Documentazione
Galleria Fotografica Geolocalizzata

Gennaro Ventrone Luigi Solaro Mariano Sommella

ANNO ACCADEMICO 2022-2023

Indice

Capitolo 1

Specifiche Progetto

1.1 Introduzione Documentazione

Nelle seguenti pagine si riporta la documentazione relativa ad una progettazione e implementazione di un sistema di basi di dati per la gestione di collezioni fotografiche geolocalizzate condivise.

Per una visione migliore del documento si utilizzi l'indice e le varie sezioni menzionate.

1.2 Elenco Specifiche

Vengono qui elencate le caratteristiche che l'applicativo deve garantire:

1. Informazioni sulla foto: Possibilità di visualizzare per ogni foto l'utente che l'ha scattata, il dispositivo utilizzato e il luogo in cui è stata scattata. Il luogo può essere identificato da coordinate geografiche o da un nome mnemonico unico all'interno del sistema.

2. Identificazione dei soggetti: Ogni foto può raffigurare diversi soggetti o utenti, in entrambi i casi devono essere identificati univocamente e categorizzati.

3. Accesso a collezioni fotografiche: Ogni utente ha sempre la possibilità di vedere la propria galleria fotografica personale, che comprende esclusivamente le foto scattate da lui. Un utente può partecipare a collezioni condivise con altri utenti che possono contenere foto scattate da questi ultimi.

4. Accesso alla foto: Ogni utente deve avere la possibilità di accedere solo alle proprie foto o a quelle condivise con gli altri utenti. Le foto private non verranno condivise con gli altri utenti.

5. Eliminazione delle foto: Gli utenti devono avere la possibilità di eliminare le proprie foto, ma solo l'amministratore del sistema può elimi-

nare un utente e tutte le sue foto (eccetto le fotografie che contengono come soggetto un altro utente).

6. Operazione di ricerca: Il sistema deve permettere la ricerca di foto in base al luogo in cui sono state scattate o al soggetto rappresentato. Inoltre, deve essere possibile visualizzare una classifica dei 3 luoghi più immortalati. (SEZIONE 4.2)

7. Gestione dei video: Ogni fotografia può essere visualizzata in sequenza, andando a formare quindi un video.

8. Sicurezza: Il sistema deve garantire la sicurezza delle informazioni, in particolare la gestione degli account degli utenti e la protezione dei dati personali. Inoltre, il sistema deve essere in grado di gestire la concorrenza tra utenti per evitare problemi di accesso simultaneo alle stesse informazioni.

Il sistema prevede che le categorie di utenti siano così rappresentate:

- **Utente:** Può effettuare i punti dal 1 al 7 (In particolare esiste un servizio riservato al ruolo di amministratore nel punto 5).
- **Amministratore:** Può effettuare il punto 5 e avere accesso agli stessi servizi di Utente.

Capitolo 2

Modello Concettuale

2.1 Passaggio dal minimondo al modello concettuale

Il mini-world è una rappresentazione semplificata del mondo reale che ci fa intuire le informazioni che devono essere salvate nel database. Superata la prima fase che consiste quindi nella raccolta e analisi dei requisiti, la seconda fase consiste nel tradurre questi ultimi in entità e relazioni mediante un costrutto grafico. In seguito alla sezione precedente si è costruito il seguente schema concettuale espresso mediante il Diagramma ER:

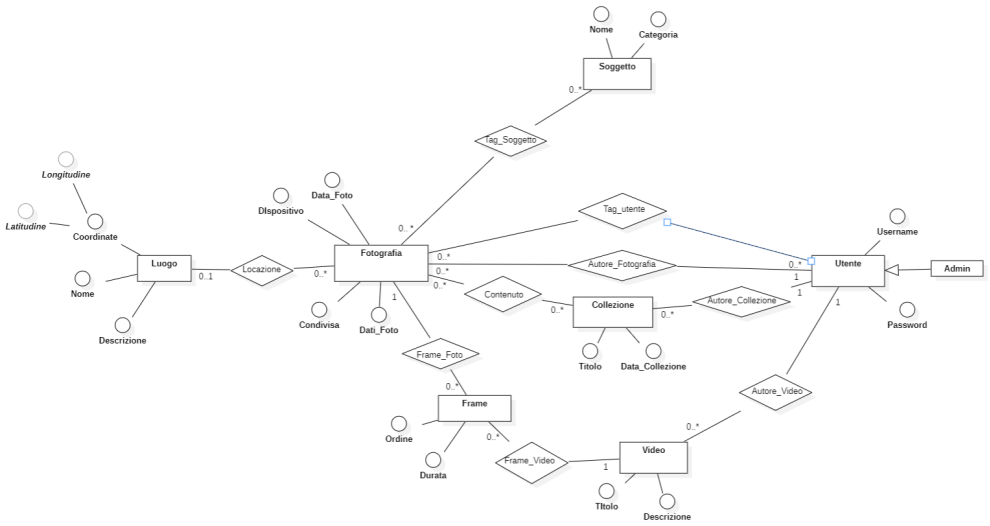

Figura 2.1: Diagramma ER

2.2 Ristrutturazione del modello concettuale

La terza fase della progettazione di un sistema di basi di dati consiste nel passaggio dal modello concettuale al modello logico. La ristrutturazione del modello concettuale è un processo che si occupa di migliorare il modello

concettuale, in questo caso quello precedentemente realizzato, per renderlo più efficiente e aggiornato alle esigenze degli utenti. Lo abbiamo realizzato nel seguento modo:

2.2.1 Analisi delle ridondanze

Le ridondanze sono costituite da dati ricavabili tramite operazioni da altri dati presenti nello schema. Nel modello realizzato precedentemente non sono presenti ridondanze, tuttavia introduciamo tre attributi derivati:

1. `Collezione.numero_elementi`
2. `Video.numero_frames`
3. `Video.durata`

Notiamo che essi sono ridondanti in quanto `collezione.numero_elementi` e `video.numero_frames` che corrispondono al numero d'istanze di associazione di tipo **contenuto** e **contenuto video** per una certa collezione o video, mentre `video.durata` corrisponde alla somma delle durate di tutti i frame contenuti in un certo video. Andremo ad implementare in modo efficiente tali attributi tramite i trigger presenti nella seguente sezione -> 4.3.1

2.2.2 Analisi delle Generalizzazioni

In un modello di dati, una relazione di generalizzazione/specializzazione rappresenta un'entità generica (superclasse) che ha una o più sottoclassi (entità specializzate) con attributi specifici aggiuntivi. Nel modello concettuale realizzato è presente un'unica istanza di specializzazione, ovvero `Admin->Utente`. Decidiamo di accorpare `Admin` nell'entità padre come attributo Booleano.

