Laboratorio di Basi di Dati: Progetto “Collectors”

Gruppo di lavoro

Matricola	Nome	Cognome	Contributo al progetto
281113	Simone	Rabottini	Analisi dei requisiti, Progettazione Concettuale, logica, fisica, documentazione
279494	Giacomo	Calcaterra	Analisi dei requisiti, Progettazione Concettuale, logica, fisica

Data di consegna progetto: 28/06/2023

Analisi dei requisiti

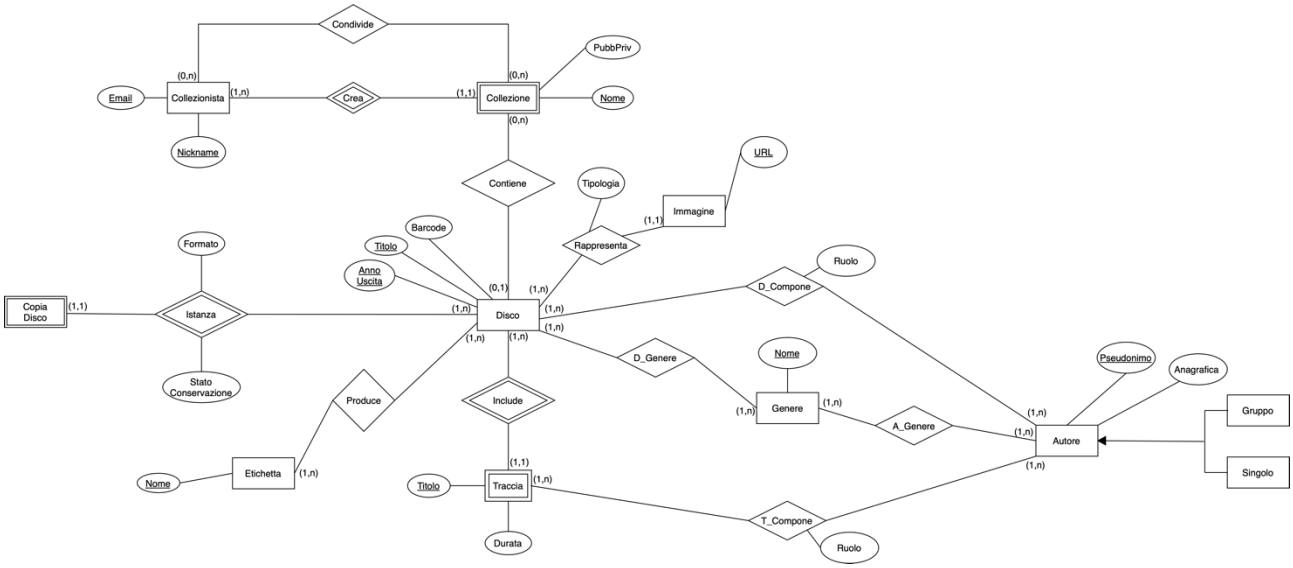
Nel progetto in questione ci occuperemo di realizzare un database che memorizza informazioni relative a collezioni di dischi.
Le entità protagoniste sono il **Collezionista**, che può creare una **Collezione**, che a sua volta può contenere **Disco** con eventualmente relativa **Copia Disco** che, come riportato sulla specifica, spesso si possono possedere magari a seguito di scambi, o magari perché se ne prevede la rivendita. Ogni **Disco**: è rappresentato da **Immagine**, possiede **Genere**, può essere prodotto da **Etichetta**, include **Traccia** che, così come il disco può essere composto da **Autore**, che possiede anch’esso un proprio **Genere**.

Il Collezionista, identificato da email e nickname, può creare una o più Collezioni ognuna con un **nome distinto:** possono esistere più collezioni con lo stesso nome, ma di collezionisti diversi (un collezionista non può creare due collezioni con lo stesso nome). La Collezione può essere privata o pubblica (impostabile con un apposito flag), e quindi condivisa con altri collezionisti del dominio. Ogni Collezione può contenere uno o più dischi con relative copie del disco, ognuna con un formato (CD, vinile, cassetta…) e uno stato di conservazione (nuovo, usato…). Il disco presenta un titolo, anno di uscita e barcode (se disponibile al momento della catalogazione). Ogni disco:

- Viene rappresentato da una o più immagini, ognuna con una propria tipologia (Copertina, Retro, facciata interna…) e URL (percorso del file memorizzato sul server)
- Viene prodotto da una o più etichette discografiche (nel caso della co-produzione), ognuna con un proprio nome
- Viene composto da uno o più autori, ognuno con un proprio ruolo (Compositore, Esecutore...)
- Possiede uno o più generi
- Include una o più tracce, ognuna con un titolo e la sua durata (in formato *ore:minuti:secondi*).

Ogni traccia potrebbe differire dall’intero disco in cui è contenuta per autori partecipanti, quindi anche la traccia può essere composta da uno o più autori, anche qui ognuno con un proprio ruolo.
L’autore può figurare come singolo o come un gruppo: nel caso di solista possiede un proprio pseudonimo e la sua anagrafica: nome, cognome e data di nascita. Nel caso del gruppo, possiede il proprio pseudonimo e la sua anagrafica riguarderà esclusivamente la data di nascita, che si riferisce alla data in cui il gruppo si è costituito. L’autore, che come già detto può comporre tracce e dischi ricoprendo un ruolo, può infine possedere uno o più generi.

