PROGETTAZIONE E SVILUPPO DI UN DATABASE PER LA GESTIONE DI UNA GALLERIA FOTOGRAFICA

Ciro Pizza - N86004131
Vincenzo Riccardo - N86004134
Salvatore Savino - N86004099

16 aprile 2023

Indice

1 INTRODUZIONE

Lo scopo di questo progetto è lo sviluppo di un sistema informativo per la **gestione di collezioni fotografiche condivise**. Il sistema sarà implementato attraverso un database ed un'applicazione Java dotata di GUI, realizzata con la tecnologia JavaFX. In particolare, per il sistema è richiesto che ogni fotografia dovrà contenere informazioni sull'utente che l'ha scattata, il dispositivo utilizzato, il luogo di scatto (identificato da coordinate geografiche o da un nome mnemonico unico) e i soggetti che la caratterizzano. Il sistema consentirà dunque agli utenti di caricare e visualizzare tali fotografie, fornendo anche filtri di ricerca delle immagini in base al loro luogo e al loro soggetto. Funzionalità importanti dovranno essere quelle che permettono all'utente di creare video (una semplice sequenza di alcune sue foto) e quella di partecipare a delle collezioni condivise con altri utenti, in cui possono essere raggruppate sia le proprie foto sia quelle degli altri partecipanti. Inoltre, gli utenti avranno sia la possibilità di gestire la "privacy" delle proprie foto (rendendole cosi esclusivamente personali e quindi non condivisibili), sia di eliminarle in qualsiasi momento (rendendole cosi non più disponibili nella propria galleria personale, senza però cancellarle definitivamente dal sistema qualora fossero presenti in delle collezione condivise). Infine, a gestione dell'utenza, dovrà esserci un amministratore con la facoltà di poter eliminare gli utenti dal sistema; all'eliminazione di un utente dovranno naturalmente essere cancellate tutte le sue foto dal sistema, eccetto quelle che contengono come soggetto un altro degli utenti della galleria condivisa in cui la foto è presente.

In questo documento andremo a trattare il progetto dal punto di vista della base di dati. Per realizzarla, è stato utilizzato un database relazionale sviluppato grazie al **DBMS Postgre**. Il sistema è stato progettato con particolare attenzione alla semplicità, alla facilità d'uso e all'intuitività dell'interfaccia grafica, al fine di rendere l'esperienza dell'utente il più piacevole e soddisfacente possibile. Verrà presentato il progetto nel dettaglio, a partire dalla scelta delle entità e degli attributi, passando per il diagramma ER e quello UML, fino ad arrivare alla scelta dello schema logico e alla descrizione delle varie funzionalità del sistema, implementate attraverso procedure, funzioni e trigger.

2 PROGETTAZIONE CONCETTUALE

2.1 Analisi dei Requisiti

Per la realizzazione del database relazionale richiesto, abbiamo innanzitutto bisogno di individuare i "concetti" del nostro **mini-world**, che useremo poi per andare a creare un diagramma completo; inizieremo rappresentando questi concetti di base attraverso *entità* ed *attributi* del **modello ER** (entità-relazione),

essendo abbastanza intuitivo e di facile comprensione. Al centro del problema, c'è sicuramente il concetto (in questo caso, l'entità) **FOTOGRAFIA**, senza il quale ovviamente non si potrebbe parlare di "galleria fotografica". Per la nostra fotografia, si va allora ad indicare il suo **valore** (che può essere ad esempio il suo path all'interno del computer), il **dispositivo** con il quale è stata scattata (specificando anche alcune sue caratteristiche) e il **luogo** in cui è stata scattata. La fotografia deve essere stata sicuramente scattata da un **utente** (che ne è naturalmente anche il "proprietario"), e può anche contenere diverse categorie di **soggetti** che la raffigurano (ndr: *un utente può essere esso stesso un soggetto, quindi lo si va a specificare*). La fotografia può inoltre essere parte di diversi **video**, ed essere inserita in svariate **collezioni**. Infine, sfruttando il **modello EER** (Enhanced ER) per modellare il nostro miniworld in maniera ancora più dettagliata, si è specificato, attraverso delle sottoclassi, che la foto può essere **privata** (ovvero disponibile solo a sè stessi) o **pubblica** (quindi condivisibile con altre persone), ma anche **eliminata** (inserita in una sorta di cestino) o **non eliminata** (visibile tranquillamente nella propria galleria personale). Si ricorda, inoltre, che gli attributi multivalore vengono indicati con due cerchi concentrici, mentre, per quanto riguarda l'EER, si indica il **vincolo di disgiunzione** con una **"d"** per specificare che una singola foto può essere membro di *al massimo* una sottoclasse, e il **vincolo di completezza totale** con una doppia linea per specificare che ogni foto deve essere *necessariamente* membro di qualche sua sottoclasse.

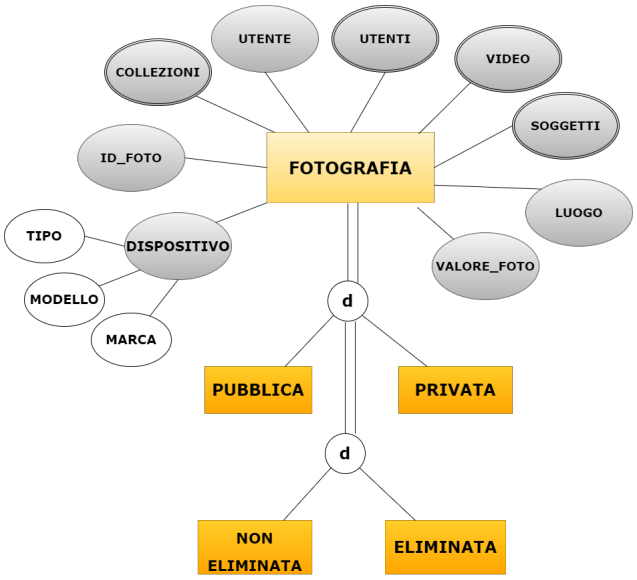

Figura 1: Entità FOTOGRAFIA

Partendo da qui, sicuramente si ha la strada spianata per "scovare" tutte le altre entità. Si è subito pensato all'entità **LUOGO**, specificato semplicemente dal nome di una **città**, e che ovviamente può essere presente in più di una **fotografia**, e all'entità **VIDEO**, creato da un **utente** e sequenza di più

fotografie. Per il video si è inserita la possibilità di poter aggiungere una breve **descrizione**, utile eventualmente per descriverne il suo contenuto.

