Università degli Studi
di Napoli Federico II

Documentazione Sistema Di Gestione Del Ciclo Di Vita Di Una Pagina Wiki

Giuseppe Pollio, Mario Lombardo

Anno accademico 2023-2024

Indice

1 Modello Concettuale **2**
- 1.1 Analisi dei requisiti . . . 2
- 1.2 Diagramma (UML) . . . 3
- 1.3 Dizionari . . . 4
  - 1.3.1 Dizionario delle Entità . . . 4
  - 1.3.2 Dizionario delle Associazioni . . . 4
  - 1.3.3 Dizionario dei vincoli . . . 5
- 1.4 Ristrutturazione del Modello Concettuale . . . 6
  - 1.4.1 Analisi delle Ridondanze . . . 6
  - 1.4.2 Analisi delle generalizzazioni . . . 6
  - 1.4.3 Eliminazione degli attributi Multivalore o Composti . . . 6
  - 1.4.4 Analisi di Entità e Associazioni . . . 6
  - 1.4.5 Scelta degli Identificatori Primari . . . 6
  - 1.4.6 Diagramma UML Ristrutturato . . . 8
  - 1.4.7 Diagramma ER Ristrutturato . . . 9

2 Modello Logico **10**
- 2.1 Traduzione di Entità e Associazioni . . . 10
- 2.2 Relazioni . . . 10

3 Modello Fisico **11**
- 3.1 Domini . . . 11
- 3.2 Tabelle . . . 11
- 3.3 View . . . 13
- 3.4 Operazioni . . . 15
- 3.5 Trigger . . . 17

1 Modello Concettuale

1.1 Analisi dei requisiti

In questa sezione si analizza la specifica con lo scopo di definire le funzionalità che la base di dati deve soddisfare. Individueremo le entità e le associazioni del mini-word, e produrremo una prima versione del modello concettuale che sarà poi rielaborato nelle fasi successive della progettazione.

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

L'introduzione individua il mini-world da rappresentare e una prima entità: **Page** avente come attributi **title** e **creation**. Inoltre nel testo della **Page** possiamo individuare un'altra entità associata alla pagina (**Page**): **PageText** contenente come attributi la riga di testo (**text**), un collegamento ad un'altra pagina **link** tramite attributo parziale e una relazione all'entità **Page**.

Una pagina di una wiki ha un titolo e un testo. Ogni pagina è creata da un determinato autore. Il testo è composto di una sequenza di frasi. Il sistema mantiene traccia anche del giorno e ora nel quale la pagina è stata creata. Una frase può contenere anche un collegamento. Ogni collegamento è caratterizzato da una frase da cui scaturisce il collegamento e da un'altra pagina destinazione del collegamento.

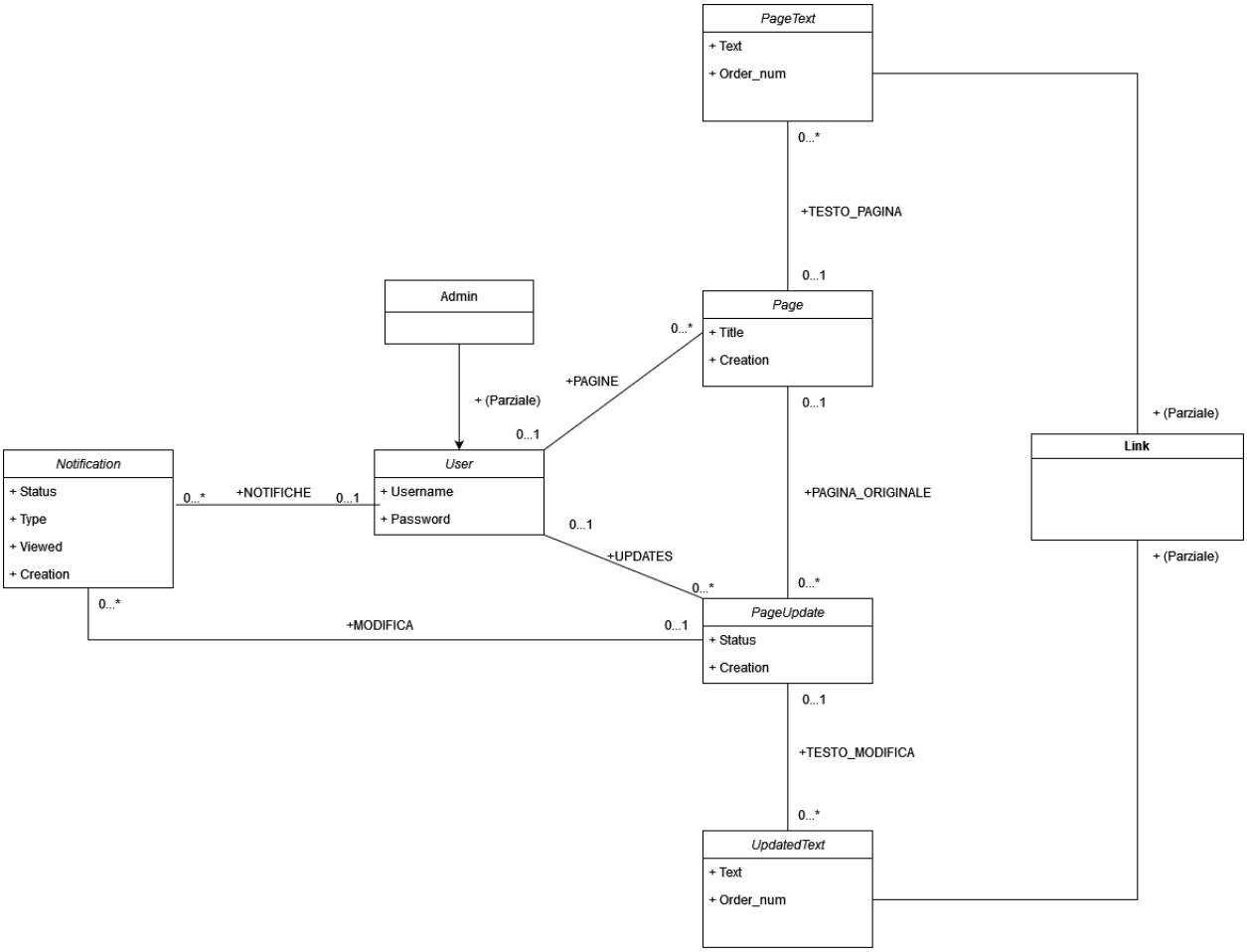
Dato che le pagine delle wiki saranno publiche oltre che modificabili, sarà necessario tenere traccia degli utenti tramite l'entità **User**, alla quale sarà possibile accederci tramite una **password** e un **username**. Di conseguenza all'entità **Page** viene inserito individuato l'autore della pagina tramite il campo **author**. Le modifiche del testo di una pagina (**Page**) saranno salvate tramite le l'entità **PageUpdate** avente attributi **status** che indica lo stato della modifica e un collegamento ad **User** che rappresenta l'autore della modifica (**author**). Il testo modificato viene individuato tramite l'entità **UpdatedText** contenente il "nuovo testo" individutato dall'attributo **text**, un eventuale **link** (collegamento) tramite attributo parziale e un collegamento all'entità **PageUpdate**, quando una modifica viene inoltrata all'autore esso invece deve essere avvisato tramite una notifica individuata dall'entità **Notification** avente come attributi **status** (stato di lettura), **type**, un collegamento a **User** per individuare il proprietario della notifica e un collegamento a **PageUpdate**.