Progettazione concettuale



- **Collezionista**:
  Chiave: email, nickname
  Ogni collezionista viene identificato univocamente dalla sua email e il suo nickname con cui si registra al dominio; sono attributi unici e che possono corrispondere a un solo utente.
- **Collezione:**
  Chiave: nome
  L'entità Collezione rappresenta un'entità debole: l'attributo nome non è sufficiente a formare una chiave, e quindi include la relazione *crea* con Collezionista per identificarsi univocamente. Con l'attributo "PubbPriv" intendiamo se la collezione creata è Pubblica o Privata.
- **Disco**:
  Chiave: Titolo, Anno Uscita
  il barcode potrebbe far pensare a una chiave che identifica univocamente il disco. Siccome al momento della catalogazione dei dischi il barcode potrebbe non essere disponibile, questo attributo non è "affidabile" per costituire una chiave. Scegliamo quindi la coppia "Titolo" - "Anno Uscita" assumendo che non possono esistere due dischi con lo stesso nome usciti lo stesso anno.
  Per esplicitare le immagini, generi ed etichette discografiche del disco, si è optato per rappresentarle come delle entità in relazione con Disco, per una migliore gestione del sistema e una efficace gestione della ridondanza dei dati.
- **Etichetta**:
  Chiave: nome
  Etichetta rappresenta la Casa discografica che svolge produzioni musicali. È opportuno rappresentarla entità non solo per perché la stessa etichetta può produrre più dischi (gestione della ridondanza), ma assumiamo anche che esista la co-produzione, e che quindi le etichette discografiche possano collaborare per produrre un disco. (Vedi relazione *produce*)
- **Immagine**:
  Chiave: URL
  Con URL indichiamo il percorso del file memorizzato sul server, esso è sicuramente univoco quindi è sufficiente a definire una chiave. Con l'entità evitiamo di appesantire il disco con troppi attributi (si intende l'inserimento di un attributo per tipologia), e apriamo alla possibilità che in un disco possono esserci più immagini della stessa tipologia (facciate interne, libretti, copertine secondarie). (Vedi relazione *rappresenta*)
- **Copia disco**:
  Con Copia disco gestiamo le copie di un disco a disposizione del collezionista che ha nella propria collezione, dove prevediamo che un collezionista può avere un'unica copia o più copie dello stesso disco (caso di doppioni). L'entità Copia disco rappresenta un'entità debole: include la relazione *istanza* con Disco per identificarsi univocamente.
- **Genere**:
  Chiave: nome
  Si introduce l'entità genere per realizzare una gestione ottimale di diverse casistiche:
  - Si costituisce una lista variabile, per includere il maggior numero di generi possibili, anche quelli che si potrebbero originare in futuro
  - Si garantisce un nome univoco e universale per ogni genere (attraverso attributo chiave "nome") a cui tutto il dominio può fare riferimento
  - Si evitano ripetizioni e ridondanze
  - Si prevede la possibilità che un disco o un autore possano avere più generi (vedi relazioni d_Genere e a_Genere).
- **Traccia**:
  Chiave: titolo
  L'entità Traccia rappresenta un'entità debole: l'attributo nome non è sufficiente a garantire l'univocità (ci possono essere più tracce con lo stesso nome). Quindi si include la relazione *include* con Disco.
- **Autore**:
  Chiave: pseudonimo
  Lo pseudonimo garantisce l'univocità dell'autore (assumendo che per copyright non possono esiste due autori con lo stesso pseudonimo). Ogni autore può figurare come un autore singolo o come un gruppo (esplicitato attraverso una generalizzazione totale). Ognuno inoltre ha la sua anagrafica (attributo composto), costituita da nome, cognome e data di nascita.

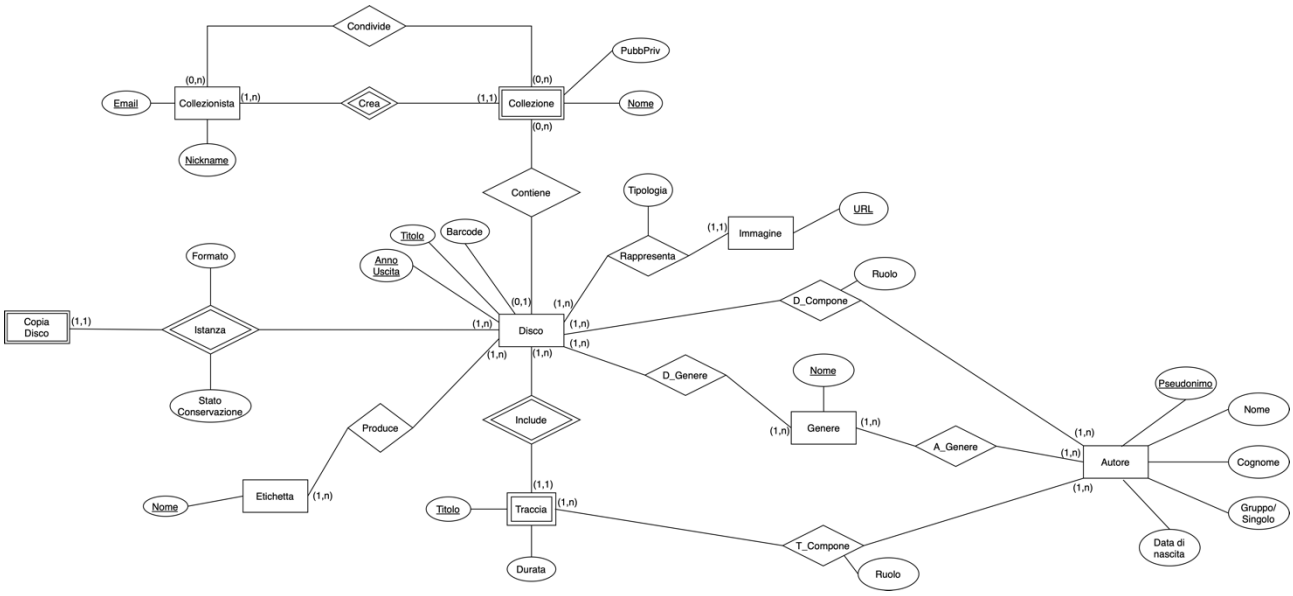
- **Collezionista → *Crea* → Collezione**: un collezionista crea una o più collezioni, una collezione può essere creata da un solo collezionista.
- **Collezione → *Condivide* → Collezionista**: una collezione può essere condivisa con uno o più collezionisti, un collezionista può ricevere la condivisione di una o più collezioni.
- **Collezione → *Contiene* → Disco:** una collezione può contenere uno o più dischi, un disco può essere contenuto in una collezione. In questo contesto è stato deciso di prevedere una ridondanza che però fa fede alla realtà. Se il disco è contenuto in una o più collezioni, a livello DBMS è sicuramente più ottimizzato, ma nella pratica è irrealizzabile: lo stesso disco con le stesse copie non può essere posseduto da più collezionisti in collezioni diverse. Ognuno ha il suo disco (con eventualmente le proprie copie) nella propria collezione. Inoltre,

