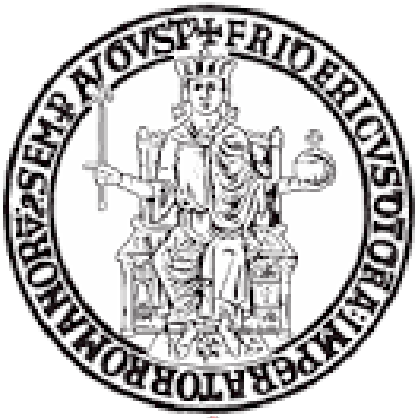


UNIVERSITÀ DEGLI STUDI DI
NAPOLI FEDERICO II

Progetto base dati

Creazione di un'applicazione che si occupa della gestione di un database di pagine wiki

Anno accademico 2022/2023

Corso informatica

Autori:
Mirko Prevenzano N86004082
Giuseppe Patalano N86004118

Indice

1 Modello concettuale

1.1 Analisi dei requisiti

Fase in cui vengono prese le informazioni richieste dal cliente passo per passo e analizzate per trovare una corretta implementazione

> *Si sviluppi un sistema informativo, composto da una base di dati relazionale 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 'e creata da un determinato autore. Il testo 'e composto di una sequenza di frasi. Il sistema mantiene traccia anche del giorno e ora nel quale la pagina 'e stata creata.*

Nel contesto del nostro progetto, è necessario sviluppare un'applicazione dedicata alla gestione del ciclo di vita di una pagina di una wiki. Nell'ambito di questa applicazione, l'entità centrale è rappresentata dalla **Pagina**. Ogni pagina deve avere un titolo unico e deve includere la data e l'ora della creazione come attributi.

Una pagina è costituita da una sequenza di frasi. Pertanto, abbiamo deciso di introdurre l'entità **Frase**, associata a una Pagina, per comporre il testo in un ordine sequenziale. Al fine di mantenere un ordine coerente delle frasi, abbiamo incluso un attributo aggiuntivo chiamato "ordine", che indica la posizione relativa delle frasi all'interno della pagina.

La creazione di una pagina è attribuita a un autore specifico.

> *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.*

All'interno del nostro progetto, abbiamo individuato nuove entità per ampliare il sistema. Iniziamo con l'entità **Collegamento**, che associa una pagina a partire da una frase di un'altra pagina. Successivamente, abbiamo l'entità **Utente**, che rappresenta un'entità generale in quanto un utente può visualizzare, proporre modifiche e, se è un autore, creare pagine. Pertanto, abbiamo deciso che l'entità Utente sia una generalizzazione, mentre Autore è una sua specializzazione.

> *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. 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*

Abbiamo introdotto l'entità **Proposta**, la quale seleziona più frasi e propone per ciascuna una nuova versione. A questo punto, è stato necessario inserire l'entità **Modifica**, che contiene la versione alternativa di una frase. Esiste un'associazione tra "Proposta" e "Modifica", e tra "Modifica" e "Frase".

Per quanto riguarda l'entità Proposta, abbiamo deciso di includere come attributi la data e l'ora di creazione, oltre a uno stato che può assumere quattro valori ('inviato', 'non inviato', 'accettato'

e 'rifiutato'). L'autore della pagina è colui che potrà gestire la proposta, solo una volta che tale proposta è inviata dall'utente. Se è un autore a creare una proposta per una sua pagina, questa è automaticamente accettata. Per elaborare le proposte, bisogna procedere partendo dalla meno recente; tutte queste funzionalità saranno garantite attraverso specifiche funzioni.

> *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*

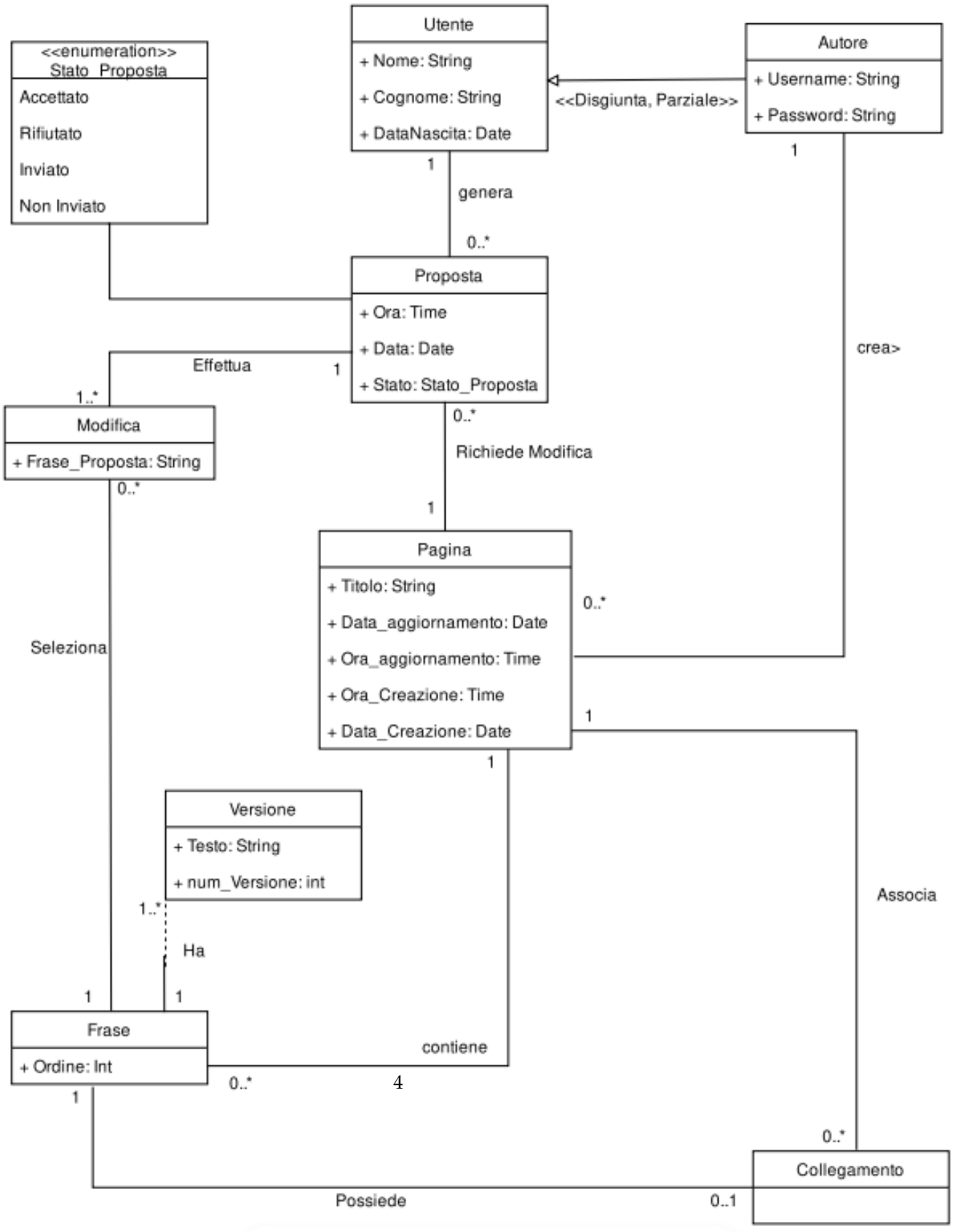
Gli Autori per poter accedere hanno bisogno di "username" e "password", pertanto sono stati aggiunti come attributi. Gli utenti possono cercare e visualizzare una pagina, mentre gli autori possono visualizzare lo storico delle pagine create oppure visualizzare quelle in cui hanno proposto una modifica. Queste funzionalità saranno implementate durante la fase di sviluppo attraverso delle apposite funzioni.