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).

Viene in aggiunta inserito il nuovo attributo **order_num** alle entità **PageText** e **UpdatedText** che rappresenta l'ordinamento all'interno del testo della **Page**. Inoltre si farà la distinzione tra un Utente (User) e Amministratore (Admin) tramite specializzazione parziale.

1.2 Diagramma (UML)

In seguito alle considerazioni espresse nella sezione precedente, si ‘e prodotto il seguente schema concettuale espresso mediante diagramma UML:



1.3 Dizionari

1.3.1 Dizionario delle Entità

Entità	Descrizione	Attributi
User	Account Utente della wiki.	**Username**(Stringa): nome identificativo dell'account utente. **Password**(Stringa): password necessaria per accedere all'account utente.
Page	Pagina presente sulla wiki.	**Title**(Stringa): Titolo della pagina. **Creation**(Data): Data e ora di creazione della pagina.
PageText	Frase di una Pagina (**Page**).	**Text**(Stringa): Contenuto della frase. **Link**(Stringa): Collegamento della frase (interno o esterno alla wiki). **Order_num**(Intero): Ordinamento della frase all'interno del testo complessivo.
PageUpdate	Modifica proposta ad pagina da parte di un altro utente.	**Status**(Intero): Stato della richiesta di modifica.
UpdatedText	Nuovo testo proposto durante la modifica (**PageUpdate**).	**Text**(Stringa): Contenuto della frase. **Link**(Stringa): Collegamento della frase (interno o esterno alla wiki). **Order_num**(Intero): Ordinamento della frase all'interno del testo complessivo.
Notification	Notifiche di un **Utente (User)**.	**Status**(Booleano): Stato di lettura di una notifica. **Type**(Intero): Tipologia di notifica.
Admin	Specializzazione parziale di un utente, un amministratore può modificare ed eliminare le pagine di altri utenti senza dover prima inviare una notifica di accettazione delle modifiche.	
Link	Specializzazione parziale di una riga di testo che rappresentà se una riga di testo è un collegamento o meno.	

1.3.2 Dizionario delle Associazioni

Associazione	Tipologia	Descrizione
Autore Pagina	uno-a-molti	Associa ad ogni pagina (Page) un utente (User) che ne rappresenta l'autore
Autore Modifica	uno-a-molti	Associa ad ogni modifica (PageUpdate) un utente (User) che ne rappresenta l'autore
Testo Pagina	uno-a-molti	Associa ad ogni pagina (Page) tutto il testo (PageText) appartenente a quella determinata pagina.
Testo Modifica	uno-a-molti	Associa ad ogni modifica (PageUpdate) tutto il testo (UpdatedText) appartenente a quella determinata modifica.
Proprietario Notifica	uno-a-molti	Associa ad ogni notifica (Notification) un utente (User) che ne rappresenta l'autore.
PageUpdate Notifica	uno-a-molti	Associa ad ogni notifica (Notification) una modifica (PageUpdate) che ne aiuta a rappresentare il contenuto.
Page PageUpdate	uno-a-molti	Associa ad ogni Modifica proposta (PageUpdate), La pagina (Page) per cui è stata proposta.