se viene modificato, la modifica effettuata da un utente non riguarderà esclusivamente la sua collezione, ma impatterà tutte le collezioni del dominio. Quindi si preferisce duplicare piuttosto che realizzare un qualcosa di concettualmente sbagliato e indesiderato.

- ◊ **Immagine → *Rappresenta* → Disco:** un'immagine rappresenta un disco, un disco è rappresentato da una o più immagini. Si definisce l'attributo Tipologia, per specificare ad esempio che l'immagine che rappresenta la copertina (tipologia) un determinato disco; stesso discorso per retro, facciate interne ecc.
- ◊ **Disco → *Istanza* → Copia Disco:** un disco ha una o più copie disco, e una copia corrisponde a un disco. Si definiscono gli attributi Stato Conservazione e Formato, per indicare rispettivamente lo stato fisico in cui il disco è conservato (nuovo, usato…) e il formato (CD, Vinile, cassetta…).
- ◊ **Etichetta → *Produce* → Disco:** un'etichetta discografica produce uno o più dischi, un disco può essere prodotto da una o più etichette discografiche.
- ◊ **Disco → *D_Genere* → Genere:** un disco possiede uno o più generi, un genere rappresenta uno o più dischi
- ◊ **Disco → *Include* → Traccia:** un disco include una (nel caso di singolo) o più tracce (nel caso di album), una traccia viene inclusa in un disco. Anche qui, si è deciso di prevedere una ridondanza per lo stesso motivo di disco (vedi descrizione della relazione *contiene*).
- ◊ **Autore → *A_Genere* → Genere:** un autore possiede uno o più generi, un genere rappresenta uno o più autori
- ◊ **Autore → *D_Compone* → Disco:** un autore compone uno o più dischi, un disco viene composto da uno o più autori. Si definisce l'attributo Ruolo per definire il ruolo che ha avuto l'autore nella realizzazione del disco (Produttore, compositore, esecutore…).
- ◊ **Autore → *T_Compone* → Traccia:** se differisce dall'intero disco, un autore compone una o più tracce, e una traccia viene composta da una o più autori. Viene definito l'attributo Ruolo per definire il ruolo che ha avuto l'autore nella realizzazione della traccia.

Progettazione logica

Ristrutturazione ed ottimizzazione del modello ER



Alla luce di: carico degli attributi delle varie entità non elevato, utilizzo degli attributi nelle query previsto, attributi derivabili non costituiti, decomposizioni di entità in relazioni già effettuate (per motivi concettuali), decomposizioni di relazioni apparentemente non necessario (in quanto non viene richiesto ad esempio uno storico), l'ottimizzazione del seguente schema riguarda in particolare l'entità Autore dove:
La generalizzazione totale di Autore diventa un attributo (mediante la fusione figli-padre) che indica semplicemente se l'autore figura come Singolo oppure Gruppo. La gestione di un attributo che può assumere due valori è meno impegnativa rispetto alla gestione di una gerarchia, e ciò sicuramente aiuta il DB a essere più efficiente.
L'attributo anagrafica viene scomposto negli attributi nome, cognome, data di nascita. Si assume che: nel caso dell'autore singolo si riportano nome, cognome e data di nascita reali. Nel caso di gruppo, nome e cognome non assumono valore, e con la data di nascita si riferirà alla data in cui il gruppo è stato fondato.

Traduzione del modello ER nel modello relazionale

Si segue il seguente schema di rappresentazione:
ENTITA': chiave primaria, attributo, chiave esterna: TABELLA RIFERITA

COLLEZIONISTA: ID, nickname, email
COLLEZIONE: ID, nome, pubbPriv, ID_Collezionista: COLLEZIONISTA
DISCO: ID, titolo, annoDiUscita, barcode, ID_Collezione: COLLEZIONE
COPIA DISCO: ID, statoConservazione, formato, ID_Disco: DISCO
IMMAGINE: URL, tipologia, ID_Disco: DISCO
ETICHETTA: nome
TRACCIA: ID, titolo, durata, ID_Disco: DISCO
AUTORE: ID, pseudonimo, nome, cognome, dataDiNascita, gruppoSingolo
GENERE: nome
condivide: ID_Collezione: COLLEZIONE, ID_Collezionista: COLLEZIONISTA
d_Compone: ruolo, ID_Disco: DISCO, ID_Autore: AUTORE
t_Compone: ruolo, ID_Traccia: TRACCIA, ID_Autore: AUTORE
d_Genere: ID, ID_Disco: DISCO, nomeGenere: GENERE
a_Genere: ID, ID_Autore: AUTORE, nomeGenere: GENERE
produce: ID, nomeEtichetta: ETICHETTA, ID_Disco: DISCO

