 Check_Password 23
5.2.4 Check_Delete_Anonymous_User 23
5.2.5 Insert_Utente 24
5.2.6 Set_Date 24
5.2.7 UpdateUtenteIntoAutore 25
5.2.8 Insert_Pagina_Frasi 26
5.2.9 Update_Date_ModificaTesto 26
5.2.10 Insert_Autore 27
5.2.11 Delete_Frasi 27
5.2.12 Insert_Cronologia 28
5.2.13 Count_Modifiche 28
5.2.14 Update_Rango 30
5.2.15 Update_PagineCreate 30
5.2.16 Insert_data_accettazione 31
5.3 **Dizionario Dei Vincoli** 32
5.3.1 Vincoli Intra-Relazionali e Vincoli Inter-Relazionali 32

1. Traccia

Si sviluppi un sistema informativo, composto da una base di dati relazionale e da un applicativo Java dotato di GUI (Swing o JavaFX), per la gestione del ciclo di vita di una pagina di una wiki.

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 in 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.

2. Output Attesi Dal Committente

2.1 **Documento Di Design Della Base Di Dati:**

2.1.1 Class Diagramm Della Base Di Dati.

2.1.2 Ristrutturazione Class Diagramm.

2.1.3 Dizionario delle Classi,delle Associazioni, e dei Vincoli.

2.1.4 Schema Logico con descrizione di Trigger e Procedure individuate.

2.2 **File SQL Contenenti:**

2.2.1 Creazione Della Struttura Della Base Di Dati.

2.2.2 Popolamento Del DB.

2.2.3 (Facoltativo, ma apprezzato) README contenente i commenti all'SQL.

3. Capitolo 2

In questo capitolo ci occuperemo della progettazione del Database, fornendo la struttura e l'organizzazione necessarie per la memorizzazione, la gestione e la fruizione efficiente dei dati. Illustreremo le metodologie che abbiamo usato per creare una base di dati efficiente su cui possa contare l'applicativo che stiamo costruendo.

4. Progettazione

4.1 Analisi Dei Dati

Le entità che possono essere individuate nel problema sono elencate all'interno della Tabella 1

4.2 Schema Concettuale

Nelle figure 4.1 e 4.2 sono presenti il class diagramm concettuale della base di dati descritta nel capitolo 1 e il diagramma Entità-Relazioni

4.3 Ristrutturazione Schema Concettuale

4.4 Analisi Ridondanze

Nello schema originale è presente una ridondanza; nello specifico quando parliamo dell'attributo *PAGINECREATE* nella classe *AUTORE*.
Andiamola ad analizzare :

TAVOLA DEI VOLUMI
UTENTE : 200
PAGINA : 400
Consideriamo l'operazione 1 di inserimento di una nuova pagina effettuata 200 volte al giorno. Esaminiamo anche l'operazione 2, cioè quella di stampa dei dati dell'utente, (compreso le pagine create) che viene eseguita 5 volte al giorno.
(Valutiamo l'accesso in scrittura il doppio dell'accesso in lettura)
Quindi le operazioni che andrò a fare per l'inserimento sono:
PAGINA E 1 S (andiamo ad inserire in pagina)
CREA R 1 S (crea la tupla)
UTENTE E 1 L (Andiamo a trovarela tupla utente)
UTENTE E 1 S (Andiamo ad aggiornare il numero di pagine create)
3S X 200/giorno + 1L X 200/giorno = 1400 accessi/giorno con l'attributo ridondante.
Se andassimo a considerare invece senza l'attributo ridondante, avremo una media di 800 accessi/giorno.

Per quanto riguarda l operazione 2, possiamo dire che tramite il dato ridondante ci sarà un solo accesso in lettura, quindi è trascurabile.
Senza il dato ridondante l'operazione 2 mi costerebbe: UTENTE E 1 L
CREA R 400 L
1L+1L*400*5/giorno = 2000 accessi/giorno.
Avremo quindi 2800 accessi/giorno VS 1400 accessi/giorno + 1kbyte(il dato ridondante).
In conclusione conserveremo il dato soprattutto per una questione di semplicità di interrogazione della base di dati.

4.5 Rimozione Attributi MultiValore

L'unico attributo multivalore presente nello schema è l'attributo *Tema* presente in *Pagina*.Non potendo replicare l'attributo nella classe per un certo numero di volte,abbiamo optato per creare una nuova classe *Tema* collegandola alla classe *Pagina*

4.6 Rimozione Classi Associazione

Nello schema concettuale è presente la classe di associazione fra Autore e Pagina, questa è stata rimossa inserendo gli attributi di quest'ultima nella classe Pagina,perchè i due attributi si riferiscono principalmente alla Pagina.

4.7 Rimozione Generalizzazioni

Nel caso delle classi *Pagina* E *Utente* è presente una generalizzazione: *Autore* è figlio della Classe *Utente*.
Per questa generalizzazione abbiamo deciso di accorpare la classe figlia nella classe padre, aggiungendo un attributo Tipo ad *Utente* che indica se l'utente del sistema è anche un *Autore*.
Cosa ci ha spinto nel fare cio? Vediamo:
Analizzando il quadro generale siamo arrivati alla conclusione che, accorpare il padre nelle figlie sarebbe stata una scelta poco saggia a causa del gran numero di valori null che ne sarebbero risultati.
Consinderando anche che, poichè alcune istanze potevano non essere degli Autori, essendo la generalizzazione non totale, non sarebbero potute essere rappresentate.
La seconda opzione era quella di aggiungere una relazione fra *Utente* e *Autore* ma sarebbe diventata una relazione 1-1 che andava poi accorpata sempre in *Utente* per non avere attributi null. Quindi a parer nostro, la scelta di accorpare la classe *Autore* in *Utente* è la scelta migliore.

4.8 Accorpamento Di Entità E Associazioni

Fra Pagina E Testo è presente una relazione [1.1] che abbiamo deciso di accorpare.
La scelta di avere due classi separate fra Pagina E Testo non è efficiente e per velocizzare gli accessi,dato che le interrogazioni che farà il database richiederanno principalmente il titolo della pagina, abbiamo deciso di accorpare Testo in Pagina.

4.9 Identificatori Primari

Risulta conveniente ai fini di una migliore traduzione delle associazioni, l'introduzione di chiavi primarie per ogni classe. Tali chiavi identificheranno le istanze in modo univoco;
in particolare si è scelto di inserire una chiave surrogata per le classi :

- Frase: Abbiamo deciso di inserire un chiave surrogata alla Classe *FRASI* perchè l'uso di un solo identificatore da parte della Classe *Pagina* non andrebbe bene, perchè non si riuscirebbero a identificare frasi uguali che sono presenti nella stessa pagina.
- Cronologia Testo: L'uso di una chiave surrogata qui è giustificato dal fatto che, pur usando una chiave di un'entità forte, non potevamo identificare frasi uguali che avevano pagine di destinazioni uguali e che erano state inserite nello stesso momento.(Questo può succedere quando il trigger dal contenuto modificato, divide le frasi, e trova due frasi con il contenuto uguale e pagina destinazione uguale).
- ModificaTesto: L'utilizzo della chiave surrogata è giustificato per semplicità. Invece di utilizzare tutto l'insieme di attributi, useremo un Id per identificare la tupla.