1.2 Elementi aggiuntivi

Durante questa fase, abbiamo considerato l'inserimento dell'entità "Testo" come indicato dalla consegna. Tuttavia, tale entità non conteneva informazioni significative oltre a essere composta da frasi. Per questo motivo, abbiamo scelto di unire il tutto aggiungendo un'associazione tra la pagina e le frasi che compongono virtualmente il testo, seguendo un certo ordine.

Poiché una pagina può essere aggiornata, modificando una frase, abbiamo deciso di includere anche gli attributi "ora_aggiornamento" e "data_aggiornamento". Inoltre, dato che una proposta può modificare una frase, è stata aggiunta l'entità "Versione" con gli attributi "num_versione" che tiene conto della versione della pagina. In questo modo è possibile reperire lo storico delle versioni che hanno caratterizzato una pagina nel tempo, oltre a contenere il testo in riferimento al contenuto della frase.

1.3 Schema UML



2 Modello concettuale ristrutturato

2.1 Analisi rindondanze

In questa fase andiamo ad analizzare il nostro modello concettuale alla ricerca di attributi derivabili. Questi attributi possono essere derivati da altri attributi della stessa entità o da altre entità, o anche eventualmente dal conteggio delle occorrenze. Nel nostro modello concettuale non sono stati individuati attributi derivabili.

2.2 Analisi delle generalizzazioni

Andiamo alla ricerca di generalizzazioni all'interno del nostro modello in modo da poterle rimuovere, in quanto nei moderni sistemi di gestioni del DB non è concessa la presenza di generalizzazioni. Tra i vari modi per poter eliminare una generalizzazione abbiamo:

- **Accorpamento della classe padre nelle figlie**, questa soluzione va utilizzata quando la generalizzazione è totale, altrimenti avremmo il rischio di non rappresentare alcune occorrenze.
- **Accorpamento delle classi figlie nel padre**, questa soluzione è utilizzabile quando non fanno molta differenza le operazioni tra le varie occorrenze e gli attributi.
- **Sostituzione della generalizzazione con un'associazione**, questa soluzione è utile applicarla quando la generalizzazione non è totale, e ci sono operazioni che fanno distinzioni tra la classe padre e la classe figlia.

Nel nostro modello è presente un'unica generalizzazione ovvero quella tra Utente e Autore. Utente è la classe padre, mentre Autore è la classe figlia, abbiamo optato per eliminare tale generalizzazione la terza soluzione, ovvero creare un'associazione tra utente e autore in quanto non è una generalizzazione totale; infatti, non tutti gli utenti sono autori, e, in aggiunta a ciò, autore e utente hanno operazioni diverse quanto interagiscono con le pagine e soprattutto con le proposte a tali pagine.

2.3 Eliminazione attributi multivalori

In questo punto cerchiamo attributi Multi-valore in modo da poterli eliminare, anche in questo caso abbiamo 3 opzioni:

- Creare una classe esterna associata.
- Trattare l'attributo multiplo come singolo.
- Replicare l'attributo n volte nella classe.

Nel nostro modello non sono stati utilizzati attributi multi-valore

2.4 Eliminazione attributi strutturati

Punto molto simile al precedente solo che qui andiamo a cercare attributi strutturati, ovvero attributi che rappresentano diverse informazioni in modo indipendente. Possono essere eliminati in 3 modi differenti:

- Introduzione di una classe per l'attributo strutturato.
- Estrazione degli attributi della classe.
- Trascurare la struttura dell'attributo.

Anche in questo caso nel nostro modello non sono stati trovati attributi strutturati.

2.5 Accorpamento di entita/associazioni

Questa fase prevede l'accorpamento di entità in altre in modo da ridurre il numero di classi, è una scelta che viene applicata soprattutto tra le entità in cui è presente una relazione di tipo 1 a 1. Nel nostro modello non è stato fatto nessun'accorpamento.

L'unica riflessione applicata su questo punto era tra la classe Frase e la classe Collegamento, in cui è presente una relazione 1 a 1, abbiamo deciso tuttavia di non accorparle perché il collegamento non è un valore che viene applicato a tutte le frasi, ma a ben poche in realtà; dunque, ci è sembrato opportuno evitare l'accorpamento in modo da evitare la presenza di molti valori null.

2.6 Identificazione chiavi primarie

Infine, c'è la scelta delle chiavi primarie per ogni classe del nostro DB.

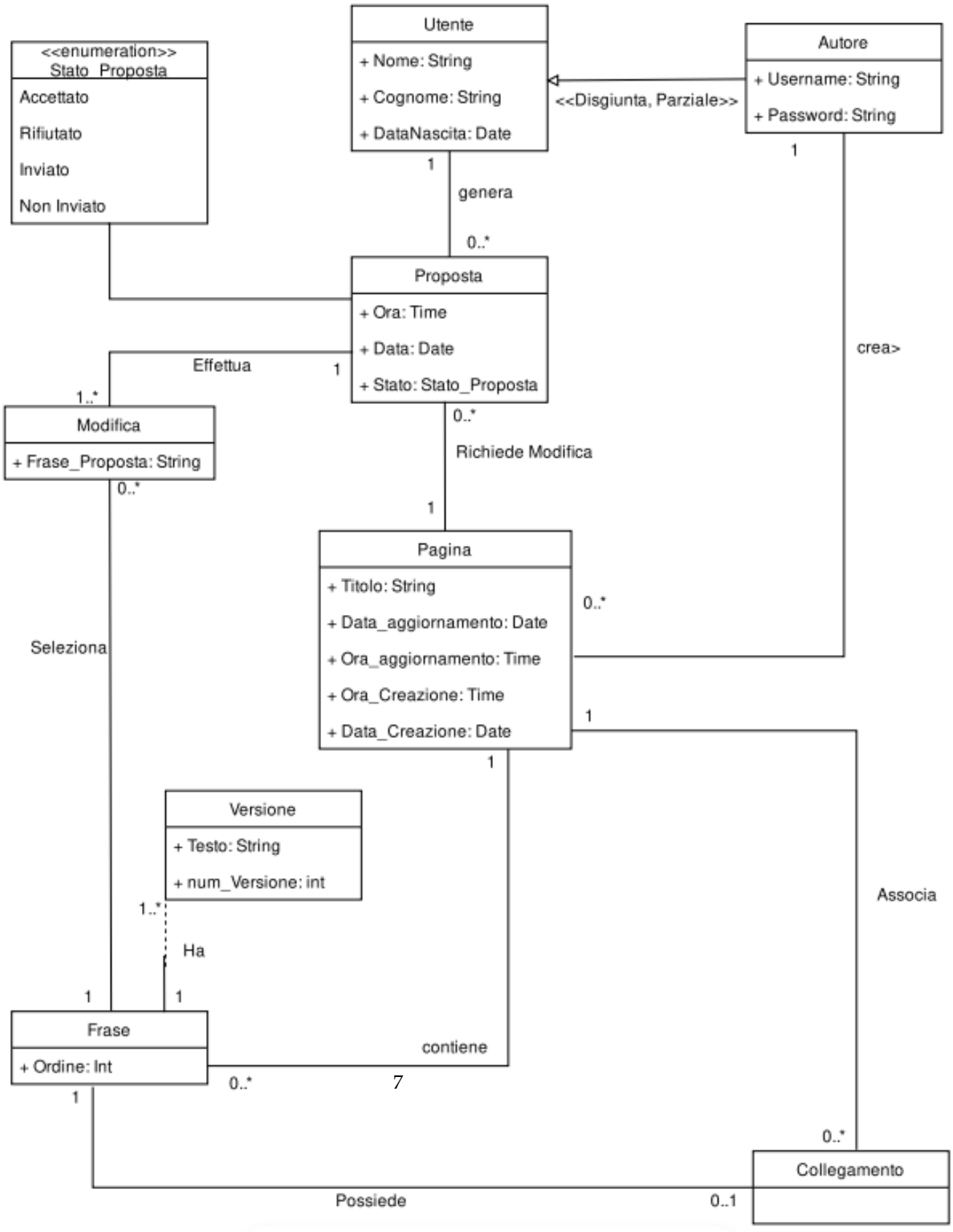
Per **Utente** è stato scelto l'attributo matricola, ereditato anche da **Autore** che insieme ad Username vanno a formare la sua chiave primaria.

