Università di Catania

DATABASE PER LA GESTIONE DI EVENTI DI INTRATTENIMENTO

Materia: Basi di dati

Docente: Pulvirenti Alfredo

Studente: Catalano Sophia Grazia

Matricola: 1000030486

Sommario

DESCRIZIONE DEL PROGETTO

Di seguito verrà descritto un progetto che riguarda l'implementazione di un database utile per la gestione di eventi di intrattenimento.

Verrà effettuata un'analisi dei requisiti richiesti, seguita dalla progettazione concettuale, logica e fisica. Successivamente si valuterà se introdurre o meno delle ridondanze.

Questo tipo di progetto è volto a creare solo la struttura del database, tutto il resto delle operazioni, come il controllo di validità di un campo, verranno successivamente gestite da un server.

Analisi dei requisiti

Come abbiamo anticipato il database si occuperà di gestire degli eventi, ognuno dei quali avrà un id che lo rappresenta, una data, un luogo e un nome.

Ovviamente si gestiranno tipologie di eventi differenti, e per questo ogni evento apparterrà ad una categoria; avremo quindi un id e una descrizione (concerto, teatro…).

Gli eventi avranno come protagonisti degli artisti, che si occuperanno dell'organizzazione dell'evento. Per ognuno di essi avremo un id, un nome (completo oppure nome d'arte) e un tipo, che rappresenta la loro arte (cantante, ballerino…).

Per ogni evento è data la possibilità di scegliere un pacchetto, rappresentato da un id, numero di posti disponibili, tipologia (standard, VIP…) e naturalmente da un costo.

Ogni utente che farà un acquisto relativo ad un pacchetto sarà identificato tramite la sua e-mail, oltre che da nominativo e data di nascita.

Infine, è stata aggiunta la possibilità di inserire dei codici promozionali, identificati dal codice stesso e da una percentuale, che verrà sottratta al costo del pacchetto, in modo da poter calcolare il prezzo totale dell'acquisto effettuato.

Glossario dei termini

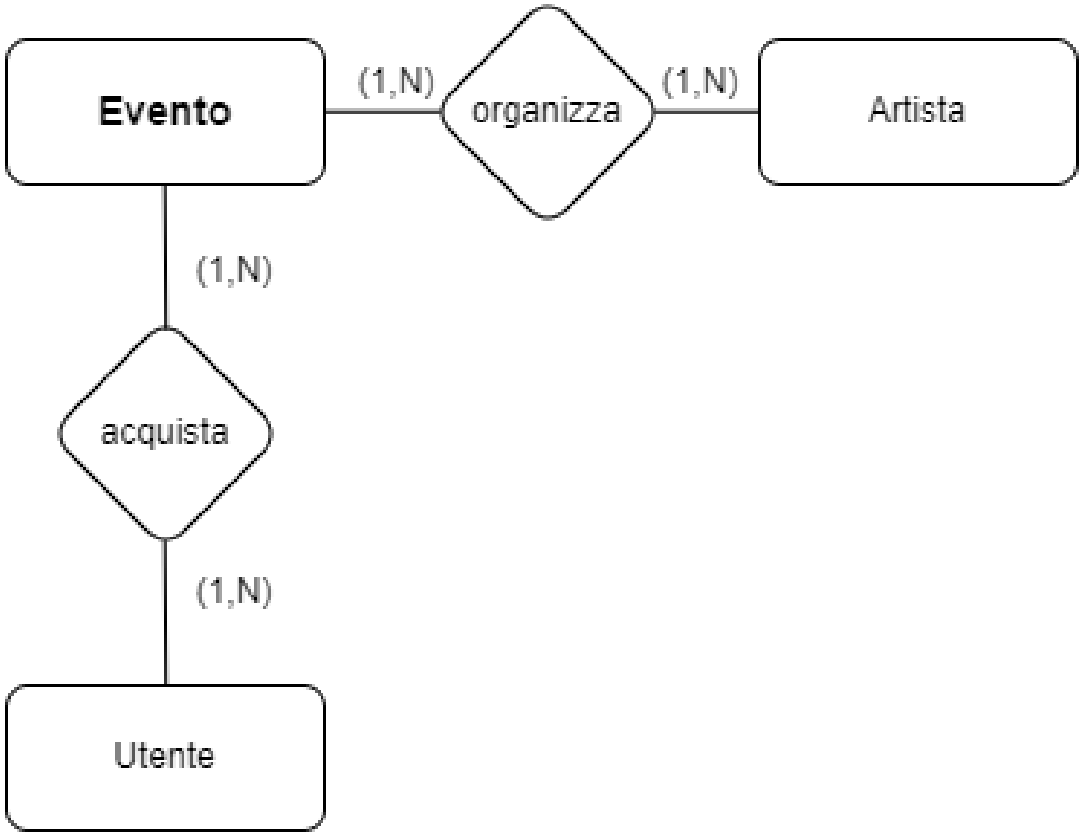
Termini	Descrizione	Sinonimi	Legami
evento	Spettacolo di intrattenimento	• spettacolo • occorrenza	• pacchetto • categoria • artista
categoria	Tipologia dell'evento	• tipologia	• evento
artista	Organizzatore e protagonista dell'evento	• protagonista • organizzatore	• evento
pacchetto	Tipologia del biglietto	• biglietto	• evento • acquisto
acquisto	Pagamento del biglietto scelto per l'evento	• pagamento	• utente • codice_promo
codice_promo	Sconto da applicare all'acquisto effettuato	• sconto	• acquisto
utente	Entità che partecipa all'evento	• partecipante	• acquisto

