DIETI. UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II

DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELLE TECNOLOGIE DELL'INFORMAZIONE

Sistema di gestione di una pagina wiki

Progetto basi && object

anomaly0197

November 2023

Indice

Indice figure

1 Progettazione concettuale

1.1 Analisi dei requisiti

Lo scopo di questa fase è verificare le informazioni fondamentali e le istruzioni di base per definire la struttura e le funzionalità del database di un sistema per la gestione di una pagina WIKI. Durante questa analisi dei requisiti verranno individuate le diverse entità e le diverse relazioni che le legano.
L'analisi sarà documentata per blocchi di requisiti, si riportano di seguito i primi requisiti individuati nel documento fornito come input al progetto:

Si sviluppi un sistema informativo per la gestione del ciclo di vita di una pagina wiki.

R1: Ogni pagina wiki ha un titolo e un testo.

R2: Ogni pagina è creata da un determinato autore.

R3: Il testo è composto di una sequenza di frasi.

R4: Il sistema mantiene traccia anche del giorno e ora nel quale la pagina è stata creata.

Considerando la richiesta di sviluppare un sistema per la gestione di pagine WIKI, viene naturalmente da pensare a distinguere ogni pagina da un'altra. Quindi per rappresentare genericamente ogni pagina del sistema, verrà utilizzata un'entità **PAGINA** caratterizzata dall'attributo chiave *Titolo* in modo tale da distinguere univocamente ogni pagina. Quest'entità avrà anche un attributo composto *DataOra* formato dagli attributi *Data* e *Ora* in modo da mantenere nel sistema l'informazione sulla data e orario di creazione della pagina.
Un ulteriore requisito è quello di mantenere nel sistema il testo di ogni pagina definito come una lista ordinata di frasi. Per questo motivo è preferibile creare direttamente un'entità **FRASE** per prendere singolarmente ogni frase e renderla univoca. In questo modo si andrebbe anche ad allegerire la base di dati, visto che non ci sarà bisogno di creare un ulteriore tabella per il testo e operare direttamente sulle singole tuple della tabella **FRASE**. Di conseguenza ci sarà una relazione tra **PAGINA** e **FRASE** di tipo 1 a N. Tra gli attributi principali abbiamo sicuramente *parola* che è un attributo multivalore in quanto una frase può avere diverse parole, e poi anche in questo caso l'attributo composto *DataOra* formato da *Data* e *Ora* di creazione della frase. Inoltre è presente un attributo *OrdineFrase* che definisce la posizione della frase all'interno della pagina.

Continuando l'analisi con un nuovo blocco di requisiti identificati nel documento iniziale:

R5: La pagina può contenere anche dei collegamenti.

R6: Ogni collegamento è caratterizzato da una frase da cui scaturisce il collegamento e da un'altra pagina destinazione del collegamento.

R7: 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.

R8: 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.

Per gestire eventuali collegamenti tra pagine è stata aggiunta un'ulteriore relazione tra pagina e frase, anche in questo caso è di tipo 1 a N. Di conseguenza una tupla di frase avrà due chiavi esterne di pagina, una che si riferisce alla pagina di appartenenza e una che si riferisce alla pagina con cui può avere un collegamento.
Il sistema deve prevedere 2 tipologie di utenti registrati nel sistema: gli utenti, chiamati autori, che hanno la possibilità di creare pagine o modificare pagine di altri utenti e gli utenti che possono leggere le pagine pubblicate nel sistema.
E' stata quindi utilizzata un'entità superclasse **UTENTE**, ed una *specializzazione* con la sottoclasse **Autore** di tipo *Totale,disgiunta*.
Ogni **UTENTE** sarà identificato dal *Codice Fiscale* che sarà inevitabilmente univoco per ogni utente e per essere valido dovrà avere 9 caratteri alfa-numerici, altrimenti non verrà accettato dal sistema. Inoltre l'entità avrà come attributi i principali campi per la registrazione di un utente (*nome,cognome,mail,password*).
Gli utenti generici del sistema (cioè coloro che non hanno eseguito l'accesso o non hanno un account) non hanno bisogno di essere registrati nella base di dati, ma potranno comunque accedere in sola lettura andando a cercare e visualizzare le pagine della WIKI.

Continuando l'analisi con l'ultimo blocco di requisiti identificati nel documento iniziale:

R9: La modifica proposta verrà notificata all'autore del testo originale la prossima volta che utilizzerà il sistema.

R10: 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.

R11: Il sistema mantiene memoria delle modifiche proposte e anche delle decisioni dell'autore (accettazione o rifiuto).

Un altro requisito fondamentale è gestire le modifiche da parte di altri utenti o magari da parte dell'utente stesso della pagina, mantenendo i record all'interno del database. Per fare ciò è necessario introdurre un attributo *type* che andremo a chiamare *stato* e indicherà lo stato di quella frase nel sistema. I possibili valori che potrà assumere quest'attributo sono: *accettato, sostituito, rifiutato, in_attesa*.

***Accettato*:** indica che la frase è ammessa nell'ultima versione della pagina.

***Rifiutato*:** indica la tupla inerente a una modifica che l'autore di quella pagina ha rifiutato da parte di un altro utente del sistema.

***Sostituito*:** indica la tupla che è stata sostituita da una nuova tupla.

***in_attesa*:** indica la tupla di una modifica proposta da un utente e che deve essere accettata o rifiutata dall'autore di quella pagina.

E' presente anche un attributo *versione* che identifica la versione di quella frase, che viene incrementata ogni volta che viene inserita una nuova modifica alla frase. Per esempio, quando viene inserita per la prima volta una frase la versione assume il valore di default 1, se viene proposta una modifica a quella frase, la nuova frase inserita avrà versione 2 e via dicendo. Inoltre è presente anche una relazione ricorsiva in modo da

garantire una coerenza delle modifiche ai testi delle pagine. Questa relazione è del tipo 1 a N. Per esempio: a una frase con stato "accettato" e quindi visibile nella pagina, vengono proposte 2 modifiche da parte di altri utenti. Quindi vengono inserite 2 nuove tuple nel database ed entrambe faranno riferimento alla stessa tupla. Mentre ogni frase inserita può avere 0 o una sola frase a cui fare riferimento.
E' presente inoltre un attributo composto *DataOraApprovata* per mantenere l'ora esatta in cui una modifica viene accettata nel sistema.