Per **Proposta**, **Modifica**, **Frase** e **Collegamento**, vengono inseriti i rispettivi ID come chiavi primarie.

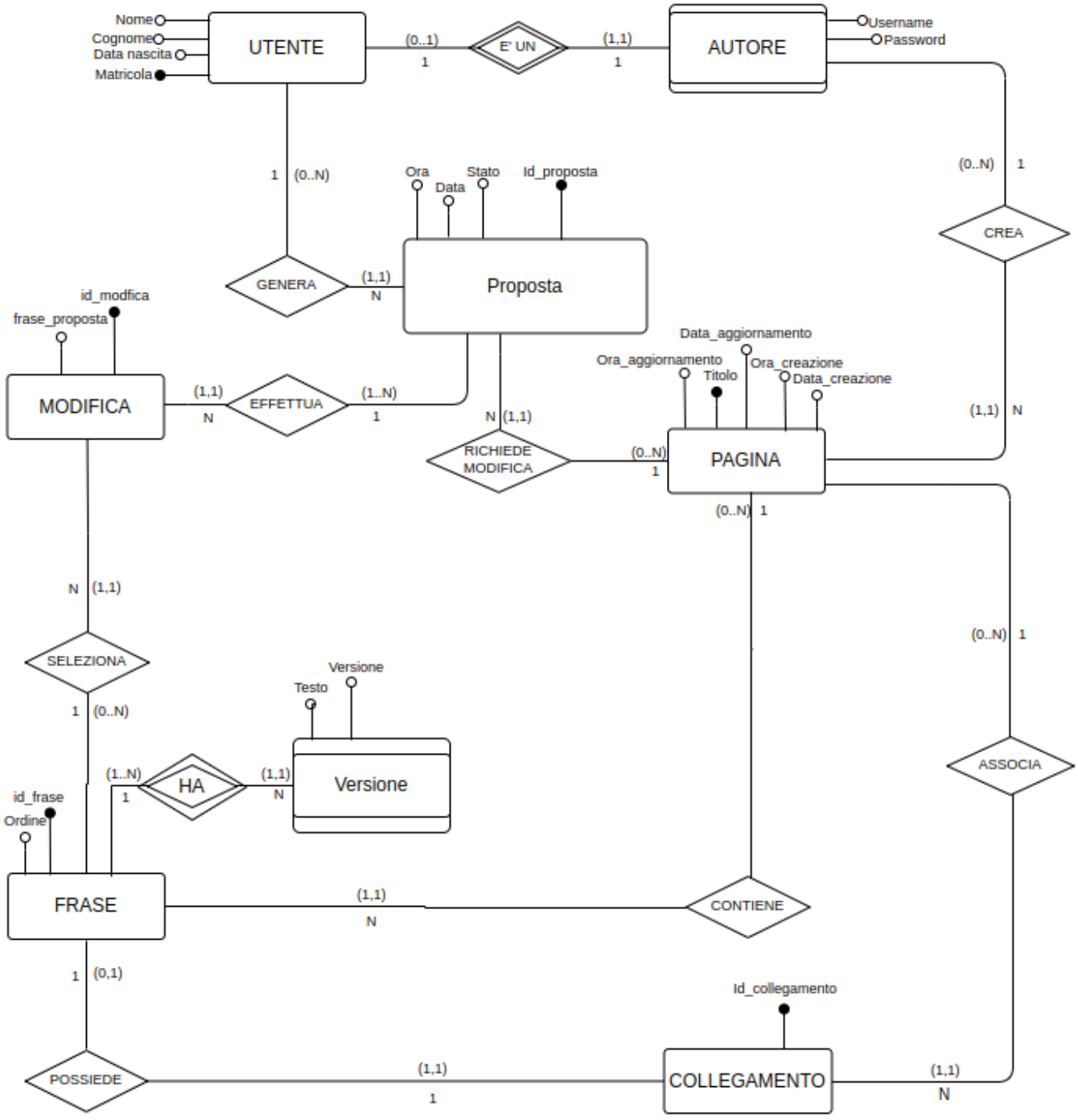
Per **Pagina** invece è stato scelto l'attributo Titolo come chiave primaria.

Infine, per **Versione** che eredita l'attributo ID_Frase viene aggiunto il suo attributo num_versione, in modo da formare la sua chiave primaria.

2.7 Schema UML ristrutturato



2.8 Schema E-R ristrutturato



3 Dizionari

3.1 Dizionario delle entità

Entità	Descrizione	Attributi
Utente	Entità che rappresenta le figure che usano l'applicazione, visualizzano le pagina e propongano modifiche	**Matricola**(stringa): attributo che identifica univocamente un utente. **Nome**(stringa) **Cognome**(stringa) **Data_nascita**(Date)
Autore	Entità che rappresenta la figura di chi crea pagina e gestisce le proposte	**Username**(stringa) **Password**(stringa)
Pagina	Entità centrale del database, composta da frasi.	**Titolo**(stringa) **Ora_creazione**(time): indica l'orario esatto in cui è creata una pagina **Data_creazione**(date): indica la data esatta in cui è creata una pagina **Data_aggiornamento**(date): indica la data esatta in cui è stata aggiornata una pagina **Ora_aggiornamento**(date): indica l'ora esatta in cui è stata aggiornata una pagina
Frase	Entità che rappresenta le singole frasi di una pagina	**id_frase**(serial): attributo univoco **Ordine**(intero): indica la posizione della frase in una pagina
Proposta	Entità che memorizza le proposte effettuate da un utente, su una pagina.	**Id_proposta**(serial):attributo univoco **Ora**(time): indica l'orario esatto in cui è creata una proposta **Data**(date): indica la data esatta in cui è creata una proposta **Stato**(stringa): indica la stato attuale di una proposta ('inviato', 'non inviato', 'accettato', 'rifiutato')
Versione	Entità che rappresenta la versione di una frase	**num_versione**(intero): attributo contenente l'informazione di versione di una frase, associata al numero versione della pagina **Testo**(stringa): contenuto frase
Modifica	Entità che contiene l'informazione di una versione proposta di una frase	**id_modifica**(serial): identifica univocamente ogni modifica