In alcune tabelle si è potuto ottimizzare l'utilizzo della memoria, risparmiando l'ID numerico come chiave primaria, infatti: Nelle tabelle d_Compone e t_Compone (che rappresentano relazioni "molti a molti"), la tripla di attributi può già garantire univocità: nello stesso disco/traccia lo stesso autore non può svolgere lo stesso ruolo per più di una volta (sarebbe un'inutile ripetizione).
In Immagine, l'URL è univoco in quanto ogni file sul server ha il proprio percorso file (che non può essere lo stesso di un altro file). In Genere ed Etichetta il nome può garantire univocità, in quanto non avrebbe senso inserire lo stesso genere (o etichetta) con lo stesso nome più di una volta. Avere nome come chiave primaria, permette inoltre di risparmiare join durante le interrogazioni, in quanto le tabelle produce, a_Genere e d_Genere possiedono già il nome del genere a cui si riferiscono.
Bisogna però valutare la seguente situazione: le relazioni produce, a_Genere e d_Genere, avendo chiavi esterne che riferiscono a chiavi primarie VARCHAR, costituire la coppia di valori come chiave potrebbe creare problemi nel caso di inserimento di più generi o etichette per più dischi. Per semplificarne la gestione inseriamo come chiave un semplice ID numerico.

Progettazione fisica

Implementazione del modello relazionale

```
CREATE DATABASE IF NOT EXISTS Collectors;
USE Collectors;

DROP TABLE IF EXISTS collezionista;
DROP TABLE IF EXISTS collezione;
DROP TABLE IF EXISTS disco;
DROP TABLE IF EXISTS immagine;
DROP TABLE IF EXISTS copiaDisco;
DROP TABLE IF EXISTS etichetta;
DROP TABLE IF EXISTS traccia;
DROP TABLE IF EXISTS autore;
DROP TABLE IF EXISTS genere;
DROP TABLE IF EXISTS condivide;
DROP TABLE IF EXISTS d_Compone;
DROP TABLE IF EXISTS t_Compone;
DROP TABLE IF EXISTS d_Genere;
DROP TABLE IF EXISTS a_Genere;
DROP TABLE IF EXISTS produce;

CREATE TABLE collezionista (
ID INTEGER UNSIGNED PRIMARY KEY AUTO_INCREMENT,
```

```
nickname VARCHAR(50) NOT NULL UNIQUE,
email VARCHAR(100) NOT NULL UNIQUE
);

CREATE TABLE collezione (
ID INTEGER UNSIGNED PRIMARY KEY AUTO_INCREMENT,
nome VARCHAR(50) NOT NULL,
pubbPriv ENUM('Pubblica','Privata') NOT NULL DEFAULT 'Privata',
ID_Collezionista INTEGER UNSIGNED,

CONSTRAINT collezione_Collezionista FOREIGN KEY (ID_Collezionista)
REFERENCES collezionista (ID)
ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE disco (
ID INTEGER UNSIGNED PRIMARY KEY AUTO_INCREMENT,
titolo VARCHAR(100) NOT NULL,
annoDiUscita CHAR(4) NOT NULL,
barcode VARCHAR(13),
ID_Collezione INTEGER UNSIGNED,

CONSTRAINT disco_Collezione FOREIGN KEY (ID_Collezione)
REFERENCES collezione (ID)
ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE copiaDisco (
ID INTEGER UNSIGNED PRIMARY KEY AUTO_INCREMENT,
statoConservazione ENUM('Nuovo','Usato'),
formato ENUM('CD','Vinile','Digitale','Cassetta'),
ID_Disco INTEGER UNSIGNED NOT NULL,

CONSTRAINT copiaDisco_Disco FOREIGN KEY (ID_Disco)
REFERENCES disco (ID)
ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE immagine (
URL VARCHAR(500) PRIMARY KEY,
tipologia VARCHAR(50),
ID_Disco INTEGER UNSIGNED NOT NULL,

CONSTRAINT immagine_Disco FOREIGN KEY (ID_Disco)
REFERENCES disco (ID)
ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE etichetta (
nome VARCHAR(100) NOT NULL PRIMARY KEY
);

CREATE TABLE traccia (
ID INTEGER UNSIGNED PRIMARY KEY AUTO_INCREMENT,
titolo VARCHAR(100) NOT NULL,
durata TIME,
ID_Disco INTEGER UNSIGNED NOT NULL,

CONSTRAINT traccia_Disco FOREIGN KEY (ID_Disco)
REFERENCES disco (ID)
ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE autore(
ID INTEGER UNSIGNED PRIMARY KEY AUTO_INCREMENT,
pseudonimo VARCHAR(100) NOT NULL UNIQUE,
nome VARCHAR(100),
cognome VARCHAR(100),
dataDiNascita DATE,
gruppoSingolo ENUM('Gruppo','Singolo') NOT NULL
);

CREATE TABLE genere (
nome VARCHAR(50) NOT NULL PRIMARY KEY
);
```

```
CREATE TABLE condivide (
ID_Collezione INTEGER UNSIGNED NOT NULL,
ID_Collezionista INTEGER UNSIGNED NOT NULL,

CONSTRAINT condivide_Collezione FOREIGN KEY (ID_Collezione)
REFERENCES collezione (ID)
ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT condivide_Collezionista FOREIGN KEY (ID_Collezionista)
REFERENCES collezionista (ID)
ON DELETE CASCADE ON UPDATE CASCADE,

PRIMARY KEY (ID_Collezione, ID_Collezionista)
);

CREATE TABLE d_Compone (
ruolo VARCHAR(50) NOT NULL,
ID_Disco INTEGER UNSIGNED NOT NULL,
ID_Autore INTEGER UNSIGNED NOT NULL,

CONSTRAINT compone_Disco FOREIGN KEY (ID_Disco)
REFERENCES disco (ID)
ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT dCompone_Autore FOREIGN KEY (ID_Autore)
REFERENCES autore (ID)
ON DELETE CASCADE ON UPDATE CASCADE,

PRIMARY KEY (ID_Disco, ID_Autore, ruolo)
);

CREATE TABLE t_Compone (
ruolo VARCHAR(50) NOT NULL,
ID_Traccia INTEGER UNSIGNED NOT NULL,
ID_Autore INTEGER UNSIGNED NOT NULL,

CONSTRAINT compone_Traccia FOREIGN KEY (ID_Traccia)
REFERENCES traccia (ID)
ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT tCompone_Autore FOREIGN KEY (ID_Autore)
REFERENCES autore (ID)
ON DELETE CASCADE ON UPDATE CASCADE,

PRIMARY KEY (ID_Traccia, ID_Autore, ruolo)
);

CREATE TABLE d_Genere (
ID INTEGER UNSIGNED PRIMARY KEY AUTO_INCREMENT,
ID_Disco INTEGER UNSIGNED NOT NULL,
nomeGenere VARCHAR(50) NOT NULL,

CONSTRAINT dGen_Disco FOREIGN KEY (ID_Disco)
REFERENCES disco (ID)
ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT dGen_Genere FOREIGN KEY (nomeGenere)
REFERENCES genere (nome)
ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE a_Genere (
ID INTEGER UNSIGNED PRIMARY KEY AUTO_INCREMENT,
ID_Autore INTEGER UNSIGNED NOT NULL,
nomeGenere VARCHAR(50) NOT NULL,

CONSTRAINT aGen_Autore FOREIGN KEY (ID_Autore)
REFERENCES autore (ID)
ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT aGen_Genere FOREIGN KEY (nomeGenere)
REFERENCES genere (nome)
ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE produce(
```

```
ID INTEGER UNSIGNED PRIMARY KEY AUTO_INCREMENT,
nomeEtichetta VARCHAR(100) NOT NULL,
ID_Disco INTEGER UNSIGNED NOT NULL,

CONSTRAINT produce_Etichetta FOREIGN KEY (nomeEtichetta)
REFERENCES etichetta (nome)
ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT produce_Disco FOREIGN KEY (ID_Disco)
REFERENCES disco (ID)
ON DELETE CASCADE ON UPDATE CASCADE
);
```