1.2 Schema concettuale

1.2.1 Modellazione ER

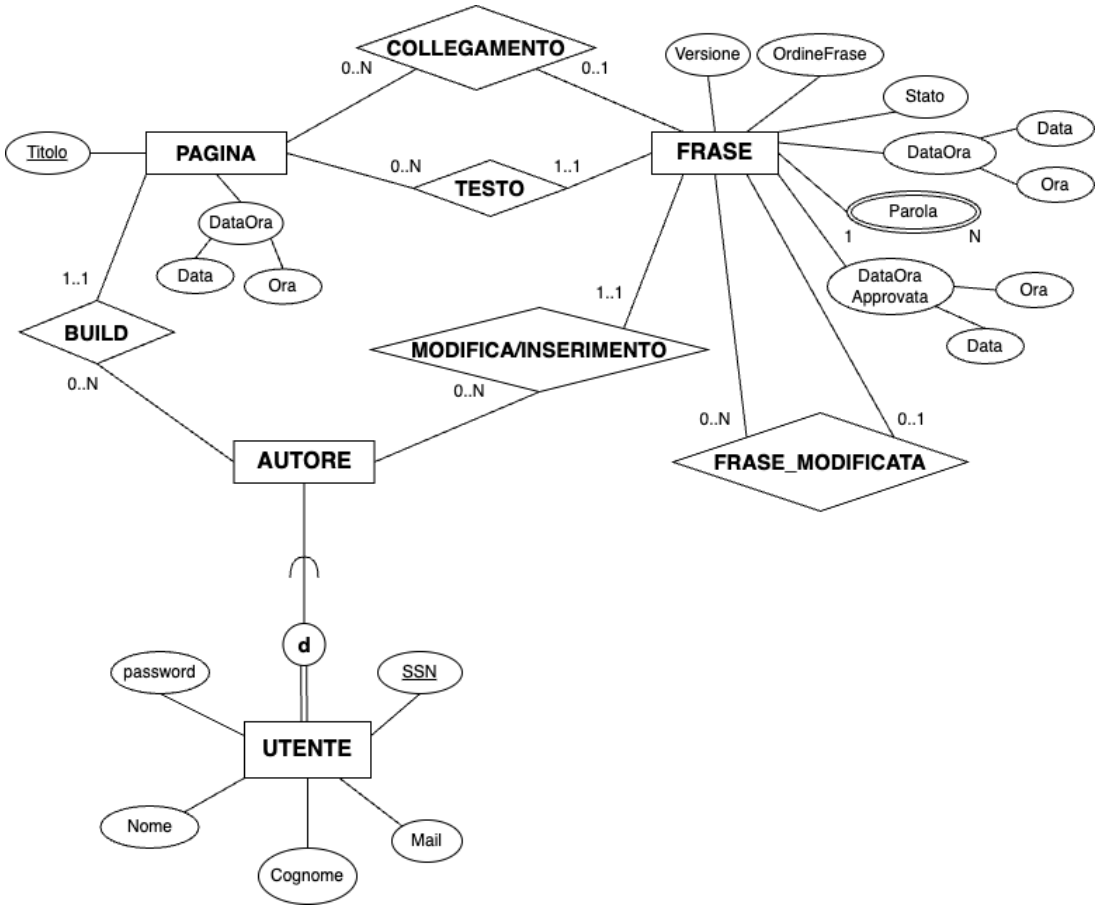

Figure 1: Modellazione ER

1.2.2 Modellazione UML

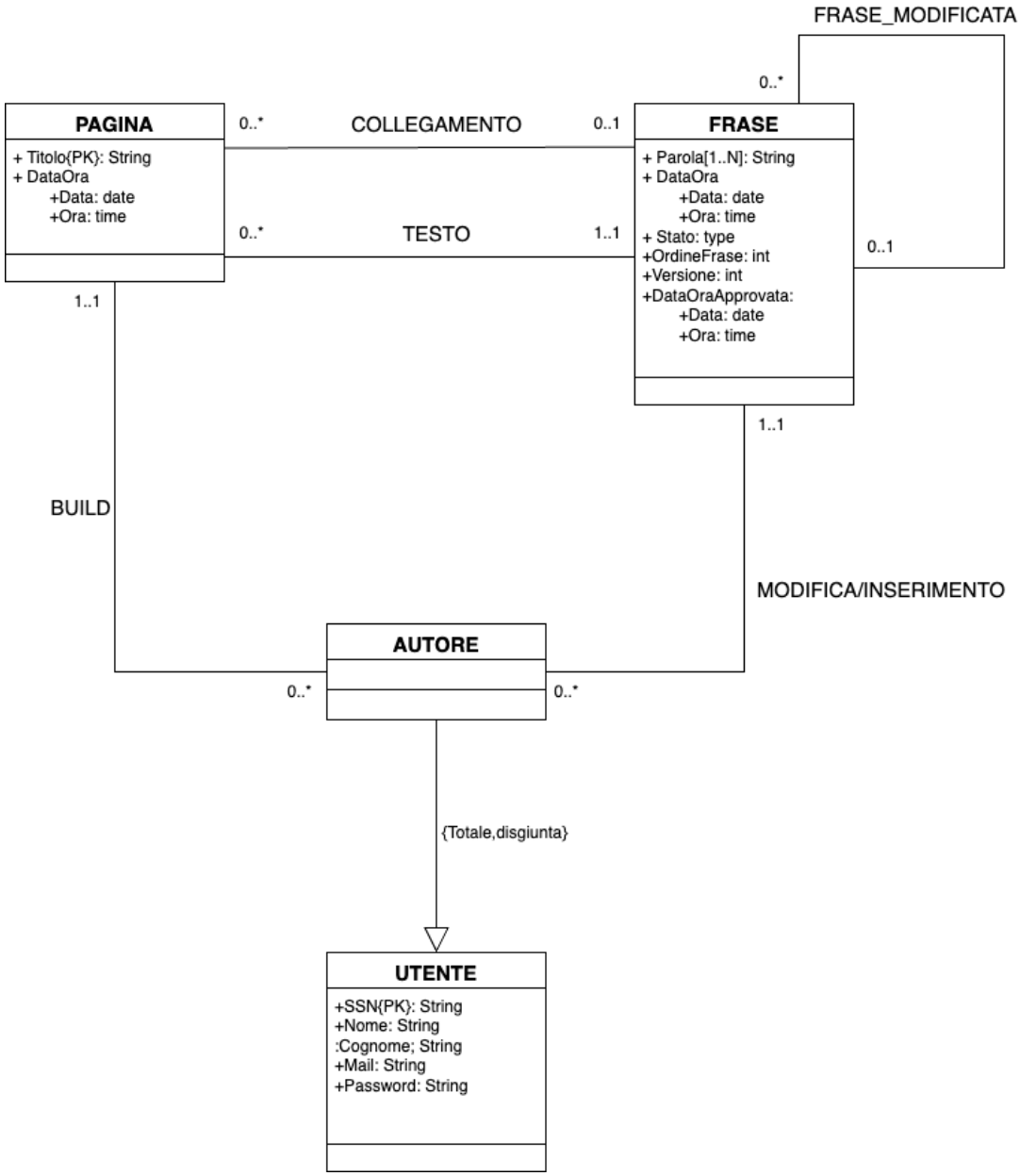

Figure 2: Modellazione UML

1.3 Dizionario delle entità e delle associazioni

1.3.1 Dizionario delle entità

ENTITA'	DESCRIZIONE	ATTRIBUTI
Pagina	Pagina generica della sistema. Permette di gestire ogni pagina che viene creata nel sistema	**Titolo**(String): titolo della pagina, è chiave primaria e permette di distinguere ogni pagina univocamente. **DataOra**: attributo composto, indica la data e l'ora di creazione della pagina. E' formato da: **Data Ora**

ENTITA'	DESCRIZIONE	ATTRIBUTI
Frase	La singola tupla di questa tabella indica tutte le caratteristiche e le proprietà di quella frase all'interno del database, in modo da mantenere tutti i cambiamenti che ci saranno all'interno del sistema senza perdere dati importanti, ad esempio quando una frase viene sostituita da un'eventuale modifica.	**Parola**[1..N]: attributo multivalore che permette di rappresentare ogni parola di quella frase. **DataOra**: attributo composto, indica la data e l'ora di creazione della frase. E' formato da: **Data** **Ora** **Stato**(Data type): attributo tipo che permette di identificare lo stato della frase nel sistema, permettendo di conseguenza di mantenere ogni modifica delle pagine all'interno della base di dati. I valori che può assumere sono: *Accettato, Rifiutato, Sostituito, In_attesa*. **OrdineFrase**(int): attributo che permette di mantenere l'ordine di quella singola frase all'interno della pagina. **Versione**(int): attributo che mantiene la versione di quella frase in base a eventuali modifiche. **DataOraApprovata**: attributo composto che indica la data e l'ora di quando una modifica a un'eventuale frase viene approvata. Può assumere anche valore *null* per le frasi che vengono inserite, quando non si tratta di una modifica. E' formato da: **Data** **Ora**

ENTITA'	DESCRIZIONE	ATTRIBUTI
Utente	Permette di identificare e mantenere i dati per ogni utente registrato nel sistema	**SSN**(String): codice fiscale dell'utente. E' chiave primaria di quest'entità e permette di rendere univoco ogni utente. Particolarità di quest'attributo è che per essere un ssn valido, deve essere formato *obbligatoriamente* da 9 caratteri alfa-numerici. **Nome**(String): indica il nome dell'utente. **Cognome**(String): indica il cognome dell'utente. **Mail**(String): indica la mail dell'utente. **Password**(String): indica la password dell'utente, necessaria per eseguire l'accesso.