2.2.3 Rimozione di Attributi Multivalori e Composti

Gli attributi multivalore sono attributi che possono contenere più di un valore per ogni tupla, mentre gli attributi composti sono attributi che contengono al loro interno più di un sotto-attributo o campo dati.

Nel modello concettuale è presente solo un attributo composto, ovvero `coordinate(Latitudine, Longitudine)`. Per ristrutturarlo scomponiamo `coordinate` rendendo i due sotto-attributi in due attributi indipendenti.

2.2.4 Analisi di Entità e Associazioni

L'analisi delle Entità e Associazioni prevede di valutare i requisiti del sistema e l'attuale modello del database per identificare eventuali problemi di

progettazione o inefficienze. Nel modello concettuale, Video è espresso come composizione di `frame`. Ristrutturiamo l'associazione come associazione identificante e `frame` come entità debole di `Video`.

2.2.5 Analisi degli Identificativi

Una chiave primaria è un attributo o un insieme di attributi che identifica univocamente ogni tupla di una tabella del database. Elenchiamo gli identificativi per ogni entità:

- **Fotografia:** La traccia richiede che ogni fotografia sia identificata univocamente, introduciamo quindi `id_foto`.
- **Luogo:** Quanto richiesto, il sistema permette di identificare un luogo tramite le coordinate geografiche, scegliamo come chiave primaria gli attributi `Latitudine` e `Longitudine`.
- **Collezione:** Viene specificato che l'applicativo debba permettere ad ogni utente di accedere a gallerie condivise da altri utenti, pertanto si è scelto di aggiungere l'identificativo `id_collezione`, in quanto gli attributi presenti nella tabella potrebbero essere inefficienti.
- **Utente:** Scegliamo `username` come chiave primaria.
- **Soggetto:** Scegliamo `nome` come chiave primaria.
- **Video:** Introduciamo l'identificante `id_video` seguendo la stessa logica adottata per l'entità `Video`.
- **Frame:** L'entità `Frame` essendo un'entità debole non ha un identificatore primario, abbiamo trovato opportuno utilizzare `ordine` come chiave parziale.