1.3.3 Dizionario dei vincoli

Vincolo	**Tipologia**	**Descrizione**
Di Leggibilità Testuale	Intrarelazionale	Il testo di una pagina, il titolo di una pagina e il testo di una modifica devono avere lunghezza non nulla.
Di Autore	Interrelazionale	Una pagina e una modifica devono sempre avere un autore. Se un utente autore di una pagina viene eliminato, anche la pagina viene eliminata.
Di Privacy	interrelazionale	Un utente deve avere un modo obbligatorio una password.
Di Sicurezza	interrelazionale	Una pagina può essere modificata direttamente solo nel caso in cui, l'utente che effettua la modifica è l'autore della pagina oppure un gestore della wiki.
Di Gestione	interrelazionale	Se esistono utenti almeno uno di essi dovrà essere il gestore della wiki (amministratore).

1.4 Ristrutturazione del Modello Concettuale

Prima di procedere con lo schema logico, è fondamentale riorganizzare il diagramma delle classi al fine di agevolare la trasformazione in schema logico. Questo processo mira a ottimizzare il progetto, rimuovere le generalizzazioni, eliminare attributi multivalore, eliminare attributi strutturati, consolidare o dividere le entità figlie, e selezionare gli identificatori appropriati per le nostre entità quando necessario.

1.4.1 Analisi delle Ridondanze

Una ridondanza è un dato che è già presente nella base di dati o può essere derivato da altri dati. Nel modello concettuale originale non sono presenti ridondanze

1.4.2 Analisi delle generalizzazioni

La specializzazione è il processo di determinazione di sottoclassi per una data entità. La generalizzazione è il suo concetto complementare.
Nel modello concettuale attuale ci sono due generalizzazioni:

La sottoclasse **Admin** dove non sono presenti attributi ma serve solo a definire i permessi di un utente. Nella ristrutturazione essa verrà accorpata all'entità genitore, come flag booleana.

La sottoclasse Link, utilizzata per capire se una riga di testo era anche un link invece verrà tolta del tutto in quanto i collegamenti verrano salvati dal programma seguendo la formattazione **href='link'**, facendo in questo modo il programma capirà in automatico quando un testo rappresenta un link e lo andrà a trattare come tale.

1.4.3 Eliminazione degli attributi Multivalore o Composti

Un attributo è multivalore se può essere associato ad un numero variabile di valori dello stesso dominio. Un attributo è composto se può essere suddiviso in sottoparti ognuna dotata di dominio. Nel modello concettuale non sono presenti attributi multivalore o attributi composti.

1.4.4 Analisi di Entità e Associazioni

Non si è ritenuto opportuno scomporre o accorpare entità. Tuttavia all'entità **PageUpdate** viene aggiunto l'attributo **old_text** che rappresenta il testo di una pagina prima che venga modificato inoltre viene aggiunto l'attributo **author** all'entità **PageText** che rappresenta chi ha scritto quel testo, queste modifiche vengono fatte in modo da poter tenere traccia della "storia" di una pagina.

Nel modello attuale si è evitato di creare relazioni di composizione.