PROGETTAZIONE CONCETTUALE

Lo schema ER verrà realizzato utilizzando il metodo top-down.

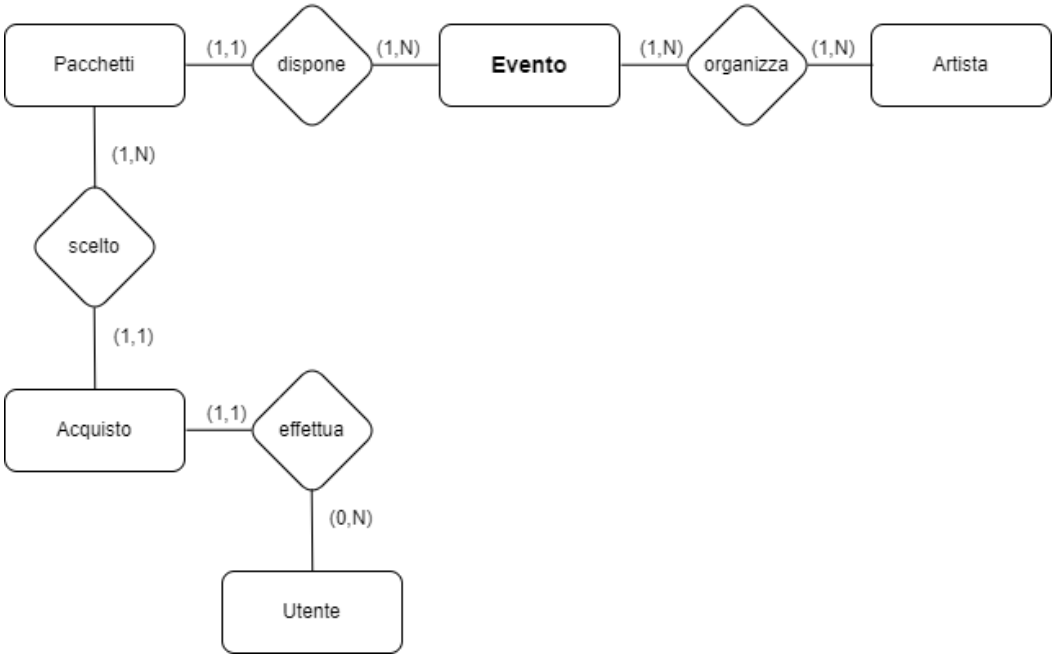
Schema generale

A seguire verrà riportato lo schema scheletro, composto dalle macro-componenti del progetto; infatti, le operazioni più importanti riguardano l'organizzazione dell'evento e la partecipazione da parte di un utente.