Implementazione dei vincoli

Sono stati implementati due trigger, relativi in particolare alla condivisione di una collezione.
Il primo vincolo stabilisce che non si può condividere una collezione privata:
Si effettua un controllo sull'attributo "PubbPriv" di *collezione* (che indica se una collezione è pubblica o privata). Se la collezione che stiamo per condividere è privata, il DBMS blocca l'operazione restituendo un messaggio di errore.
Viene definita una funzione che restituisce lo stato di pubblicazione di una collezione. Quest'ultima viene richiamata all'interno di un trigger che si aziona in INSERT nella tabella condivide, ricevendo in input l'ID della collezione che sta per essere condivisa

```
DELIMITER $$
CREATE FUNCTION flagpubbPriv (ID_param CHAR(5)) RETURNS CHAR(8) DETERMINISTIC
BEGIN
RETURN (SELECT pubbPriv FROM collezione WHERE ID=ID_param);
 END$$


DELIMITER $$
CREATE TRIGGER condivide_privata BEFORE INSERT ON condivide FOR EACH ROW
BEGIN
DECLARE ID_trigger CHAR(5);
SET ID_trigger = NEW.ID_Collezione;
IF flagpubbPriv(ID_trigger)='Privata' THEN
 SIGNAL SQLSTATE '45000' SET message_text = 'Non puoi condividere una collezione privata!';
END IF;
END$$
```

Il secondo vincolo stabilisce che non si può condividere una collezione con se stessi:
Si effettua un controllo sull'attributo "ID_Collezionista" di *collezione*, per verificarne il proprietario. Se il proprietario della collezione che stiamo per condividere è lo stesso che riceve la condivisione, il DBMS blocca l'operazione restituendo un messaggio di errore.
Viene definita una funzione che restituisce l'ID del collezionista proprietario della collezione. Quest'ultima viene richiamata all'interno di un trigger che si aziona in INSERT nella tabella condivide (trigger che segue il precedente), ricevendo in input l'ID della collezione che sta per essere condivisa, e l'ID del collezionista che la riceve.

```
DELIMITER $$
CREATE FUNCTION funzCollezionista (ID_param_1 CHAR(5), ID_param_2 CHAR(5)) RETURNS CHAR(5)
DETERMINISTIC
BEGIN
RETURN (SELECT ID_Collezionista FROM collezione WHERE ID=ID_param_1 AND
ID_Collezionista=ID_param_2);
 END$$


DELIMITER $$
CREATE TRIGGER condivide_stesso BEFORE INSERT ON condivide FOR EACH ROW FOLLOWS condivide_privata
BEGIN
DECLARE ID_trigger_1 CHAR(5);
    DECLARE ID_trigger_2 CHAR(5);
SET ID_trigger_1 = NEW.ID_Collezione;
    SET ID_trigger_2 = NEW.ID_Collezionista;
IF NEW.ID_Collezionista=funzCollezionista(ID_trigger_1, ID_trigger_2) THEN
 SIGNAL SQLSTATE '45000' SET message_text = 'Non puoi condividere una collezione con te stesso!';
END IF;
END$$
```