1.4.5 Scelta degli Identificatori Primari

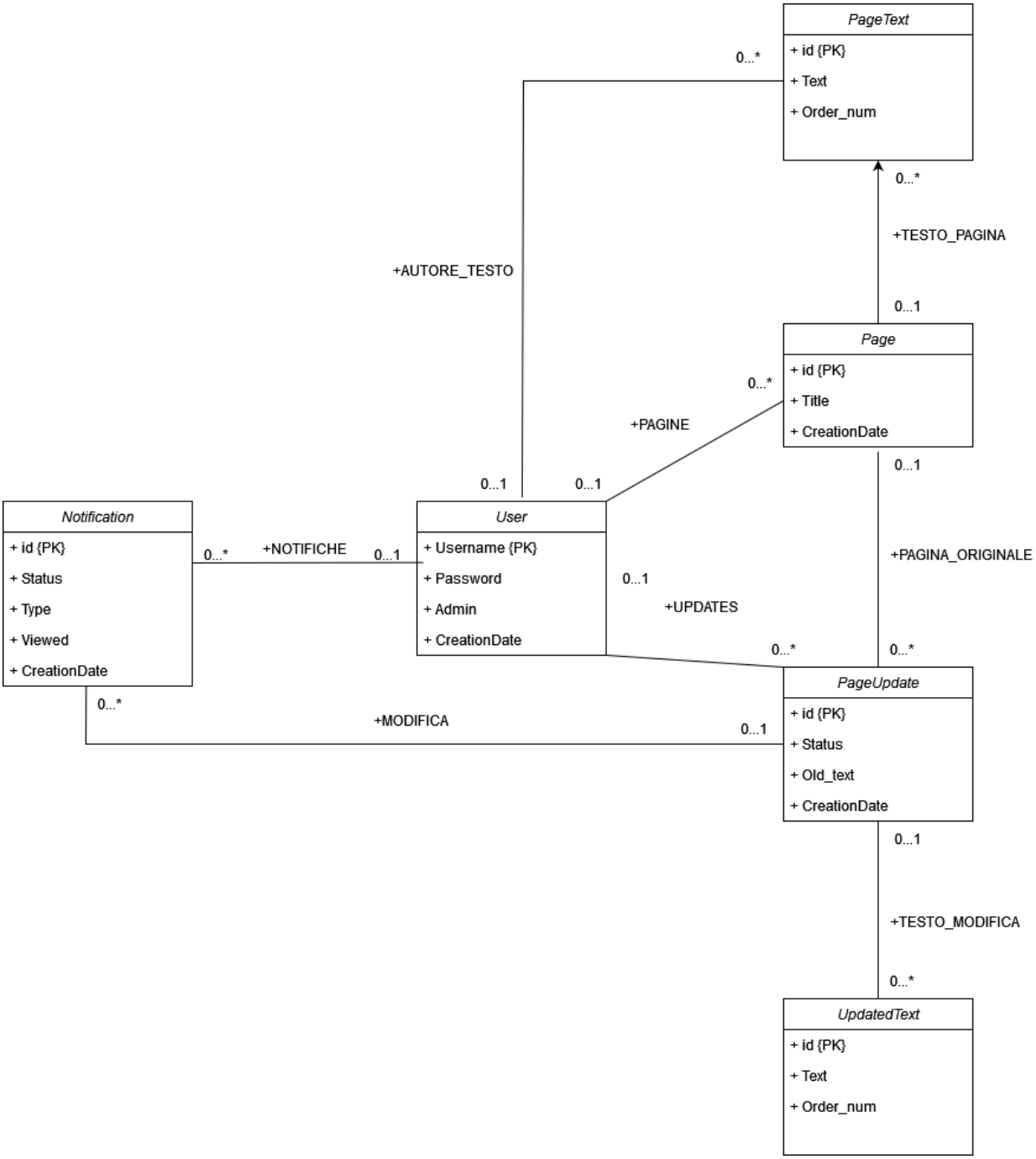
Un identificatore o chiave minimale è un insieme di attributi che identificano univocamente un'entità. È possibile che un'entità sia dotata di molteplici identificatori, fra i quali ne sceglieremo uno principale che agirà da chiave primaria.

- **User**: Viene scelto **username** come identificatore primario.
- **Notification**: Viene introdotto l'identificativo primario **id**.
- **Page**: Viene introdotto l'identificativo primario **id**.

- **PageText**: Viene introdotto l'identificativo primario **id**.
- **PageUpdate**: Viene introdotto l'identificativo primario **id**.
- **UpdatedText**: Viene introdotto l'identificativo primario **id**.