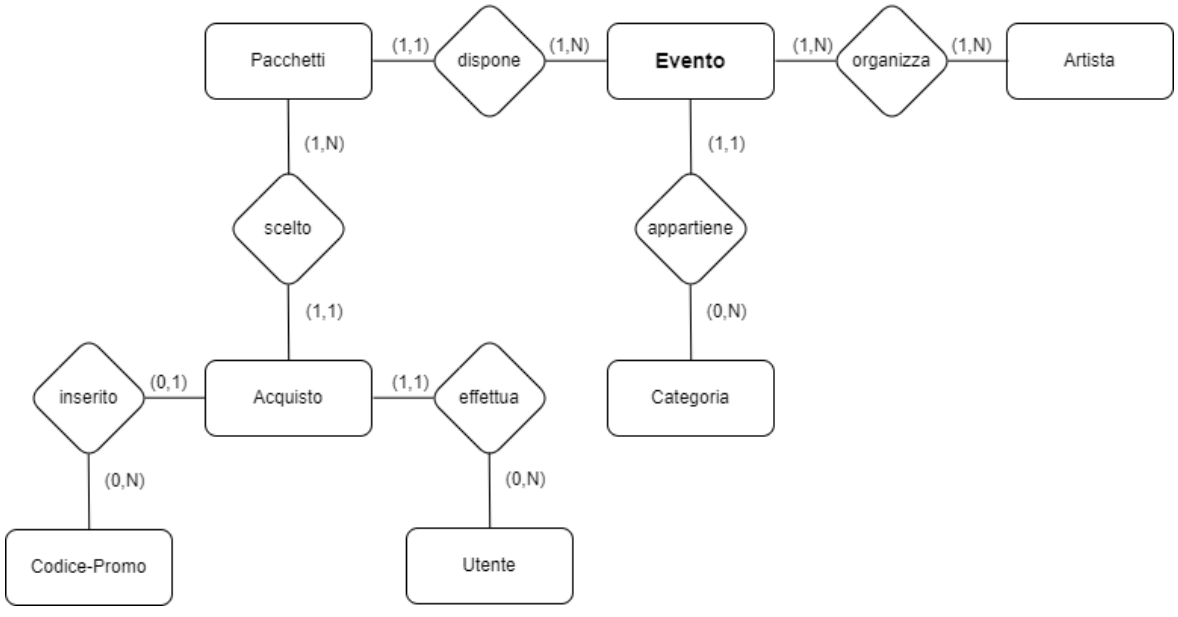
Espansione schema scheletro

A questo punto viene aggiunta la possibilità di scegliere una tipologia di pacchetto per l'evento.



