Basi di dati e laboratorio web

Relazione per il progetto d'esame
Anno accademico 2023/24
Sara Seidita 579517

1.2 Specifica delle operazioni del progetto

Principali

1. **Funzionalità di ricerca per blog e contenuto**
    a. tramite parole che compaiono nel titolo blog
    b. tramite parole che compaiono nella categoria del blog
    c. tramite parole che compaiono nel nome utente (autore del blog)

2. **Fase di registrazione**
    a. L'utente deve inserire un nome utente univoco
    b. L'utente deve inserire la propria email
    c. L'utente deve inserire una password
    d. L'utente deve inserire un conferma password

3. **Fase di login**
    a. L'utente deve inserire il proprio nome utente
    b. L'utente deve inserire la password

4. **Visualizzazione blog e post**
    a. Vedere il blog e i post pubblicati
    b. Vedere i commenti sotto i post pubblicati

Utente anonimo

1. L'utente anonimo visualizza la pagina principale che si visita per prima

Utente loggato (semplice)

1. Visualizzare il proprio profilo:
    a. Visualizza i propri dati personali
    b. Modificare i propri dati personali
    c. Cancellare il proprio profilo
    d. Accedere tramite link alla gestione blog:
        i. Creare fino a 5 blog
        ii. Visualizzare il blog
        iii. Cancellare il blog
    e. Accedere tramite link alla gestione del blog in cui è co-autore:
        i. Visualizzare il blog
    f. Accedere tramite un bottone per diventare utente premium

2. Diventare premium

    a. Bottone per diventare premium tramite l'inserimento di alcuni dati

3. Visualizzare l'header del sito
    a. Accedere all'homepage tramite un link
    b. Accedere alla lista dei blog tramite un link
    c. Accedere al profilo utente
    d. Fare il logout

4. Visualizzare il proprio blog (sezione lista blog, tramite ricerca, etc)
    a. Accedere e visualizzare i post del blog
    b. Bottone per la creazione di un post
    c. Bottone per la cancellazione del post
    d. Visualizzare i commenti
    e. Inserire un commento
    f. Cancellare il proprio commento
    g. Inserire il like
    h. Visualizzare il conteggio del like