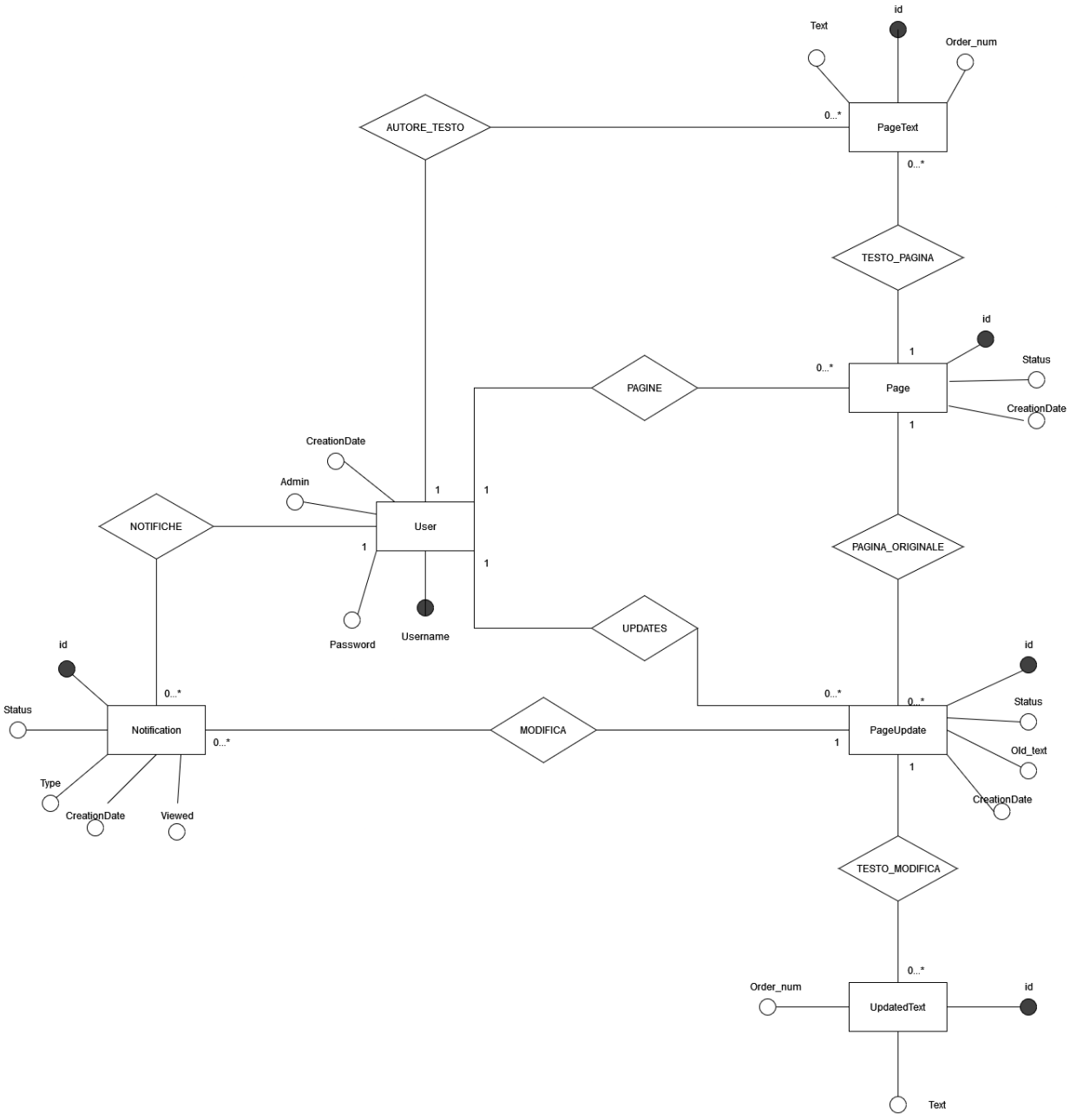
1.4.6 Diagramma UML Ristrutturato

Dopo aver ristrutturato il Class Diagram come descritto precedentemente, possiamo produrre il seguente schema concettuale ristrutturato espresso mediante diagramma UML:



1.4.7 Diagramma ER Ristrutturato

Ulteriore notazione per poter esprimere lo schema concettuale è l'ER. Il seguente è il diagramma ER:



2 Modello Logico

2.1 Traduzione di Entità e Associazioni

Avendo completato il processo di ristrutturazione, possiamo procedere col *mapping* di entità e associazioni. Per ogni entità del modello ristrutturato, definiremo una relazione equivalente con gli stessi attributi. Il processo di traduzione per le associazioni è invece più complesso e richiede un'analisi individuale di ogni associazione.

- **User, Page**: sono associazioni *uno-a-molti* che espriamiamo tramite chiave esterna, solo l'entità **Page** avrà come attributo **author**.
- **User, PageUpdate**: sono associazioni *uno-a-molti* che espriamiamo tramite chiave esterna, solo l'entità **PageUpdate** avrà come attributo **author**.
- **PageText, Page**: sono associazioni *uno-a-molti* che espriamiamo tramite chiave esterna, solo l'entità **PageText** avrà come attributo **page_id**.
- **UpdatedText, PageUpdate**: sono associazioni *uno-a-molti* che espriamiamo tramite chiave esterna, solo l'entità **UpdatedText** avrà come attributo **update_id**.
- **PageText, User**: sono associazioni *uno-a-molti* che espriamiamo tramite chiave esterna, solo l'entità **PageText** avrà come attributo **author**.
- **User, Notification**: sono associazioni *uno-a-molti* che espriamiamo tramite chiave esterna, solo l'entità **UpdatedText** avrà come attributo **user**.
- **PageUpdate, Notification**: sono associazioni *uno-a-molti* che espriamiamo tramite chiave esterna, solo l'entità **Notification** avrà come attributo **update_id**.

2.2 Relazioni

Legenda: *ChiavePrimaria*, Chiave Esterna↑, Attributo Nullabile?, Attributo Derivato

- **USER** (*username*, password)
- **PAGE** (*id*, title, creation, author↑) - author → USER.username
- **PAGETEXT** (*id*, order_num, text, page_id↑, author↑)
  - page_id → PAGE.id
  - author → USER.username
- **UPDATE** (*id*, status, creation, page_id↑, author↑)
  - page_id → PAGE.id
  - author → USER.username
- **NOTIFICATION** (*id*, status, type, user↑, update_id↑)
  - update_id → UPDATE.id
  - user → USER.username
- **UPDATEDTEXT** (*id*, text, order_num, type, update_id↑)
  - update_id → UPDATE.id

3 Modello Fisico

3.1 Domini

Notazioni: I domini usano SNAKE_CASE maiuscolo.

```
CREATE DOMAIN SHORT_TEXT VARCHAR(128)
CHECK(LENGTH(VALUE) > 0)
```

Il dominio SHORT_TEXT rappresenta un tipo di testo i cui valori sono vincolati a essere non nulli in termini di lunghezza. Applichiamo questo tipo per la maggior parte dei titoli e nomi nello schema, garantendo così il rispetto del *Vincolo di Leggibilità Testuale*.