Per le altre classi abbiamo deciso di utilizzare chiavi naturali:
Per Utente utilizziamo il NomeUtente univoco in tutto il sistema.
Per Pagina utlizziamo il titolo(ogni pagina nel sistema sarà con titolo diverso).
Per il Tema usiamo la ParolaChiave come identificativo primario.

Classe	Descrizione
Utente	Per i vari utenti che navigano nell'applicativo. Di ogni utente si conserva Il NomeUtente, l'email, la password, il numero di modifiche effettuate e il rango dell'utente.
Autore	Per gli utenti che creano pagine, quest'ultimi diventano autori. Di ogni autore si conserva il numero di pagine create.
Pagina	Per le pagine che vengono create dagli utenti. Di ogni pagina si conserva il titolo, la datacreazione, l'oracreazione e i temi collegati.
Testo	Per il testo corrente della pagina. Di ogni testo si conserva la dataultimamodifica e l'oraultimamodifica.
Frase	Per le frasi del testo. Di ogni frase si riporta il contenuto.
CronologiaTesto	Per contenere la storia dei testi (modifiche subite). Si riporta il contenuto della frase,la pagina di destinazione della frase,la data e l'ora. Ad esempio una tupla di cronologiaTesto potrebbe essere così strutturata:La frase precedente alla modifica corrente, con la relativa pagina di destinazione. (cosicchè abbiamo la possibilità di vedere i linkpassati), la data di inserimento e l'ora di inserimento. Via cosi per tutte le frasi che componevano quel testo in quella determinata ora. Cosi sapremo per certo che frasi aveva quel testo, in quella determinata ora.
ModificaTesto	Per contenere le modifiche proposte dall'utente.Si conserva Il testo richiesto,l'esito,la datacreazione,l'ora creazione la data di accettazione e l'ora di accettazione della modifica.(Non conserviamo la datadelrifiuto e l'ora del rifiuto per non aggiungere ulteriori valori a null semplicemente se la proposta non viene accettata questi due attributi sono null)

Tabella 1

4.10 Progettazione Logica

Una volta ristrutturato lo schema concettuale, si procede traducendo le varie associazioni. Iniziamo col tradurre direttamente tutte le classi.
Man mano che si andranno a tradurre le varie associazioni, andremo a modificare la struttura dei vari schemi relazionali laddove necessario.

4.11 `Traduzione Delle Classi`

Traduciamo ciascun entità individuata nello schema concettuale ristrutturato in una relazione aggiungendo i vari identificatori primari:

Utente

NomeUtente	Password	E-mail	N°Modifiche	Tipo	N°PagineCreate	Rango

Pagina

Titolo	DataCreazione	OraCreazione	DataUtimaModifica	OraUltimaModifica

Frase

IdFrase	Contenuto

ModificaTesto

IdModifica	TestoRichiesto	Esito	DataCreazione	OraCreazione	DataAcc.	OraAcc.

CronologiaTesto

IdCronologia	ContenutoFrase	PaginaDestinazione	DataInserimento	OraInserimento

Tema

ParolaChiave

4.12 Traduzione Delle Associazioni

4.12.1 Traduzione Delle Associazioni Molti A Molti

Traduciamo le associazioni *..* mediante la realizzazione di apposite tabelle ponte:

1. Associazione *PaginaFrasi* fra *Pagina* e *Frase* :

PaginaFrase

IdFrase	TitoloPagina

2. Associazione *PaginaTema* fra *Pagina* e *Tema* :

PaginaTema

TitoloPagina	Tema

4.12.2 Traduzione Delle Associazioni Uno A Molti

Per ciascuna delle associazioni binarie di tipo uno a molti, si identificano le entità deboli e quelle forti che partecipano all'associazione. Per tradurre l'associazione in relazioni, basterà includere la chiave dell'entità forte all'interno della relazione dell'entità debole. Avremo quindi:

(a) Un *Utente* può creare più *Pagine*:

Pagina

Titolo	DataCreazione	OraCreazione	DataUtimaModifica	OraUltimaModifica	Autore

(b) Una *Pagina* può avere più *Frasi* che linkano ad altre pagine :

Frase

IdFrase	Contenuto	PaginaDestinazione

(c) Una *Pagina* può avere più *PaginePassate*:

CronologiaPagina

IdCronologia	ContenutoFrase	PaginaDestinazione	DataIns	OraIns	PaginaRiferimento

(d) Un *Utente* può proporre più *Modifiche* al testo:

ModificaTesto

IdModifica	TestoRichiesto	Esito	DataCreazione	OraCreazione	DataAcc.
OraAcc.	AutoreModifica				

(e) L' *Autore* della *Pagina* può ricevere più notifiche riguardanti le modifiche apportate al testo:

ModificaTesto

IdModifica	TestoRichiesto	Esito	DataCreazione	OraCreazione	DataAcc.	OraAcc.
AutoreModifica	AutorePagina					

(f) L'*Utente* può essere proprietario di più modifiche accettate e poi inserite in CronologiaPagina (-¿ è l'autore della modifica effettuata):

CronologiaPagina

IdCronologia	ContenutoFrase	PaginaDestinazione	DataIns	OraIns	PaginaRiferimento	AutoreModifica

(g) Alla *Pagina* possono appartenere più *Modifiche Del Testo*:

ModificaTesto

IdModifica	TestoRichiesto	Esito	DataCreazione	OraCreazione	DataAcc.
OraAcc.	AutoreModifica	AutorePagina	PaginaRiferimento		

4.13 Dizionario Dei Dati

Classe	**Descrizione**	**Attributi**
Utente	Tabella per i vari utenti che useranno l'applicativo.	*NomeUtente (string) (totale)*: NomeUtente che userà l'utente per navigare e farsi riconoscere nell'applicativo. *Password (string)(totale)*:Password dell'utente che userà per il register e il login. *E-mail (string)(totale)*:E-mail dell'utente che inserisce nel login. *Tipo (boolean)(parziale)*:Attributo booleano che indica di che tipo è l'utente, se un utente del sistema non ha creato una pagina (non è un autore in quel caso) o un utente che ha creato una pagina (in quel caso si diventa immediatamente autori). *N°Modifiche (int)(parziale)*:Attributo che indica il numero di modifiche effettuate dall'utente. *PagineCreate (int)(parziale)*:Attributo che indica le pagine create da un utente(parziale perchè è un attributo attivo solo quando tipo è true).
Pagina	Tabella che conserva le pagine create dagli utenti.	*Titolo (string) (totale)*: Titolo identificativo della pagina. *DataUltimaModifica(date)(parziale)*:DataUltimaModifica della pagina. *OraUltimaModifica(time)(parziale)*:OraUltimaModifica della pagina. *DataCreazione(date)(parziale)*:Datacreazione della pagina. *OraCreazione(time)(parziale)*:Oracreazione della pagina.