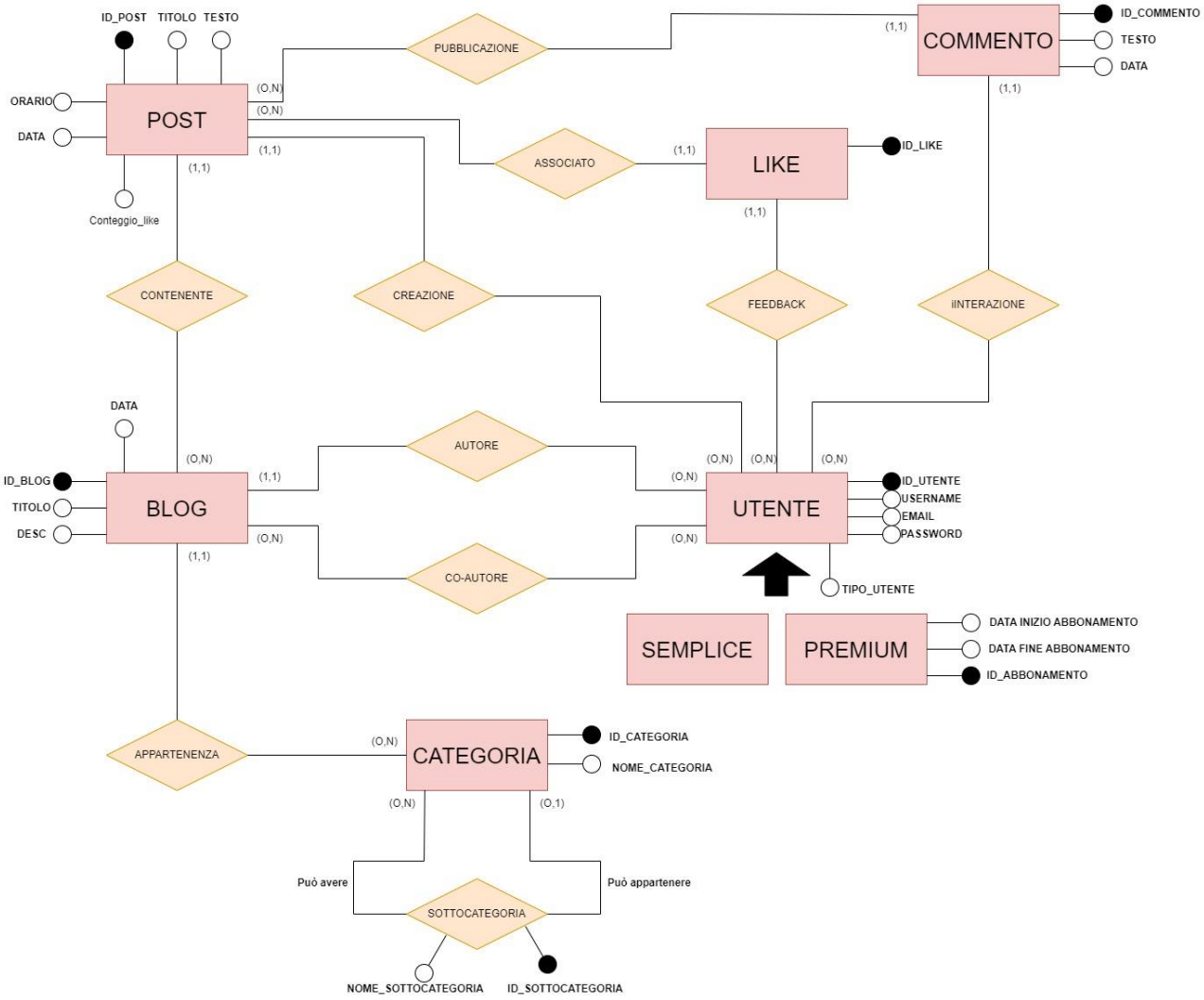
5. Visualizzare gli altri blog in cui non è autore e non è co-autore
    a. Visualizzare il blog
    b. Visualizzare i post
    c. Inserire un comento
    d. Cancellare i propri commenti
    e. Inserire un solo like
    f. Cancellare il solo like
    g. Visualizzare il conteggio del like

Utente loggato (premium)

1. Tutte le funzionalità precedenti
2. Eccezione:
    a. Creare infiniti blog

FASE 2: PROGETTAZIONE CONCETTUALE

2.1 Diagramma E-R



2.2 Dizionario delle entità

ENTITA'	DESCRIZIONE	ATTRIBUTI
Utente	Utente del sistema	ID_utente, username, email, password
Premium	Utente premium del sistema	ID_abbonamento, data_inizio_abbonamento, data_fine_abbonamento
Blog	Funzionalità centrale dell'applicazione	ID_blog, titolo, descrizione, img
Post	Contenuto dei blog	ID_post, titolo, img, orario, data, testo, conteggio_like
Categoria	Tema del blog	ID_categoria, nome_categoria
Commento	Interazione dell'utente	ID_commento, testo, data
Like	Feedback dell'utente	ID_like