Continua nella pagina successiva

Tabella 1 – Continuato dalla pagina precedente

Entità	**Descrizione**	**Attributi**
		Frase_proposta(stringa):contenuto della frase proposta
Collegamento	Entità che assegna ad una frase un collegamento ad una pagina	**id_collegamento**(serial): attributo univoco

3.2 Dizionario delle Associazioni

Associazione	**Descrizione**	**Tipologia**
Effettua	Associazione Uno-a-Molti	Una proposta effettua una o più modifiche, la modifica è di una sola proposta.
Genera	Associazione Uno-a-Molti	Un utente può generare più proposte, una proposta è generata da un utente
E' un	Associazione Uno-a-Uno	Un utente può essere un autore.
Crea	Associazione Uno-a-Molti	Un autore può creare più pagine, una pagina ha un solo autore.
Richiede modifica	Associazione Uno-a-Molti	Un proposte richiede la modifica di una pagina, una pagina può ricevere più proposte.
Contiene	Associazione Uno-a-Molti	Una pagina contiene delle frasi, una frase è contenuta in una pagina.
Seleziona	Associazione Uno-a-Molti	Una modifica seleziona una frase, una frase può essere selezionata da più modifiche
Ha	Associazione Uno-a-Molti	Una frase nel tempo può avere più versione, una singola versione è di una frase.
Possiede	Associazione Uno-a-Uno	Una frase può possedere un collegamento, mentre un collegamento è di una frase
Associa	Associazione Uno-a-Molti	Un collegamento associa una singola pagina, mentre una pagina può essere associata a più collegamenti

4 Modello logico

Terminata la ristrutturazione del modello concettuale, in questa fase avviene il mapping tra entità e associazioni.

4.1 Entità

Ogni entità sarà trasformata in relazione, così com'è, con gli stessi attributi.

4.2 Associazioni

- **Autore "è un" Utente**: Ogni autore che è presente nel nostro DB è automaticamente un utente; infatti, esso oltre a poter creare le pagine ha le stesse opportunità concessa ad utente.
- **Utente "genera" Proposta**: Ogni utente può effettuare zero o più proposte di modifica a frasi di una pagina creata da un Autore, andando così a generare la proposta che sarà poi valutata dall'autore.
- **Proposta "richiede modifica" Pagina**: Ogni pagina può avere più proposte di modifica del contenuto delle sue frasi.
- **Autore "crea" Pagina**: Una pagina viene creata da un autore, un autore a sua volta può creare una o più pagine.
- **Proposta "effettua" Modifica**: Generata la proposta si vanno a selezionare le modifiche sulle frasi, andando a proporre il nuovo testo delle frasi, creando così istanze per la tabella modifica.
- **Modifica "seleziona" Frase**: Ogni modifica è relativa al testo di una frase, andando a proporne uno nuovo.
- **Frase "ha" Versione**: Le frasi possono avere una o più versioni, a seconda delle modifiche che vengono accettate su di esse.
- **Pagina "contiene" Frase**: Le pagine sono composte da una o più frasi che unite vanno a formare il contenuto vero è proprio della pagina.
- **Frase "possiede" Collegamento**: Le frasi di una determinata pagina a loro volta possono avere dei collegamenti che si riferiscono ad altre pagine esistenti.
- **Collegamento "associa" Pagina**: Come già accennato il collegamento ad una frase di una pagina permette l'associazione ad un'altra pagina esistente nel nostro DB.

4.3 Schema Logico

- **UTENTE** (matricola, nome, cognome, data_nascita,)
- **AUTORE** (username, matricola↑)
  - **AUTORE**.matricola → **UTENTE**.matricola
- **PAGINA** (Titolo, data_creazione, ora_creazione, data_aggiornamento, ora_aggiornamento, username↑,matricola↑)
  - **PAGINA**.username → **AUTORE**.username
  - **PAGINA**.matricola → **AUTORE**.matricola
- **FRASE** (id_frase, ordine, titolo ↑)
  - **FRASE**.titolo → **PAGINA**.titolo
- **VERSIONE** (num_versione,id_frase ↑, testo)
  - **VERSIONE**.id_frase → **FRASE**.id_frase
- **PROPOSTA** (id_proposta, data, ora, stato,titolo ↑, username ↑, matricola_autore ↑, matricola ↑)
  - **PROPOSTA**.titolo → **PAGINA**.titolo
  - **PROPOSTA**.username → **AUTORE**.username
  - **PROPOSTA**.matricola_autore → **AUTORE**.matricola
  - **PROPOSTA**.matricola → **UTENTE**.matricola
- **MODIFICA** (id_modifica, frase_proposta, id_frase ↑, id_proposta ↑)
  - **MODIFICA**.id_frase → **FRASE**.id_frase
  - **MODIFICA**.id_proposta → **PROPOSTA**.id_proposta
- **COLLEGAMENTO** id_collegamento, id_frase↑, titolo ↑)
  - **COLLEGAMENTO**.titolo → **PAGINA**.titolo
  - **COLLEGAMENTO**.id_frase→ **FRASE**.id_frase

5 Tabelle database

- **UTENTE**

```
        CREATE TABLE Utente(
    Matricola CHAR(9) PRIMARY KEY,
    Nome VARCHAR NOT NULL,
    Cognome VARCHAR NOT NULL,
    Data_Nascita DATE
);
```

- **AUTORE**

```
  CREATE TABLE Autore (
    username VARCHAR UNIQUE NOT NULL,
    Matricola VARCHAR UNIQUE NOT NULL,
    Password VARCHAR NOT NULL,

    CONSTRAINT autore_fk FOREIGN KEY (Matricola) REFERENCES Utente(Matricola)
    ON UPDATE CASCADE ON DELETE CASCADE,

    CONSTRAINT autore_pk PRIMARY KEY(username,Matricola)
);
```

- **PAGINA**

```
   CREATE TABLE Pagina (
    Titolo VARCHAR PRIMARY KEY,
    Data_creazione DATE DEFAULT CURRENT_DATE,
    Ora_creazione TIME DEFAULT CURRENT_TIME,
    Data_aggiornamento DATE DEFAULT CURRENT_DATE,
    Ora_aggiornamento TIME DEFAULT CURRENT_TIME,
    Matricola VARCHAR NOT NULL,
    username VARCHAR NOT NULL,

    CONSTRAINT pagina_fk1 FOREIGN KEY (Matricola, Username)
    REFERENCES Autore(Matricola, Username)
    ON DELETE CASCADE ON UPDATE CASCADE,

);
```

- **FRASE**

```
    CREATE TABLE Frase (
    id_Frase SERIAL PRIMARY KEY,
    titolo varchar NOT NULL,
    ordine int,
    CONSTRAINT frase_fk FOREIGN KEY (titolo) REFERENCES pagina(titolo)
    ON UPDATE CASCADE ON DELETE CASCADE
);
```