Continua Nella Prossima Pagina

Continua Dalla Pagina Precedente

Classe	**Descrizione**	**Attributi**
Frase	Tabella per memorizzare le frasi contenute nella pagine.	*IdFrase (serial) (totale)*: Identificativo seriale per le frasi. *Contenuto (text)(totale)*:Contenuto della frase.
CronologiaTesto	Per memorizzare la storia delle pagine.Qui vengono messe tutte le frasi del testo corrente ovvero della modifica appena accettata.	*IdCronologia (serial) (totale)*: Id seriale identificativo. *ContenutoFrase(text)(totale)*:Contenuto della frase del testo passato. *PaginaDestinazione(text)(parziale)*:Pagina destinazione della frase del testo passato. *DataInserimento(date)(totale)*:Data in cui quel testo è stato cambiato. *OraInserimento(time)(totale)*:Ora in cui quel testo è stato cambiato.
ModificaTesto	Tabella per memorizzare le modifiche proposte dall'utente.	*IdModifica (serial) (totale)*: Id seriale identificativo. *ContenutoRichiesto(text)(totale)*:Contenuto del testo richiesto. *Esito(boolean)(parziale)*:Esito della modifica. *DataCreazione(date)(parziale)*:Datacreazione della modifica. *OraCreazione(time)(parziale)*:OraCreazione della modifica. *DataAccettazione(Date)(parziale)*:DataAccettazion della modifica. *OraAccettazione(time)(parziale)*:OraAccettazione della modifica.
Tema	Tabella per memorizzare i temi delle pagine.	*ParolaChiave (String) (totale)*: Parola Chiave identificativa del tema.

4.14 Dizionario Delle Associazioni

Associazione	**Descrizione**	**Classi Coinvolte**
Crea	Rappresenta la creazione delle pagine da parte dell'autore.	**Utente** [**1..1**]: La pagina viene creata da un solo utente. **Pagina** [**0..***]: Un utente può creare più pagine.
Collegamento	Rappresenta il possesso di frasi linkate da parte di una pagina e il collegamento delle frasi ad una pagina di destinazione.	**Pagina** [**0..1**]: Una frase può linkare a 0 o 1 pagina di destinazione. **Frase** [**0..***]: Una pagina può avere 0 o più frasi linkate.
Ha	Questa associazione viene inserita per poter tenere traccia dei testi correnti passati che ha avuto il testo attuale durante tutta la sua vita e quindi poter avere un suo storico.	**Pagina** [**1..1**]: Un testo passato fa riferimento ad un solo testo. **CronologiaTesto** [**0..***]: Una pagina può avere 0 o più pagine passate.
Propone	Ogni utente può proporre più modifiche al testo.	**Utente** [**1..1**]: Una modifica fa riferimento ad un solo utente. **ModificaTesto** [**0..***]: Un utente può proporre 0 o più modifiche.
Notificato	Rappresenta la notifica fra l'autore del testo e la modifica proposta dall'utente.	**Utente** [**1..1**]: Viene notificato un autore quando viene creata una modifica. **ModificaTesto** [**0..***]: Un utente può proporre 0 o più modifiche ad un testo.

Continua Nella Prossima Pagina

Continua Dalla Pagina Precedente

Appartenenza	Rappresenta l'appartenenza di una determinata modifica effettuata dal un utente ad una pagina.	**Pagina** [**1..1**]: Una richiesta di modifica appartiene ad una sola pagina. **ModificaTesto** [**0..***]: Una pagina può avere più richieste di modifica.
ProprietarioModifica	Questa associazione viene inserita per memorizzare chi è l'autore che ha causato il cambiamento di una pagina,in fase di visual. L'utente potrà vedere chi è stato l'autore del cambiamento del testo.	**Utente** [**1..1**]: Una modifica accettata appartiene ad un solo utente. **CronologiaTesto** [**0..***]: Un utente può avere più modifiche accettate e di conseguenza più testi originali che sono passati in testi passati.
PaginaFrase	Ogni pagina può contenere 0 o più frasi. Una frase può appartenere a 0 o più pagine	**Pagina** [**0..***]: Indica le frasi che ha la pagina. **Frase** [**0..***]: Indica le frasi che fanno parte della pagina.
Pagina/Tema	Ogni pagina può avere 0 o più temi.Un tema può appartenere a 0 o più pagine	**Pagina** [**0..***]:Un tema può appartenere a più pagine. **Tema** [**0..***]: Indica i temi che può possedere una pagina.