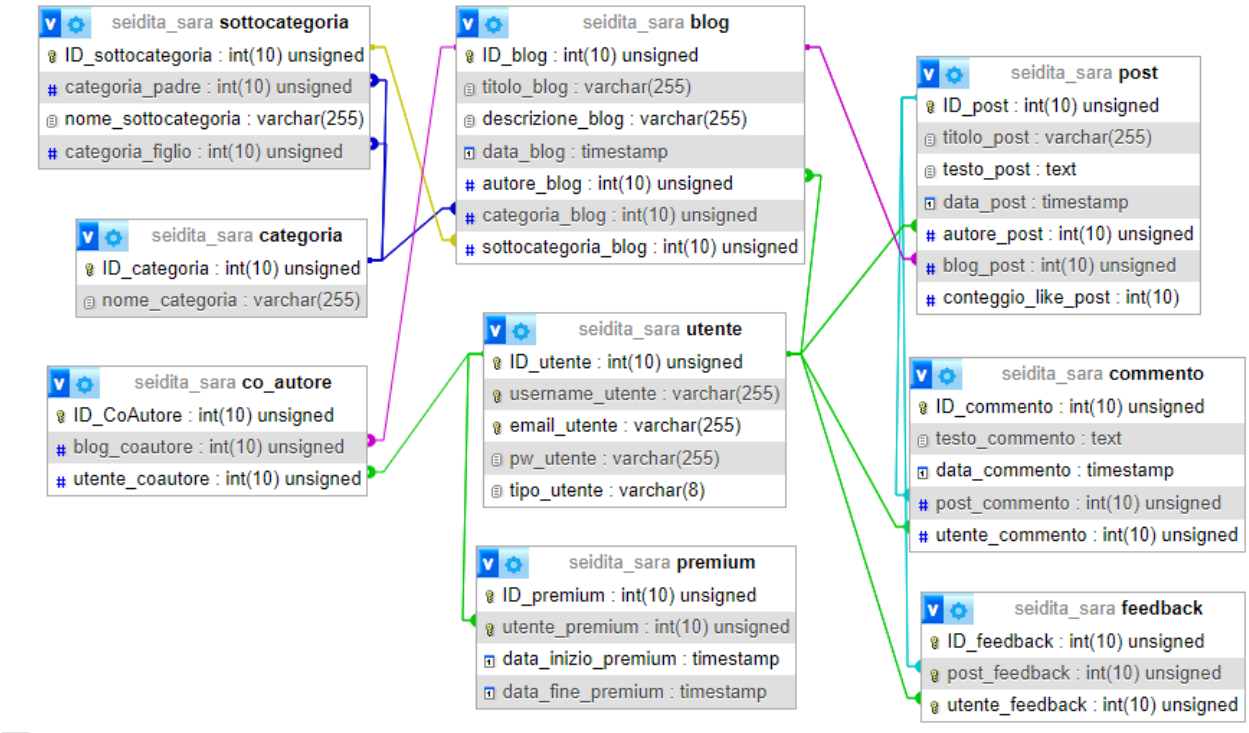
2.3 Dizionario delle relazioni

RELAZIONE	DESCRIZIONE	COMPONENTI
Autore	Associa blog e utente (autore del blog)	Utente, blog
Creazione	Associa utente a post	Utente, post
Contenente	Associa blog ai post	Blog, post
Appartenenza	Associa blog alla categoria	Blog, categoria
Sottocategoria	Relazione ricorsiva di categoria	Categoria
Feedback	Associa utente a like	Utente, like
Associato	Associa like a post	Like, post
Interazione	Associa utente al commento	Utente, commento
Pubblicazione	Associa il commento al post	Commento, post

2.4 Regole di vincolo

1. **Utente registrato, loggato, non premium**
    a. Può creare fino a 5 blog

Schema relazionale delle tabelle



FASE 3: PROGETTAZIONE LOGICA

Schema logico

UTENTE(ID_utente, username, email, password, tipo_utente)
BLOG(ID_blog, titolo_blog, descrizione_blog, data_blog, autore_blog*, categoria_blog*)
POST(ID_post, titolo_post, testo_post, data_post, orario_post, autore_post*, blog_post*, conteggio_like)
COMMENTO(ID_commento, testo_commento, data_commento, post_commento*, utente_commento*)
FEEDBACK(ID_feedback, post_feedback*, utente_feedback*)
CATEGORIA(ID_categoria, nome_categoria)
SOTTOCATEGORIA(ID_sottocategoria, categoria_padre*, categoria_figlio*, nome_sottocategoria)
PREMIUM(ID_premium, ID_utente_premium, data_inizio_premium, data_fine_premium)

FASE 4: IMPLEMENTAZIONE

Pagine PHP dell'applicazione – alcune scelte effettuate

register.php

- Pagina php che include la form per la registrazione dell'utente
- La form prevede il metodo POST e come action lo script php "*server.php*" e prende in input quattro valori:
    - L'username dell'utente
    - L'email dell'utente
    - La password dell'utente
    - La conferma password inserita dall'utente
- È inclusa la validazione della form utilizzando **jQuery validation**, dove abbiamo:
    - Uno script che utilizza *addMethod*, che aggiunge un metodo di validazione personalizzato, che include il nome di tipo stringa ("regex"), il metodo di tipo *function*, ossia la funzione che ha come parametri il valore ("*value*") e l'elemento ("*element")*.
        - Lo script ritorna la validazione personalizzata per l'email utilizzando un'espressione regolare: */^\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}\b$/*
            - Questa espressione regolare ci consente di avere il controllo dell'input dell'email controllando che ci sia il "." e una stringa successiva, in modo tale che non si controlli soltanto la presenza di "@".
    - Lo script *validate()*, dove ho aggiunto regole come:
        - I required, tutti i campi devono essere compilati
        - L'username deve essere lungo almeno 3 caratteri
        - "regex" per il campo email, che controlla la validazione impostata con lo script precedente
        - La password deve essere lunga almeno 8 caratteri
        - La conferma password deve essere uguale (*equalTo*) alla password
        - Sono inseriti i vari messaggi che appaiono in caso di compilazione errata dei campi
- Se la compilazione è corretta, viene inviato il form-data tramite il submit al file server.php, che effettua l'inserimento dei dati dell'utente del database