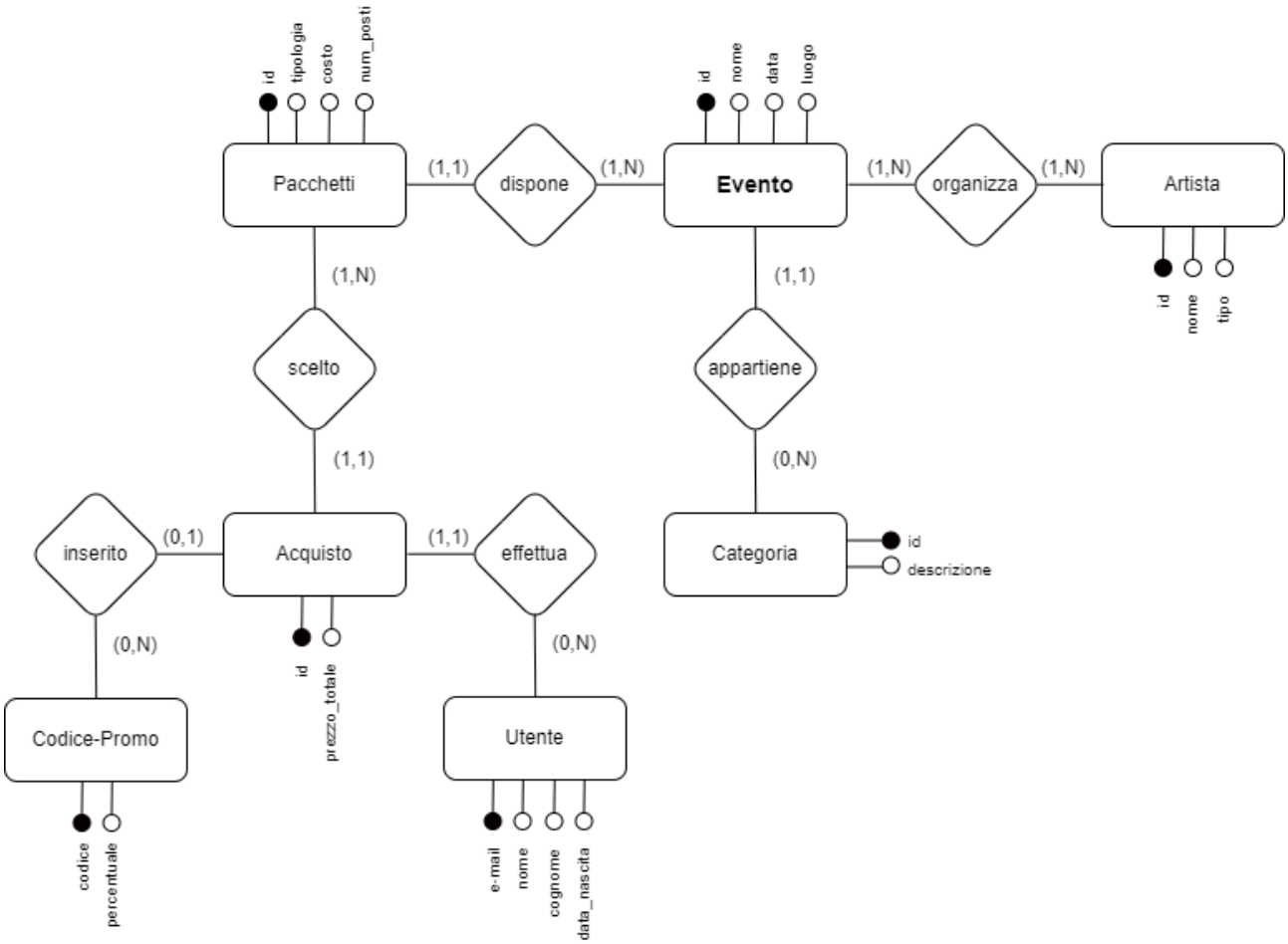
Codici-Promozionali

Adesso verranno categorizzate le tipologie degli eventi. E per finire verrà aggiunta la possibilità di inserire dei codici promozionali sugli acquisti.



Schema completo

Infine, viene proposto lo schema presente sopra, a cui verranno aggiunti tutti gli attributi delle entità, e se dovessero esserci, anche quelli delle relazioni.