Autore	Superclasse di utente, in quanto ogni utente registrato è un autore di N pagine o N modifiche a pagine di altri autori.	

1.3.2 Dizionario delle associazioni

ASSOCIAZIONI	DESCRIZIONE
Build	Associazione **uno-a-molti**. Si riferisce ad AUTORE e PAGINA, in quanto un autore può creare 0 o N pagine, mentre una pagina che esiste può essere creata solo da un'autore.
Collegamento	Associazione **uno-a-molti**. Si riferisce a PAGINA e FRASE, in quanto una frase può avere (non per forza) un collegamento a un'altra pagina, e una può avere 0 o N collegamenti a diverse altre frasi. Per esempio a una pagina *X*, possono essere collegate le frasi *Y,Z* ma non la frase *L*.
Testo	Associazione **uno-a-molti**. Si riferisce a PAGINA e FRASE. Una pagina può avere 0 (se la pagina all'inizio non dovesse avere un testo in base alle scelte dell'autore) o N frasi che vanno a formare il testo di quella pagina. Ogni frase invece avrà per forza una sola pagina a cui fa riferimento.
Modifica/Inserimento	Associazione **uno-a-molti**. Si riferisce ad AUTORE e FRASE. Un autore può effettuare 0 o N inserimenti/modifiche di frasi all'interno di una sua pagina o talvolta anche proporre modifiche a pagine non sue. Ovviamente per non perdere dati all'interno del sistema, sia l'inserimento di una frase che una sua eventuale modifica, corrisponde all'inserimento di una nuova tupla all'interno della base di dati, pertanto la frase dovrà avere obbligatoriamente solo un riferimento all'autore che l'avra inserita nel sistema.
Frase_Modificata	Associazione **uno-a-molti**. E' una relazione ricorsiva su FRASE. Permette di mantenere sulle modifiche il riferimento alla frase alla quale quella modifica si riferiva. Questo processo dal punto di vista pratico, verrà visto in maniera più approfondita più avanti, nel modello logico.

2 Ristrutturazione del modello concettuale

L'obiettivo di questa fase è modificare il modello concettuale per renderlo più adatto ad una traduzione al livello logico.

2.1 Analisi delle ridondanze

Non sono presenti particolari ridonanze da gestire o segnalare.