login.php

- Pagina php che include la form per la fase di login dell'utente
- La struttura del codice si presenta simile a quella di *register.php*, ma all'utente viene chiesto di inserire come campi di input il proprio nome utente e la password corrispondente, campi che vengono validati da jQuery validation, per il controllo della lunghezza che deve essere uguale a quando viene fatta la registrazione (quindi minima di 3 caratteri per l'username e minima di 8 per la password)
- La form viene inviata a *server.php* come per la registrazione

server.php

- Questo script PHP include la connessione al database per:
    - Poter completare la registrazione dell'utente, e quindi poter inserire i dati nella base di dati;
    - Poter effettuare il login dell'utente (quindi con il controllo dei dati se inseriti correttamente e corrispondenti a quelli presenti nel database), facendo poi startare la sessione tramite session_start()

Registrazione

- Il codice registrazione utente prevede if(isset($_POST['usernameUtente']), dove *isset* verifica se una variabile è definita e non è NULL, e $_POST contiene un array di variabili ricevuti tramite un metodo HTTP POST, in questo caso dalla form di registrazione si chiede la variabile dell'usernameUtente, valore ricevuto dall'input dell'utente.
    - Se la variabile è definita, recupero i valori inseriti nella form register.php inserendoli in una variabile, in particolare:
        - L'username
        - L'email
        - La password (e non la conferma password)
    - La password viene poi criptata tramite md5()
    - Controllo poi se il nome utente o la email è già presente nel database. Per farlo:
        - Una query che seleziona l'username e mail dalla tabella utenti dove l'username o l'email corrispondono agli input inseriti dall'utente
        - Mysqli_query che esegue la query nel database
        - Utilizzo poi mysqli_num_rows che ritorna il numero di righe del risultato della query, dove:
            - Se restituisce 0, allora si possono inserire i dati nel database perché disponibili e l'utente verrà riportato alla fase di login e setto la variabile di sessione con $_SESSION
            - Se restituisce almeno una riga, allora l'utente dovrà reinserire i dati perchè o l'username o l'email è già esistente
    - Se ci sono stati altri errori, viene incluso *error.php* che è un semplice messaggio di errore e che consente di ritornare alla homepage

Login

- Se la variabile $_POST['loginUsername'] è definita:
    - Creo una variabile $errors che inizializza un array, inizialmente vuoto, che dovrà contenere eventuali errori
    - Recupero i valori inseriti salvandoli in una variabile, utilizzando:

        - mysqli_real_escape_string, funzione che è necessaria per l'escape di caratteri speciali e prende come parametri $conn, la variabile di connessione al database, e $_POST, ossia i valori passati tramite POST dalla form
    - Se le variabili del nome utente e password sono vuoti, allora si utilizza array_push, ossia accodo un elemento all'array $errors
    - Se la variabile $errors ha 0 elementi, significa che non ci sono errori, e quindi andrò a criptare la password (come per la registrazione) e ad effettuare la query di SELECT sul database, per verificare che username e password corrispondono e che siano corretti
        - Utilizzando mysqli_num_rows, se il risultato è uguale a 1, allora il login viene effettuato correttamente, setto la variabile di sessione e l'utente verrà reindirizzato alla homepage del sito (*index.php*)
        - Altrimenti, se i dati sono errati, l'utente deve reinserire i dati corretti