Cardinalità delle relazioni

Evento <-> Artista

- Un evento può essere organizzato da diversi artisti (1,N)
- Un'artista può organizzare uno o più eventi (1,N)

Evento <-> Categoria

- Un evento appartiene ad una sola categoria (1,1)
- Ad una categoria possono appartenere zero o più eventi (0,N)

Pacchetto <-> Evento

- Un evento dispone di più pacchetti (1,N)
- Uno specifico pacchetto è a disposizione di un solo evento (1,1)

Acquisto <-> Pacchetto

- Un pacchetto può essere scelto in più acquisti (1,N)
- In un acquisto può essere scelto un solo tipo di pacchetto (1,1)

Codice-Promo <-> Acquisto

- Un codice promozionale può essere inserito in zero o più acquisti (0,N)
- In un acquisto può essere, o non, inserito un solo codice promozionale (0,1)

Utente <-> Acquisto

- Un utente può effettuare zero o più acquisti (0,N)
- Un acquisto può essere effettuato da un solo utente (0,1)

Dizionario Dati-Entità

Entità	Descrizione	Attributi	Identificatore
evento	Spettacolo di intrattenimento	id, nome, data, luogo	id
categoria	Tipologie di eventi	id, descrizione	id
artista	Organizzatore dell'evento	id, nome, tipo	id
pacchetto	Tipologia di biglietto	id, tipologia, costo, num_posti	id
acquisto	Acquisto del biglietto per l'evento	id , prezzo_totale	id
utente	Cliente che acquista nel sito	email, nome, cognome, data nascita	email
codice_promo	Sconto sull'acquisto	codice, percentuale	codice

Dizionario Dati-Relazioni

Relazione	Entità partecipanti	Descrizione	Attributi
effettua	utente, acquisto	Ogni acquisto viene effettuato da un cliente	
inserito	codice_promo, acquisto	Un cliente può scegliere di inserire un codice nel suo acquisto	
scelto	pacchetto, acquisto	In ogni acquisto deve essere scelto un pacchetto	
dispone	evento, pacchetto	Ogni evento ha diversi tipi di pacchetti	
appartiene	evento, categoria	Associa ogni evento alla sua categoria	
organizza	artista, evento	Un evento viene organizzato da degli artisti	artista(FK) evento(FK)

N.B. FK = foreign key (chiave esterna).

PROGGETTAZIONE LOGICA

Tabella dei volumi

Concetto	Tipo	Volume
artista	*E*	*200*
evento	*E*	*1 000*
categoria	*E*	*50*
pacchetto	*E*	*3 000 (3 pacchetti per evento)*
acquisto	*E*	*2 000 000*
codice_promo	*E*	*5 000*
utente	*E*	*20 000*
organizza	*R*	*1 000 (circa 5 eventi per artista)*

Tabella delle frequenze

Operazione	***Descrizione***	***Frequenza***	***Tipo***
O1	Inserisci un nuovo evento	200/giorno	I
O2	Inserisci un acquisto	1 000/giorno	I
O3	Modifica la data di un evento	6/giorno	I
O4	Elimina un utente	10/mese	B
O5	Stampa l'artista che ha organizzato più eventi	1/mese	B
O6	Stampa tutte le date dei concerti di un artista	1 /giorno	I
07	Stampa gli utenti che hanno effettuato il maggior numero di acquisti	1/mese	B
08	Stampa i codici promozionali che non sono mai stati usati	1/mese	B

Analisi delle ridondanze

Considerando la progettazione concettuale, lo schema attualmente non presenta alcuna ridondanza.

Sappiamo di avere 2 000 000 acquisti e 20 000 utenti nel nostro database. Ipotizziamo quindi di dover svolgere le seguenti operazioni:

01: Inserimento di un acquisto da parte di un utente. Frequenza 200 al giorno.

02: Calcolo del numero di acquisti di un utente. Frequenza 1 al giorno.

Dobbiamo stabilire se conviene inserire o meno l'attributo ridondante "numero_acquisti" su utente.