2.3 Diagramma Ristrutturato

2.3.1 Diagramma ER

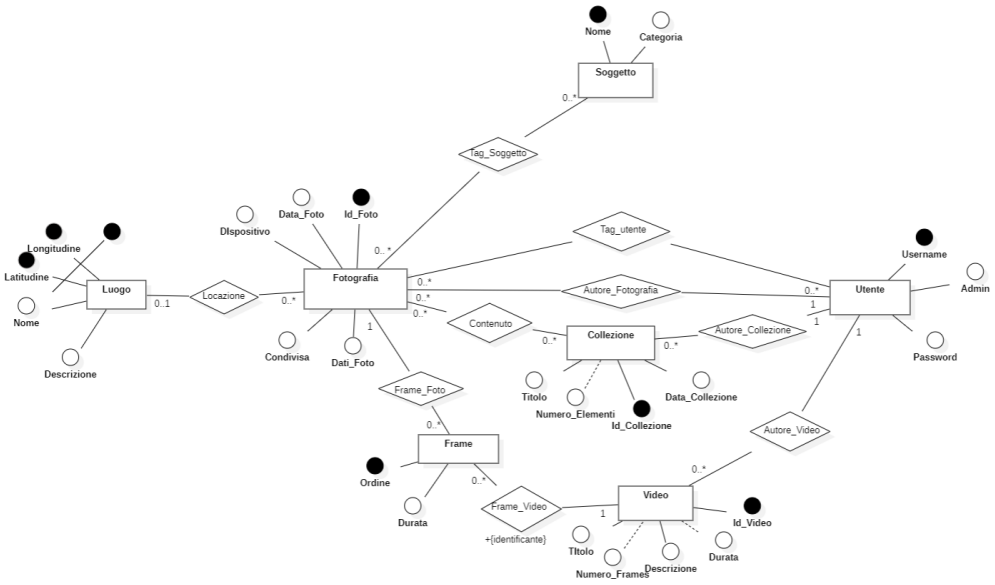

Figura 2.2: Diagramma ER Ristrutturato

2.3.2 Diagramma UML

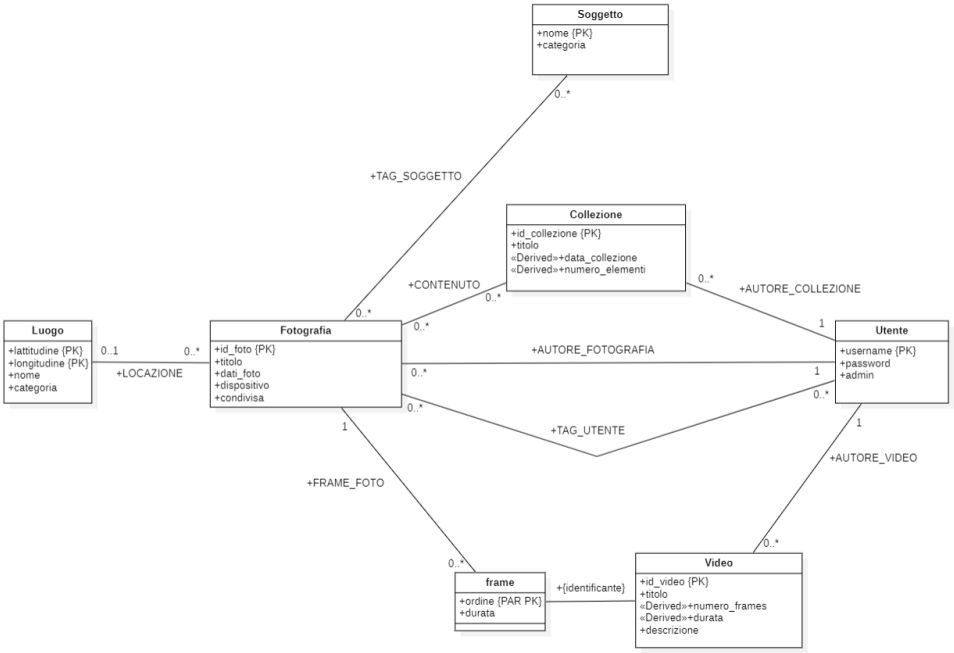

Figura 2.3: Diagramma UML Ristrutturato

2.4 Dizionario delle Entità

Fotografia	**E' l'entità principale del nostro database, specifica tutte le caratteristiche legate alle fotografie**
ID_Foto:	L'identificativo primario di ogni foto
Titolo:	Titolo con cui è stata salvata la foto, per riconoscere il contenuto in modo immediato.
Dati_foto:	Sono i dati di codifica di ogni fotografia, essendo unici è una possibile chiave primaria.
Dispositivo:	Specifica il dispositivo utilizzato per scattare la fotografia.
Condivisa:	E' il flag per gestire la privacy delle fotografie.
Username_Autore:	Chiave esterna dell'username che ha scattato la foto all'interno di Fotografia, fa riferimento alla relazione "Autore_Fotogafia".
Latitudine e Longitudine:	Sono chiavi esterne della tabella Luogo, fanno riferimento alla relazione "Locazione".
Luogo	**Luogo è l'entità che specifica la locazione dello scatto delle fotografia**
Latitudine:	Coordinata di Latitudine, usata come identificativo.
Longitudine:	Coordinata di Longitudine, usata come identificativo.
Nome:	Nome mnemonico, serve per facilitare all'utente il riconoscimento immediato del luogo dello scatto.
Descrizione:	Descrizione dettagliata del luogo, poiché lo stesso luogo può essere molto ampio, c'è bisogno di una descrizione.
Collezione	**Collezione è l'entità dove andranno inserite tutte le fotografie condivise**
ID_Collezione:	Identificativo univoco della Collezione
Username:	Chiave Esterna della tabella "Utente", specifica l'autore che ha creato la Galleria Condivisa
Titolo:	Titolo che sintetizza brevemente il contenuto della Galleria
Data_Collezione:	Data di creazione della Galleria
Numero_Elementi:	Numero di fotografie che sono all'interno della Galleria
Soggetto	**Soggetto è l'entità che specifica il soggetto di una fotografia**
Nome:	Nome generico associato ad un soggetto, utilizzato come identificativo.
Categoria:	Categoria d'appartenenza del soggetto.

Tabella 2.1: Dizionario delle entità (Parte 1)

Utente	**Utente è l'entità che specifica gli utenti dell'applicativo**
Username:	Username unico per ogni utente, così viene identificato all'interno del sistema.
Password:	Password dell'utente, indispensabile per poter accedere al sistema.
Admin:	Attributo booleano che conferisce il ruolo di amministratore di sistema ad un utente.
Video	**Video è l'entità che identifica i video creati dagli utenti**
ID_Video:	Identificativo del Video
Autore:	Chiave esterna della tabella "Utente", specifica l'autore del Video.
Titolo:	Titolo del video che ne sintetizza brevemente il contenuto.
Numero_Frames:	Il numero di frame che compongono il video.
Durata:	Tempo totale del video.
Descrizione:	Descrizione dettagliata del contenuto del video.
Frame	**Frame è l'entità che si frappone tra "Video" e "Fotografia", serve ad associare ogni frame singolo, composto da una fotografia, ad un Video**
ID_Video:	Chiave esterna della tabella "Video" e chiave primaria composta.
Ordine:	Indice in cui è localizzato il frame all'interno del Video, Chiave primaria composta.
ID_Foto:	Chiave esterna della tabella "Fotografia",
Durata:	Specifica il tempo che occupa un determinato frame all'interno di un video.

Tabella 2.2: Dizionario delle entità (Parte 2)

2.5 Dizionario delle Associazioni

Autore_Collezione
Tipologia: Uno a Molti (1:N)
Associa un'Utente con le Collezione da esso create.
Autore_Fotografia
Tipologia: Uno a Molti (1:N)
Associa un'utente alle fotografie che ha scattato.
Autore_Video
Tipologia: Uno a Molti (1:N)
Associa un'utente ai video da esso creati.
Frame_Foto
Tipologia: Uno a Molti (1:N)
Associa un Frame di un Video alla fotografia che rappresenta.
Frame_Video
Tipologia: Uno a Molti (1:N)
Associa un singolo Frame al video di appartenenza.
Locazione
Tipologia: Uno a Molti (1:N)
Associa una fotografia al Luogo in cui è stata scattata.
Contenuto
Tipologia: Molti a Molti (N:M)
Associa Le Collezioni con le Fotografie che esse contengono.
Tag_Soggetto
Tipologia: Molti a Molti (N:M)
Associa i Soggetti con le fotografie in cui sono taggati.
Tag_Utente
Tipologia: Molti a Molti (N:M)
Associa Un'Utente alle fotografie in cui è taggato.