2.2 Analisi delle generalizzazioni

Durante quest'analisi, è necessario verificare le generalizzazioni presenti nella struttura, che dovranno essere ristrutturate prima della traduzione al livello logico.
Nell'attuale modello concettuale é possibile riconoscere una *generalizzazione* con una *specializzazione*:

- **Utente**, che è superclasse dell'unica sottoclasse *Autore*.

Si tratta di un vincolo **disgiunta-totale**, di conseguenza significa che ogni UTENTE è per forza un AUTORE. Questa specializzazione è stata aggiunta solo a scopo illustrativo e per una maggiore comprensione concettuale del sistema. Pertanto a questo punto si può tranquillamente eliminare l'entità AUTORE e continuare a operare sui singoli UTENTI del sistema.

2.3 Eliminazione degli attributi multi-valore

E' presente solo un attributo multivalore cioè **Parola** che permette di gestire le parole che vanno a formare la frase. Tuttavia questa è solo una definizione a livello concettuale in quanto andare a gestire singolarmente ogni parola all'interno della base di dati, sarebbe insostenibile a lungo termine, pertanto questo attributo multivalore diventerà un attributo come gli altri e verrà trattato come stringa.

2.4 Eliminazione attributi strutturati

Gli attributi strutturati o composti presenti sono tutti quelli che riguardano la data e l'ora per l'inserimento di una pagina o di una frase nel sistema. Pertanto andremo a trattarli come un unico attributo **DataOra** che sarà di tipo *timestamp* permettendo di gestire sia la data che l'ora.

2.5 Partizione/Accorpamento delle entità/associazioni

Non sono presenti casi di partizione o accorpamento di entità/associazioni.

2.6 Identificazione chiavi primarie

Quest'ultima fase della ristrutturazione prevede l'analisi e la motivazione della presenza o meno di chiavi primarie per ogni entità.
L'entità **Pagina** ha come chiave primaria *Titolo* che rende univoca ogni pagina. Tuttavia per motivi di praticità e per un acesso agli indici più rapido, è preferibile inserire un id, che verra chiamato **id_pagina** e sarà la nuova chiave primaria. Per quanto riguarda il titolo, rimarrà in ogni caso una chiave candidata e avrà la caratteristica di essere UNIQUE.
L'entità **Frase** attualmente non ha chiavi primarie, e questo è un problema sia per quanto riguarda l'accesso agli indici sia anche per la relazione ricorsiva di quest'entità. Pertanto è necessario aggiungere un id per ogni tupla di frase, che verrà chiamato **id_frase** e sarà chiave primaria.
L'entità **Utente** ha come chiave primaria **SSN**. Sicuramente anche in questo caso per avere un rapido accesso agli indici sarebbe utile introdurre un id, tuttavia per motivi di sicurezza ha più senso rimanere il codice fiscale come chiave primaria.