Implementazione funzionalità richieste

Funzionalità 1

Inserimento di una nuova collezione

Si assume di inserire una nuova collezione nel database denominata “Collezione query”, che viene assegnata al collezionista Luca Franchi. Viene ripreso il suo ID mediante una sottoquery, che cerca l’ID del collezionista fornendo il suo nickname. Questo procedimento viene effettuato anche nelle query successive, per riprendere in modo facile e sicuro l’ID univoco di collezioni, dischi ecc.

```
INSERT INTO collezione (nome, pubbPriv, ID_Collezionista) VALUES
('Collezione query', 'Privata', (SELECT ID FROM collezionista WHERE nickname='luca14'));
```

Funzionalità 2

Aggiunta di dischi a una collezione e di tracce a un disco

Si assume di inserire due dischi (e due tracce all’interno di uno dei due) nella collezione inserita con la query precedente.

```
INSERT INTO disco (titolo, annodiUscita, barcode, ID_Collezione) VALUES
('Disco query', 2023, 1987509384765, (SELECT ID FROM collezione WHERE nome='Collezione query'));

INSERT INTO disco (titolo, annodiUscita, barcode, ID_Collezione) VALUES
('Disco query: Remastered', 2024, 1457596544767, (SELECT ID FROM collezione WHERE nome='Collezione
query'));

INSERT INTO traccia (titolo, durata, ID_Disco) VALUES
('Canzone Query', '03:14', (SELECT ID FROM disco WHERE titolo='Disco query'));

INSERT INTO traccia (titolo, durata, ID_Disco) VALUES
('Canzone Query 2', '05:10', (SELECT ID FROM disco WHERE titolo='Disco query'));
```

Funzionalità 3

Modifica dello stato di pubblicazione di una collezione (da privata a pubblica e viceversa) e aggiunta di nuove condivisioni a una collezione

Prendendo sempre in esame la collezione inserita con la query 1, modifichiamo lo stato di pubblicazione, e in seguito ne effettuiamo delle condivisioni con due collezionisti. La modifica dello stato di pubblicazione può essere effettuata anche al contrario, l’importante è che al momento della condivisione la collezione sia Pubblica, in quanto se si condivide una collezione privata, il DBMS bloccherà l’operazione (vedi sezione “Implementazione dei vincoli”).

```
UPDATE collezione SET pubbPriv='Privata' where nome='Collezione query';
UPDATE collezione SET pubbPriv='Pubblica' where nome='Collezione query';

INSERT INTO condivide (ID_Collezione, ID_Collezionista) VALUES
((SELECT ID FROM collezione WHERE nome='Collezione query'),
(SELECT ID FROM collezionista WHERE nickname='carlos56'));

INSERT INTO condivide (ID_Collezione, ID_Collezionista) VALUES
((SELECT ID FROM collezione WHERE nome='Collezione query'),
(SELECT ID FROM collezionista WHERE nickname='marios22'));
```

Funzionalità 4

Rimozione di un disco da una collezione

Assumiamo di rimuovere un disco con titolo “OUT” dalla collezione inserita con la query 1.

```
DELETE FROM disco WHERE titolo='OUT' AND
ID_Collezione= (SELECT ID FROM collezione WHERE nome='Collezione query');
```

Funzionalità 5

Rimozione di una collezione

Assumiamo di rimuovere una collezione nominata “Collezione out”. Siccome vi sono possono essere due collezioni con lo stesso nome ma di collezionisti diversi, per identificare correttamente la collezione da eliminare forniamo anche l’ID del collezionista proprietario della collezione.

```
DELETE FROM collezione WHERE nome='Collezione out' AND
ID_Collezionista= (SELECT ID FROM collezionista WHERE nickname='marios22');
```

Funzionalità 6

Lista di tutti i dischi in una collezione

Viene fornita la lista di tutti i dischi, assumendo che la collezione di interesse sia “Musica in movimento” del collezionista con nickname “carlos56”. Forniamo anche il genere del disco che riportiamo con `GROUP_CONCAT`, per fare in modo che: se il disco possiede più generi, questi ultimi compongono un'unica riga (quella del relativo disco).

Nome Disco	Anno di Uscita	Genere
Broken Glass	1986	Musica classica
Rockstar	2018	Trap,Rap,Rock

```
SELECT disco.titolo AS 'Nome Disco', disco.annoDiUscita AS 'Anno di Uscita',
GROUP_CONCAT(d_Genere.nomeGenere) AS 'Genere'
FROM disco JOIN collezione ON (disco.ID_Collezione = collezione.ID)
JOIN collezionista ON (collezione.ID_Collezionista = collezionista.ID)
JOIN d_Genere ON (disco.ID = d_Genere.ID_disco)
WHERE collezione.nome = 'Musica in movimento' AND collezionista.nickname = 'carlos56'
GROUP BY disco.titolo, disco.annoDiUscita;
```

Funzionalità 7

Track list di un disco

Viene fornita la lista di tutte le tracce, assumendo che il disco di interesse sia “Dream”. Per garantire che i risultati si riferiscono solo ad un determinato disco, nella clausola WHERE, oltre al titolo, viene incluso l’anno di uscita del disco e la collezione che lo contiene (*vedi sezione “Progettazione concettuale”, Relazione “contiene”*).

```
SELECT traccia.titolo AS 'Titolo', traccia.durata AS 'Durata'
FROM traccia JOIN disco ON (traccia.ID_Disco=disco.ID)
WHERE disco.titolo='Dream' AND disco.annodiUscita=2022 AND disco.ID_Collezione=1;
```

Funzionalità 8

Ricerca di dischi in base a nomi di autori/compositori/interpreti e/o titoli. Si potrà decidere di includere nella ricerca le collezioni di un certo collezionista e/o quelle condivise con lo stesso collezionista e/o quelle pubbliche