	Con ridondanza	***Senza ridondanza***
O1	1 scrittura su utente 1 scrittura su acquisto **Tot: 2S = 4L -> 800**	1 scrittura su acquisto **Tot: 1S = 2L -> 400**
02	1 lettura su utente **Tot: 1L -> 1**	100 letture su acquisto **Tot: 100L -> 100**
Costo finale	801	500

In conclusione, non conviene aggiungere l'attributo ridondante in quanto le letture effettuate all'interno del database che non lo contiene sono nettamente inferiori.

Traduzione Entità/Relazioni

Si indicheranno le chiavi primarie con sottolineature continue, e le chiavi esterne tramite sottolineature tratteggiate.

CATEGORIA: (id, descrizione)

EVENTO: (id, categoria, nome, data, luogo)

ARTISTA: (id, nome, tipo)

ORGANIZZA (artista, evento)

PACCHETTO: (id, evento , tipologia, costo, num_posti)

UTENTE: (email, nome, cognome, data_nascita)

CODICE_PROMO: (codice, percentuale)

ACQUISTO: (id, pacchetto, utente, promo, prezzo_totale)

PROGETTAZIONE FISICA

In questa parte del documento verranno implementate le tabelle, i trigger e le operazioni, il tutto attraverso il linguaggio di interrogazione SQL.

Implementazione delle tabelle

```
CREATE TABLE categoria (
  id int(5) AUTO_INCREMENT PRIMARY KEY,
  descrizione  varchar(40) NOT NULL
);

CREATE TABLE evento (
id int(5) AUTO_INCREMENT PRIMARY KEY,
categoria int(5),
nome varchar(40),
data datetime NOT NULL,
luogo varchar(20) NOT NULL,
FOREING KEY (categoria) REFERENCES categoria(id)
ON UPDATE RESTRICT ON DELETE RESTRICT
);
```

```
CREATE TABLE artista (
id int(5) AUTO_INCREMENT PRIMARY KEY,
nome varchar(20),
tipo varchar(15) NOT NULL
);

CREATE TABLE organizza (
artista int(5),
evento int(5),
FOREIGN KEY (artista) REFERENCES artista(id)
ON UPDATE RESTRICT ON DELETE RESTRICT,
FOREIGN KEY (evento) REFERENCES evento(id)
ON UPDATE RESTRICT ON DELETE RESTRICT,
PRIMARY KEY (artista, evento)
);

CREATE TABLE pacchetto (
id int(5) AUTO_INCREMENT PRIMARY KEY,
evento int(5),
tipologia varchar(30) NOT NULL,
costo float NOT NULL,
num_posti int(10) NOT NULL,
FOREIGN KEY evento REFERENCES evento(id)
ON UPDATE RESTRICT ON DELETE RESTRICT
);
```

```
CREATE TABLE utente (
email varchar(30) PRIMARY KEY,
nome varchar(15),
cognome varchar(15),
data_nascita date
);

CREATE TABLE codice_promo (
codice varchar(20) PRIMARY KEY,
percentuale float NOT NULL
);

CREATE TABLE acquisto (
id int(5) AUTO_INCREMENT PRIMARY KEY,
pacchetto int(5),
utente varchar(30),
promo varchar(15),
prezzo_totale float DEFAULT NULL,
FOREIGN KEY pacchetto REFERENCES pacchetto(id),
ON UPDATE RESTRICT ON DELETE RESTRICT,
FOREIGN KEY utente REFERENCES utente(e-mail)
ON UPDATE RESTRICT ON DELETE RESTRICT,
FOREIGN KEY promo REFERENCES codice-promo(codice)
ON UPDATE RESTRICT ON DELETE RESTRICT
);
```

Implementazione dei trigger

- Controlla se l'utente ha già utilizzato un determinato codice sconto:

```
CREATE TRIGGER controllo_codice_usato
BEFORE INSERT ON acquisto
FOR EACH ROW
BEGIN
    DECLARE x INT;
    SELECT COUNT(*) INTO x
    FROM acquisto
    WHERE promo = NEW.promo
    AND utente = NEW.utente;
        IF x > 0 THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Questo codice promo è già stato utilizzato
            da questo utente.';
    END IF;
END;
```