2.7 Schema ristrutturato

2.7.1 Modellazione ER

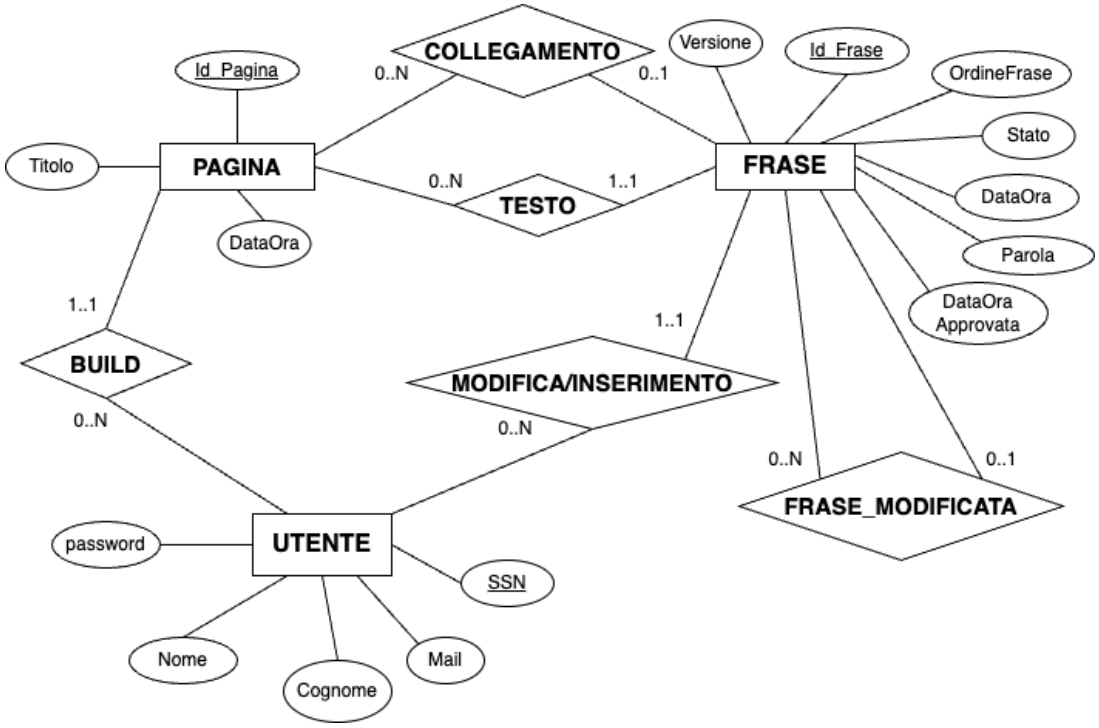

Figure 3: Schema ristrutturato: Modellazione ER

2.7.2 Modellazione UML

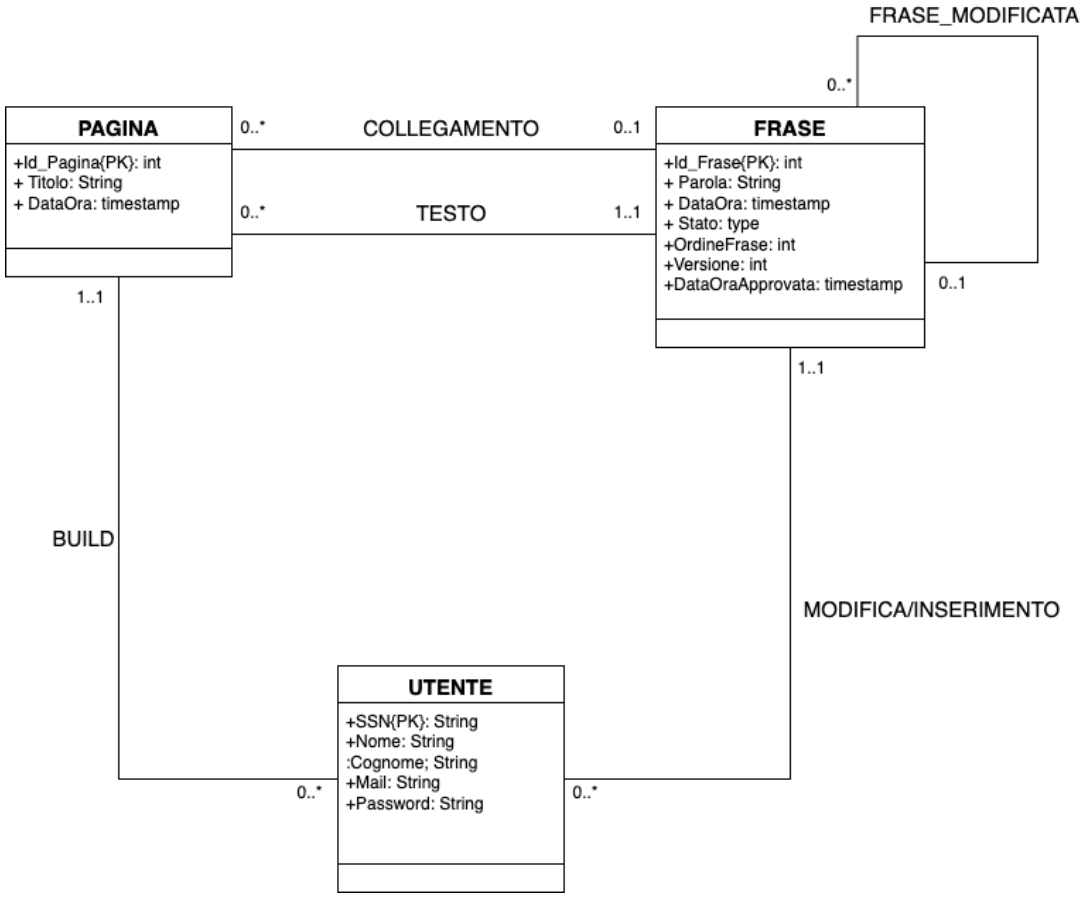

Figure 4: Schema ristrutturato: Modellazione UML

2.8 Dizionario delle entità e associazioni ristrutturato

2.8.1 Dizionario delle entità

ENTITA'	DESCRIZIONE	ATTRIBUTI
Pagina	Pagina generica della sistema. Permette di gestire ogni pagina che viene creata nel sistema	**Id_pagina**(int): chiave primaria, rappresenta univocamente ogni pagina all'interno della base di dati. **Titolo**(String): chiave candidata, indica il titolo della pagina che dovrà essere unico all'interno del sistema. **DataOra**(timestamp): indica la data e l'ora di creazione della pagina.

ENTITA'	DESCRIZIONE	ATTRIBUTI
Frase	La singola tupla di questa tabella indica tutte le caratteristiche e le proprietà di quella frase all'interno del database, in modo da mantenere tutti i cambiamenti che ci saranno all'interno del sistema senza perdere dati importanti, ad esempio quando una frase viene sostituita da un'eventuale modifica.	**Id_frase**(int): chiave primaria di frase. Permette di rendere univoca ogni frase nel sistema. **Parola**(String): indica l'insieme delle parole che vanno a formare quella frase. **DataOra**(timestamp): indica la data e l'ora di inserimento della frase nel database. **Stato**(Data type): attributo tipo che permette di identificare lo stato della frase nel sistema, permettendo di conseguenza di mantenere ogni modifica delle pagine all'interno della base di dati. I valori che può assumere sono: *Accettato, Rifiutato, Sostituito, In_attesa*. **OrdineFrase**(int): attributo che permette di mantenere l'ordine di quella singola frase all'interno della pagina. **Versione**(int): attributo che mantiene la versione di quella frase in base a eventuali modifiche. **DataOraApprovata**(timestamp): indica la data e l'ora di quando una modifica a un'eventuale frase viene approvata. Può assumere anche valore *null* per le frasi che vengono inserite, quando non si tratta di una modifica.

ENTITA'	DESCRIZIONE	ATTRIBUTI
Utente	Permette di identificare e mantenere i dati per ogni utente registrato nel sistema	**SSN**(String): codice fiscale dell'utente. E' chiave primaria di quest'entità e permette di rendere univoco ogni utente. Particolarità di quest'attributo è che per essere un ssn valido, deve essere formato *obbligatoriamente* da 9 caratteri alfa-numerici. **Nome**(String): indica il nome dell'utente. **Cognome**(String): indica il cognome dell'utente. **Mail**(String): indica la mail dell'utente. **Password**(String): indica la password dell'utente, necessaria per eseguire l'accesso.