Si assume di voler cercare dischi in base al titolo “Compilation 2003” oppure al nome dell’autore “John Brave” che può ricoprire il ruolo di Autore, compositore o interprete in quel disco. Per realizzare la decisione di espandere la ricerca in varie collezioni, è stata scritta una procedura che prende come parametri in ingresso tre valori booleani. Per ogni decisione, (quindi ricerca in collezioni personali (si assume di Mario Rossi), ricerca in collezioni condivise con il collezionista(si assume con Mario Rossi) e ricerca in collezioni pubbliche) è stata realizzata una temporary table che viene popolata con la relativa query. In base ai valori booleani scelti nella chiamata della procedura (esempio `CALL query8 (1,1,1)`), le varie tabelle vengono combinate in UNION, e ciò è possibile grazie a una gestione delle varie casistiche con costrutti IF. Alla fine della procedura, le tabelle vengono eliminate per permettere di rieseguire la procedura, magari con valori diversi.

```
DELIMITER $$
CREATE PROCEDURE query8 (personal BOOLEAN, condivise BOOLEAN, pubbliche BOOLEAN)
BEGIN

CREATE TEMPORARY TABLE query8_Start AS
(
SELECT DISTINCT disco.* FROM collezione
JOIN collezionista ON (collezione.ID_Collezionista=collezionista.ID)
JOIN condivide ON (collezione.ID=condivide.ID_Collezione)
JOIN disco on (collezione.ID=disco.ID_Collezione)
JOIN d_Compone ON (disco.ID=d_Compone.ID_Disco)
JOIN autore ON (d_Compone.ID_Autore=autore.ID)
WHERE (autore.pseudonimo='John Brave' AND (d_Compone.ruolo='Autore' OR d_Compone.ruolo='Compositore'
OR d_Compone.ruolo='Interprete')) OR disco.titolo='Compilation 2003'
);

CREATE TEMPORARY TABLE query8_Personal AS
(
SELECT DISTINCT disco.* FROM collezione
JOIN collezionista ON (collezione.ID_Collezionista=collezionista.ID)
JOIN condivide ON (collezione.ID=condivide.ID_Collezione)
JOIN disco on (collezione.ID=disco.ID_Collezione)
```

```
JOIN d_Compone ON (disco.ID=d_Compone.ID_Disco)
JOIN autore ON (d_Compone.ID_Autore=autore.ID)
WHERE collezionista.nickname='marios22'
);

CREATE TEMPORARY TABLE query8_Condivise AS
(
SELECT disco.* FROM collezione
JOIN collezionista ON (collezione.ID_Collezionista=collezionista.ID)
JOIN condivide ON (collezione.ID=condivide.ID_Collezione)
JOIN disco on (collezione.ID=disco.ID_Collezione)
JOIN d_Compone ON (disco.ID=d_Compone.ID_Disco)
JOIN autore ON (d_Compone.ID_Autore=autore.ID)
WHERE condivide.ID_Collezionista=(SELECT ID FROM collezionista WHERE nickname='marios22')
);

CREATE TEMPORARY TABLE query8_Pubbliche AS
(
SELECT disco.* FROM collezione
JOIN collezionista ON (collezione.ID_Collezionista=collezionista.ID)
JOIN condivide ON (collezione.ID=condivide.ID_Collezione)
JOIN disco on (collezione.ID=disco.ID_Collezione)
JOIN d_Compone ON (disco.ID=d_Compone.ID_Disco)
JOIN autore ON (d_Compone.ID_Autore=autore.ID)
WHERE pubbPriv='Pubblica'
);
IF (personal AND condivise AND pubbliche) THEN
SELECT * FROM query8_Start
UNION
SELECT * FROM query8_Personal
UNION
SELECT * FROM query8_Condivise
UNION
SELECT * FROM query8_Pubbliche;

ELSEIF (personal AND condivise) THEN
SELECT * FROM query8_Start
UNION
SELECT * FROM query8_Personal
UNION
SELECT * FROM query8_Condivise;

ELSEIF (personal AND pubbliche) THEN
SELECT * FROM query8_Start
UNION
SELECT * FROM query8_Personal
UNION
SELECT * FROM query8_Pubbliche;

ELSEIF (condivise AND pubbliche) THEN
SELECT * FROM query8_Start
UNION
SELECT * FROM query8_Condivise
UNION
SELECT * FROM query8_Pubbliche;

ELSEIF (personal) THEN
SELECT * FROM query8_Start
UNION
SELECT * FROM query8_Personal;

ELSEIF (condivise) THEN
SELECT * FROM query8_Start
UNION
SELECT * FROM query8_Condivise;

ELSEIF (pubbliche) THEN
SELECT * FROM query8_Start
UNION
SELECT * FROM query8_Pubbliche;

ELSEIF NOT(personal AND condivise AND pubbliche) THEN
SELECT * FROM query8_Start;
END IF;
```

```
DROP TABLE query8_Start;
DROP TABLE query8_Personal;
DROP TABLE query8_Condivise;
DROP TABLE query8_Pubbliche;

END$$
```

Funzionalità 9

Verifica della visibilità di una collezione da parte di un collezionista

Si assume di verificare la visibilità da parte del collezionista Mario Rossi della collezione inserita con la query 1. Una collezione è visibile a un collezionista se: è sua, è condivisa con sé, oppure è pubblica. Se rispetta una di queste tre casistiche, la collezione verrà visualizzata in output.

```
SELECT DISTINCT collezione.nome
FROM collezione JOIN collezionista ON (collezione.ID_Collezionista=collezionista.ID)
JOIN condivide ON (collezionista.ID=condivide.ID_Collezionista)
WHERE collezione.nome='Collezione query' AND
(collezione.ID_Collezionista= (SELECT ID FROM collezionista WHERE nickname='marios22') OR
condivide.ID_Collezionista= (SELECT ID FROM collezionista WHERE nickname='marios22') OR
collezione.pubbPriv='Pubblica');
```