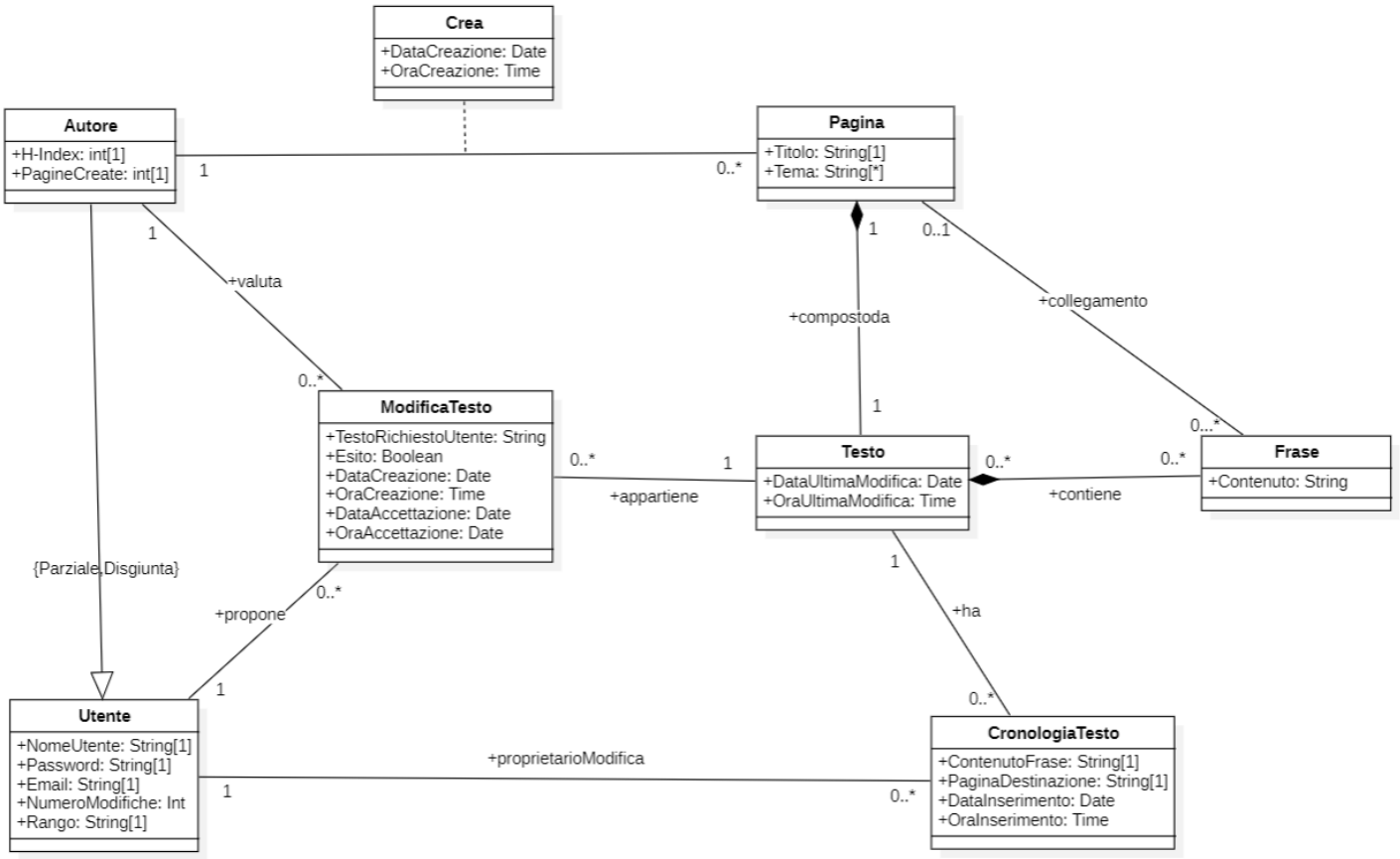

Figura 1: Figura 4.1 Schema Concettuale Del Problema

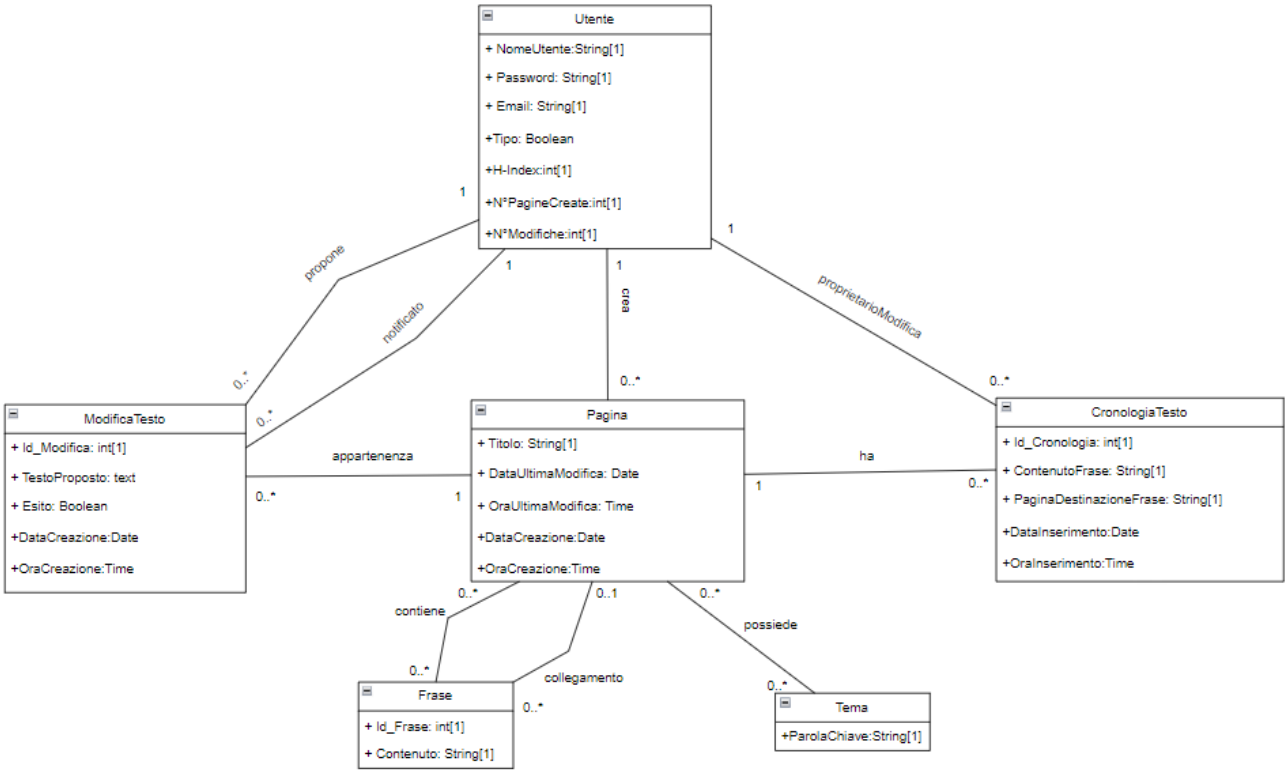

Figura 2: Figura 4.2 Schema Concettuale Ristrutturato Del Problema

Figura 3: Figura 4.3 Modello Logico Del Problema

5. Capitolo 3

In questo capitolo presentiamo l'implementazione fisica della base di dati utilizzando Postgres come sistema di gestione di database. Postgres, o PostgreSQL, è un DBMS open-source con ampie funzionalità, stabilità e una comunità di sviluppatori attiva che offre il supporto completo del linguaggio SQL e fornisce strumenti avanzati per l'ottimizzazione delle query, l'indicizzazione dei dati e la gestione delle transazioni.

Durante il capitolo, mostreremo il processo di traduzione dello schema logico a quello fisico, concentrandoci su elementi chiave come tabelle, la definizione dei trigger e delle varie procedure.
Tutto ciò, al fine di creare un database relazionale efficiente, affidabile e soprattutto attivo, e che soddisfi le esigenze di un applicativo che possa appoggiarsi su di essa, quale piattaforma di gestione dei dati.

5.1 Definizione Delle Tabelle

5.1.1 Utente

```
CREATE DOMAIN Email AS varchar(200) CHECK (VALUE ~* '^[A-Za-z0-9._%-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}$');
CREATE TYPE rango_utente as enum ('Principiante','Intermedio','Esperto');
CREATE TABLE Utente
(
    nomeutente varchar(200),
    CHECK (TRIM(nomeUtente) <> ''),
    password text not null,
    email Email not null unique,
    numeromodifiche int ,
    tipoutente boolean,
    numeropaginecreate int ,
    h_index int ,
    rango rango_utente ,
    primary key(nomeutente)
);
```

Listato 3.1

5.1.2 Pagina

```
CREATE TABLE Pagina
(
    titolo varchar(200),
    CHECK (TRIM(titolo) <> ''),
    datacreazione date,
    oracreazione time ,
    dataultimamodifica date,
    oraUltimaModifica time ,
    autore varchar(200),
    primary key(titolo),
    foreign key(autore) references Utente(nomeutente) on update cascade
);
```