Tabella 2.3: Dizionario delle Associazioni

2.6 Dizionario dei Vincoli

Vincolo d'integrità Referenziale
Nelle tabelle molti a molti come "Tag_Utente", "Tag_Soggetto" e "contenuto" bisogna mantenere la coerenza tra le tabelle che mettono in relazione durante l'inserimento o aggiornamento dei dati.
Vincoli Amministratore
"L'amministratore del sistema può eliminare un utente: in tal caso, tutte le foto dell'utente verranno cancellate dalla libreria, eccetto quelle che contengono come soggetto un altro degli utenti della galleria condivisa." Deve esistere almeno un amministratore. Quando un'amministratore elimina un utente, tutte le foto che contengono almeno un tag_utente di altre persone vengono conservate con autore nullo.
Vincoli D'autore
Fotografie, Video e Collezioni devono sempre avere un Autore inizialmente. Quando un'utente sarà eliminito verrà rispettato il *"Vincolo Amministatore"*
Vincoli Di Privacy
"Un utente può eliminare una foto ed in tal caso, questa non verrà più vista nella sua galleria, ma resterà disponibile nelle gallerie degli altri utenti." Quando una fotografia passa dallo stato "pubblico" allo stato "privato" sarà eliminata da tutte le Collezioni. Quando un'utente elimina una fotografia pubblica, se essa è presente in una Collezione non verrà eliminata all'interno di essa. Utilizzeremo lo stesso criterio per quanto riguarda la gestione dei Video. Un utente può incorporare fotografie all'interno di collezioni o video solo se è l'autore di tali fotografie o se le fotografie sono state condivise.

Tabella 2.4: Dizionario dei Vincoli (Parte 1)

Capitolo 3

Progettazione Logica

3.1 Passaggio dal Diagramma ER al Modello Logico

Il passaggio dal modello concettuale al modello logico implica la traduzione di ogni entità e relazione nel modello concettuale in una tabella nel modello logico. Analizziamo le relazioni del nostro applicativo. Nel nostro database, possiamo trovare due tipi principali di relazioni: Uno a Molti (1:N) e Molti a Molti (N:M). Le relazioni Uno a Molti implicano che un record in una tabella può essere associato a molti record in un'altra tabella, mentre le relazioni Molti a Molti indicano che molti record in una tabella possono essere associati a molti record in un'altra tabella. L'implementazione di queste relazioni verrà poi esplicata all'interno del (Mapping 3.2).

- **Relazioni Uno a Molti (1:N):** In questo tipo di relazione, un record in una tabella può essere associato a molti record in un'altra tabella. Per implementare queste relazioni, generalmente viene utilizzata una chiave esterna nella tabella "molti" che fa riferimento alla chiave primaria della tabella "uno".

  - **Autore_Collezione**: La relazione tra la tabella "utente" e la tabella "collezione" è una relazione Uno a Molti (1:N), in cui un utente può creare più collezioni condivise, ma ogni collezione può essere creata da un solo utente.
    Questa relazione è **parziale** poichè un utente può non avere collezioni, quindi non tutti gli utenti devono necessariamente avere una collezione correlata.
  - **Autore_Fotografia**: La relazione tra la tabella "utente" e la tabella "fotografia" è una relazione Uno a Molti (1:N), in cui un utente può aver scattato molte fotografie, ma ogni fotografia è stata scattata da solo utente.
    Questa relazione è **parziale** poichè non tutti gli utenti devono necessariamente avere fotografie correlate.

- **Autore_Video**: La relazione tra la tabella "utente" e la tabella "video" è una relazione Uno a Molti (1:N), in cui un utente può aver creato molti video, ma ogni video è stato creato da un unico utente.
  Questa relazione è **parziale** poichè un utente può non avere video, quindi non tutti gli utenti devono necessariamente avere un video correlato.
- **Frame_Foto**: Un frame può fare riferimento ad una sola fotografia, ma una fotografia può essere utilizzata in molti frame di diversi video.
  Questa associazione è di tipo molti-a-uno (N:1). Questa relazione è **parziale** poichè non tutte le foto devono necessariamente essere utilizzate in un frame.
- **Frame_Video**: Un video può avere molti frame, e ogni frame appartiene ad un solo video. Questa associazione è di tipo uno-a-molti (1:N).
  Questa relazione è **totale** poichè ogni video deve avere almeno un frame, quindi ogni entità nel lato "uno" (video) ha almeno una entità correlata nel lato "molti" (frame).
- **Locazione**: La relazione tra la tabella "luogo" e la tabella "fotografia" è una relazione Uno a Molti (1:N), in cui un luogo può essere associato a molte fotografie, ma ogni fotografia è associata a un solo luogo.
  Questa relazione è **parziale** poichè non tutte le foto devono necessariamente essere correlate a un luogo. Questo è visto anche nella traccia: *"Per ogni fotografia, identificata da un identificativo è necessario specificare l'utente che l'ha scattata, il dispositivo con cui è stata scattata e,* ***se necessario, il luogo in cui è stata scattata.****"*

- **Relazioni Molti a Molti (N:M):** In questo tipo di relazione, molti record in una tabella possono essere associati a molti record in un'altra tabella. Per implementare queste relazioni, viene utilizzata una tabella di associazione che contiene le chiavi primarie delle due tabelle coinvolte come chiavi esterne.

  - **Contenuto**: La relazione tra la tabella "Collezione" e la tabella "Fotografia" è una relazione Molti a Molti (N:M), in cui una collezione può contenere molte fotografie e una fotografia può essere contenuta in molte collezioni.
  - **Tag_Soggetto**: La relazione tra la tabella "soggetto" e la tabella "fotografia" è una relazione Molti a Molti (N:M), in cui un soggetto può avere molti tag e un tag può essere associato a molti soggetti.
  - **Tag_Utente**: La relazione tra la tabella "fotografia" e la tabella "utente" è una relazione Molti a Molti (N:M), in cui una

fotografia può avere molti tag degli utenti e un utente può essere associato come tag a molte fotografie.