- Aggiorna il numero di posti disponibili dopo ogni acquisto:

```
CREATE TRIGGER aggiorna_posti_disponibili
AFTER INSERT ON acquisto
FOR EACH ROW
BEGIN
    UPDATE pacchetto
    SET num_posti = num_posti - 1
    WHERE id = NEW.pacchetto;
END;
```

- Calcola il prezzo totale dell’acquisto:

```
CREATE TRIGGER calcola_prezzo_totale
BEFORE INSERT ON acquisto
FOR EACH ROW
BEGIN
    DECLARE prezzo_pacchetto DECIMAL(10,2);
    DECLARE sconto DECIMAL(5,2);
    SELECT costo INTO prezzo_pacchetto
        FROM pacchetto WHERE id = NEW.pacchetto;
    SELECT percentuale INTO sconto
        FROM codice_promo WHERE codice = NEW.promo;
    IF sconto IS NOT NULL THEN
        SET NEW.prezzo_totale = prezzo_pacchetto - (prezzo_pacchetto / 100 * sconto);
    ELSE
        SET NEW.prezzo_totale = prezzo_pacchetto;
    END IF;
END;
```

- Controlla se il pacchetto scelto ha ancora dei posti disponibili

```
CREATE TRIGGER controllo_posti_disponibili
BEFORE INSERT ON acquisto
FOR EACH ROW
BEGIN
        IF (SELECT num_posti
            FROM pacchetto
            WHERE id = NEW.pacchetto) < 1
        THEN
                SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'il pacchetto selezionato ';
        END IF;
END;
```

Implementazione delle operazioni

- Inserisci un nuovo evento:

```
INSERT INTO evento (id, nome, data, luogo, categoria)
VALUES ([id_evento], [nome_evento], [data_evento], [luogo_evento],
        [categoria_evento]);
```

- Inserisci un acquisto:

```
INSERT INTO acquisto (id, prezzo_totale, pacchetto, utente, promo)
VALUES ([id_acquisto], [prezzo_finale_acquisto], [pacchetto_scelto],
        [utente_che_acquista], [promo_usata]);
```

- Modifica la data di un evento:

```
UPADE evento
SET data = [data_nuova]
WHERE id = [id_evento];
```

- Elimina un utente:

```
DELETE
FROM utente
WHERE email = [email_utente];
```

- Stampa l’artista che ha organizzato più eventi:

```
CREATE VIEW counteventi AS
SELECT artista.nome, COUNT(*) AS numEventi
FROM organizza, evento, artista
WHERE evento = evento.id
AND artista = artista.id
GROUP BY artista

SELECT *
FROM counteventi
WHERE numEventi >= (SELECT MAX(numEventi)
                    FROM counteventi);
```

- Stampa tutte le date dei concerti di Blanco:

```
SELECT E.nome, E.data, E.luogo
FROM categoria C
JOIN evento E ON C.id = E.categoria
JOIN organizza O ON E.id = O.evento
JOIN artista A ON O.artista = A.id
WHERE C.descrizione = ‘Concerto’
AND A.nome = ‘Blanco’;
```

- Stampa gli utenti che hanno effettuato il minor numero di acquisti:

```
CREATE VIEW numacquisti AS
SELECT COUNT(*) AS acquistiEffettuati, utente
FROM acquisto
GROUP BY utente


SELECT U.email, U.nome, U.cognome, acquistiEffettuati
FROM utente U, numacquisti N
WHERE email = utente
AND acquistiEffettuati <= (SELECT MIN(acquistiEffettuati)
                           FROM numacquisti);
```

- Stampa i codici promozionali che non sono mai stati usati:

```
SELECT codice
FROM codice_promo LEFT JOIN acquisto ON codice = promo
AND id IS NULL;
```