profilo.php

- Se la variabile di sessione è definita, recupero i dati dell'utente, i dati premium se presente, e il numero di blog creati dall'utente
- Per recuperare i dati:
    - Faccio una query SELECT
    - Eseguo la query con mysqli_query
    - Uso mysqli_fetch_assoc, che prende l'oggetto risultato come parametro e restituisce un'array associativo, così posso elabolarlo secondo le necessità
    - Nel profilo l'utente può visualizzare:
        - I suoi dati personali
        - Che tipo di utente è
        - Se è tipo semplice, visualizza un banner con un bottone che lo riporta a *premium.php*
        - Se è tipo premium, allora vedrà la data in cui si è abbonato e la scadenza
        - Un banner dove c'è il numero di blog che ha creato e il numero di blog in cui è co-autore e potrà aprire un link in entrambi i casi per gestire i blog
        - Un banner *impostazione utente* con due bottoni che indirizzano l'utente a modificare i dati del profilo o a cancellarlo direttamente

modifica-profilo.php

- Inclusa una form dove nel campo sono presenti i dati dell'utente. L'utente in questione può decidere se modificare l'username o l'email modificando il campo e inviando con il submit, il quale tramite l'action effettuerà il codice php, il quale controlla se il campo modificato è esistente o meno, e in caso è disponibile, aggiorna il database. Se la modifica viene effettuata con successo, l'utente dovrà effettuare nuovamente il login per confermare.
- Come per la registrazione e il login, anche questa form presenta lo script del jquery validation per la validazione dei dati inseriti (es. la validazione dell'email)

premium.php

- L'utente che accede tramite link a questa pagina potrà inserire alcuni dati in una form per poter passare ad un profilo premium, tra i quali:
    - Numero della carta di credito, che necessita di esattamente 16 caratteri numerici
    - Il nome della carta
    - Il cognome
    - La data di scadenza
    - Il codice di sicurezza, che necessita di avere esattamente 3 caratteri numerici

- La form viene validata con jquery validation, dove uso la funzione addMethod() per aggiungere una regola che controlla che la lunghezza esatta di due campi in particolare, e poi il validate() dei campi
- Dopo aver recuperato i dati inseriti dall'utente e aver recuperato l'ID dell'utente, inserisco i dati nella tabella premium, inserendo come date la data di inizio premium (data attuale) e data di fine premium (+1 mese).
- Viene poi modificato il campo tipo_utente nella tabella utente, da "Semplice" (di default) a "Premium"
- Se tutto va a buon fine, l'utente verrà riportato al profilo dove vedrà che è diventato utente premium.

crea-blog.php

- Se l'utente è di tipo Premium o se l'utente è di tipo Semplice e ha creato meno di cinque blog, potrà compilare una form dove dovrà inserire alcuni valori per creare il proprio blog, tra cui:
    - Titolo del blog
    - Descrizione del blog, la quale dovrà avere massimo 160 caratteri
    - Categoria del blog
    - Sottocategoria del blog
- La form viene validata con jQuery validation per il controllo dei campi
- Lo script php che permette la creazione del blog prevede inizialmente il recupero dell'ID utente, dei blog creati per fare il count, recupero tipo utente, per poter fare accedere alla creazione del blog soltanto a chi può farlo.
    - Dopodiché inizializzo le variabili, inizialmente vuote, che conterranno i valori presi in input
    - Inizializzo l'array associativo $errors, inizialmente vuoto, che conterrà eventuali errori
    - Mi prendo le categorie e sottocategorie per il controllo dell'inserimento delle stesse per la creazione del blog
        - Uso il *mysqli_fetch_all* perché porta tutte le righe risultato come un array associativo (MYSQLI_ASSOC)
    - Dopodiché, dopo aver definito $_POST['creablog_invia'], ossia il submit della form, eseguo i vari check:
        - Check titolo, verifica che è stato inserito
        - Check descrizione, verifica che è stata inserita
        - Check categoria, non solo verifica che è stata inserita, ma:
            - Verifica che la categoria deve contenere soltanto spazi e lettere
            - Controllo attraverso un while se la categoria è già presente nella tabella, verificandolo con il nome messo input e verificando se corrisponde a un nome_categoria già presente nella tabella. Se presente, prendo l'ID di tale categoria. Se non è presente, inserisco la nuova categoria, e uso mysqli_insert_id dopo aver eseguito la query per poter inserire l'ID e per poterlo salvare nella variabile. Questo mi servirà per l'inserimento dei dati del blog, in quanto mi serve l'ID della categoria.
        - Check sottocategoria, stesso procedimento della categoria
        - Recupero la data della creazione
        - Se non ci sono errori, prevedo a recuperare i dati inseriti nella form, facendo l'escape delle stringhe e salvandoli nelle variabili.
            - Recupero l'id dell'utente