Listato 3.2

5.1.3 Frase

```
CREATE TABLE Frase
(
    idFrase serial,
    contenuto text not null,
    PaginaDestinazione varchar(200) ,
    primary key(idFrase),
    foreign key (PaginaDestinazione) references Pagina(Titolo) on delete set null on update cascade
);
```

Listato 3.3

5.1.4 Tema

```
CREATE TABLE Tema
(
    ParolaChiave text,
    primary key(ParolaChiave)
);
```

Listato 3.4

5.1.5 ModificaTesto

```
CREATE TABLE ModificaTesto
(
    idModifica serial,
    testoRichiesto text not null,
    esito boolean,
    dataCreazione date ,
    oraCreazione time ,
    dataAccettazione time,
    oraAccettazione date,
    AutoreRichiedente varchar(200),
    AutorePagina varchar(200) ,
    PaginaRiferimento varchar(200),
    primary key(idModifica),
    foreign key(AutoreRichiedente) references Utente(nomeutente) ON UPDATE CASCADE,
    foreign key(AutorePagina) references Utente(nomeutente) ON UPDATE CASCADE,
    foreign key(PaginaRiferimento) references Pagina(titolo) ON DELETE CASCADE ON UPDATE CASCADE,
);
```

Listato 3.5

5.1.6 CronologiaTesto

```
CREATE TABLE CronologiaTesto
(
    idCronologia serial,
    contenutoFrase text not null,
    paginaDestinazione varchar(200),
    oraInserimento time,
    dataInserimento date,
    ProprietarioModifica varchar(200),
    PaginaRiferimento varchar(200),
    primary key(idCronologia),
    foreign key(ProprietarioModifica) references Utente(nomeutente)on update cascade,
    foreign key(PaginaRiferimento) references Pagina(titolo) on delete cascade on update cascade,
);
```

Listato 3.6

5.1.7 Pagina_Frase

```
CREATE TABLE pagina_frase
(
    TitoloPagina varchar(200),
    idFrase serial,
    primary key(TitoloPagina, idFrase),
    foreign key(TitoloPagina) references Pagina(titolo) on delete cascade on update cascade,
    foreign key(idFrase) references Frase(idFrase) on delete cascade
);
```

Listato 3.7

5.1.8 Pagina_Tema

```
CREATE TABLE pagina_tema
(
    TitoloPagina varchar(200),
    ParolaChiave text,
    primary key(TitoloPagina,ParolaChiave),
    foreign key(TitoloPagina) references Pagina(titolo) on delete cascade on update cascade,
    foreign key(ParolaChiave) references Tema(ParolaChiave) on delete cascade on update cascade
);
```

Listato 3.8

5.2 Definizione Dei Trigger

5.2.1 Check_DeletePagina

Stando al vincolo *Check_DeletePagina*, quando un utente decide di cancellare una pagina, deve essere aggiornato il numero di pagine create dall'utente e diminuito di una unità.
Deve essere aggiornata la tabella frasi e settare a null i link che avevano come pagina di destinazione la pagina cancellata.
Ciò implica di eliminare da *Pagina_Frase* tutti i collegamenti, lefrasi, che aveva la pagina,ed eliminare da modifica e cronologia tutti i riferimenti alla pagina cancellata (questo possibile grazie ai vincoli di fk).
Quindi quando l'utente cancella una pagina, non ci deve essere più riferimento a quella pagina e gli utenti non potranno cliccare sui link relativi che portavano a quella pagina.

```
CREATE OR REPLACE FUNCTION check_deletepagina() RETURNS TRIGGER AS $$
DECLARE
     numeropagine int;
BEGIN
        SELECT U.numeropaginecreate into numeropagine
        FROM UTENTE U
        WHERE U.nomeutente = OLD.autore;

        if(numeropagine > 0) then
        UPDATE UTENTE
        set numeropaginecreate = numeropaginecreate - 1
        where nomeutente = OLD.autore;
        END IF;
        RETURN NEW;

END;
$$ language plpgsql;
CREATE OR REPLACE TRIGGER check_deletepagina
AFTER DELETE ON PAGINA
FOR EACH ROW
WHEN (OLD.titolo is not null)
EXECUTE function check_deletepagina();
```

5.2.2 Check_DeleteUtente

Abbiamo deciso che nel momento in cui si cancella una tupla da utente, non vengono cancellate tutte le altre tuple in pagina o nelle altre tabelle, ma viene settato l'utente *ANONIMO*.
L'utente anonimo è un utente settato all'inizio del database a cui vengono referenziate le tuple a cui viene cancellato l'autore.(per questo motivo non ci sono vincoli di cancellazione quando si cancella una tupla)

```
CREATE OR REPLACE FUNCTION check_deleteutente() RETURNS TRIGGER AS
$$
BEGIN
    UPDATE PAGINA
    SET autore = 'Anonimo'
    where OLD.nomeutente = autore;

    UPDATE MODIFICATESTO
    set autorerichiedente = 'Anonimo'
    where OLD.nomeutente = autorerichiedente;


    UPDATE MODIFICATESTO
    set autorepagina = 'Anonimo'
    where OLD.nomeutente = autorepagina;


    UPDATE Cronologiatesto
    set proprietariomodifica = 'Anonimo'
    where OLD.nomeutente = proprietariomodifica;
    RETURN OLD;
END;
$$ language plpgsql;
CREATE OR REPLACE TRIGGER check_deleteutente
BEFORE DELETE ON Utente
FOR EACH ROW
WHEN (OLD.nomeutente is not null)
EXECUTE FUNCTION check_deleteutente();
```

5.2.3 Check_Password

Prima di inserire una tupla in Utente bisogna controllare se la password inserita dall'utente non sia uguale al suo nomeUtente e abbia una lunghezza di minimo 8 caratteri.

```
CREATE OR REPLACE FUNCTION check_password() RETURNS TRIGGER AS $$
BEGIN
    IF NEW.password <> NEW.nomeutente THEN
        IF LENGTH(NEW.password) >= 8 THEN
            RETURN NEW;
        ELSE
            RAISE EXCEPTION 'La password deve essere lunga almeno 8 caratteri';
        END IF;
    ELSE
        RAISE EXCEPTION 'La password non può essere uguale al nome utente';
    END IF;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER check_password
BEFORE INSERT ON utente
FOR EACH ROW
WHEN (NEW.password IS NOT NULL)
EXECUTE FUNCTION check_password();
```

5.2.4 Check_Delete_Anonymous_User

Quando si cerca di cancellare l'utente Anonimo, viene impedito da questo trigger lanciando un eccezione.(per integrità della base di dati)

```
CREATE OR REPLACE FUNCTION check_delete_anonymous_user() RETURNS TRIGGER AS $$
BEGIN
    if(OLD.nomeutente = 'Anonimo') THEN
        RAISE EXCEPTION 'Impossibile cancellare l utente anonimo';
    ELSE
    RETURN OLD;
    END IF;
END;
$$ language plpgsql;

CREATE TRIGGER check_delete_anonymous_user_trigger
BEFORE DELETE ON utente
FOR EACH ROW
EXECUTE FUNCTION check_delete_anonymous_user();
```