3.2 Mapping delle Entità e delle Associazioni

Legenda: Chiave Primaria, Chiave Esterna$^{\uparrow}$, Attributo Derivato

- **UTENTE**(Username, Password, Admin)

- **LUOGO**(Latitudine, Longitudine, Nome, Descrizione)

- **FOTOGRAFIA**(Id_foto, Username_autore$^{\uparrow}$, Dati_foto, Dispositivo, Data_foto, Latitudine$^{\uparrow}$, Longitudine$^{\uparrow}$, Condivisa, Titolo)

  Username_autore → UTENTE.Username

  Latitudine → LUOGO.Latitudine

  Longitudine → LUOGO.Longitudine

- **COLLEZIONE** (Id_collezione, Username$^{\uparrow}$, Titolo, Data_collezione, Numero_elementi)

  Username → UTENTE.Username

- **CONTENUTO**(Id_collezione$^{\uparrow}$, Id_foto$^{\uparrow}$)

  Id_collezione → COLLEZIONE.Id_collezione

  Id_foto → FOTOGRAFIA.Id_foto

- **TAG_UTENTE**(Username$^{\uparrow}$, Id_foto$^{\uparrow}$)

  Username → UTENTE.Username

  Id_foto → FOTOGRAFIA.Id_foto

- **VIDEO**(Id_video, Autore$^{\uparrow}$, Titolo, Numero_frames, Durata, Descrizione)

  Autore → UTENTE.Username

- **FRAME**(Id_video$^{\uparrow}$, Id_foto$^{\uparrow}$, durata, ordine)

  Id_video → VIDEO.Id_video
  Id_foto → FOTOGRAFIA.Id_foto

- **SOGGETTO**(Nome, Categoria)

- **TAG_SOGGETTO** (nome_soggetto$^{\uparrow}$, Id_foto $^{\uparrow}$)

  Nome_soggetto → SOGGETTO.Nome
  Id_foto → FOTOGRAFIA.Id_foto

Capitolo 4

Codice SQL

4.1 Tabelle

In questa sezione sono riportati gli scripts inerenti alle tabelle che compongono il database. Sono presenti alcuni commenti inerenti alla struttura e scelte di implemetazione. Per ulteriori informazioni inerenti alle tabelle e il loro scopo si visiti la seguente sezione -> SEZIONE 2.4

```
1 CREATE TABLE IF NOT EXISTS utente (
2     username VARCHAR(30) NOT NULL,
3     password VARCHAR(30) NOT NULL,
4     admin BOOLEAN NOT NULL DEFAULT false,
5
6     CONSTRAINT utente_pk PRIMARY KEY (username)
7 );
```

```
1 CREATE TABLE IF NOT EXISTS luogo (
2     latitudine  FLOAT NOT NULL,
3     longitudine FLOAT NOT NULL,
4     nome VARCHAR(50) UNIQUE,
5     descrizione VARCHAR(225),
6
7     CONSTRAINT luogo_pk PRIMARY KEY (latitudine,
          longitudine)
8 );
```

Si è scelto di impostare il vincolo `UNIQUE` al nome di un luogo in quanto non è possibile avere vari luoghi dotati dello stesso nome ma è possibile avere luoghi privi di nome.

```
1 CREATE TABLE IF NOT EXISTS fotografia (
2     id_foto SERIAL NOT NULL,
3     username_autore VARCHAR(30),
4     dati_foto BYTEA,
```

```
5       dispositivo VARCHAR(30) NOT NULL DEFAULT
            'Sconosciuto',
6       data_foto TIMESTAMP NOT NULL DEFAULT
            CURRENT_TIMESTAMP,
7       latitudine FLOAT,
8       longitudine FLOAT,
9       condivisa BOOLEAN NOT NULL default false,
10
11      CONSTRAINT fotografia_pk PRIMARY KEY (id_foto),
12      CONSTRAINT fotografia_autore_fk FOREIGN KEY
            (username_autore) REFERENCES
            utente(username) ON DELETE CASCADE,
13      titolo VARCHAR(30) NOT NULL default 'foto.jpg',
14      CONSTRAINT fotografia_luogo_fk FOREIGN KEY
            (latitudine, longitudine) REFERENCES
            luogo(latitudine,longitudine)
15  );
```

Le fotografie presentano il vincolo `ON DELETE CASCADE`, quindi sono eliminate a cascata se l'autore viene eliminato. La richiesta della specifica, ovvero che non vengano eliminate fotografie che presentano un tag utente, è stato implementato tramite il seguente trigger -> 4.3.3

```
1   CREATE TABLE IF NOT EXISTS collezione (
2       id_collezione INTEGER NOT NULL,
3       username VARCHAR(30) NOT NULL,
4       titolo VARCHAR(30) NOT NULL,
5       data_collezione TIMESTAMP NOT NULL DEFAULT
            CURRENT_TIMESTAMP,
6       numero_elementi INTEGER NOT NULL DEFAULT 0,
7
8       CONSTRAINT collezione_pk PRIMARY KEY
            (id_collezione),
9       CONSTRAINT collezione_utente_fk FOREIGN KEY
            (username) REFERENCES utente(username) ON
            DELETE CASCADE
10  );
```

```
1   CREATE TABLE IF NOT EXISTS contenuto(
2       id_collezione INTEGER NOT NULL
3       id_foto INTEGER NOT NULL
4
5       CONSTRAINT contenuto_pk PRIMARY KEY
            (id_collezione, id_foto
6       CONSTRAINT contenuto_collezione_fk FOREIGN KEY
            (id_collezione) REFERENCES
            collezione(id_collezione) ON DELETE CASCADE,
```

```
7       CONSTRAINT contenuto_fotografia_fk FOREIGN KEY
            (id_foto) REFERENCES fotografia(id_foto) ON
            DELETE CASCADE
8   );
```

```
1   CREATE TABLE IF NOT EXISTS tag_utente(
2       username VARCHAR(30) NOT NULL,
3       id_foto INTEGER NOT NULL,
4
5       CONSTRAINT tag_utente_pk PRIMARY KEY (username,
            id_foto),
6       CONSTRAINT tagutente_utente_fk FOREIGN
            KEY(username) REFERENCES utente(username) ON
            DELETE CASCADE,
7       CONSTRAINT tagutente_fotografia_fk FOREIGN
            KEY(id_foto) REFERENCES fotografia(id_foto)
            ON DELETE CASCADE
8   );
```

```
1   CREATE TABLE IF NOT EXISTS video (
2       id_video SERIAL NOT NULL,
3       autore VARCHAR(30) NOT NULL,
4       titolo VARCHAR(30) NOT NULL DEFAULT 'video.mp4',
5       numero_frames INTEGER NOT NULL DEFAULT 0,
6       durata INTEGER NOT NULL DEFAULT 0,
7       descrizione VARCHAR(225),
8
9       CONSTRAINT video_pk PRIMARY KEY (id_video),
10      CONSTRAINT video_autore_fk FOREIGN KEY (autore)
            REFERENCES utente(username) ON DELETE CASCADE
11  );
```

```
1   CREATE TABLE IF NOT EXISTS frame(
2       id_video INTEGER NOT NULL,
3       id_foto INTEGER,
4       durata INTEGER NOT NULL DEFAULT 0,
5       ordine INTEGER NOT NULL DEFAULT 0,
6
7       CONSTRAINT frame_pk PRIMARY KEY (id_video,
            ordine),
8       CONSTRAINT frame_video_fk FOREIGN KEY
            (id_video) REFERENCES video(id_video) ON
            DELETE CASCADE,
9       CONSTRAINT frame_fotografia_fk FOREIGN KEY
            (id_foto) REFERENCES fotografia(id_foto) ON
            DELETE SET NULL
10  );
```

```
CREATE TABLE IF NOT EXISTS soggetto (
    nome VARCHAR(30) NOT NULL,
    categoria VARCHAR(30) NOT NULL DEFAULT '-',

    CONSTRAINT soggetto_pk PRIMARY KEY (nome)
);
```

```
CREATE TABLE IF NOT EXISTS tag_soggetto(
    nome_soggetto VARCHAR(30) NOT NULL,
    id_foto INTEGER NOT NULL,
    CONSTRAINT tag_soggetto_pk PRIMARY KEY
       (nome_soggetto, id_foto),
    CONSTRAINT tagsottetto_soggetto_fk FOREIGN KEY
       (nome_soggetto) REFERENCES soggetto(nome),
    CONSTRAINT tagsoggetto_fotografia_fk FOREIGN
       KEY (id_foto) REFERENCES fotografia(id_foto)
);
```