- **VERSIONE**

```
   CREATE TABLE Versione(
        testo VARCHAR,
        num_versione int NOT NULL,
        id_frase int NOT NULL,

        CONSTRAINT versione_fk FOREIGN KEY (id_frase) REFERENCES Frase(id_frase)
        ON UPDATE CASCADE ON DELETE CASCADE,

        CONSTRAINT versione_pk PRIMARY KEY(id_frase, num_versione)
);
```

- **PROPOSTA**

```
   CREATE TYPE Stato_proposta AS ENUM ('non inviato', 'inviato', 'accettato', 'rifiutato');
CREATE TABLE Proposta (
    Id_Proposta SERIAL PRIMARY KEY,
    Data DATE DEFAULT CURRENT_DATE,
    Ora TIME DEFAULT CURRENT_TIME,
    Stato STATO_PROPOSTA DEFAULT 'non inviato',
    Titolo VARCHAR NOT NULL,
    Matricola VARCHAR NOT NULL,

    CONSTRAINT proposta_fk3 FOREIGN KEY (Matricola) REFERENCES Utente(Matricola)
    ON DELETE SET NULL ON UPDATE CASCADE,

    CONSTRAINT proposta_fk4 FOREIGN KEY (Titolo) REFERENCES Pagina(Titolo)
    ON UPDATE CASCADE
);
```

Nota: È stato definito un tipo che può assumere 4 valori distinti, identificando così i diversi stati in cui può trovarsi una proposta. Si è scelto l'inclusione di uno stato "non inviato", consentendo agli utenti di apportare modifiche alla proposta prima di decidere di inviarla all'autore.

- **MODIFICA**

```
    CREATE TABLE Modifica (
    id_Modifica SERIAL PRIMARY KEY,
    Frase_Proposta VARCHAR,
    id_Frase INT NOT NULL,
```

```
    id_Proposta INT NOT NULL,

    CONSTRAINT modifica_fk1 FOREIGN KEY (id_Frase) REFERENCES Frase(id_Frase),

    CONSTRAINT modifica_fk2 FOREIGN KEY (id_Proposta) REFERENCES Proposta(Id_Proposta)
);
```

- **COLLEGAMENTO**

```
    CREATE TABLE Collegamento (
    id_Collegamento SERIAL PRIMARY KEY,
    id_Frase INT NOT NULL,
    Titolo VARCHAR NOT NULL,
    CONSTRAINT collegamento_fk1 FOREIGN KEY (id_Frase) REFERENCES Frase(id_Frase)
        ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT collegamento_fk2 FOREIGN KEY (Titolo) REFERENCES Pagina(Titolo)
        ON DELETE CASCADE ON UPDATE CASCADE

);
```

6 Operazioni su database

6.1 Trigger di sistema

- **before_insert_frase()**
  Trigger lanciato prima dell'inserimento di una frase, esso serve a gestire l'ordine di tale frase, inizializzandolo a 1 qualora si trattasse della prima frase inserita nella pagina, altrimenti prende l'ordine massimo delle frasi di quella pagina e lo incrementa, permettendo quindi di capire in che ordine vengono inserite le frasi.

```
1  CREATE OR REPLACE FUNCTION before_insert_frase()
2  RETURNS TRIGGER AS $$
3  DECLARE
4      ordine_max frase.ordine%TYPE;
5      num_righe INTEGER;
6  BEGIN
7   --verifica se la frase inserita    la prima di tale pagina
8      SELECT COUNT(*) INTO num_righe
9      FROM Frase
10     WHERE titolo = NEW.titolo;
11
12     IF num_righe = 0 THEN
13         NEW.Ordine := 1;
14     ELSE
15         SELECT MAX(Ordine) INTO ordine_max  --seleziona massimo valore di ordine
16         FROM Frase
17         WHERE titolo = NEW.titolo;
18
19         NEW.Ordine := ordine_max + 1;
20     END IF;
21
22     RETURN NEW;
23 END;
24 $$ LANGUAGE plpgsql;
25
26
27 CREATE TRIGGER before_insert_frase
28 BEFORE INSERT ON frase
29 FOR EACH ROW
30 EXECUTE FUNCTION before_insert_frase();
```

- **before_modifica()**
  Trigger lanciato prima dell'inserimento nella tabella modifica, esso verifica che lo stato della proposta associata sia 'non inviato', altrimenti lancia l'eccezione rendendo noto che tale proposta è già stata inviata o elaborata.

```
1   CREATE OR REPLACE FUNCTION before_modifica()
2  RETURNS TRIGGER AS $$
3  DECLARE
4      stato_proposta_attuale proposta.stato%TYPE;
5      var INTEGER;
6  BEGIN
7      SELECT stato INTO stato_proposta_attuale --ricava stato proposta
8      FROM proposta
9      WHERE id_proposta = NEW.id_proposta;
10
11     SELECT COUNT(*) INTO var
12     FROM modifica
13     WHERE id_frase = NEW.id_frase AND id_proposta = NEW.id_proposta;
14
15     IF stato_proposta_attuale <> 'non␣inviato' THEN
16 RAISE EXCEPTION 'Proposta␣gi ␣inviata␣all␣autore,␣non␣pu ␣essere␣ulteriormente␣modificata'
17         RETURN OLD;
18     ELSIF var > 0 THEN
19         UPDATE modifica
20         SET frase_proposta = NEW.frase_proposta
21         WHERE id_frase = NEW.id_frase AND id_proposta = NEW.id_proposta;
22         RETURN OLD;
23     ELSE
24         RETURN NEW;
25     END IF;
26 END;
27 $$ LANGUAGE PLPGSQL;
28
29 CREATE TRIGGER before_modifica
30 BEFORE INSERT ON modifica
31 FOR EACH ROW
32 EXECUTE FUNCTION before_modifica();
```

- **before_delete_modifica** Trigger lanciato prima dell'aggiornamento o eliminazione della modifica, esso controlla lo stato della proposta verificando che sia 'non inviato', altrimenti lancia l'eccezione rendendo noto che tale proposta è già stata inviata o elaborata.

```
1      CREATE OR REPLACE FUNCTION before_delete_modifica()
2  RETURNS TRIGGER AS $$
3  DECLARE
4      stato_proposta_attuale proposta.stato%TYPE;
5  BEGIN
6      SELECT stato INTO stato_proposta_attuale  --ricava stato proposta
7      FROM proposta
8      WHERE id_proposta = OLD.id_proposta;
9
10     IF stato_proposta_attuale <> 'non␣inviato' THEN
11 RAISE EXCEPTION 'Proposta␣gi ␣inviata␣all␣autore,␣non␣pu ␣essere␣ulteriormente␣modificata'
12         RETURN OLD;
13     ELSE
14         RETURN NEW;
15     END IF;
16 END;
17 $$ LANGUAGE PLPGSQL;
18
19 CREATE TRIGGER before_delete_modifica
20 BEFORE DELETE OR UPDATE ON modifica
21 FOR EACH ROW
22 EXECUTE FUNCTION before_delete_modifica();
```