2.8.2 Dizionario delle associazioni

ASSOCIAZIONI	DESCRIZIONE
Build	Associazione **uno-a-molti**. Si riferisce ad AUTORE e PAGINA, in quanto un autore può creare 0 o N pagine, mentre una pagina che esiste può essere creata solo da un'autore.
Collegamento	Associazione **uno-a-molti**. Si riferisce a PAGINA e FRASE, in quanto una frase può avere (non per forza) un collegamento a un'altra pagina, e una può avere 0 o N collegamenti a diverse altre frasi. Per esempio a una pagina *X*, possono essere collegate le frasi *Y,Z* ma non la frase *L*.
Testo	Associazione **uno-a-molti**. Si riferisce a PAGINA e FRASE. Una pagina può avere 0 (se la pagina all'inizio non dovesse avere un testo in base alle scelte dell'autore) o N frasi che vanno a formare il testo di quella pagina. Ogni frase invece avrà per forza una sola pagina a cui fa riferimento.
Modifica/Inserimento	Associazione **uno-a-molti**. Si riferisce ad AUTORE e FRASE. Un autore può effettuare 0 o N inserimenti/modifiche di frasi all'interno di una sua pagina o talvolta anche proporre modifiche a pagine non sue. Ovviamente per non perdere dati all'interno del sistema, sia l'inserimento di una frase che una sua eventuale modifica, corrisponde all'inserimento di una nuova tupla all'interno della base di dati, pertanto la frase dovrà avere obbligatoriamente solo un riferimento all'autore che l'avra inserita nel sistema.
Frase_Modificata	Associazione **uno-a-molti**. E' una relazione ricorsiva su FRASE. Permette di mantenere sulle modifiche il riferimento alla frase alla quale quella modifica si riferiva. Questo processo dal punto di vista pratico, verrà visto in maniera più approfondita più avanti, nel modello logico.

3 Traduzione al modello logico

3.1 Mapping associazioni

3.1.1 Associazioni 1 a N

- Per l'associazione **Build** tra *Autore* e *Pagina*, è stato deciso di inserire una chiave esterna di *autore* in *pagina*.
- Per l'associazione **Collegamento** tra *Pagina* e *Frase*, è stato deciso di inserire una chiave esterna di *pagina* in *frase*.
- Per l'associazione **Testo** tra *Pagina* e *Frase*, è stato deciso di inserire una chiave esterna di *pagina* in *frase*.
- Per l'associazione **Modifica/Inserimento** tra *Autore* e *Frase*, è stato deciso di inserire una chiave esterna di *autore* in *frase*.
- L'associazione **Frase_Modificata** è un'associazione ricorsiva di **Frase** e viene gestita inserendo una chiave esterna di frase in frase, che potrà avere anche valore *null*. Dal punto di vista pratico, una modifica a una frase sarà una nuova tupla inserita nella base di dati, che avrà la chiave esterna di frase (*id_frase*) valorizzata con l'id della frase a cui è associata quella modifica. Se una frase viene semplicemente inserita, allora quel campo sarà valorizzato a *null*. Se una o più modifiche proposte da uno o più utenti, dovessero essere rifiutate, quel campo verrà in ogni caso valorizzato con l'id della frase a cui fa riferimento.

3.1.2 Associazioni N a N

Non sono presenti associazioni N a N.

3.1.3 Associazioni 1 a 1

Non sono presenti associazioni 1 a 1.

3.2 Modello logico

Gli attributi sottolineati rappresentano la chiave primaria, quelli in *corsivo* le chiavi esterne.

Utente (ssn, Nome, Cognome, Mail, Password)

Pagina (*Id_pagina*, Titolo, DataOra, *ssn_autore*)
Pagina.ssn_autore -⟩ Utente.ssn

Frase

(*Id_frase*, Parola, DataOra,Stato,OrdinaFrase, Versione,
DataOraApprovata, *ssn_utente*, *Id_pagina*, *Id_collegamento*, *Id_FraseMod*)
Frase.ssn_utente -⟩ Utente.ssn
Frase.Id_pagina -⟩ Pagina.Id_pagina
Frase.Id_collegamento -⟩ Pagina.Id_pagina
Frase.Id_FraseMod -⟩ Frase.Id_frase

4 Modello fisico

4.1 Creazione del database

Per verificare il sistema nell'ambiente di sviluppo, la base di dati è stata creata su una istanza di PostgreSQL attivata in un "container" tramite un engine installato in locale (**Podman**), per la gestione di contenitori in ambiente Linux. Questi contenitori o container da un punto di vista pratico sono delle piccolissime macchine virtuali che permettono di eseguire applicazioni in modo sicuro e comodo.
Per connettersi all'istanza di PostgreSQL, verrà utilizzato **DBeaver**, un'applicazione di amministrazione di database gratuita e open source che fornisce un'interfaccia grafica per interagire con diversi tipi di database, semplificandone tutte le attività di gestione.

4.2 Creazione delle tabelle

In questa sezione sono riportate le istruzioni DDL utilizzate per la creazione delle tabelle all'interno della base di dati.

Utente:

```
CREATE TABLE public.utente (
    ssn varchar(9) NOT NULL,
    nome varchar(45) NOT NULL,
    cognome varchar(45) NOT NULL,
    mail varchar(100) NOT NULL,
    "password" varchar(255) NOT NULL,
    CONSTRAINT ssn_check CHECK ((length(ltrim((ssn)::text)) = 9)),
    CONSTRAINT utente_pkey PRIMARY KEY (ssn)
);
```

Pagina:

```
CREATE TABLE public.pagina (
    id_pagina serial4 NOT NULL,
    titolo varchar(125) NOT NULL,
    dataora timestamp NOT NULL,
    ssn_autore bpchar(9) NOT NULL,
    CONSTRAINT pagina_pkey PRIMARY KEY (id_pagina),
    CONSTRAINT titolo_unico UNIQUE (titolo)
);

ALTER TABLE public.pagina ADD CONSTRAINT fk_autore FOREIGN KEY (ssn_autore)
REFERENCES public.utente(ssn) ON DELETE CASCADE ON UPDATE CASCADE;
```

Frase:

```
CREATE TABLE public.frase (
    id_frase serial4 NOT NULL,
    parola varchar(255) NOT NULL,
    dataora timestamp NOT NULL,
    stato public."stato_type" NOT NULL,
    ordinefrase int4 NOT NULL,
    versione int4 DEFAULT 1 NOT NULL,
    dataoraapprovata timestamp NULL,
    ssn_utente bpchar(9) NOT NULL,
    id_pagina int4 NOT NULL,
    id_collegamento int4 NULL,
    id_frasemod int4 NULL,
    CONSTRAINT frase_pkey PRIMARY KEY (id_frase)
);
```

ALTER TABLE public.frase ADD CONSTRAINT fk_collegamento FOREIGN KEY (id_collegamento) REFERENCES public.pagina(id_pagina) ON DELETE SET NULL ON UPDATE CASCADE;