4.2 View

Le viste utente possono nascondere dettagli complessi o sensibili delle tabelle del database, fornendo una visione focalizzata e personalizzata dei dati. Abbiamo quindi implementato le user views in SQL, tramite il comando `CREATE VIEW`, e in funzioni plpgsql.

```
CREATE OR REPLACE FUNCTION GalleriaUtente (Utente
   utente.username%TYPE, Richiedente
   utente.username%TYPE) RETURNS SETOF fotografia AS
$$
BEGIN
    RETURN QUERY (
        SELECT * FROM fotografia
        WHERE fotografia.username_autore = Utente
           AND (
            fotografia.username_autore =
               Richiedente OR fotografia.condivisa
               = TRUE
        )
    );
END;
$$ LANGUAGE plpgsql;
```

La funzione "GalleriaUtente()" è una funzione in linguaggio PL/pgSQL (PostgreSQL) che restituisce un insieme di fotografie pubbliche. Nel caso in cui l'utente sia anche l'autore, visualizzerà sia le foto condivise e sia le foto private.

```
CREATE OR REPLACE FUNCTION CollezioniUtente(utente
   utente.username%TYPE) RETURNS SETOF collezione AS
$$
BEGIN
    RETURN QUERY (
        SELECT * FROM collezione
        WHERE collezione.username = utente
    );
END;
$$LANGUAGE plpgsql;
```

La funzione "CollezioniUtente()" è una funzione in linguaggio PL/pgSQL (PostgreSQL) che restituisce un insieme di righe della tabella "collezione", stilando una lista dei vari album fotografici di un utente.

```
CREATE OR REPLACE FUNCTION ContenutoCollezione
   (collezione collezione.id_collezione%TYPE)
   RETURNS SETOF fotografia AS
$$
BEGIN
    RETURN QUERY(
        SELECT fotografia.*
        FROM contenuto NATURAL JOIN fotografia
        WHERE id_collezione = collezione
    );
END;
$$ LANGUAGE plpgsql;
```

La funzione "ContenutoCollezione()" è una funzione in linguaggio PL/pgSQL (PostgreSQL) che restituisce un insieme di fotografie contenute nella collezione indicata.

```
CREATE OR REPLACE VIEW ClassificaLuoghi AS
SELECT latitudine, longitudine, nome, descrizione,
   COUNT(id_foto) AS NumeroFotografie
FROM luogo NATURAL LEFT JOIN fotografia
GROUP BY latitudine, longitudine, nome, descrizione
ORDER BY NumeroFotografie DESC, nome ASC
LIMIT 3;
```

La seguente vista corrisponde alla richiesta dalla specifica, ovvero che fosse fornita la classifica dei 3 luoghi più immortalati. Abbiamo quindi raggruppato le foto per luogo, ordinate per numero di scatti e mostrato appunto solo 3 luoghi con il comando `LIMIT 3`.

```
CREATE OR REPLACE FUNCTION foto_per_luogo(in_nome
   luogo.nome%TYPE, utente
   fotografia.username_autore%TYPE)
RETURNS SETOF fotografia AS
$$
BEGIN
    RETURN QUERY (
        SELECT fotografia.*
        FROM fotografia
        JOIN luogo ON fotografia.latitudine =
           luogo.latitudine AND
           fotografia.longitudine =
           luogo.longitudine
        WHERE luogo.nome = in_nome AND (
            fotografia.username_autore = utente OR
               fotografia.condivisa = TRUE
        )
    );
END;
$$ LANGUAGE plpgsql;
```

La funzione "foto_per_luogo()" è una funzione in linguaggio PL/pgSQL (PostgreSQL) che restituisce un insieme di fotografie scattate nel luogo indicato.

```
CREATE OR REPLACE FUNCTION
   foto_per_tag_utente(in_username
   tag_utente.username%TYPE)
RETURNS SETOF fotografia AS
$$
BEGIN
    RETURN QUERY (
        SELECT fotografia.*
        FROM fotografia
        JOIN tag_utente ON tag_utente.id_foto =
           fotografia.id_foto
        WHERE tag_utente.username = in_username
    );
END;
$$ LANGUAGE plpgsql;
```