- **after_update_proposta()**
  Trigger lanciato prima dell'aggiornamento di una proposta.
  - Se lo stato della proposta passa da 'inviato' ad 'accettato', seleziono le modifiche richieste e vengono apportate creando una nuova versione (unica per tutte le modifiche).Inoltre eliminiamo dal database tutte le proposte di quella pagina, ormai obsolete, non inviate.
  - Se lo stato passa da 'inviato' a 'rifiutato', le frasi restano invariate
  - Se lo stato passa da 'non inviato' a 'inviato', si verifica se la proposta ha delle modifiche altrimenti lancia eccezione. Controlla se la proposta è da parte dell'autore stesso della pagina, se così fosse vengono rifiutate in ordine dalla meno recente tutte le proposte e in automatico accettata quella dell'autore
  - In tutti gli altri casi, la modifica di stato non è consentita

```
1     CREATE OR REPLACE FUNCTION after_update_proposta()
2 RETURNS TRIGGER AS $$
3 DECLARE
4     var INTEGER;
5     num_versione_frase INTEGER;
6     id_frase_selezionata frase.id_frase%TYPE;
7     testo_frase VARCHAR;
8         id_proposta_sel proposta.id_proposta%TYPE;
9  --cursore che prende tutte le modifiche di una determinata proposta
10    frasi_modifica CURSOR FOR
11        SELECT id_frase, frase_proposta
12        FROM modifica
13        WHERE id_proposta = OLD.id_proposta;
14
15    --cursore che prende tutte le proposte non inviate di tale pagina in ordine dalla meno r
16    proposte CURSOR FOR
17        SELECT id_proposta
18        FROM proposta
19        WHERE titolo=NEW.titolo AND stato='inviato' AND matricola<>NEW.matricola
20        ORDER BY(data,ora);
21 BEGIN
22    IF NEW.stato='accettato' AND OLD.stato='non inviato' THEN
23        RAISE EXCEPTION 'La proposta ancora non   stata
24 inviata, quindi non pu essere gestita';
25        RETURN OLD;
26    ELSIF NEW.stato='accettato' AND OLD.stato='inviato'  THEN
27        OPEN frasi_modifica;
28        FETCH frasi_modifica INTO id_frase_selezionata, testo_frase;
29        --elimino le proposte non inviate, ormai obsolete
30        DELETE FROM Proposta WHERE titolo=OLD.titolo AND stato='non inviato';
31        --seleziono massimo valore di versione delle frasi di tale pagina
32        SELECT MAX(num_versione) INTO num_versione_frase
33        FROM Versione NATURAL INNER JOIN Frase
34        WHERE titolo IN(SELECT titolo FROM frase WHERE id_frase=id_frase_selezionata);
35        INSERT INTO versione(id_frase, testo, num_versione)
36        VALUES(id_frase_selezionata, testo_frase, num_versione_frase+1);
```

```
37      --scorro tutte le modifiche e inserisco nuove versioni
38          LOOP
39              FETCH frasi_modifica INTO id_frase_selezionata, testo_frase;
40              EXIT WHEN NOT FOUND;
41              INSERT INTO versione(id_frase, testo, num_versione) VALUES(id_frase_selezionata,
42          END LOOP;
43
44          CLOSE frasi_modifica;
45          RETURN NEW;
46       ELSIF NEW.stato='rifiutato' AND OLD.stato='inviato' THEN
47          RETURN NEW;
48       ELSIF NEW.stato='inviato' AND OLD.stato='non inviato' THEN
49          SELECT COUNT(*) INTO var  --controllo se ci sono delle modifiche richieste
50          FROM modifica
51          WHERE id_proposta=new.id_proposta;
52          IF var=0 THEN
53                  RAISE EXCEPTION 'Proposta senza nessuna modifica proposta';
54                  RETURN OLD;
55          END IF;
56          SELECT COUNT(*) INTO var   --controllo se ci sono proposte inviate, non elaborate
57          FROM Pagina
58           WHERE titolo = NEW.titolo AND matricola = NEW.matricola;
59          IF var> 0 THEN
60        --scorro tali proposte e le rifiuto in ordine dalla meno recente.
61                  OPEN proposte;
62                  LOOP
63                  FETCH proposte INTO id_proposta_sel;
64                  EXIT WHEN NOT FOUND;
65                  UPDATE proposta SET stato='rifiutato' WHERE id_proposta=id_proposta_sel;
66                  END LOOP;
67                  CLOSE proposte;
68          UPDATE proposta SET stato = 'accettato' WHERE id_proposta = NEW.id_proposta;
69          END IF;
70          RETURN NEW;
71       ELSE
72          RETURN OLD;
73       END IF;
74  END;
75  $$ LANGUAGE PLPGSQL;
76
77
78  CREATE TRIGGER after_update_proposta
79  AFTER UPDATE ON PROPOSTA
80  FOR EACH ROW
81  EXECUTE FUNCTION after_update_proposta();
```

- **before_gestione_proposta()**
  Trigger lanciato prima dell'aggiornamento di una proposta, esso controlla la data e l'orario in cui vengono effettuate le proposte in modo da selezionare le meno recenti, e lanciare un eccezione qualora si volessero valutare altre proposte più recenti.

```
1    CREATE OR REPLACE FUNCTION before_gestione_proposta() RETURNS TRIGGER AS $$
2  BEGIN
3      IF (old.stato='inviato' AND (NEW.stato='accettato' OR NEW.stato='rifiutato')) THEN
4
5          IF EXISTS (
6          SELECT 1
7          FROM proposta
8          WHERE titolo = NEW.Titolo
9              AND (data < NEW.Data OR (Data = NEW.Data AND Ora < NEW.Ora))
10             AND stato = 'inviato'
11             AND id_proposta <> OLD.id_proposta
12     ) THEN
13                 RAISE EXCEPTION 'Ci␣sono␣altre␣proposte
14 ␣␣ancora␣da␣dover␣gestire,␣meno␣recenti';
15         RETURN OLD;
16     ELSE
17         RETURN NEW;
18     END IF;
19 END IF;
20     RETURN NEW;
21 END;
22 $$ LANGUAGE PLPGSQL;
23
24
25 CREATE TRIGGER before_gestione_proposta
26 BEFORE UPDATE ON PROPOSTA
27 FOR EACH ROW
28 EXECUTE FUNCTION before_gestione_proposta();
```

6.2 Procedure di popolazione

- Procedura per la creazione di una pagina

```
CREATE OR REPLACE PROCEDURE crea_pagina(titolo_pagina VARCHAR,
matricola_autore CHAR(9),
username_autore VARCHAR)
AS $$
BEGIN
    INSERT INTO pagina(titolo, username, matricola) VALUES (titolo_pagina, username_autore,
END;
$$ LANGUAGE PLPGSQL;
```

- Procedura per l'inserimento di un utente

```
    CREATE OR REPLACE PROCEDURE aggiungi_utente(nome_utente VARCHAR, cognome_utente VARCHAR,
    data_nascita_utente DATE, matricola_utente CHAR(9))
AS $$
BEGIN
INSERT INTO utente(nome,cognome,data_nascita, matricola)
VALUES(nome_utente, cognome_utente, data_nascita_utente, matricola_utente);
END
$$ LANGUAGE PLPGSQL;
```

- Procedura per l'aggiunta di un autore. Prima di completare l'inserimento è chiamata la procedura che inserisce un utente in quanto un autore è tale solo se sia anche utente.

```
CREATE OR REPLACE PROCEDURE aggiungi_autore(nome_utente VARCHAR, cognome_utente VARCHAR,
data_nascita_utente DATE, matricola_utente CHAR(9),
username_autore VARCHAR, password_autore VARCHAR)
AS $$
BEGIN
call aggiungi_utente(nome_utente, cognome_utente, data_nascita_utente, matricola_utente);
INSERT INTO autore(username, matricola, password)
VALUES(username_autore, matricola_utente, password_autore);
END
$$ LANGUAGE PLPGSQL;
```