ALTER TABLE public.frase ADD CONSTRAINT fk_frasemod FOREIGN KEY (id_frasemod) REFERENCES public.frase(id_frase) ON DELETE SET NULL ON UPDATE CASCADE;

ALTER TABLE public.frase ADD CONSTRAINT fk_pagina FOREIGN KEY (id_pagina) REFERENCES public.pagina(id_pagina) ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE public.frase ADD CONSTRAINT fk_utente FOREIGN KEY (ssn_utente) REFERENCES public.utente(ssn) ON DELETE CASCADE ON UPDATE CASCADE;

4.3 Vincoli

4.3.1 Dizionario dei vincoli

VINCOLO	TIPO DI VINCOLO	INTRA/INTER
stato_type	Vincolo di dominio Si tratta di un **data type**, e in quanto tale è possibile considerarlo come un vincolo di dominio, poichè si sta definendo un insieme di regole o restrizioni che si applicano a una specifica colonna o attributo	Vincolo intra-relazionale
ssn_check	Vincolo di check	Vincolo intra-relazionale
utente_pkey	Vincolo di chiave	Vincolo intra-relazionale
pagina_pkey	Vincolo di chiave	Vincolo intra-relazionale
titolo_unico	Vincolo di unicità	Vincolo intra-relazionale
frase_pkey	Vincolo di chiave	Vincolo intra-relazionale
fk_autore	Vincolo di integrità referenziale	Vincolo inter-relazionale
fk_collegamento	Vincolo di integrità referenziale	Vincolo inter-relazionale
fk_frasemod	Vincolo di integrità referenziale	Vincolo inter-relazionale
fk_pagina	Vincolo di integrità referenziale	Vincolo inter-relazionale
fk_utente	Vincolo di integrità referenziale	Vincolo inter-relazionale

Ci sono da considerare inoltre i vincoli di **INTEGRITA' DI ENTITA'**, che si riferiscono al mantenimento della correttezza e della coerenza dei dati all'interno di una tabella di un database. Ciò implica garantire che ogni record in una tabella sia unico e che tutte le colonne necessarie contengano valori validi. Quindi in questi vincoli sono inclusi tutti i **Vincoli di NOT NULL**, che garantiscono che quella o quelle specifiche colonne non abbiano valori nulli. Ad esempio, nel caso delle tabelle **Utente** e **Pagina**, per creare un record ogni campo dovrà essere valorizzato.
I vincoli di integrità di entità includono anche i Vincoli di chiave e i Vincoli di unicità.

4.4 Trigger

Il trigger è un tipo di istruzione che viene eseguita in risposta a determinati eventi o azioni nel sistema di gestione del database.
In questa base di dati è stato inserito un trigger che permette di cambiare automaticamente lo *stato* di una frase, in seguito a delle modifiche da parte dell'utente di quella pagina. In questo modo si terrà constantemente aggiornata l'ultima versione di quella pagina, ma sarà possibile mantenere all'interno del database anche tutte le modifiche, sia quelle accettate che rifiuatate.
Questo trigger agisce sulla tabella frase.

Frase:
Trigger:
imposta_stato_sostituito:
create trigger imposta_stato_sostituito
after insert or update on public.frase
for each row
when (((new.id_frasemod is not null)
and (new.stato = 'accettato'::stato_type))) execute function update_stato_frase()

Tramite un'accurata ricerca sulla documentazione PostgreSQL, per permettere il corretto funzionamento di un trigger, è necessario utilizzare una funzione che restituisce un trigger. Pertanto è stata utilizzata la funzione **update_stato_frase**.

```
CREATE OR REPLACE FUNCTION public.update_stato_frase()
RETURNS trigger
LANGUAGE plpgsql
AS $function$
begin
    update frase
    set stato = 'sostituito'
    where ordinefrase = new.ordinefrase
    and id_pagina = new.id_pagina
    and stato = 'accettato'
    and id_frase ¡¿ new.id_frase;
return new;
end;
$function$
;
```

4.5 Funzioni

Ogni funzione presente ha uno scopo specifico all'interno del sistema. Sono state implementate con il linguaggio PLSQL per fare in modo che all'applicazione venga passato direttamente l'output richiesto, che nella maggior parte dei casi sarà una stringa di testo. In altri casi invece vengono utilizzate delle query che vengono scritte e mandate direttamente dall'applicazione e che verranno viste in seguito. La prima funzione visualizzata è stata quella per l'implementazione e il corretto funzionamento del trigger, ossia **update_stato_frase**, che permette di aggiornare lo stato di una frase a "sostituito" in base a uno specifico evento del trigger.

Di seguito le altre funzioni utilizzate:

testo_pagina:

```
 CREATE OR REPLACE FUNCTION public.testo_pagina(v_idpagina integer)
 RETURNS text
LANGUAGE plpgsql
AS $function$
declare
testo text := '';
vfrase varchar(100);
cursore refcursor;
begin
    open cursore for
       select parola
       from frase
       where id_pagina = v_idpagina and stato = 'accettato'
       order by ordinefrase ;
    loop
       fetch cursore into vfrase;
       exit when not found;
       testo := testo || vfrase || E'\n';
    end loop;
    return testo;
end;
$function$
;
```

Questa funzione prende in input l'id di una pagina, e ne restituisce in un'unica stringa tutte le frasi di quella pagina con stato "accettato". In altre parole è la funzione generica che restituisce il testo della pagina visibile a ogni utente generico del sistema.

cronologia_frasi:

```
CREATE OR REPLACE FUNCTION public.cronologia_frasi(v_idpagina integer,
v_ordine integer)
 RETURNS text
 LANGUAGE plpgsql
AS function
declare
testo text := ' ';
vfrase varchar(100);
v_versione integer;
cursore refcursor;
cursore2 refcursor;
begin
    open cursore for
        select parola, versione
        from frase
        where id_pagina = v_idpagina
        and stato in ('accettato','sostituito') and ordinefrase = v_ordine
        order by versione;
    loop
    fetch cursore into vfrase, v_versione;
    exit when not found;
    testo := testo || v_versione || ')' || vfrase || E'\n';
    end loop;
    return testo;
end;
function
;
```