5.2.5 Insert_Utente

s Quando viene inserito un utente per la prima volta, gli viene attributo un ruolo 'False' che indica che l'utente non è un autore nel database e vengono settati gli altri attributi come l'h_index = 0,il numero di modifiche = 0,il rango di principiante,e il numero di pagine create = 0, tutti simboli che indicano che l'autore è appena entrato nella base di dati, o si è registrato da poco.

```
CREATE OR REPLACE FUNCTION insertutente_update() RETURNS TRIGGER AS
$$
    BEGIN
        UPDATE Utente
        SET tipoutente = false,
        h_index = 0,
        numeromodifiche=0,
        rango = 'Principiante',
        numeropaginecreate=0
        WHERE nomeutente = NEW.nomeutente;
        RETURN NEW;
    END;
$$ language plpgsql;

CREATE OR REPLACE TRIGGER insertUtente
BEFORE INSERT ON Utente
FOR EACH ROW
WHEN (NEW.NomeUtente is not NULL)
EXECUTE FUNCTION insertUtente_update();
```

5.2.6 Set_Date

Quando viene inserita una pagina, questo trigger permette di inserire insieme ad essa la data di creazione, l'ora di creazione la dataultimamodifica e l'oraultimamodifica della pagina.

```
CREATE OR REPLACE FUNCTION setdate() RETURNS TRIGGER AS $$
BEGIN
    update pagina
    set
    dataultimamodifica = CURRENT_DATE,
    oraUltimamodifica = CURRENT_TIME,
    datacreazione = CURRENT_DATE,
    oraCreazione = CURRENT_TIME
    where titolo = NEW.titolo;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE OR REPLACE TRIGGER InsertPagina
AFTER INSERT ON Pagina
FOR EACH ROW
WHEN (NEW.Titolo IS NOT NULL)
EXECUTE FUNCTION setdate();
```

5.2.7 UpdateUtenteIntoAutore

Quando viene creata una pagina, bisogna impostare a true il tipo dell'utente; il che significa che è diventato un autore.
Dopo che viene inserita una pagina viene fatto un update della tabella utente per settare il tipo a true.

```
CREATE OR REPLACE FUNCTION UpdateUtenteIntoAutore() RETURNS TRIGGER AS $$
BEGIN
    UPDATE utente
    set tipoutente = true
    where nomeutente = NEW.autore;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE OR REPLACE TRIGGER UpdateUtenteIntoAutore
    AFTER INSERT ON PAGINA
    FOR EACH ROW
    WHEN (NEW.TITOLO IS NOT NULL)
EXECUTE FUNCTION UpdateUtenteIntoAutore();
```

5.2.8 Insert_Pagina_Frasi

Il trigger *Insert_Pagina_Frasi*, serve a garantire che nella tabella *PAGINA_FRASE* ci siano le tuple coerenti (delle frasi di quel testo).
Funzionamento trigger:
Dall'Applicativo andiamo a generare le frasi che l'utente crea e le inseriamo una alla volta nella tabella frasi. Fatto ciò il trigger viene attivato,ed esegue la funzione.
Il trigger fa un insert nella tabella *PAGINA_FRASE*, dove seleziona la nuova frase da inserire, e la inserisce nella pagina che ha la data di modifica più recente(quella appena modificata). Quindi, la query nel trigger mi prende il titolo della pagina inserita dall'utente e mi inserisce nella tabella (Pagina-Frase) la nuova frase con la pagina nuova, avendo la coppia giusta da inserire. Questo viene fatto per ogni frase del testo.

```
create function insert_pagina_frasi() returns trigger as
$$
BEGIN
    INSERT INTO pagina_frase (titolopagina, idfrase)
    SELECT p.titolo,NEW.idfrase
    FROM pagina p
    ORDER BY p.dataultimamodifica DESC, p.oraultimamodifica DESC
    LIMIT 1;


    RETURN NEW;
END;
$$ language plpgsql;

CREATE OR REPLACE TRIGGER insert_in_pagina_frasi
AFTER INSERT ON FRASE
FOR EACH ROW
WHEN (NEW.idfrase IS NOT NULL)
EXECUTE FUNCTION insert_pagina_frasi();
```

5.2.9 Update_Date_ModificaTesto

Quando viene inserita una proposta dall'utente questo trigger inserisce la dataCreazione e l'ora creazione della modifica.

```
create or replace function updatedatemodifiche_testo() returns trigger as
$$
BEGIN
    UPDATE ModificaTesto
    SET
        datacreazione = CURRENT_DATE,
        oracreazione = CURRENT_TIME
    WHERE paginariferimento = NEW.paginariferimento;
    RETURN NEW;
END;
$$ language plpgsql;

CREATE OR REPLACE TRIGGER InsertModifiche_testo
AFTER INSERT ON modificatesto
FOR EACH ROW
WHEN (NEW.Paginariferimento IS NOT NULL)
EXECUTE FUNCTION updateDateModifiche_testo();
```

5.2.10 Insert_Autore

Quando viene inserita una tupla in modificatesto, questo trigger inserisce anche l'autore della pagina originale a cui è stata indirizzata la modifica.
Il trigger quindi fa un update alla tabella modificatesto e setta l'autore della pagina grazie alla query annidata fra le parentesi. Questa query ci permette di recuperare l'autore della pagina originale e settarlo alla tupla modificatesto.

```
create function inserisci_autore() returns trigger
    language plpgsql
as
$$
    BEGIN
        UPDATE modificatesto
        set autorepagina = (SELECT p.autore
                                   from pagina p
                                   where p.titolo = new.paginariferimento)
        where NEW.idmodifica = idmodifica;
        RETURN NEW;
    END;
$$;


CREATE OR REPLACE TRIGGER inserisci_autore
AFTER INSERT ON MODIFICATESTO
FOR EACH ROW
WHEN (NEW.idmodifica is not NULL)
EXECUTE FUNCTION inserisci_autore();
```

5.2.11 Delete_Frasi

Questo trigger consente all'utente, nel momento in cui clicca il tasto "Accetta Modifiche", di cambiare il testo corrente con la nuova modifica proposta dall'autore.
Si tratta semplicemente di distruggere le frasi che sono collegate alla pagina corrente. Questo trigger quindi viene attivato quando l'esito della modifica è true e cancella dalla tabella *PAGINA_FRASE* le frasi collegate alla pagina modificata, in modo che quelle non siano più collegate alla pagina in questione.

```
create or replace function delete_frasi() returns trigger
    language plpgsql
as
$$
    BEGIN
        DELETE
        FROM pagina_frase pf
        where pf.titolopagina = NEW.paginariferimento;
        RETURN NEW;
    END;
$$;


CREATE OR REPLACE TRIGGER delete_frasi
AFTER UPDATE OF esito ON modificatesto
FOR EACH ROW
WHEN (NEW.esito is true)
EXECUTE FUNCTION delete_frasi();
```

5.2.12 Insert_Cronologia

Questo trigger consente all'utente di modificare il testo e lavorare sulla tabella *CRONOLOGIA_TESTO*.
Ora partiamo da un presupposto, vogliamo giustificare prima di tutto perchè è stato scelto uno schema cosi per la tabella Cronologia .
La scelta dello schema della tabella cronologia non è casuale ovviamente, ma deriva dal fatto che, da traccia, viene chiesto di far visualizzare la storia della pagina agli utenti.
Facciamo visualizzare la storia all'utente della pagina, visualizzando anche i link passati; questa tabella mi conserva il contenuto delle frasi, con le relative pagina di destinazione.
Queste due informazioni, insieme alla data di inserimento e all'ora di inserimento mi permettono di recuperare il testo di quella specifica pagina(in quella determinata ora) e anche le eventuali frasi linkate.