- Procedura per l'inserimento delle frasi. Inizializza a 1 la versione quando non sono presenti altre versioni ¿1 delle frasi di tale pagina, qualora siano presenti avviene un incremento di 1 al valore massimo presente.

```
  CREATE OR REPLACE PROCEDURE inserisci_frase(titolo_pagina VARCHAR, testo_frase VARCHAR)
  AS $$
DECLARE
    id_frase_nuova INTEGER;
    var INTEGER;
    num_versione_frase INTEGER :=1;
BEGIN

    INSERT INTO frase(titolo) VALUES(titolo_pagina) RETURNING id_frase INTO id_frase_nuova;
    SELECT MAX(num_versione) INTO var
    FROM Versione NATURAL INNER JOIN Frase
    WHERE titolo = titolo_Pagina;
    IF var>1 THEN
        num_versione_frase=var+1;
    END IF;
    INSERT INTO versione(id_frase, testo, num_versione) VALUES(id_frase_nuova, testo_frase,
END;
$$ LANGUAGE PLPGSQL;
```

- procedura che permette di creare una modifica da associare ad una proposta già esistente, selezionando una frase attraverso la sua posizione

```
 CREATE OR REPLACE PROCEDURE proponi_modifica(
 posizione INTEGER,
 id_proposta_modifica INTEGER,
 modifica_proposta VARCHAR)
AS $$
DECLARE
    titolo_pagina pagina.titolo%TYPE;
    id_frase_selezionata frase.id_frase%TYPE;
BEGIN
--ricava il titolo della pagina di cui si propone la modifica
    SELECT titolo INTO titolo_pagina
    FROM proposta
    WHERE id_proposta = id_proposta_modifica;
--ricava l'id_frase della frase selezionata
    SELECT f.id_frase INTO id_frase_selezionata
    FROM frase f JOIN pagina p ON f.titolo = p.titolo
    WHERE p.titolo = titolo_pagina AND f.ordine=posizione;

    INSERT INTO modifica(frase_proposta, id_proposta, id_frase)
    VALUES(modifica_proposta, id_proposta_modifica, id_frase_selezionata);
END;
$$ LANGUAGE PLPGSQL;
```

- Procedura per l'inserimento di un collegamento, funziona tramite una pagina sorgente dalla quale viene selezionata la frase dove applicare il collegamento, presa tramite l'ordine e la pagina di destinazione, ovvero la pagina dove porta tale collegamento.

```
1      CREATE OR REPLACE PROCEDURE inserisci_collegamento(titolo_pagina_sorgente VARCHAR,
2      titolo_pagina_destinazione VARCHAR,
3      ordine_frase INTEGER)
4
5  AS $$
6
7  DECLARE
8
9      id_frase_collegamento Frase.id_frase%TYPE;
10
11 BEGIN
```

6.3 Funzioni

Richieste dalla traccia abbiamo sviluppato quattro funzioni che ci restituiscono determinate informazioni richieste.

- **Versione corrente e collegamenti associati di una determinata pagina** Funzione che permette attraverso il titolo di una pagina di visualizzare la versione corrente di tale pagina e i collegamenti alle eventuali altre pagina, permettendo inoltre di visualizzare la frase 'NO LINK' laddove le frasi non posseggano collegamenti verso altre pagine.

```
1      CREATE OR REPLACE FUNCTION visualizza_pagina(titolo_pagina VARCHAR)
2  RETURNS TABLE (
3      testo_frase VARCHAR,
4      Titolo_collegamento_destinazione VARCHAR
5  ) AS $$
6
7  BEGIN
8
9
10     RETURN QUERY
11     SELECT v.testo AS testo_frase,
12            CASE
13                WHEN c.Titolo IS NOT NULL THEN c.Titolo
14                ELSE 'NO LINK'
15            END AS Titolo_collegamento_destinazione
16     FROM Versione v
17     JOIN Frase f ON v.id_frase = f.id_Frase
18     LEFT JOIN Collegamento c ON f.id_frase = c.id_frase
19     WHERE f.titolo = titolo_pagina
20         AND v.num_versione = (
21             SELECT MAX(v2.num_versione)
22             FROM Versione v2
23             WHERE v2.id_frase = f.id_frase
24         )
25     ORDER BY f.ordine;
26
27 END;
28 $$ LANGUAGE plpgsql;
```

- **Recupero di versione corrente, affiancato dalla versione proposta** Funzione che dato un autore e una pagina verifica se tale pagina è stata creata dall'autore, se non si verifica ciò è lanciate un eccezione. Se la pagina è stata creata dall'autore permette di visualizzare la versione corrente di quella pagina e le eventuali modifiche proposte a tali frasi.

```
1      CREATE OR REPLACE FUNCTION Visualizza_Versione_E_Proposte(
2      username_autore VARCHAR,
3      matricola_autore VARCHAR,
4      titolo_pagina VARCHAR
5  )
6  RETURNS TABLE (
7      ordine INTEGER,
```

```
8       versione_corrente VARCHAR,
9       versione_proposta VARCHAR
10  ) AS $$
11  DECLARE
12
13      id_proposta_recente INTEGER;
14      autore_count INT;
15  BEGIN
16      -- Controlla se l'autore    associato alla pagina
17      SELECT COUNT(*)
18      INTO autore_count
19      FROM Pagina
20      WHERE Titolo = titolo_pagina
21          AND Matricola = matricola_autore
22          AND username = username_autore;
23
24      -- Se l'autore non    associato alla pagina, lancia un'eccezione
25      IF autore_count = 0 THEN
26          RAISE EXCEPTION 'autore␣specificato␣non␣associato␣a␣questa␣pagina.';
27      ELSE
28          SELECT id_proposta INTO id_proposta_recente
29          FROM proposta
30          WHERE titolo=titolo_pagina AND stato='inviato';
31
32          RETURN QUERY
33
34   --ricavo le frasi che hanno ricevuto una proposta di modifica
35  SELECT f.ordine, v.testo AS versione_corrente, m.frase_proposta AS versione_proposta
36  FROM frase f
37  JOIN versione v ON f.id_frase = v.id_frase
38  JOIN modifica m ON f.id_frase = m.id_frase
39  WHERE m.id_proposta = id_proposta_recente
40      AND f.titolo = titolo_pagina
41      AND v.num_versione IN (
42          SELECT MAX(v1.num_versione)
43          FROM versione v1
44          WHERE v1.id_frase = f.id_frase
45      )
46
47  UNION
48
49  SELECT f.ordine, v.testo AS versione_corrente, v.testo AS versione_proposta
    --ricavo le frasi che non hanno ricevuto modifica
50  FROM frase f
51  JOIN versione v ON f.id_frase = v.id_frase
52  WHERE f.titolo = titolo_pagina
53      AND v.num_versione IN (
54          SELECT MAX(v1.num_versione)
55          FROM versione v1
56          WHERE v1.id_frase = f.id_frase
```

```
57      )
58      AND (f.ordine, v.testo) NOT IN (
59          SELECT f1.ordine, v1.testo
60          FROM frase f1
61          JOIN versione v1 ON f1.id_frase = v1.id_frase
62          JOIN modifica m1 ON f1.id_frase = m1.id_frase
63          WHERE m1.id_proposta = id_proposta_recente
64              AND f1.titolo = titolo_pagina
65              AND v1.num_versione IN (
66                  SELECT MAX(v2.num_versione)
67                  FROM versione v2
68                  WHERE v2.id_frase = f1.id_frase
69              )
70      )
71  ORDER BY ordine;
72
73      END IF;
74  END;
75  $$ LANGUAGE plpgsql;
```