La funzione "foto_per_tag_utente()" è una funzione in linguaggio PL/-pgSQL (PostgreSQL) che restituisce un insieme di fotografie aventi come tag_utente l'utente indicato.

```
CREATE OR REPLACE FUNCTION
   foto_per_tag_soggetto(in_nome_soggetto
   tag_soggetto.nome_soggetto%TYPE,
```

```
    utente_richiedente
    fotografia.username_autore%TYPE)
RETURNS SETOF fotografia AS
$$
BEGIN
    RETURN QUERY (
        SELECT fotografia.*
        FROM fotografia
        JOIN tag_soggetto ON tag_soggetto.id_foto =
           fotografia.id_foto
        WHERE tag_soggetto.nome_soggetto =
           in_nome_soggetto AND (
            fotografia.username_autore=
               utente_richiedente OR
               fotografia.condivisa = TRUE)
    );
END;
$$ LANGUAGE plpgsql;
```

La funzione "foto_per_tag_soggetto()" è una funzione in linguaggio PL/-pgSQL (PostgreSQL) che restituisce un insieme di fotografie aventi come tag_soggetto il nome del soggetto indicato.

```
CREATE OR REPLACE FUNCTION
   visualizza_video(in_titolo video.titolo%TYPE)
RETURNS SETOF frame AS
$$
BEGIN
    RETURN QUERY (
        SELECT frame.*
        FROM frame
        JOIN video ON video.id_video =
           frame.id_video
        WHERE video.titolo = in_titolo
        ORDER BY ordine
    );
END;
$$ LANGUAGE plpgsql;
```

La funzione "visualizza_video()" è una funzione in linguaggio PL/pgSQL (PostgreSQL) che restituisce un insieme di frame ordinati in sequenza del video indicato, andando quindi a visualizzare il video.

In quest'ultima parte della sezione vengono definite alcune viste aggiunte per una rapida visualizzazione del database.

```
--View per mostrare gli utenti dell'applicativo
CREATE OR REPLACE VIEW ShowUser AS
SELECT DISTINCT username
FROM utente;
```

```
--View per mostrare gli admin dell'applicativo
CREATE OR REPLACE VIEW ShowAdmin AS
SELECT DISTINCT username, admin
FROM utente
WHERE admin = true;
```

```
--View per visualizzare i dati di ogni frame
   utilizzato in almeno un video
CREATE OR REPLACE VIEW ContenutoFrame AS
SELECT fotografia.dati_foto, frame.*
FROM fotografia
INNER JOIN frame ON fotografia.id_foto =
   frame.id_foto;
```

```
--View per visualizzare una lista delle categorie
   dei soggetti nel database
CREATE OR REPLACE VIEW CategoriaSoggetto AS
SELECT DISTINCT soggetto.categoria
FROM soggetto;
```

```
--View per lista di tutti i video
CREATE OR REPLACE VIEW ShowVideos AS
SELECT titolo AS "Titolo", autore AS "Autore",
   descrizione AS "Info"
FROM video;
```

4.3 Trigger

In questa sezione vengono rappresentati i trigger necessari per l'implementazione di vincoli e altre caratteristiche della base di dati. Si noti che ogni trigger è composto dal trigger in sè e la funzione che lo esegue.

4.3.1 Attributi Derivati

```
CREATE OR REPLACE FUNCTION
   aggiorna_elementi_galleria() RETURNS TRIGGER AS
   $$
BEGIN
    IF (TG_OP = 'INSERT') THEN
```

```
4           UPDATE collezione SET numero_elementi =
                numero_elementi + 1 WHERE id_collezione
                = NEW.id_collezione;
5       ELSIF (TG_OP = 'DELETE') THEN
6           UPDATE collezione SET numero_elementi =
                numero_elementi - 1 WHERE id_collezione
                = OLD.id_collezione;
7       END IF;
8       RETURN NULL;
9   END;
10  $$ LANGUAGE PLPGSQL;
```

```
1   CREATE TRIGGER aggiorna_elementi_galleria_trigger
2   AFTER INSERT OR DELETE ON contenuto
3   FOR EACH ROW
4   EXECUTE FUNCTION aggiorna_elementi_galleria();
```

La funzione "aggiorna_elementi_galleria()" viene eseguita ogni volta che viene attivato il trigger. La funzione aggiorna il numero di elementi di una collezione dopo l'inserimento di un nuovo elemento o l'eliminazione .

```
1   CREATE OR REPLACE FUNCTION
       update_video_frame_count() RETURNS TRIGGER AS $$
2   BEGIN
3       IF (TG_OP = 'INSERT') THEN
4           UPDATE video SET numero_frames =
                numero_frames + 1 WHERE id_video =
                NEW.id_video;
5       ELSIF (TG_OP = 'DELETE') THEN
6           UPDATE video SET numero_frames =
                numero_frames - 1 WHERE id_video =
                OLD.id_video;
7       END IF;
8       RETURN NULL;
9   END;
10  $$ LANGUAGE plpgsql;
```

```
1   CREATE TRIGGER update_video_frame_count_trigger
2   AFTER INSERT OR DELETE ON frame
3   FOR EACH ROW
4   EXECUTE FUNCTION update_video_frame_count();
```

La funzione "update_video_frame_count()" viene eseguita ogni volta che viene attivato il trigger. La funzione aggiorna il numero di frame totali di un video dopo l'inserimento di un nuovo elemento o l'eliminazione.

```
CREATE OR REPLACE FUNCTION generate_frame_order()
   RETURNS TRIGGER AS $$
BEGIN
    NEW.ordine := (SELECT COALESCE(MAX(ordine), 0)
       FROM frame WHERE id_video = NEW.id_video) +
       1;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER frame_order_trigger
BEFORE INSERT ON frame
FOR EACH ROW
EXECUTE FUNCTION generate_frame_order();
```

La funzione "generate_frame_order()" viene eseguita ogni volta che viene attivato il trigger. La funzione aggiunge l'ordine al nuovo frame inserito.

```
CREATE OR REPLACE FUNCTION update_video_duration()
   RETURNS TRIGGER AS $$
BEGIN
  UPDATE video
  SET durata = (
    SELECT SUM(durata)
    FROM frame
    WHERE id_video = NEW.id_video
  )
  WHERE id_video = NEW.id_video;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER update_video_duration
AFTER INSERT OR UPDATE ON frame
FOR EACH ROW
EXECUTE FUNCTION update_video_duration();
```

La funzione "update_video_duration()" viene eseguita ogni volta che viene attivato il trigger. La funzione aggiorna la durata totale di un video dopo l'inserimento di un frame.