3.2 Tabelle

Di seguito sono indicate le istruzioni DDL necessarie per definire le tabelle del database relazionale. Poiché queste sono una traduzione diretta delle relazioni definite nella sezione sul modello logico, si è preferito mantenere i commenti al minimo. Questi ultimi sono inclusi solo per chiarire concetti di SQL. Per informazioni sulla progettazione, fare riferimento alla sezione 3, Modello Logico.
Notazioni: Tutte le tabelle condividono lo stesso nome delle relazioni corrispondenti nel modello logico e utilizzano PascalCase. Data una tabella Sorgente, tutti i vincoli di chiave primaria espliciti seguono il formato **Sorgente_pk**. Per ogni chiave esterna che fa riferimento a una tabella Destinazione, si avrà il corrispondente vincolo **Sorgente_fk_Destinazione**.

```
CREATE TABLE IF NOT EXISTS User (
username SHORT_TEXT NOT NULL,
password SHORT_TEXT NOT NULL,
admin BOOLEAN DEFAULT FALSE,
creationDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP(),
CONSTRAINT User_pk PRIMARY KEY (username)
);
```

```
CREATE TABLE IF NOT EXISTS Page (
id INT AUTO_INCREMENT,
title SHORT_TEXT NOT NULL,
creationDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP(),
author SHORT_TEXT NOT NULL,
CONSTRAINT Page_pk PRIMARY KEY (id),
CONSTRAINT Page_fk_User FOREIGN KEY (author) REFERENCES User(username)
ON DELETE CASCADE
);
```

```
CREATE TABLE IF NOT EXISTS PageText (
  id INT AUTO_INCREMENT,
  order_num INT NOT NULL,
  text TEXT NOT NULL,
  page_id INT NOT NULL,
  author SHORT_TEXT NOT NULL,
  CONSTRAINT PageText_pk PRIMARY KEY (id),
  CONSTRAINT PageText_fk_Page FOREIGN KEY (page_id)
  REFERENCES Page(id) ON DELETE CASCADE,
  CONSTRAINT PageText_pk_User FOREIGN KEY (author)
  REFERENCES User(username)
);
```

```
CREATE TABLE IF NOT EXISTS `PageUpdate` (
id INT AUTO_INCREMENT,
page_id INT DEFAULT NULL,
author SHORT_TEXT DEFAULT NULL,
status INT DEFAULT NULL,
creationDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP(),
old_text LONGTEXT DEFAULT NULL,
CONSTRAINT Update_pk PRIMARY KEY (id),
CONSTRAINT Update_fk_Page FOREIGN KEY (page_id)
REFERENCES Page(id) ON DELETE CASCADE,
CONSTRAINT Update_fk_User FOREIGN KEY (author)
REFERENCES User(username) ON DELETE CASCADE
);
```

```
CREATE TABLE IF NOT EXISTS `UpdatedText` (
id INT NOT NULL,
text LONGTEXT DEFAULT NULL,
order_num INT DEFAULT NULL,
update_id INT DEFAULT NULL,
type INT NOT NULL,
CONSTRAINT UpdatedText_pk PRIMARY KEY (id),
CONSTRAINT UpdatedText_fk_Update FOREIGN KEY (update_id)
REFERENCES `PageUpdate`(id) ON DELETE CASCADE
);
```

```
CREATE TABLE IF NOT EXISTS Notification (
id INT AUTO_INCREMENT,
user SHORT_TEXT NOT NULL,
status BOOLEAN,
update_id INT NOT NULL,
type INT NOT NULL,
viewed BOOLEAN DEFAULT FALSE,
creationDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP(),
CONSTRAINT Notification_pk PRIMARY KEY (id),
CONSTRAINT Notification_fk_User FOREIGN KEY (user)
REFERENCES User(username)
ON DELETE CASCADE,
CONSTRAINT Notification_fk_User FOREIGN KEY (update_id)
REFERENCES `Update`(id)
ON DELETE CASCADE
);
```

3.3 View

Le seguenti viste sono state create per fornire una visione strutturata e semplificata dei dati nel database. Ogni vista corrisponde a una specifica interrogazione o aggregazione di dati utili per analizzare e monitorare l'attività degli utenti e le modifiche alle pagine nel sistema.

Listing 1: Vista PagineDiUnUtente

```
-- Restituisce gli username degli utenti e i titoli delle pagine associate a un utente specifico.
CREATE OR REPLACE VIEW PagineDiUnUtente AS
SELECT User.username, Page.title
FROM User
JOIN Page ON User.username = Page.author;
```

Listing 2: Vista ModificheDiUnUtenteAdUnaPagina

```
-- Restituisce gli username degli utenti, i titoli delle pagine e i nuovi testi delle modifiche effettuate da un utente su una pagina.
CREATE OR REPLACE VIEW ModificheDiUnUtenteAdUnaPagina AS
SELECT User.username, Page.title, PageText.text AS new_text
FROM User
JOIN Page ON User.username = Page.author
JOIN PageText ON Page.id = PageText.page_id;
```

Listing 3: Vista PaginaConTesto

```
-- Restituisce il titolo, l'ID della pagina, l'ID del testo e l'autore di una pagina insieme al testo associato.
CREATE OR REPLACE VIEW PaginaConTesto AS
SELECT Page.title, Page.id AS page_id, PageText.id AS text_id, Page.author
FROM Page
LEFT JOIN PageText ON Page.id = PageText.page_id;
```

Listing 4: Vista NotificheDiUnUtente

```
-- Restituisce gli username degli utenti, lo stato, l'ID dell'aggiornamento e il tipo di notifiche associate a un utente.
CREATE OR REPLACE VIEW NotificheDiUnUtente AS
SELECT User.username, Notifications.status, Notifications.update_id, Notifications.type
FROM Notifications
JOIN User ON Notifications.user = User.username;
```

Listing 5: Vista UpdatePerPagina

```
-- Restituisce l'ID della pagina e il numero di aggiornamenti accettati per una pagina specifica.
CREATE OR REPLACE VIEW UpdatePerPagina AS
SELECT Page.id AS page_id, COUNT(PageUpdate.id) AS num_updates
FROM Page
LEFT JOIN PageUpdate ON Page.id = PageUpdate.page_id
WHERE PageUpdate.status IS NOT NULL
GROUP BY Page.id;
```