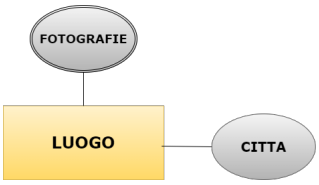

Figura 2: Entità LUOGO

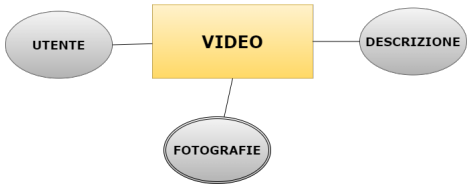

Figura 3: Entità VIDEO

Altra entità che si è presa in considerazione è la **COLLEZIONE**, raggruppamento di una o più **fotografie**, e dunque fulcro del nostro sistema. Da notare che, cosi come può avere più foto, una singola collezione può anche essere "posseduta" da più **utenti** che vogliono poter condividere le loro foto; di conseguenza, si può anche qui sfruttare il modello EER per specificare due diverse sottoclassi di collezioni: **personale**, ad indicare la collezione di default a cui ha accesso esclusivamente il singolo utente, composta quindi dalle sole foto caricate / scattate da lui, e **condivisa**, ad indicare il fatto che la collezione in questione è stata creata in seguito da più utenti (partecipanti ad essa), e che può dunque essere formata da fotografie provenienti da diversi utenti. Inoltre, per quanto riguarda la collezione condivisa, si è pensato alla possibilità di farla assegnare, al momento della sua reazione, un **nominativo** per poterla riconoscere.

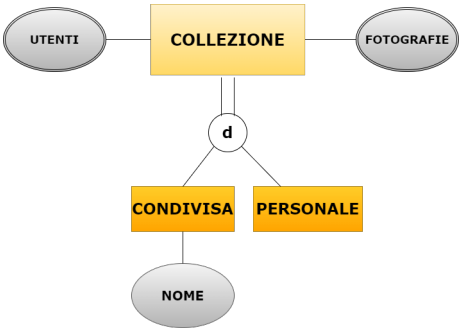

Figura 4: Entità COLLEZIONE

Arrivati a questo punto, si può sicuramente notare che COLLEZIONE, FOTOGRAFIA e VIDEO non possono esistere senza un determinato utente che li possegga, li scatta e li crea. Di conseguenza, non può mancare l'entità **UTENTE**; si può descrivere quest'ultima con delle **informazioni personali** (come nome, cognome ecc...) e delle **credenziali** (come e-mail e password) con il quale può accedere univocamente al sistema.

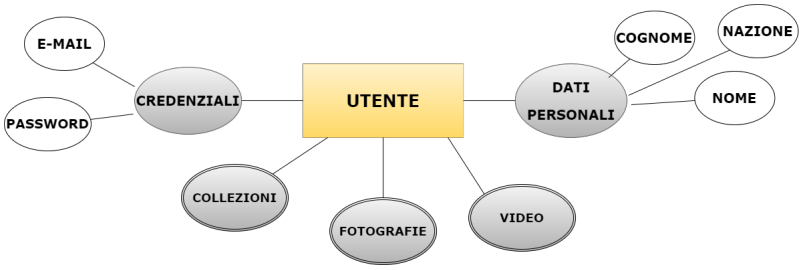

Figura 5: Entità UTENTE

Infine, dato che a gestione degli stessi **utenti** è richiesto un amministratore con facoltà di poterli eliminare in qualsiasi momento, si è pensato, come ultima entità, all'**AMMINISTRATORE DEL SISTEMA**, una sorta di "superutente" che non ha bisogno di specifiche informazioni personali, ma per cui basta e avanza una speciale **password** d'accesso.

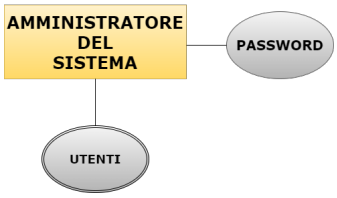

Figura 6: Entità AMMINISTRATORE DEL SISTEMA

2.2 ER Diagram

Una volta trovate le varie astrazioni del nostro mini-world, e modificate intuitivamente in entità con relativi attributi, si può facilmente notare che esistono numerosi *rapporti impliciti* che intercorrono tra loro:

- L'attributo *Utenti* di AMMINISTRATORE si riferisce all'entità UTENTE
- Gli attributi *Collezioni*, *Fotografie* e *Video* di UTENTE si riferiscono alle rispettive entità COLLEZIONE, FOTOGRAFIA e VIDEO

- e cosi via...

Inizialmente si possono rappresentare questi rapporti come attributi, ma, in un modello ER che si rispetti, occorre invece trasformarli in delle **relazioni**, con rispettive **cardinalità**. Di conseguenza, un volta effettuato questo passaggio, ci si ritrova con il seguente ER diagram finale:

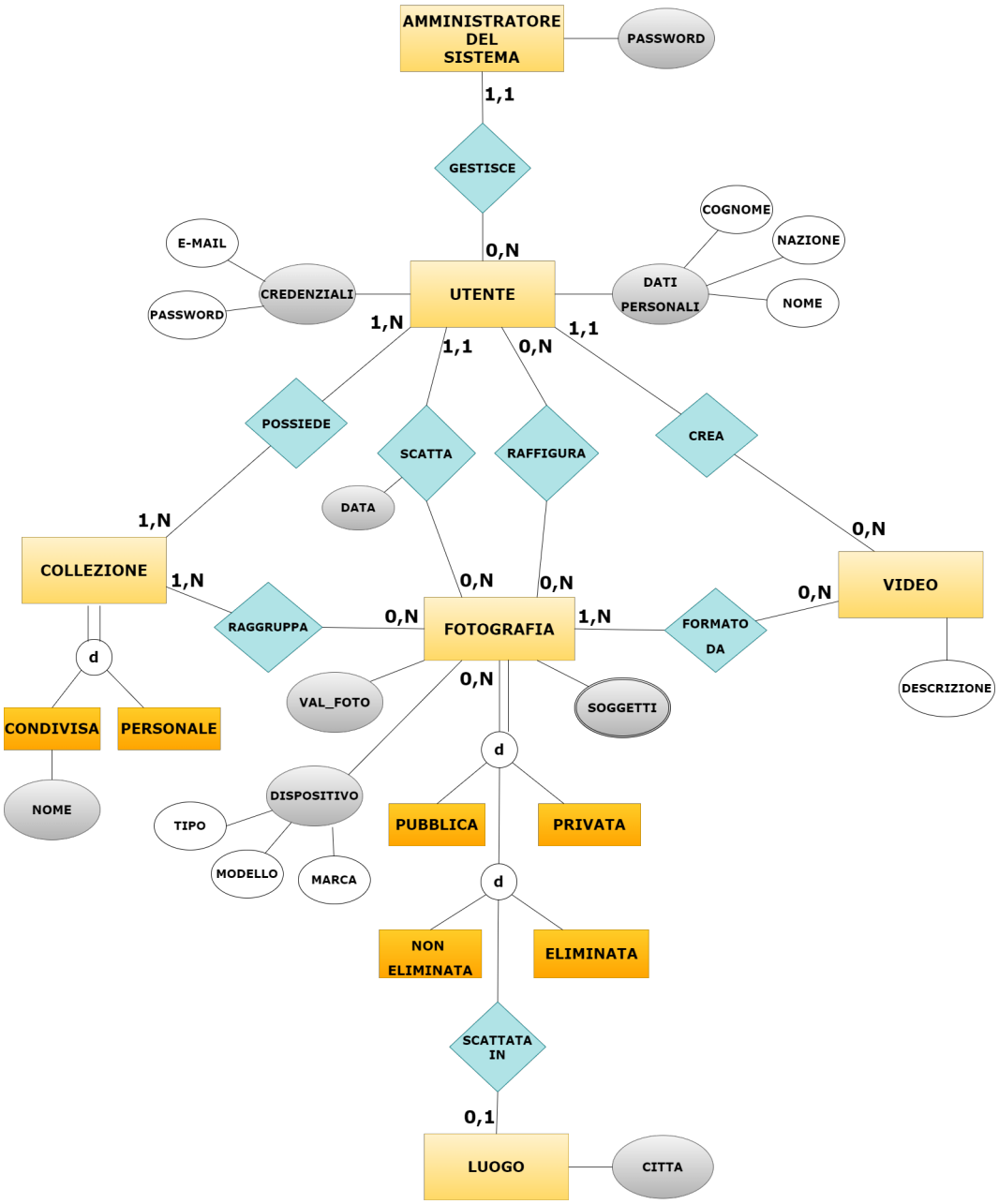

Figura 7: ER Diagram

2.3 Class Diagram

Completato il diagramma ER, si è poi optato, soprattutto per questione di praticità e compatibilità col modello a oggetti di JAVA, alla trasformazione di quest'ultimo in un **Class Diagram UML**, sostituendo le entità con le **classi**, e le relazioni con le **associazioni**:

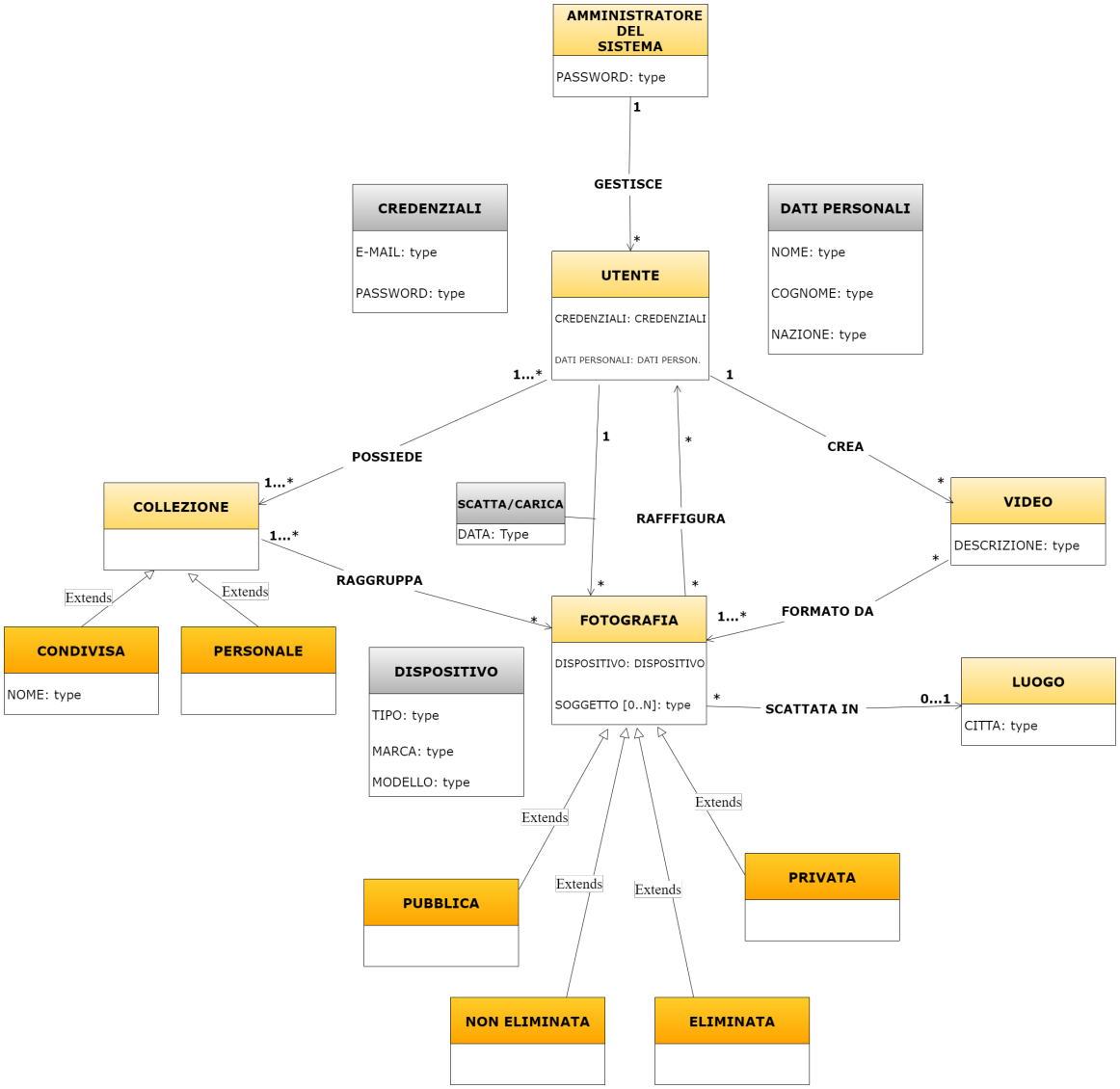

Figura 8: Class Diagram UML

Si può notare che, in questo determinato frangente, si è deciso di lasciare intenzionalmente il valore dei tipi degli attributi come *"type"*, inteso come un tipo non ancora specificato.