4.3.2 Vincolo di Privacy

I vincoli di Privacy implementati consentono di inserire fotografie private a collezioni o video se e solo se l'autore della fotografia è anche il possessore della collezione o l'autore del video.

```
1  CREATE OR REPLACE FUNCTION controllo_autore()
       RETURNS TRIGGER AS $$
2  DECLARE
3  proprietario_foto VARCHAR;
4  proprietario_collezione VARCHAR;
5  BEGIN
6      SELECT username_autore INTO proprietario_foto
           FROM fotografia WHERE id_foto=NEW.id_foto;
7      SELECT username INTO proprietario_collezione
           FROM collezione WHERE
           id_collezione=NEW.id_collezione;
8      IF NOT EXISTS (SELECT * FROM fotografia WHERE
           id_foto = NEW.id_foto AND ((condivisa =
           true) OR (proprietario_foto =
           proprietario_collezione))) THEN
9           RAISE EXCEPTION 'Non sei autorizzato ad
               utilizzare questa foto.';
10     END IF;
11     RETURN NEW;
12 END;
13 $$ LANGUAGE plpgsql;
```

```
1  CREATE TRIGGER inserimento_Galleria
2      BEFORE INSERT ON Contenuto
3      FOR EACH ROW
4      EXECUTE FUNCTION controllo_autore();
```

La funzione "controllo_autore()" viene eseguita prima dell'inserimento di una riga nella tabella "Contenuto". La funzione verifica se l'autore della fotografia privata che si sta cercando di inserire è anche il possessore della collezione.

```
1  CREATE OR REPLACE FUNCTION
       controllo_privacy_video() RETURNS TRIGGER AS $$
2  DECLARE
3  proprietario_foto VARCHAR;
4  proprietario_video VARCHAR;
5  BEGIN
6    IF (SELECT condivisa FROM fotografia WHERE
        NEW.id_foto = fotografia.id_foto) = false THEN
7      RAISE EXCEPTION 'Non puoi utilizzare foto
           private per video.';
```

```
8     END IF;
9     RETURN NEW;
10  END;
11  $$ LANGUAGE plpgsql;
```

```
1  CREATE TRIGGER inserimento_frame
2      BEFORE INSERT ON frame
3      FOR EACH ROW
4      EXECUTE FUNCTION controllo_privacy_video();
```

La funzione "controllo_privacy _video()" viene eseguita prima dell'inserimento di una riga nella tabella "Frame". La funzione verifica se la fotografia che si sta cercando di inserire nel video non sia privata.

```
1  CREATE OR REPLACE FUNCTION rimuovi_foto_privata()
       RETURNS TRIGGER AS $$
2  BEGIN
3      DELETE FROM frame WHERE id_foto = OLD.id_foto;
4    RETURN NEW;
5  END;
6  $$ LANGUAGE plpgsql;
```

```
1  CREATE TRIGGER rimuovi_foto_privata_dopo_update
2  AFTER UPDATE OF condivisa ON fotografia
3  FOR EACH ROW WHEN (NEW.condivisa = false)
4  EXECUTE PROCEDURE rimuovi_foto_privata();
```

La funzione "rimuovi_foto _privata()" viene eseguita dopo aer reso privata una fotografia. La funzione elimina la fotografia se essa è presente come frame all'interno di un video.

4.3.3 Vincolo di Amministrazione

```
1  CREATE OR REPLACE FUNCTION verify_admin()
2  RETURNS TRIGGER AS $$
3  BEGIN
4      IF OLD.admin = TRUE AND (SELECT COUNT(*) FROM
           utente WHERE admin = true) = 1 THEN
5          RAISE EXCEPTION 'Non    possibile eliminare
               l''unico utente amministratore';
6      END IF;
7
8      RETURN OLD;
9  END;
10 $$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER verify_admin_trigger
BEFORE DELETE ON utente
FOR EACH ROW
EXECUTE FUNCTION verify_admin();
```

La funzione "verify_admin()" viene eseguita ogni volta che viene attivato il trigger. La funzione non permette di eliminare l'unico admin presente nel database.

```
CREATE OR REPLACE FUNCTION save_tagphoto()
RETURNS TRIGGER AS
$$
BEGIN
    UPDATE fotografia
    SET username_autore = NULL
    WHERE username_autore= OLD.username AND
       condivisa = TRUE AND EXISTS (
        SELECT * FROM tag_utente
        WHERE tag_utente.id_foto =
           fotografia.id_foto AND
           tag_utente.username<>fotografia.username_autore

    );
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER save_tagphoto_trigger
BEFORE DELETE ON utente
FOR EACH ROW
EXECUTE FUNCTION save_tagphoto();
```

La funzione "save_tagphoto()" viene eseguita ogni volta che viene attivato il trigger. La funzione non permette di eliminare le fotografie condivise con taggati utenti presenti nel database.

```
CREATE OR REPLACE FUNCTION riordina_frame() RETURNS
   TRIGGER AS $$
BEGIN
    -- Riduci l'ordine dei frame successivi a
       quello eliminato
    UPDATE frame
    SET ordine = ordine - 1
    WHERE id_video = OLD.id_video AND ordine >
       OLD.ordine;

```

```
8       RETURN NULL; -- il valore di ritorno non
            rilevante per un trigger AFTER DELETE
9   END;
10  $$ LANGUAGE plpgsql;
```

```
1   CREATE TRIGGER riordina_frame_trigger
2   AFTER DELETE ON frame
3   FOR EACH ROW EXECUTE FUNCTION riordina_frame();
```

La funzione "riordina_frame()" viene eseguita ogni volta che viene attivato il trigger. La funzione aggiorna automaticamente l'ordine dei frame successivi dopo l'eliminazione di un determinato frame.