Funzionalità 10

Numero dei brani (tracce di dischi) distinti di un certo autore (compositore, musicista) presenti nelle collezioni pubbliche

Si assume di voler visualizzare il numero di brani dell'Autore "Mac Hollister" presenti nelle collezioni pubbliche, dove ha contribuito come Musicista o Compositore. Vengono utilizzate le funzioni `COUNT` e `DISTINCT` per costituire il numero di brani totali distinti.

```
SELECT COUNT(DISTINCT traccia.titolo) AS 'Numero brani'
FROM traccia JOIN disco ON (traccia.ID_Disco=disco.ID)
JOIN collezione ON (disco.ID_Collezione=collezione.ID)
JOIN t_Compone ON (traccia.ID=t_Compone.ID_Traccia)
JOIN autore ON (t_Compone.ID_Autore=autore.ID)
WHERE autore.pseudonimo='Mac Hollister' AND
(t_Compone.ruolo='Musicista' OR t_Compone.ruolo='Compositore') AND
collezione.pubbPriv='Pubblica';
```

Funzionalità 11

Minuti totali di musica riferibili a un certo autore (compositore, musicista) memorizzati nelle collezioni pubbliche

Si assume di voler visualizzare i minuti totali di musica dell'Autore "Mac Hollister" presenti nelle collezioni pubbliche, dove ha contribuito come Musicista o Compositore. In questo caso, avendo memorizzato la durata dei brani in TIME, viene anzitutto effettuato un "cast" della durata dal formato TIME in secondi (con la funzione `TIME_TO_SEC`), poi viene effettuata la somma di tutti i secondi (riferiti alla durata dei brani), e infine il valore finale viene convertito nuovamente in TIME, visualizzando la durata totale in formato *ore:minuti:secondi.*

```
SELECT SEC_TO_TIME(SUM(TIME_TO_SEC(traccia.durata))) AS 'Minuti totali'
FROM traccia JOIN disco ON (traccia.ID_Disco=disco.ID)
JOIN collezione ON (disco.ID_Collezione=collezione.ID)
JOIN t_Compone ON (traccia.ID=t_Compone.ID_Traccia) JOIN autore ON (t_Compone.ID_Autore=autore.ID)
WHERE autore.pseudonimo='Mac Hollister' AND (t_Compone.ruolo='Musicista' OR
t_Compone.ruolo='Compositore') AND
collezione.pubbPriv='Pubblica';
```

Funzionalità 12

Statistiche (una query per ciascun valore): numero di collezioni di ciascun collezionista, numero di dischi per genere nel sistema

Si utilizza la funzione `GROUP BY` per raggruppare le collezioni per nickname del collezionista (e i dischi per genere nel secondo caso), `COUNT` per evidenziare il numero di collezioni che ogni collezionista possiede(e il numero di dischi per ogni genere nel secondo caso).

```
SELECT collezionista.nickname, COUNT(collezione.ID_Collezionista) AS numCollezioni
FROM collezionista JOIN collezione ON (collezionista.ID=collezione.ID_Collezionista)
GROUP BY collezionista.nickname
```

```
SELECT d_Genere.nomeGenere, COUNT(d_Genere.ID_Disco) AS numDischi
FROM d_Genere
GROUP BY d_Genere.nomeGenere
```

Funzionalità 13

Opzionalmente, dati un numero di barcode, un titolo e il nome di un autore, individuare tutti i dischi presenti nelle collezioni che sono più coerenti con questi dati (funzionalità utile, ad esempio, per individuare un disco già presente nel sistema prima di inserirne un doppione). L'idea è che il barcode è univoco, quindi i dischi con lo stesso barcode sono senz'altro molto coerenti, dopodichè è possibile cercare dischi con titolo simile e/o con l'autore dato, assegnando maggior punteggio di somiglianza a quelli che hanno più corrispondenze

Si scrive una procedura, che prende come parametro in input tre valori (barcode di un disco, titolo di un disco, pseudonimo di un autore), e cerca tutti i dischi i cui attributi “si avvicinano ai valori forniti”, attraverso la funzione `LIKE` (`CONCAT` utilizzato per far si che potessimo inserire il parametro della procedura tra i metacaratteri “%”).

```
DELIMITER $$
CREATE PROCEDURE query13(barcode_param VARCHAR(13), titolo_param VARCHAR(50), pseudonimo_param
VARCHAR(50))
BEGIN
SELECT disco.* FROM disco
JOIN d_Compone ON (disco.ID=d_compone.ID_Disco)
JOIN autore ON (d_Compone.ID_Autore=Autore.ID)
WHERE disco.barcode LIKE CONCAT('%',barcode_param,'%') OR
disco.titolo LIKE CONCAT('%',titolo_param,'%') OR
autore.pseudonimo LIKE CONCAT('%',pseudonimo_param,'%');
END$$
```

Interfaccia verso il database

Abbiamo provato a realizzare una semplice interfaccia grafica in PHP (codice in allegato) che permette, attraverso un’iterazione con pulsanti, di eseguire le funzionalità precedentemente realizzate e visualizzarne il risultato.
L’idea di base è quella di costituire una pagina HTML/CSS responsive con all’interno un form che contiene tutti i pulsanti con cui eseguire le query. Alla pressione di uno dei pulsanti il form invia la richiesta, e il PHP, attraverso funzioni dell’estensione “MySQLi”, si connette al DB, esegue la relativa query, e chiude la connessione.

Tuttavia, il codice non funziona perfettamente, in quanto non siamo riusciti a visualizzare il risultato della query una volta richiesto di eseguirla.