2.4 Analisi di ristrutturazione del Class Diagram

Non tutti i costrutti che si sono utilizzati nei precedenti diagrammi hanno una traduzione naturale nei modelli logici che saranno poi implementati nei database relazionali; dunque, al fine di rendere il nostro Class Diagram idoneo, e per migliorare l'efficienza generale dell'implementazione, si procede alla sua **ristrutturazione**. La fase di ristrutturazione di uno schema concettuale è suddiviso generalmente in:

1. Analisi delle ridondanze
2. Eliminazione delle Generalizzazioni
3. Eliminazione Attributi Multipli
4. Eliminazione Attributi Strutturati
5. Partizionamento/Accorpamento di classi e associazioni
6. Analisi degli Identificativi

2.4.1 Analisi delle Ridondanze

Non si sono rilevate particolari ridondanze nel Class Diagram.

2.4.2 Eliminazione delle Generalizzazioni

Sono presenti ben due generalizzazioni nel Class Diagram: COLLEZIONI e FOTOGRAFIA, con le loro rispettive sottoclassi. Si ricorda che entrambe le classi generalizzate hanno un **vincolo di specializzazione totale**, e dunque, per poterle eliminare, la scelta più opportuna è sicuramente quella di applicare una delle seguenti opzioni:

a) Accorpamento delle figlie della generalizzazione nel padre

b) Accorpamento del padre della generalizzazione nelle figlie

Si è optato alla fine per la scelta **a)**, considerando soprattutto che:

- Le classi di specializzazioni non hanno particolari attributi che li distinguono tra loro (nel caso di FOTOGRAFIA ne abbiamo addirittura 0)
- Le operazioni che andremo a fare su classe padre e classi figlie sono praticamente le stesse

All'interno della classe COLLEZIONE sono stati cosi aggiunti l'attributo *nome* (appartenente alla vecchia sottoclasse CONDIVISA), e l'attributo *personale* (flag a cui verrà assegnato un semplice valore intero, 0 od 1, utile distinguere le due ex sottoclassi implicite). Nella classe FOTOGRAFIA vengono aggiunti

invece gli attributi *pubblica* ed *eliminata*, che hanno la medesima funzionalità di *personale*. Si è dunque evitato di introdurre più classi, ottenendo sicuramente un **minor numero di accessi**, con l'unica trascurabile pecca di introdurre dei valori nulli nel caso una collezione sia *personale* (ndr: una collezione personale non ha un nome).

2.4.3 Eliminazione degli Attributi Multipli

È presente un unico attributo multivalore, ovvero *soggetti* (FOTOGRAFIA). Per eliminarlo, tra le tecniche a disposizione che si possono sfruttare, ovvero:

a) Creare una entità esterna associata

b) Trattare l'attributo come singolo

c) Replicare l'attributo nella classe

Si è optato alla fine per la **creazione di una entità esterna associata**, con l'aggiunta dell'attributo *categoria* che va a specificare il tipo di soggetto in questione. La scelta è giustificata dal fatto che, avendo l'idea di inizializzare il database con diversi **soggetti di default**, si è ritenuto molto utile avere una determinata classe in cui poterli "immagazzinare". Inoltre, si è considerato abbastanza sprecato sia trattare l'attributo come se fosse singolo (riducendo ad una foto la possibilità di avere diversi soggetti), sia replicarlo all'interno della classe (rischiando cosi di avere eccessivi valori nulli).

2.4.4 Eliminazione degli Attributi Strutturati

Per quanto riguarda gli attributi strutturati, ne sono presenti tre: *credenziali* (UTENTE), *dati personali* (UTENTE) e *dispositivo* (FOTOGRAFIA). Anche qui, come nel caso degli attributi multipli, si può ricorrere a 3 modi per eliminarli:

a) Introduzione di una classe

b) Estrazione degli attributi

c) Eliminazione degli attributi

Non si è ritenuto nessun attributo all'altezza di avere una specifica classe. Si è invece optato, per l'attributo *dispositivo*, ad una netta **eliminazione** dei suoi attributi, considerati abbastanza superflui per le funzionalità del progetto, mentre, per quanto riguarda *dati personali* e *credenziali*, alla loro **estrazione** nella classe UTENTE, essendo valutati entrambi di notevole importanza.

2.4.5 Partizione/Accorpamento di classi e associazioni

Non sono stati individuate particolari classi che necessitano di un'eventuale partizione o accorpamento. Si era inizialmente pensato, per una questione di maggior praticità, ad accorpare la classe LUOGO alla classe FOTOGRAFIA, sia considerando il fatto che LUOGO avesse un solo attributo, sia considerando il potenziale numero elevato di città/località da dover immagazzinare in essa (molto maggiore rispetto alle categorie dei soggetti, ad esempio). Però, per rispettare la richiesta di avere ogni luogo identificato univocamente, si è deciso alla fine di rimanere con la classe LUOGO. Per quanto riguarda le associazioni, si è deciso invece di accorpare l'attributo dell'associazione ***"scatta"***, ovvero *data*, nella classe FOTOGRAFIA.

2.4.6 Analisi degli identificativi

Infine, per quanto riguarda la scelta degli identificativi del Class Diagram, si è optato per i seguenti attributi:

- *ID_amminist* (AMMINISTRATORE DEL SISTEMA)
- *ID_utente*, *email* (UTENTE)
- *ID_foto* (FOTOGRAFIA)
- *ID_collezione*, *nome* (COLLEZIONE)
- *ID_video* (VIDEO)
- *ID_soggetto*, *categoria* (SOGGETTO)
- *città* (LUOGO)

Da come si può notare, si è scelto di identificare ogni classe, ad eccezione di LUOGO (per cui si è pensato sufficiente l'attributo *città*), con uno specifico attributo di **chiave primaria** in cui nel nome è incluso il prefisso "**ID**". Ci sono poi tre **chiavi secondarie**, anch'esse uniche all'interno del sistema, ovvero *email* (UTENTE), *nome* (COLLEZIONE) e *categoria* (SOGGETTO). Per quanto riguarda *email*, è abbastanza scontato il fatto che, per questione di univocità, due utenti non possano avere una stesso indirizzo di posta elettronica. Per quanto riguarda *categoria*, anche qui è abbastanza intuitivo il fatto che è assolutamente inutile avere due tipologie di soggetto uguali all'interno della stessa classe. Infine, per quanto riguarda il *nome* della collezione, pur essendo in teoria assolutamente concepibile avere due nomi uguali all'interno del sistema, si è deciso di usarla come chiave semplicemente per una questione di praticità, rendendo la ricerca più semplice da gestire all'interno dell'applicativo JAVA.