Ora che abbiamo chiarito il perchè viene utilizzata questa tabella, andiamo a discutere del **perchè** abbiamo creato questo trigger.
Il trigger viene fatto in risposta all'accettazione di un utente ad una modifica(o da una modifica di una pagina da parte dell'autore stesso), quando un utente quindi clicca il tasto salva, automaticamente questo trigger risale alle frasi della pagina a cui è riferita la modifica, e le inserisce nella tabella.

Quello che fa il trigger è inserire:
1)La pagina di riferimento(cioè la pagina a cui stiamo cambiando il testo)
2)La data e l'ora(per facilità di confronti queste vengono troncate)
3)Il contenuto della frase
4)La pagina di destinazione della frase
5)L'autore che ha fatto la modifica.
Con la join possiamo risalire alle frasi del testo corrente e quindi metterle nella tabella, questo ovviamente viene fatto prima del trigger(vedi 5.2.11), altrimenti non troveremo le frasi del testo, perchè cancellate.
Successivamente quello che fa il trigger è fare un update dell'ultima modifica della pagina.

```
create OR REPLACE function insertcronologia_testi() returns trigger
    language plpgsql
as
$$
DECLARE
    sql_istruzione TEXT;
    sql_istruzione1 TEXT;
BEGIN
    INSERT INTO cronologiatesto
        (paginariferimento,datainserimento,orainserimento,contenutofrase, paginadestinazione,proprietariomodifica)
        SELECT NEW.paginariferimento,CURRENT_DATE,DATE_TRUNC('second', CURRENT_TIMESTAMP),
        f.contenuto, f.paginadestinazione, NEW.autorerichiedente
        FROM pagina p, pagina_frase pf, frase f
        WHERE p.titolo = pf.titolopagina AND pf.idfrase = f.idfrase AND p.titolo = NEW.paginariferimento;

    sql_istruzione1 := 'UPDATE Pagina SET dataultimamodifica = CURRENT_DATE,
                        oraultimamodifica = CURRENT_TIME
                        WHERE titolo = $1.paginariferimento';
    EXECUTE sql_istruzione1 USING NEW;
    RETURN NEW;
END;
$$;
CREATE OR REPLACE TRIGGER insertCronologia_Testi
BEFORE UPDATE OF esito ON modificatesto
FOR EACH ROW
WHEN (NEW.esito IS TRUE)
EXECUTE FUNCTION insertCronologia_Testi();
```

5.2.13 Count_Modifiche

Questo trigger permette incrementare il numero di testi modificati da parte dell'autore della modifica, se questa è stata accettata.

```
create or replace function update_numeromodifiche() returns trigger
    language plpgsql
as
$$
DECLARE
BEGIN
    UPDATE utente
    set numeromodifiche=numeromodifiche+1
    where nomeutente = NEW.autorerichiedente;
    RETURN NEW;
END;
$$;

CREATE OR REPLACE trigger count_modifiche
AFTER UPDATE OF ESITO ON modificatesto
FOR EACH ROW
WHEN (NEW.esito is true)
EXECUTE function  update_NumeroModifiche();
```

5.2.14 Update_Rango

Questo vincolo mi permette di avere una maggiore completezza nella fase di visualizzazione dell'utente.
Quando l'autore di una pagina vedrà una richiesta dall'utente, potrà cliccare sul bottone visualizza utente e vedere tutte le sue statistiche comprese il rango, che è un indice di bravura dell'utente,, che viene aggiornato ogni qualvolta l'utente manda una notifica che andrà a buon fine(verrà accettata).

```
create function update_rango() returns trigger as
$$
BEGIN
    IF (NEW.numeromodifiche >= 5 AND NEW.numeromodifiche < 10) THEN
        UPDATE utente
        SET rango = 'Intermedio'
        WHERE nomeutente = NEW.nomeutente;
    ELSIF (NEW.numeromodifiche >= 10) THEN
        UPDATE utente
        SET rango = 'Esperto'
        WHERE nomeutente = NEW.nomeutente;
    END IF;
    RETURN NEW;
END;
$$ language plpgsql;
CREATE OR REPLACE TRIGGER update_rango
AFTER UPDATE OF numeromodifiche ON utente
FOR EACH ROW
WHEN (NEW.nomeutente is not null)
EXECUTE FUNCTION update_rango();
```

5.2.15 Update_PagineCreate

Questo vincolo ci permette di tener traccia delle pagine create dall'utente e in fase di visualizzazione delle statistiche, di far visualizzare all'utente quante pagine ha creato.Il trigger aumenta di uno ogni qualvolta si inserisce una tupla in pagina.

```
create or replace function update_paginecreate() returns trigger
    language plpgsql
as
$$
BEGIN
    UPDATE UTENTE
    SET numeropagineCreate = numeropagineCreate + 1
    WHERE NEW.autore = nomeutente;
    RETURN NEW;
END;
$$;


CREATE OR REPLACE TRIGGER update_PagineCreate
AFTER INSERT ON PAGINA
FOR EACH ROW
WHEN (NEW.titolo is not null)
EXECUTE FUNCTION update_PagineCreate();
```

5.2.16 Insert_data_accettazione

Questo vincolo ci permette di inserire la data e l'ora di accettazione quando l'utente accetta la modifica.
Il trigger si aziona quando l'esito della modifica diventa true ed è in grado di settare la data e l'ora di accettazione della modifica,non inseriamo la data e l'ora del rifiuto come detto prima perchè aggiungeremo campi a null aggiuntivi.
Semplicemente quando la modifica non viene accettata, questi due campi non vengono riempiti.

```
create OR REPLACE  function insert_data_accettazione() returns trigger
    language plpgsql
as
$$
DECLARE
    sql_istruzione TEXT;
BEGIN
    UPDATE modificatesto
    SET dataaccettazione = CURRENT_DATE,
        oraaccettazione = DATE_TRUNC('second', CURRENT_TIMESTAMP)
    WHERE idmodifica = NEW.idmodifica;

    RETURN NEW;
END;
$$;


CREATE OR REPLACE TRIGGER insert_data_accettazione
AFTER UPDATE OF ESITO ON modificatesto
FOR EACH ROW
WHEN (NEW.esito is true)
EXECUTE FUNCTION insert_data_acettazione();
```