Listing 6: Vista UtentiConModificheProposte

```
-- Restituisce gli username degli utenti che hanno proposto modifiche per una pagina specifica.
CREATE OR REPLACE VIEW UtentiConModificheProposte AS
SELECT User.username, PageUpdate.page_id
FROM PageUpdate
JOIN User ON PageUpdate.author = User.username;
```

Listing 7: Vista PagineConPiuModificheAccettate

```
-- Restituisce il titolo delle pagine e il numero massimo di modifiche accettate per ciascuna pagina.
CREATE OR REPLACE VIEW PagineConPiuModificheAccettate AS
SELECT Page.title, MAX(num_accepted) AS num_accepted
FROM (
SELECT Page.id, Page.title, COUNT(PageUpdate.id) AS num_accepted
FROM Page
LEFT JOIN PageUpdate ON Page.id = PageUpdate.page_id
LEFT JOIN Notifications ON PageUpdate.id = Notifications.update_id
WHERE Notifications.status = 1
GROUP BY Page.id, Page.title
) AS Subquery
GROUP BY Subquery.id, Subquery.title;
```

Listing 8: Vista PagineConPiuModificheProposte

```
-- Restituisce il titolo delle pagine e il numero massimo di modifiche proposte (non accettate) per ciascuna pagina.
CREATE OR REPLACE VIEW PagineConPiuModificheProposte AS
SELECT Page.title, MAX(num_proposed) AS num_proposed
FROM (
SELECT Page.id, Page.title, COUNT(PageUpdate.id) AS num_proposed
FROM Page
LEFT JOIN PageUpdate ON Page.id = PageUpdate.page_id
LEFT JOIN Notifications ON PageUpdate.id = Notifications.update_id
WHERE Notifications.status = 0
GROUP BY Page.id, Page.title
) AS Subquery
GROUP BY Subquery.id, Subquery.title;
```

3.4 Operazioni

In questa sezione sono presenti operazioni di tipo programmatico.

Notazioni: si utilizza PascalCase per i nomi delle funzioni/procedure e camelCase per i parametri

```
CREATE OR REPLACE FUNCTION search_pages(
search_text VARCHAR(255),
search_in_title BOOLEAN DEFAULT TRUE,
search_in_text BOOLEAN DEFAULT FALSE,
search_in_author BOOLEAN DEFAULT FALSE
) RETURNS TABLE (
page_id INT,
title VARCHAR(255),
creation DATE,
author VARCHAR(255)
) AS $$
BEGIN
RETURN QUERY
SELECT id, title, creation, author
FROM Page
WHERE
(search_in_title AND title ILIKE '%' || search_text || '%') OR
(search_in_text AND id IN (SELECT page_id FROM PageText WHERE text
   ILIKE '%' || search_text || '%')) OR
(search_in_author AND author ILIKE '%' || search_text || '%');
END;
$$ LANGUAGE plpgsql;
```

Questa funzione PL/pgSQL, chiamata `search_pages`, accetta un testo di ricerca e tre parametri booleani come input. Restituisce una tabella contenente page_id, titolo, data di creazione e autore delle pagine che corrispondono ai criteri di ricerca specificati.

Parametri:

- `search_text`: Il testo da cercare all'interno dei titoli, dei testi o degli autori delle pagine.
- `search_in_title` (Default: `TRUE`): Booleano che indica se cercare all'interno dei titoli delle pagine.
- `search_in_text` (Default: `FALSE`): Booleano che indica se cercare all'interno dei testi delle pagine.
- `search_in_author` (Default: `FALSE`): Booleano che indica se cercare all'interno degli autori delle pagine.

Utilizzo:

```
SELECT * FROM search_pages('testo_di_ricerca', TRUE, TRUE, FALSE);
```

Note:

- La funzione esegue una ricerca case-insensitive utilizzando l'operatore ILIKE.
- Il risultato è una tabella con colonne: page_id, titolo, data di creazione e autore.

```
CREATE OR REPLACE FUNCTION search_notifications(
titolo_pagina_text TEXT,
tipo_notifica INT,
stato_notifica INT,
notifica_letta BOOLEAN
)
RETURNS TABLE (
id INT,
user_name SHORT_TEXT,
status BOOLEAN,
update_id INT,
notification_type INT,
viewed BOOLEAN,
creation_date TIMESTAMP
) AS
$$
BEGIN
RETURN QUERY
SELECT
n.id,
n.user,
n.status,
n.update_id,
n.type,
n.viewed,
n.creationDate
FROM Notification n
JOIN "Update" u ON n.update_id = u.id
JOIN Page p ON u.page_id = p.id
WHERE p.title ILIKE '%' || titolo_pagina_text || '%'
AND n.type = tipo_notifica
AND n.status = stato_notifica
AND n.viewed = notifica_letta;
END;
$$
LANGUAGE plpgsql;
```

La funzione PL/pgSQL, denominata `search_notifications`, consente di effettuare una ricerca avanzata tra le notifiche del sistema. Restituisce una tabella contenente dettagli specifici delle notifiche che corrispondono ai criteri di ricerca specificati.