2.5 Class Diagram Ristrutturato

Completata la ristrutturazione, ci si ritrova infine con il seguente Class Diagram:

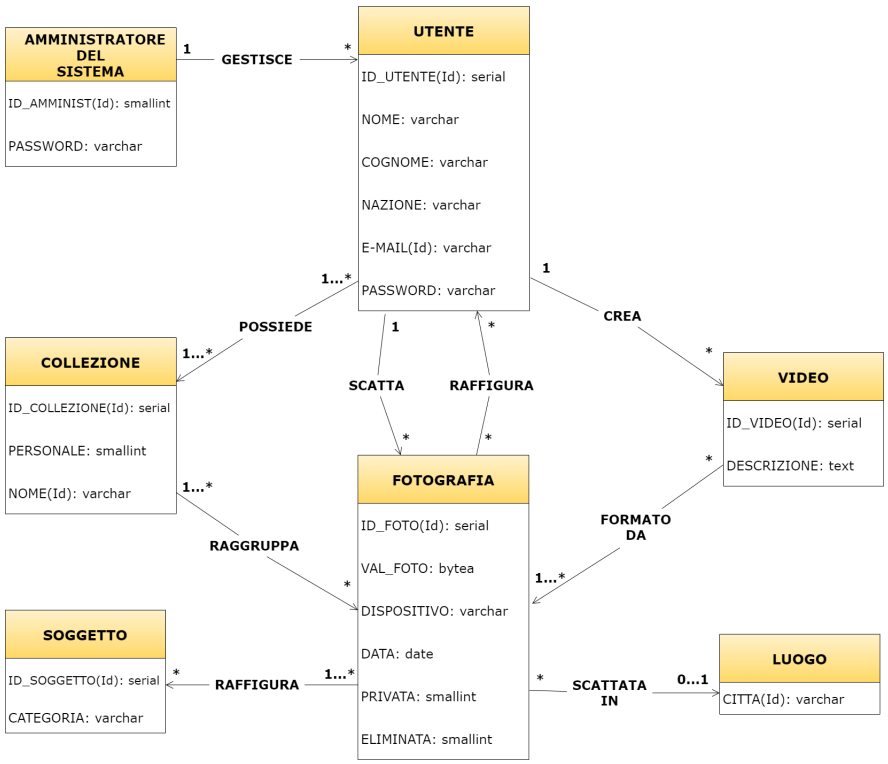

Figura 9: Class Diagram UML - Ristrutturato

Si può notare che, in questo frangente, sono stati specificati tutti i tipi che saranno poi assegnati agli attributi al momento dell'implementazione definitiva. Inoltre, si sottolinea che a tutte le chiavi con prefisso *"ID"* (eccetto *id_ amministst*, che non ne ha bisogno) è stato assegnato il tipo **serial**. Quest'ultimo, in Postgre, non è altro che un intero con aumento automatico del proprio valore ad ogni inserimento di una nuova tupla, caratterizzato da un **valore iniziale** e da un **incremento** impostati e gestiti a nostra preferenza.

3 DIZIONARIO DELLE CLASSI, DELLE ASSOCIAZIONI e DEI VINCOLI

Andremo ora a vedere, nel dettaglio, attraverso un **dizionario** in formato tabellare, tutte le classi, le associazioni e i vincoli coinvolte nel nostro Class Diagram.

3.1 Dizionario delle Classi

Classe	Descrizione	Attributi
Amministratore del Sistema	Classe che rappresenta il responsabile del sistema, incaricato alla gestione dell'utenza	**ID_amminist** (*smallint*): chiave primaria della classe che identifica univocamente l'amministratore. **Password** (*varchar*): stringa d'accesso all'esclusiva sezione di gestione dell'amministratore.
Utente	Classe che rappresenta i clienti utilizzatori della galleria fotografica	**ID_utente** (*serial*): chiave primaria della classe che identifica univocamente ogni singolo utente. Il suo valore iniziale è impostato ad 1, cosi come il suo incremento. **Nome** (*varchar*): nome dell'utente. **Cognome** (*varchar*): cognome dell'utente. **Nazione** (*varchar*): nazione d'origine dell'utente. **Email** (*varchar*): chiave secondaria della classe, specifica l'indirizzo di posta elettronica dell'utente. **Password** (*varchar*): stringa che permetta all'utente di accedere al sistema.
Fotografia	Classe che rappresenta le fotografie	**ID_foto** (*serial*): chiave primaria della classe che identifica univocamente ogni singola fotografia. Il suo valore iniziale è impostato a 100, mentre il suo incremento ad 1. **Val_foto** (*bytea*): valore della fotografia, inteso come la rappresentazione che ha la foto all'interno del database. Si è preferito assegnargli il tipo bytea (il binary string di Postgre, simile al BLOB), andando cosi a memorizzare i byte della foto direttamente nel database. **Dispositivo** (*varchar*): dispositivo con il quale è stata scattata la foto. **Data** (*date*): data in cui la foto è stata caricata nel sistema. **Pubblica** (*smallint*): indica se la fotografia è pubblica (1) oppure privata (0). Essendo vincolata ai soli valori 0 ed 1, si è preferito assegnarli il tipo smallint per questione di efficienza. Il valore di default è impostato a 1. **Eliminata** (*smallint*): indica se la fotografia è nella collezione personale (0) oppure nel cestino (1). Il valore di default è impostato a 0.

Collezione	Classe che rappresenta le collezioni di fotografie	**ID_collezione**(*serial*): chiave primaria della classe che identifica univocamente ogni singola collezione. Il suo valore iniziale è impostato a 100.000, mentre il suo incremento a 5. **Personale** (*smallint*): indica se la collezione è personale (1) oppure condivisa (0). Il valore di default è impostato a 1. **Nome** (*varchar*): nome di una collezione pubblica; il nome di una collezione personale è invece impostato a NULL.
Video	Classe che rappresenta i video	**ID_video** (*serial*): chiave primaria della classe che identifica univocamente ogni singolo video. Il suo valore iniziale è impostato a 1.000, mentre il suo incremento a 5. **Descrizione** (*text*): breve resoconto di ciò che viene raffigurato nel video.
Luogo	Classe che rappresenta il luogo in cui la fotografia è scattata	**Città** (*varchar*): chiave primaria della classe che identifica univocamente ogni singolo luogo. In questo caso, non è altro che il nome di una città; verranno inseriti alcuni luoghi di default, ma l'utente avrà la facoltà di specificare e aggiungere un qualsiasi altra località.
Soggetto	Classe che rappresenta i soggetti delle fotografie	**ID_amminist** (*serial*): chiave primaria della classe che identifica univocamente ogni singolo soggetto. Il suo valore iniziale è impostato a 1, cosi come il suo incremento. **Categoria** (*varchar*): chiave secondaria della classe, specifica il nome della tipologia di soggetto. Verranno inserite delle categorie di default a cui l'utente potrà fare riferimento.