            - Faccio ed eseguo la query di inserimento; se eseguita con successo, l'utente verrà riportato alla *gestione-blog.php*
- Gli utenti che non sono ammessi a questa pagina vedranno *error.php* e/o ciò che vedono gli utenti anonimi (*user_anon.php*)

visual_blog.php

- Pagina PHP che permette di visualizzare i blog, a cui si accede tramite link *visual_blog.php?ID_Blog="ID del blog"*
- Definendo l'ID blog dalla GET, è possibile visualizzare uno specifico blog
- Lo script php recupera l'ID del blog dalla GET, recupera tutti i dati del blog, dei post del blog, dei commenti dei post, i coautori del blog, categorie e sottocategorie.
    - Viene gestito il co-autore, il quale potrà vedere soltanto il crea_post.php, ma non potrà cancellare i post o gestire il blog
- Ciò che l'utente loggato vedrà in questa pagina è:
    - Una sezione dedicata ai dati del blog, ossia: titolo del blog, autore del blog, data del blog, categoria e sottocategoria, e co-autore del blog
    - Se l'utente è autore del blog vedrà i seguenti bottoni:
        - Crea_post?ID_Blog
        - Gestione_blog
    - Se l'utente è co-autore, vedrà solo crea_post
    - Una sezione dedicata ai post, dove:
        - Se non sono presenti, vedrà un semplice messaggio
        - Se presenti, attraverso un foreach, vedrà ogni post creato con i propri dati (titolo, testo, data, autore del post, commenti, like)
            - La sezione dei commenti viene gestita anch'essa tramite un foreach, e l'autore del commento potrà cancellarlo
        - Un form aggiungi commento, che in action riporta a *insert_commento.php*, frammento che ne permette l'inserimento, e con il submit vedrà aggiornare la pagina con il commento inserito
        - Un bottone per aggiungere il like, che in action riporta a insert_like.php e con il submit vedrà il conteggio del like aumentato di 1. In caso avesse già inserito un like vedrà il conteggio scendere. Questo è gestito tramite trigger.
        - Se l'utente è autore del blog, potrà cancellare i vari post, con annessi commenti e like.

crea_post.php

- Pagina PHP che permette l'inserimento di un post ad uno specifico blog
- I controlli della form sono simili a quello di crea-blog.php
- Lo script PHP prevede l'inizializzazione delle variabili dei dati inseriti e degli errori (salvati in un array, inizialmente vuoto).
    - Viene definita la variabile dell'ID del blog presa dalla GET
    - Si effettuano i vari controlli dei dati, come per la creazione del blog
    - Se tutto è in regola, e quindi non ci sono errori, si salvano gli input nelle variabili facendo l'escape delle stringhe, e si crea e si esegue la query di inserimento. Dopo che la query viene eseguita, l'utente verrà riportato al blog per poter vedere il post appena creato e per poterne creare altri

Blog_list.php / visual_list.php

- Il menu del sito prevede un link che riporta alla visualizzazione della lista dei blog (blog_list.php)
- La paginazione della pagina è stata gestita tramite Ajax, creato un numero di elementi da visualizzare per pagina e un offset iniziale
    - La chiamata $.ajax viene eseguita tramite GET e l'url dello script visual_list.php
    - Se eseguita con successo, si vedrà la response della funzione loadProducts()
    - Se si clicca il bottone $next o $prev, si andrà rispettivamente avanti o indietro di pagina
- In *visual_list.php* c'è lo script PHP che effettua la sql di selezione dei blog, ordinando per titolo blog e usando LIMIT $limit – variabile presa tramite GET con il valore imposto nello script per la chiamata Ajax – e come OFFSET $offset – come per la variabile $limit .
- Attraverso un while dove mostra le righe risultato, si visualizza la lista del blog, ai quali si potrà accedere per visualizzarli tramite il link