Questa funzione prende in input un id di una pagina, e un intero che rappresenta l'ordine di una frase. Restituisce lo storico di tutte le versioni di quella frase.

storico_pagina:

```
CREATE OR REPLACE FUNCTION public.storico_pagina(datapubb timestamp
without time zone, v_idpagina integer)
 RETURNS text
LANGUAGE plpgsql
AS function
declare
testo text := ' ';
vfrase varchar(100);
cursore refcursor;
begin
    open cursore for
       select f1.parola
       from frase as f1
       where (f1.dataoraapprovata ¡= datapubb or f1.dataoraapprovata is null)
       and f1.id_pagina = v_idpagina
       and (f1.stato = 'accettato' or f1.stato = 'sostituito')
       and f1.versione = (select max(f2.versione)
          from frase as f2
          where (f2.dataoraapprovata ¡= datapubb or f2.dataoraapprovata is null)
          and f2.id_pagina = f1.id_pagina
          and (f2.stato = 'accettato' or f2.stato = 'sostituito')
          and f2.ordinefrase = f1.ordinefrase)
       order by f1.ordinefrase;
    loop
    fetch cursore into vfrase;
    exit when not found;
    testo := testo || vfrase || E'\n';
    end loop;
 return testo;
end;
function
;
```

Questa funzione prende in input un timestamp e l'id di una pagina, e in base al timestamp ti recupera lo stato di quella pagina in quel preciso momento.

4.6 Query

In quest'ultima sezione ci sarà un riepilogo di tutte le query che è possibile utilizzare all'interno del sisteme suddivise per la tabella a cui fanno riferimento.

Utente:

SELECT * FROM utente WHERE ssn = 'stringa'
Query che in base a un ssn dato in input, restituisce tutti i campi di quell'utente.

INSERT INTO utente
VALUES ('stringa','stringa','stringa','stringa','stringa')
Query che permette l'inserimento di un utente all'interno del database ed è necessario valorizzare ogni campo altrimenti si va a violare il vincolo di integrità di entità.

DELETE FROM utente WHERE ssn = 'stringa'
Query che permette la cancellazione di un utente nel sistema dato un certo ssn.

Pagina:

INSERT INTO pagina (titolo,dataora,ssn_autore)
VALUES ('stringa',current_timestamp,'stringa')
Query che permette l'inserimento di una pagina nel sistema. L'id_pagina viene valorizzato e incrementato da solo in quanto è un seriale, mentre il titolo deve essere diverso da un altro titolo esistente nel database, altrimenti si va a violare il vincolo di unicità.

SELECT id_pagina,titolo,ssn_autore
FROM pagina
WHERE ssn_autore = 'stringa'
Query che restituisce tutti i campi della pagina necessari per il funzionamento del sistema dato uno specifico ssn.

DELETE FROM pagina
WHERE id_pagina = 'int'
Query che rimuove una pagina specifica dato uno specifico id.

SELECT id_pagina,titolo,ssn_autore
FROM pagina
WHERE titolo = 'stringa'
Query che restituisce tutti i campi della pagina necessari per il funzionamento del sistema tramite la ricerca del titolo della pagina.

Frase:

INSERT INTO frase (parola,dataora,stato,ordinefrase,ssn_utente,id_pagina)
VALUES ('stringa', current_timestamp,'accettato', 'int','stringa','int')
Query che permette l'inserimento di una frase nel sistema. L'id_frase viene valorizzato ed incrementato da solo in quanto è un seriale.

SELECT id_frase,parola,stato,ordinefrase,versione,ssn_utente,id_pagina
FROM frase
WHERE id_pagina = 'int' AND stato = 'accettato'
ORDER BY ordinefrase
Query che seleziona tutte le frasi con stato accettato in base a uno specifico id_pagina.
INSERT INTO frase (parola,dataora,stato,ordinefrase,versione,dataoraapprovata,ssn_utente,id_pagina,id_

VALUES ('stringa', current_timestamp,'accettato','int,'int',current_timestamp, 'stringa','int','int')
Query che inserisce una frase modificata nel sistema.

SELECT ordinefrase
FROM frase
WHERE id_pagina = 'int' AND stato = 'accettato'
Query che recupera l'ordine della frasi di quella pagina a cui fa riferimento con stato accettato.

INSERT INTO frase (parola,dataora,stato,ordinefrase,versione,ssn_utente,id_pagina,id_frasemod)
VALUES ('stringa',current_timestamp,'in_attesa','int,'int','stringa','int','int')
Query che inserisce una proposta di modifica a una frase nel sistema.

SELECT F1.parola AS F1_parola, F2.parola AS F2_parola, P.titolo AS P_titolo, F2.id_frase AS F2_id_frase
FROM frase AS F1, frase AS F2, pagina AS P
WHERE F1.id_frase = F2.id_frasemod AND F1.stato = 'accettato' AND F2.stato = 'in_attesa' AND F1.id_pagina = P.id_pagina
ORDER BY F2.dataora
Query che restituisce tutte le proposte e le rispettive frasi a cui fanno riferimento, compreso il titolo e l'id della pagina a cui appartengono.

UPDATE frase
SET id_collegamento = 'int'
WHERE id_frase = 'int'
Query che permette di creare un collegamento di una frase con una pagina, quindi in termini di database consiste nell'effettuare un update.

SELECT P.titolo AS P_titolo
FROM frase AS F, pagina AS P
WHERE P.id_pagina = F.id_collegamento AND F.id_frase = 'int'
Query che permette di visualizzare il collegamento di una frase con una pagina dato un certo id_frase.

SELECT dataoraapprovata
FROM frase
WHERE dataoraapprovata is not null AND id_pagina = 'int'
Query che seleziona tutti i timestamp delle modifiche approvate di una rispettiva pagina.

UPDATE frase
SET stato = 'accettato' , dataoraapprovata = current_timestamp
WHERE id_frase = 'int'
Query che permette l'accettazione di una proposta di modifica da parte di un utente nel sistema.

UPDATE frase
SET stato = 'rifiutato'
WHERE id_frase = 'int'
Query che permette il rifiuto di una proposta di modifica da parte di un utente nel sistema.