3.2 Dizionario delle Associazioni

Nome	Classi coinvolte
Gestisce	**Amministratore del Sistema** [1] ruolo *gestisce*: indica che un singolo utente è obbligatoriamente gestito da un unico amministratore. **Utente** [*] ruolo *è gestito*: indica che un amministratore gestisce più utenti, ma potrebbe anche averne 0.
Scatta	**Utente** [1] ruolo *scatta*: indica che una singola foto è scattata necessariamente da un solo utente. **Fotografia** [*] ruolo *è scattata*: indica che un singolo utente può scattare più foto, ma potrebbe anche averne scattate 0.
Raffigura	**Utente** [*] ruolo *è raffigurato*: indica che un singolo utente può essere raffigurato in 0 o più fotografie. **Fotografia** [*] ruolo *raffigura*: indica che una singola fotografia può raffigurare 0 o più utenti.
Possiede	**Utente** [1...*] ruolo *possiede*: indica che una singola collezione può essere posseduta da 1 o più utenti. **Collezione** [1...*] ruolo *è posseduta*: indica che un singolo utente può possedere più collezioni, ma necessariamente una di esse, ovvero quella personale.
Raggruppa	**Collezione** [1...*] ruolo *raggruppa*: indica che una singola fotografia può essere raggruppata in più collezioni, ma deve essere presente naturalmente in almeno una di esse. **Fotografia** [*] ruolo *è raggruppata*: indica che una singola collezione può raggruppare più di una fotografia, ma potrebbe anche raggrupparne 0.
Crea	**Utente** [1] ruolo *crea*: indica che un singolo video è creato necessariamente da un solo utente. **Video** [*] ruolo *è creato*: indica che un singolo utente può creare più video, ma potrebbe anche averne creati 0.

Formato da	**Video** [*] ruolo *formato da*: indica che 1 o più fotografie possono far parte di un video, ma potrebbero anche non essere presenti in nessuno di essi. **Fotografia** [2...*] ruolo *è parte di*: indica che un singolo video può essere formato da più foto, ma per essere tale ne deve quantomeno avere 2.
Scattata in	**Fotografia** [*] ruolo *scattata in*: indica che un singolo luogo può essere presente in 0 o più fotografie. **Luogo** [0...1] ruolo *è presente in*: indica che una singola fotografia è scattata in un determinato luogo, ma potrebbe anche non essere specificato nessuno di esso.
Raffigura	**Fotografia** [*] ruolo *raffigura*: indica che un singolo soggetto può essere raffigurato in 0 o più fotografie. **Soggetto** [*] ruolo *è raffigurato*: indica che una singola fotografia può raffigurare 0 o più soggetti.

3.3 Dizionario dei Vincoli

Descrizione	Implementazione
La password dell'amministratore non può essere NULL.	Vincolo CHECK.
Il nome di un utente non può essere NULL.	Vincolo CHECK.
Il cognome di un utente non può essere NULL.	Vincolo CHECK.
L'email di un utente non può essere NULL.	Vincolo CHECK.
L'email di un utente deve essere unica all'interno del sistema.	Vincolo UNIQUE.
L'email di un utente deve rispettare determinati criteri di validità.	TRIGGER.
La password di un utente non può essere NULL.	Vincolo CHECK.
La password di un utente deve rispettare determinati criteri di validità.	TRIGGER.
Il nome di una collezione deve essere unico all'interno del sistema.	Vincolo UNIQUE.

L'attributo *personale* di COLLEZIONE può assumere solo i valori interi 0 ed 1.	Vincolo CHECK.
Il valore di una foto non può essere NULL.	Vincolo CHECK.
L'attributo *pubblica* di FOTOGRAFIA può assumere solo i valori interi 0 ed 1.	Vincolo CHECK.
L'attributo *eliminata* di FOTOGRAFIA può assumere solo i valori interi 0 ed 1.	Vincolo CHECK.
La categoria di un soggetto non può essere NULL.	Vincolo CHECK.
La categoria di un soggetto deve essere unica all'interno del sistema.	Vincolo UNIQUE.
La creazione di un nuovo utente comporta la creazione automatica di una sua collezione personale.	TRIGGER.
Le fotografie caricate dall'utente sono direttamente inserite nella sua collezione personale.	TRIGGER.
L'eliminazione di una fotografia ne comporta l'eliminazione dalla collezione personale di sua appartenenza, ma non dalle gallerie condivise in cui essa è presente.	PROCEDURA.
L'eliminazione di un utente comporta l'eliminazione della sua collezione personale.	TRIGGER.
L'eliminazione di un utente comporta l'eliminazione di tutte le sue foto, ad eccezione di quelle raffiguranti altri utenti che partecipano alla stessa collezione condivisa in cui esse sono presenti.	TRIGGER.
Non si può aggiungere ad una collezione condivisa un utente che già partecipa ad essa.	TRIGGER.
Se un utente rende una sua foto privata, essa sarà eliminata da qualsiasi collezione condivisa in cui è presente.	TRIGGER.
Un utente può caricare al massimo 1000 fotografie.	TRIGGER.

4 PROGETTAZIONE LOGICA

4.1 Analisi di traduzione ad uno Schema Logico

Prima di passare all'implementazione definitiva del Class Diagram ristrutturato, esso necessita di un ultimo passaggio, ovvero la traduzione ad un **modello logico** che si avvicini ancor più al modello relazionale richiesto dal database. Per effettuare ciò, si va ad effettuare il cosiddetto **mapping**, sia per le classi che per le associazioni.