Gestione-coauthor.php

- Pagina PHP dove un utente può gestire il co-autore del proprio blog.
- L'utente potrà avere un solo co-autore per ogni blog.
- Ciò che vedrà l'utente autore del blog, è il nome del co-autore aggiunto, il quale potrà essere rimosso se necessario
- Se non ci sono co-autori, ci sarà una form dove potrà inserire un nome utente da aggiungere.
    - Una chiamata AJAX suggerirà sottostante i nomi utenti presenti nel database
        - Ci sarà uno script search_user.php che effettua la ricerca del nome utente dal database tramite il valore che l'utente sta inserendo nell'input
    - La form prevede un'action che riporta i dati allo script *add-coauthor.php*
        - Lo script prende tramite la GET l'id del Blog
        - Si selezionano i dati del blog per estrarre l'ID che viene salvato tramite variabile
        - Si selezionano i dati dell'utente dove il nome utente corrisponde al valore inserito nella form, per poter estrarre l'ID
        - Si controlla se l'utente è già stato inserito (controllo che potrebbe essere omesso, dato che se ne può inserire soltanto uno)
        - Se non ci sono problemi, viene inserito l'utente nella tabella co_autore e l'utente autore del blog verrà riportato alla gestione del suo co-autore
        - Se l'utente non esiste, verrà segnalato tramite popup

Funzione di ricerca blog

La sezione di ricerca blog è stata inserita nella pagina principale *index.php*.
In questa sezione, l'utente potrà ricercare il blog per:

- Titolo
- Categoria
- Autore del blog

Per fare questo, ho usato Ajax. Per ogni tipo di ricerca, è stata creata una form, dove l'utente potrà scegliere come ricercare un blog desiderato. Ogni form avrà poi un div, dove verrà mostrato il risultato tramite le chiamate Ajax.
In generale, la chiamata Ajax:

- Funzione fill(Value) che serve per assegnare il valore all'id #ricerca – ID che corrisponde a quello dell'input dove viene inserita la stringa – e per nascondere il div #display, ossia il div dove verrà mostrato il risultato una volta che l'utente inizierà a inserire il valore

- Funzione principale con la chiamata $ajax di tipo POST e che viene eseguita tramite i vari script PHP. In caso di successo, viene mostrato il risultato.

Gli script PHP sono simili tra loro, cambia soltanto la condizione nella query. Se si cerca un utente, nel WHERE ci sarà username_utente = '$variabile', dove la variabile corrisponde all'input inserito.
Il risultato mostrerà i blog che rispettano le condizioni di ricerca e ci sarà la possibilità di visualizzare tali blog.

Gestione feedback – conteggio like post
Una delle funzionalità che può fare un utente nel sito è quello di dare un feedback ai post dei vari blog. Questo feedback prevede un bottone like, che se selezionato aumenta il conteggio dei like di un determinato post, e se selezionato nuovamente, diminuisce il conteggio.
L'aumento e diminuzione del conteggio del like è stato gestito tramite TRIGGER.

```
DELIMITER $$
CREATE TRIGGER `deleteLikeCounter` AFTER DELETE ON `feedback` FOR EACH ROW update post
set conteggio_like_post = COALESCE(conteggio_like_post, 0) - 1
where OLD.post_feedback = post.ID_post
$$
DELIMITER ;
DELIMITER $$
CREATE TRIGGER `updateLikeCounter` AFTER INSERT ON `feedback` FOR EACH ROW update post
set conteggio_like_post = COALESCE(conteggio_like_post, 0) + 1
where NEW.post_feedback = post.ID_post
$$
DELIMITER ;
```

Se un utente clicca like e non l'ha fatto prima, viene inserita una riga nella tabella feedback e l'attributo conteggio_like_post viene aggiornato (+1).
In caso ci fosse stato già l'inserimento del like da parte dell'utente, il trigger provvederà ad aggiornare l'attributo (-1).

ACCESSIBILITÀ
Per quanto riguarda l'accessibilità del sito:

- ho tenuto conto dei colori utilizzati per il sito stesso, preferendo colori neutri, facendo attenzione al contrasto che c'è tra i colori di sfondo e i colori del testo.
- Ho evitato font particolari o troppo elaborati, utilizzandone solo uno. In caso di titoli, è stato aumentato il font-size. In caso di informazioni specifiche (esempio: nel profilo utente, **username:** nomeutente) è stato usato il grassetto