Parametri:

- `titolo_pagina_text`: Il testo da cercare all'interno dei titoli delle pagine associate alle notifiche.
- `tipo_notifica`: Il tipo di notifica da cercare.
- `stato_notifica`: Lo stato della notifica da cercare.
- `notifica_letta` (Default: `FALSE`): Booleano che indica se cercare notifiche lette o non lette.

Output:
La funzione restituisce una tabella con le seguenti colonne:

- `id`: L'identificativo univoco della notifica.

- `user_name`: Lo username dell'utente associato alla notifica.
- `status`: Lo stato della notifica.
- `update_id`: L'identificativo univoco dell'aggiornamento associato alla notifica.
- `notification_type`: Il tipo della notifica.
- `viewed`: Booleano che indica se la notifica è stata letta o meno.
- `creation_date`: La data di creazione della notifica.

Esempio di Utilizzo:

```
1       SELECT * FROM search_notifications('Pagina', 1, 0, FALSE);
```

Questa query restituirà tutte le notifiche relative alle pagine che contengono il testo "Pagina", hanno tipo di notifica 1, stato 0 e non sono state ancora lette.

3.5 Trigger

In questa sezione sono definiti trigger necessari per l'implementazione di vincoli o altre caratteristiche della base di dati oppure necessari per l'automatizzazione di certe cose.

I trigger nel modello vengono utilizzati principalmente per gestire le notifiche degli updates di una pagina.

Notazione: Ogni trigger viene definito dalla notazione **Trigger_NomeTabellaListen_Caratteristica_NomeTabellaEdit** dove:

- **NomeTabellaListen**: Rappresenta la tabella sulla quale il trigger "guarda" le modifiche.
- **NomeTabellaEdit**: Rappresenta la tabella modificata dal trigger
- **Caratteristica**: Rappresenta una caratteristica determinata da ciò che compie il trigger.

Ogni trigger è composto da due parti, il trigger vero e proprio e la funzione che esso esegue

```
1  CREATE OR REPLACE FUNCTION notification_request_update()
2  RETURNS TRIGGER AS $$
3  DECLARE
4  page_creator VARCHAR(255);
5  TYPE_REQUEST_UPDATE INT := 0;
6  BEGIN
7  SELECT author INTO page_creator
8  FROM Page
9  WHERE id = NEW.page_id;
10
11 INSERT INTO notification ("user", update_id, type)
12 VALUES (page_creator, NEW.id, TYPE_REQUEST_UPDATE);
13
14 RETURN NEW;
15 END;
16 $$ LANGUAGE plpgsql;
17
18 CREATE TRIGGER Trigger_Update_Request_Notifica
19 AFTER INSERT ON PageUpdate
20 FOR EACH ROW
21 EXECUTE FUNCTION notification_request_update();
```

```
1  CREATE OR REPLACE FUNCTION notification_update_accepted()
2  RETURNS TRIGGER AS $$
3  DECLARE
4  page_creator VARCHAR(255);
5  TYPE_UPDATE_ACCEPTED INT := 1;
6  NOTIFICATION_READ INT := 1;
7  STATUS_PENDING INT := -1;
8  STATUS_ACCEPTED INT := 1;
9  BEGIN
10 SELECT author INTO page_creator
11 FROM Page
12 WHERE id = NEW.page_id;
13
14 IF OLD.status = STATUS_PENDING AND NEW.status = STATUS_ACCEPTED THEN
15 INSERT INTO Notification ("user", update_id, type)
```

```
    VALUES (NEW.author, NEW.id, TYPE_UPDATE_ACCEPTED);

    UPDATE Notification
    SET type = TYPE_UPDATE_ACCEPTED, status = NOTIFICATION_READ
    WHERE "user" = page_creator AND update_id = NEW.id;
    END IF;

    RETURN NEW;
    END;
    $$ LANGUAGE plpgsql;

    CREATE TRIGGER Trigger_Update_Accettazione_Notifica
    AFTER UPDATE ON PageUpdate
    FOR EACH ROW
    EXECUTE FUNCTION notification_update_accepted();
```

```
    CREATE OR REPLACE FUNCTION notification_update_rejected()
    RETURNS TRIGGER AS $$
    DECLARE
    page_creator VARCHAR(255);
    TYPE_UPDATE_REJECTED INT := 2;
    NOTIFICATION_READ INT := 1;
    STATUS_PENDING INT := -1;
    STATUS_REJECTED INT := 0;
    BEGIN
    SELECT author INTO page_creator
    FROM Page
    WHERE id = NEW.page_id;

    IF OLD.status = STATUS_PENDING AND NEW.status = STATUS_REJECTED THEN
    INSERT INTO notification ("user", update_id, type)
    VALUES (NEW.author, NEW.id, TYPE_UPDATE_REJECTED);

    UPDATE notification
    SET type = TYPE_UPDATE_REJECTED, status = NOTIFICATION_READ
    WHERE "user" = page_creator AND update_id = NEW.id;
    END IF;

    RETURN NEW;
    END;
    $$ LANGUAGE plpgsql;

    CREATE TRIGGER Trigger_Update_Rifiuto_Notifica
    AFTER UPDATE ON PageUpdate
    FOR EACH ROW
    EXECUTE FUNCTION notification_update_rejected();
```