5.3 **Dizionario Dei Vincoli**

5.3.1 Vincoli Intra-Relazionali e Vincoli Inter-Relazionali

Vincolo	**Tipo**	**Descrizione**	**Vedi**
utente_nomeutente_check	IntraRelazionale	Vincolo di check, controlla che il nome utente non è vuoto.	vedi 5.1.1
utente_pkey	IntraRelazionale	Vincolo di chiave primaria.	vedi 5.1.1
utente_email_key	IntraRelazionale	Vincolo di check che controlla se l'email è in un formato ammissibile.	vedi 5.1.1
pagina_titolo_check	IntraRelazionale	Vincolo di check che controlla se il titolo non è vuoto.	vedi 5.1.2
pagina_pkey	IntraRelazionale	Vincolo di chiave primaria.	vedi 5.1.2
pagina_autore_fkey	IntraRelazionale	Vincolo di foreign key.Se viene aggiornata una tupla in utente, vengono aggiornate anche quelle referenziate in Pagina	vedi 5.1.2
fk_frase	IntraRelazionale	Vincolo di foreign key, se viene cancellata una tupla da pagina viene settata a null la pagina di destinazione.Se viene aggiornata una tupla in pagina, vengono aggiornate tutte le tuple referenziate in Frase	vedi 5.1.3
frase_pkey	IntraRelazionale	Vincolo di chiave primaria.	vedi 5.1.3
Tema_pkey	IntraRelazionale	Vincolo di chiave primaria	vedi 5.1.4
fk_modificatesto_pagina	IntraRelazionale	Vincolo di foreign key.Se viene cancellata o aggiornata una tupla in pagina viene cancellata anche in modificatesto	vedi 5.1.5

modificatesto_autorepagina_fkey	IntraRelazionale	Vincolo di foreign key.Se viene aggiornata una tupla in utente, vengono aggiornate tutte quelle referenziate in ModificaTesto	vedi 5.1.5
modificatesto_autorerichiedente_fkey	IntraRelazionale	Vincolo di foreign key.Se viene aggiornata una tupla in utente, vengono aggiornate tutte quelle referenziate in ModificaTesto.	vedi 5.1.5
modificatesto_pkey	IntraRelazionale	Vincolo di chiave primaria.	vedi 5.1.5
cronologiatesto_pkey	IntraRelazionale	Vincolo di chiave primaria.	vedi 5.1.6
cronologiatesto_proprietariomodifica_fk	IntraRelazionale	Vincolo di foreign key.	vedi 5.1.6
fk_cronologiatesto_pagina	IntraRelazionale	Vincolo di foreign key.Se viene eliminata una tupla da pagina,vengono eliminate anche le tuple referenziate in CronologiaTesto	vedi 5.1.6
fk_pagina_frase_idfrase	IntraRelazionale	Vincolo di foreign key.Se viene eliminata una tupla in Frase vengono eliminate anche le tuple referenziate in Pagina_Frase	vedi 5.1.7
fk_pagina_frase_titolopagina	IntraRelazionale	Vincolo di foreign key.Se viene eliminata una tupla in Pagina vengono eliminate anche le tuple referenziate in Pagina_Frase	vedi 5.1.7
pagina_frase_pkey	IntraRelazionale	Vincolo di chiave primaria	vedi 5.1.7
pagina_tema_pkey	IntraRelazionale	Vincolo di chiave primaria	vedi 5.1.8
pagina_tema_parolachiave_fkey	IntraRelazionale	Vincolo di foreign key, se viene eliminata una tupla in temi, vengono eliminate anche in pagina_tema le tuple referenziate	vedi 5.1.8
pagina_tema_titolopagina_fkey	IntraRelazionale	Vincolo di foreign key, se viene eliminata una tupla in Pagina, vengono eliminate anche in pagina_tema le tuple referenziate	vedi 5.1.8

Vincolo	Tipo	Descrizione	Vedi
check_deletepagina	InterRelazionale	Trigger che si attiva quando viene cancellata una pagina,diminuisce di uno il numero di pagine create.	vedi 5.2.1
check_deleteutente	InterRelazionale	Trigger che si attiva quando viene cancellato un utente,setta la stringa Anonimo a tutte le tuple in cui era referenziato l'utente.	vedi 5.2.2
check_password	InterRelazionale	Trigger che si attiva quando viene inserito un utente,controlla se la password è maggiore di 8 caratteri e che non è uguale al nomeutente.	vedi 5.2.3
delete_frasi	InterRelazionale	Trigger che si attiva dopo che viene settato l'esito a true della modifica dell'utente.è in grado di distruggere le frasi del testo corrente a cui è referenziata la modifica	vedi 5.2.11
inserisci_autore	InterRelazionale	Trigger che si attiva quando viene inserita la pagina,setta a true il campo Tipo per aggiornare la tupla di utente e settarla in autore	vedi 5.2.10
inserisci_data_accettazione	InterRelazionale	Trigger che si attiva quando viene settato l'esito a true della modifica, setta la data e l'ora accettazione	vedi 5.2.16
inserisci_pagina_frasi	InterRelazionale	Trigger che permette di inserire nella tabella la coppia esatta di Pagina-Frase che inserisce l'utente nell'applicativo	vedi 5.2.8
inserisci_cronologia_testi	InterRelazionale	Trigger che si attiva quando viene accettata una modifica e permette di inserire le frasi del testo corrente nella tabella *CRONOLOGIATESTO* e di aggiornare la data e ora della modifica	vedi 5.2.12

Update_Utente_Into_Autore	InterRelazionale	Trigger che aggiona il campo tipo in utente lo setta a true.	vedi 5.2.7
set_date	InterRelazionale	Trigger che si attiva quando viene inserita una tupla in pagina,questo trigger permette di inserire la datacreazione,l'ora creazione la data ultimamodifica e l'oraultimamodifica.	vedi 5.2.6
Update_Numero_Modifiche	InterRelazionale	Trigger che si attiva quando viene accettata una modifica,aumenta il campo numeromodifiche dell'utente referenziato.	vedi 5.2.13
Update_Pagine_Create	InterRelazionale	Trigger che si attiva quando viene inserita una pagina,aumenta il numero di pagine create dall'utente.	vedi 5.2.15
Update_Rango	InterRelazionale	Trigger che si attiva quando l'utente supera un certo numero di modifiche, aumenta il rango dell'utente in base alle sue modifiche.	vedi 5.2.14
Update_Date_ModificaTesto	InterRelazionale	Trigger che si attiva quando viene inserita una modifica,setta la data e l'ora di creazione.	vedi 5.2.9
insert_utente_update	InterRelazionale	Trigger che si attiva quandoviene inserito l'utente, setta i suoi campi come un utente che ha appena fatto l'accesso.	vedi 5.2.16
check_delete_anonymous_user	InterRelazionale	Trigger che impedisce la cancellazione della tupla *ANONIMO*.	vedi 5.2.4