- **Storico di pagine create o di cui esiste una proposta** Funzioneche permette ad un autore di visualizzare tutte le pagine da lui create, o le pagine dove egli ha proposto una modifica con il numero di versioni che esse hanno. Mentre la funzione visualizza_pagina_autore permette di visualizzare una determinata versione indicata di una determianta pagi a.

```
1         CREATE OR REPLACE FUNCTION visualizza_storico_autore(username_autore VARCHAR,
2         matricola_autore VARCHAR)
3  RETURNS TABLE (
4      tipo_pagina TEXT,
5      titolo_pagina VARCHAR,
6      num_versione INT
7  ) AS $$
8  BEGIN
9      RETURN QUERY
10     SELECT
11         'Creata' AS tipo_pagina,
12         p.Titolo AS titolo_pagina,
13         MAX(v.num_versione) AS num_versione
14     FROM Pagina p
15     JOIN Frase f ON p.titolo = f.titolo
16     JOIN Versione v ON f.id_Frase = v.id_Frase
17     WHERE p.username = username_autore AND p.matricola = matricola_autore
18     GROUP BY p.Titolo
19
20     UNION
21
22     SELECT
23         'Proposta' AS tipo_pagina,
24         p.Titolo AS titolo_pagina,
25         MAX(v.num_versione) AS num_versione
26     FROM Pagina p
27     JOIN Frase f ON p.titolo = f.titolo
28     JOIN Versione v ON f.id_Frase = v.id_Frase
29     JOIN Modifica m ON m.id_frase = f.id_frase
30     JOIN Proposta pr ON m.id_proposta = pr.id_proposta
31     WHERE pr.Matricola = matricola_autore AND p.matricola != matricola_autore
32     AND p.username != username_autore
33     GROUP BY p.Titolo
34     ORDER BY titolo_pagina;
35
36 END;
37 $$ LANGUAGE plpgsql;
38
39
40 ------------------------------------------------------------------------------------
41
42
43
44 CREATE OR REPLACE FUNCTION visualizza_pagina_autore(
45     IN username_autore VARCHAR,
46     IN matricola_autore CHAR(9),
```

```
47      IN titolo_pagina VARCHAR,
48      IN numero_versione INT
49  )
50  RETURNS TABLE (
51      titolo_pag VARCHAR,
52      testo_frase VARCHAR,
53      n_versione INT
54  )
55  AS $$
56  DECLARE
57      tipo_pagina TEXT;
58  BEGIN
59      tipo_pagina := (
60          SELECT 'Creata'
61          FROM Pagina
62          WHERE username = username_autore AND matricola = matricola_autore
63          AND Titolo = titolo_pagina
64          UNION
65          SELECT 'Proposta'
66          FROM Modifica m
67          JOIN Proposta pr ON m.id_proposta = pr.id_proposta
68          WHERE pr.Matricola = matricola_autore AND pr.Titolo = titolo_pagina
69          LIMIT 1
70      );
71
72      IF tipo_pagina IS NULL THEN
73          RAISE EXCEPTION 'Non␣hai␣accesso␣a␣questa␣pagina';
74      ELSE
75          RETURN QUERY
76          SELECT
77              p.Titolo AS titolo_pag,
78              v.testo AS testo_frase,
79              v.num_versione
80          FROM versione v
81          JOIN Frase f ON v.id_frase = f.id_frase
82          JOIN Pagina p ON f.titolo = p.titolo
83          WHERE p.Titolo = titolo_pagina AND v.num_versione <= numero_versione
84          AND v.num_versione = (
85              SELECT MAX(num_versione)
86              FROM Versione
87              WHERE id_frase = f.id_frase AND num_versione <= numero_versione
88          );
89      END IF;
90  END;
91  $$ LANGUAGE plpgsql;
```

7 Popolazione

- **Utente**

```
CALL aggiungi_utente('Paolo', 'Sinardi', '1980-04-12', '10101010');
CALL aggiungi_utente('Teresa', 'Leopardi', '1995-01-10', '36144771');
```

- **Autore** Note:questa procedura effettua anche la creazione degli utenti

```
CALL aggiungi_autore('Mario', 'Rossi', '1990-05-15', '123456789', 'mario01', 'password1');
CALL aggiungi_autore('Anna', 'Verdi', '1985-10-20', '987654321', 'anna84', 'password2');
CALL aggiungi_autore('Luca', 'Bianchi', '1993-03-28', '456789123', 'luca93', 'password3');
CALL aggiungi_autore('Giulia', 'Neri', '1998-07-10', '789123456', 'giulia98', 'password4');
CALL aggiungi_autore('Marco', 'Gialli', '1996-12-05', '654321987', 'marco96', 'password5');
```

- **Pagina**

```
CALL crea_pagina('Informatica', '123456789', 'mario01');
CALL crea_pagina('Tecnologie del Futuro', '987654321', 'anna84');
CALL crea_pagina('Calcio', '456789123', 'luca93');
CALL crea_pagina('Viaggi nella Galassia', '789123456', 'giulia98');
CALL crea_pagina('Economia Globale', '654321987', 'marco96');
```

- **Frase** Note:questa porcedura effettua anche l'inserimento nella tabella versione.

```
CALL inserisci_frase('Informatica', 'Mouse e tastiera');
CALL inserisci_frase('Informatica', 'Processori e Stampanti');
CALL inserisci_frase('Informatica', 'Tecnologie Hardwere');
CALL inserisci_frase('Calcio', 'Allenatori esperti');
CALL inserisci_frase('Calcio', 'I migliori stadi');
CALL inserisci_frase('Calcio', 'La var');
CALL inserisci_frase('Viaggi nella Galassia', 'Astronavi');
```

- **Collegamento**

```
CALL inserisci_collegamento('Informatica', 'Tecnologie del Futuro', 1);
CALL inserisci_collegamento('Informatica', 'Tecnologie del Futuro', 2);
CALL inserisci_collegamento('Informatica', 'Viaggi nella Galassia', 3);
CALL inserisci_collegamento('Calcio', 'Tecnologie del Futuro', 3);
CALL inserisci_collegamento('Viaggi nella Galassia', 'Tecnologie del Futuro', 1);
```

- **Proposta**

```
INSERT INTO Proposta (Titolo, Matricola)
VALUES
    ('Informatica', '123456789'),
    ('Informatica', '987654321'),
    ( 'Calcio', '10101010'),
    ( 'Calcio', '789123456'),
    ('Viaggi nella Galassia', '36144771');
```

- **Modifica**

```
1  CALL proponi_modifica(1, 1, 'Touch screen');
2  CALL proponi_modifica(2, 3, 'Tecnologie Softwere');
3  CALL proponi_modifica(3, 2, 'I migliori Inni');
4  CALL proponi_modifica(3, 3, 'Calcio moderno');
5  CALL proponi_modifica(3, 1, 'Shuttle');
```