4.1.1 Mapping delle Classi

Per ogni classe si andrà quindi a scrivere uno **schema di relazione** con lo stesso nome, i medesimi attributi e i rispettivi identificatori (si indicheranno con una sottolineatura le chiavi primarie, con il corsivo le chiavi secondarie). Di conseguenza:

Amministratore(ID amministratore, Password)

Utente(ID utente, Nome, Cognome, Nazione, *Email*, Password)

Fotografia(ID foto, Val_foto, Dispositivo, Data, Pubblica, Eliminata)

Collezione(ID collezione, Personale, *Nome*)

Video(ID video, Descrizione)

Soggetto(ID soggetto, *Categoria*)

Luogo(Città)

4.1.2 Mapping delle Associazioni

Per quanto riguarda le associazioni, si dovranno mappare considerando principalmente la loro **cardinalità**. Non avendo associazioni 1:1, ci basterà considerare quelle N:N (o *:*) e quelle 1:N (o anche 1:*).

- Per le associazioni ***:*** si andrà a scrivere uno schema di relazione con il nome della associazione (in questo caso però si è preferito scegliere nomi più chiari), e come attributi gli identificatori primari delle classi coinvolte, che in questo caso avranno vincolo di **chiave esterna** (e indichiamo con una doppia sottolineatura). Di conseguenza:

  Foto_raffigura_utente(ID foto, ID utente)

Utente_possiede_collezione(ID utente, ID collezione)

Collezione_raggruppa_foto(ID collezione, ID foto)

Video_formato_da_foto(ID video, ID foto)

Foto_raffigura_soggetto(ID foto, ID soggetto)

- Per le associazioni **1:*** si andrà invece a modificare opportunamente gli schemi di relazione ottenuti nel mapping delle classi, andando ad inserire, nello schema la cui classe ha cardinalità (*), l'identificatore della classe che ha cardinalità (1). Anche questo identificatore, come nel caso precedente, sarà una chiave esterna, e lo si indicherà con doppia sottolineatura. Di conseguenza:

  Utente(ID utente, Nome, Cognome,Nazione, *Email*,Password, ID amministratore)

  Fotografia(ID foto, Val_foto, Dispositivo, Data, Pubblica, Eliminata, ID utente, Città)

4.2 Schema Logico

Dunque, il nostro **schema logico** finale sarà:

Amministratore	(ID amministratore, Password).
Utente	(ID utente, Nome, Cognome, Nazione, *Email*,Password, ID amministratore)
Fotografia	(ID foto, Val_foto, Dispositivo, Data, Pubblica, Eliminata, ID utente, Città)
Collezione	(ID collezione, Personale, *Nome*)
Video	(ID video, Descrizione)
Soggetto	(ID soggetto, *Categoria*)
Luogo	(Città)

Foto_raffigura_utente	(ID foto, ID utente)
Utente_possiede_collezione	(ID utente, ID collezione)
Collezione_raggruppa_foto	(ID collezione, ID foto)
Video_formato_da_foto	(ID video, ID foto)
Foto_raffigura_soggetto	(ID foto, ID soggetto)

5 PROCEDURE, FUNZIONI e TRIGGER

Concludiamo con una panoramica di procedure, funzioni e trigger implementati nel nostro database, con relative descrizioni.

5.1 Procedure

Nome	Descrizione
aggiungi_fotografia	Procedura che effettua l'inserimento di una nuova foto nella tabella FOTOGRAFIA.
aggiungi_soggetto	Procedura che effettua l'inserimento di una nuova categoria nella tabella SOGGETTO.
aggiungi_luogo	Procedura che effettua l'inserimento di una nuova città nella tabella LUOGO.
crea_amministratore	Procedura che effettua l'inserimento di una nuovo amministratore nella tabella AMMINISTRATORE.
crea_utente	Procedura che effettua l'inserimento di un nuovo utente nella tabella UTENTE.
crea_video	Procedura che effettua l'inserimento di un nuovo video nella tabella VIDEO.
crea_collezione_condivisa	Procedura che effettua la creazione di una nuova collezione condivisa. La collezione condivisa dovrà essere creata, almeno inizialmente, con un solo altro utente partecipante ad essa.
aggiungi_utente_in collezione_condivisa	Procedura che aggiunge un nuovo utente ad una collezione condivisa già esistente.

elimina_fotografia	Procedura che elimina una foto dalla galleria (collezione) personale dell'utente. Si occuperà di controllare se la fotografia da eliminare è presente in delle gallerie condivise; in tal caso, come da vincolo richiesto, provvederà a far in modo che la fotografia resti in quelle gallerie condivise. In caso contrario, la eliminerà completamente dal sistema.
elimina_fotografia_in collezione_condivisa	Procedura che elimina una foto da una collezione condivisa. Si occuperà di controllare se la fotografia da eliminare è presente nella collezione personale d'appartenenza o in altre collezioni condivise; in tal caso, provvederà a far in modo che essa venga eliminata solo nella galleria condivisa in cui ne è richiesta l'eliminazione. In caso contrario, la eliminerà completamente dal sistema.
elimina_fotografie_in collezione_condivisa	Procedura che elimina TUTTE le foto da una collezione condivisa. Segue lo stesso vincolo di *elimina_fotografia_in collezione_condivisa* .
elimina_utente	Procedura che elimina un utente dal database.
elimina_utente_in collezione_condivisa	Procedura che elimina un utente da una collezione condivisa.
elimina_video	Procedura che elimina un video dal database.
inserisci_fotografia_in in_cestino	Procedura che inserisce una foto in un cestino "virtuale", dove le foto possono poi essere definitivamente eliminate.
rimuovi_fotografia_in da_cestino	Procedura che rimuove una foto dal cestino, reinserendola nella collezione personale dell'utente.
rendi_fotografia_privata o_pubblica	Procedura che gestisce la privacy di una foto, o rendendola privata, o effettuando il processo inverso.
inserisci_fotografia_in collezione_condivisa	Procedura che aggiunge una foto in una collezione condivisa.
inserisci_fotografie_in collezione_condivisa	Procedura che aggiunge ad una collezione condivisa TUTTE le foto dei due utenti (ovviamente non eliminate e non private) che l'hanno creata. Essa è utilizzabile solo nel momento immediato alla creazione, e verrà richiesta come prompt dall'applicazione.
inserisci_in_foto raffigura_soggetto	Procedura che aggiunge le chiavi esterne della coppia foto-soggetto (relazione *:*) nella rispettiva tabella.
inserisci_in_foto raffigura_utente	Procedura che aggiunge le chiavi esterne della coppia foto-utente (relazione *:*) nella rispettiva tabella.

inserisci_in_utente possiede_collezione	Procedura che aggiunge le chiavi esterne della coppia utente-collezione (relazione *:*) nella rispettiva tabella.
inserisci_in_collezione raggruppa_foto	Procedura che aggiunge le chiavi esterne della coppia collezione-foto (relazione *:*) nella rispettiva tabella.
inserisci_in_video formato_da_foto	Procedura che aggiunge le chiavi esterne della coppia video-foto (relazione *:*) nella rispettiva tabella.
truncate_tabelle	Procedura che sfrutta il comando TRUNCATE (Postgre) per eliminare i dati di alcune tabelle e, di conseguenza, resettare il popolamento del database.

5.2 Funzioni

Nome	Tipo di Ritorno	Descrizione
cestino	*TABLE*	Funzione che restituisce una *table*, quindi un record di tuple. In questo caso, restituisce il risultato di una query che seleziona tutte le foto eliminate (momentaneamente) di un utente.
collezione_personale	*TABLE*	Funzione che restituisce il risultato di una query che seleziona tutte le foto della collezione personale di un utente.
collezione_condivisa	*TABLE*	Funzione che restituisce il risultato di una query che seleziona tutte le foto di una determinata collezione condivisa a cui partecipa un utente.
foto_non_presenti in_collezione_condivisa	*TABLE*	Funzione che restituisce il risultato di una query che seleziona tutte le foto (non private) di una collezione personale che non sono state ancora condivise in una determinata collezione. Utile per evitare di condividere foto già presenti in una collezione condivisa.
stesso_luogo	*TABLE*	Funzione che restituisce il risultato di una query che seleziona tutte le foto di una collezione personale accomunate da uno stesso luogo, specificato come parametro.

stesso_soggetto	*TABLE*	Funzione che restituisce il risultato di una query che seleziona tutte le foto di una collezione personale accomunate da uno stesso soggetto, specificato come parametro.
top_3_luoghi	*TABLE*	Funzione che restituisce il risultato di una query che seleziona i 3 luoghi più presenti (con relativo numero di foto) tra le foto della collezione personale di un utente.
recupera_id_utente	*INT*	Funzione che, data l'email di un utente, restituisce il suo ID.
recupera_id_collezione	*INT*	Funzione che, dato il nome di una collezione, restituisce il suo ID.
recupera_id_soggetto	*INT*	Funzione che, dato il nome di una categoria di soggetto, restituisce il suo ID.
recupera_id_foto	*INT*	Funzione che recupera l'ID dell'ultima foto inserita.
recupera_id_video	*INT*	Funzione che recupera l'ID dell'ultimo video inserito.
recupera_soggetti_foto	*TABLE*	Funzione che restituisce il risultato di una query che seleziona tutti i soggetti che sono raffigurati in una determinata foto.
recupera_utenti_foto	*TABLE*	Funzione che restituisce il risultato di una query che seleziona tutti gli utenti che sono raffigurati in una determinata foto.

5.3 Trigger

Nome	Modalità di Attivazione	Descrizione
collezione_già_condivisa	*AFTER INSERT*	Trigger che si attiva dopo l'inserimento di una tupla nella tabella UTENTE_POSSIEDE_COLLEZIONE. Controlla se l'utente che sta per essere aggiunto ad una collezione condivisa sia già presente in essa; in tal caso, solleva un'eccezione.

creazione_collezione personale	*AFTER INSERT*	Trigger che si attiva dopo l'inserimento di una tupla nella tabella UTENTE. Si occupa di creare, naturalmente, la collezione personale di proprietà del nuovo utente; utente e rispettiva collezione personale avranno lo stesso valore di chiave primaria.
collezione_personale dopo_eliminazione utente	*AFTER DELETE*	Trigger che si attiva dopo l'eliminazione di una tupla dalla tabella UTENTE. Si occupa di eliminare la collezione personale di proprietà dell'utente appena eliminato.
collezione_condivisa dopo_foto privata	*AFTER UPDATE*	Trigger che si attiva dopo la modifica dell'attributo *pubblica* nella tabella FOTOGRAFIA. Si occupa di eliminare una foto resa privata dalle collezioni condivise in cui essa è presente.
controllo_email	*AFTER INSERT*	Trigger che si attiva dopo l'inserimento di una nuova tupla nella tabella UTENTE. Si occupa di controllare se l'email dell'utente appena registrato rispetti un determinato pattern di validità.
controllo_password	*AFTER INSERT*	Trigger che si attiva dopo l'inserimento di una nuova tupla nella tabella UTENTE. Si occupa di controllare se la password dell'utente appena registrato rispetti un determinato pattern di validità.
fotografie_dopo eliminazione_utente	*AFTER DELETE*	Trigger che si attiva dopo l'eliminazione di una tupla dalla tabella UTENTE. Come da vincolo richiesto, si occupa di eliminare tutte le fotografie appartenenti all'utente eliminato, eccetto quelle in cui sono raffigurati altri utenti della galleria condivisa in cui esse sono presenti.
inserimento_data in_fotografia	*AFTER INSERT*	Trigger che si attiva dopo l'inserimento di una tupla nella tabella FOTOGRAFIA. Si occupa di impostare la *data* della fotografia a quella corrente, ovvero al momento in cui la fotografia è caricata nel sistema.

inserimento_id amminist_in_utente	*AFTER INSERT*	Trigger che si attiva dopo l'inserimento di una tupla nella tabella UTENTE. Si occupa di associare l' *id_amminist* dell'utente a l'unico amministratore presente nel sistema.
inserimento_fotografia in_collezione_personale	*AFTER INSERT*	Trigger che si attiva dopo l'inserimento di una tupla nella tabella FOTOGRAFIA. Si occupa di aggiungere la fotografia appena caricata dall'utente nella sua collezione personale.
limite_fotografie per_utente	*AFTER INSERT*	Trigger che si attiva dopo l'inserimento di una tupla nella tabella FOTOGRAFIA. Controlla se la collezione personale di utente ha più di 1000 fotografie scattate/caricate da lui; in tal caso, solleva un'eccezione.
limite_utenti sistema	*AFTER INSERT*	Trigger che si attiva dopo l'inserimento di una tupla nella tabella UTENTE. Controlla se sono presenti più di 99.999 utenti registrati al sistema; in tal caso, viene sollevata un'eccezione. Questo trigger è giustificato dal fatto che, dato che una collezione personale ha lo stesso ID dell'utente che la possiede e le collezioni condivise hanno un ID che invece parte dal valore 100.000, alla creazione del 100.000esimo utente si avrà un conflitto di ID in COLLEZIONE, che va a violare il vincolo di chiave primaria. Si provvederà dunque, nell'eventualità, ad aumentare gli ID delle collezioni che hanno un valore maggiore (o uguale) di 100.000, permettendo la registrazione di nuovi utenti.