Università di Catania

Dipartimento di Matematica e Informatica
Corso di Laurea Triennale in Informatica

Database Project

Event Management System

A MySQL Relational Database Implementation

Autore
Alessandro Ferrante

Docente
Prof. Alfredo Pulvirenti

Anno Accademico 2024-2025
Dicembre 2025

Sommario

Il presente elaborato illustra l'analisi, la progettazione e l'implementazione di una base di dati relazionale a supporto di un sistema di *Event Management*. Il progetto nasce dall'esigenza di gestire in modo integrato l'intero processo organizzativo degli eventi, comprendendo la pianificazione logistica in location a capienza limitata, la gestione dei contratti di sponsorizzazione e il complesso processo di vendita dei biglietti e gestione degli ingressi.

Partendo dall'analisi dei requisiti, è stato sviluppato uno schema concettuale ed evoluto successivamente in uno schema logico ottimizzato, ponendo particolare attenzione alla normalizzazione e alla gestione delle ridondanze. L'implementazione fisica, realizzata su piattaforma MySQL/MariaDB, fa ampio uso di logica attiva e procedurale per garantire l'integrità dei dati.

Indice

Capitolo 1

Analisi dei Requisiti

1.1 Analisi dei Requisiti

Input Iniziale

Il progetto richiede la progettazione di una base di dati sulla base della seguente specifica assegnata dal docente:

> *"Sistema di Gestione di Eventi. La base di dati deve gestire eventi, partecipanti, iscrizioni, location e sponsor. Ogni evento ha un nome, data, orario, descrizione e luogo. I partecipanti possono registrarsi agli eventi e l'organizzazione deve monitorare le iscrizioni. Gli sponsor devono essere associati a eventi specifici per la gestione dei contratti e degli sponsorizzati."*

Descrizione della Realtà

Sulla base dell'input iniziale, è stato analizzato il dominio applicativo per individuare le specifiche necessarie al funzionamento del sistema.

Il sistema è concepito per un'organizzazione che pianifica eventi di vario genere (es. conferenze, concerti) in diversi luoghi fisici. Ogni **Evento** è collocato in una specifica dimensione temporale (Data e Ora) e spaziale (**Location**). Un vincolo fondamentale emerso dall'analisi è la necessità di rispettare la **capienza massima** delle strutture per garantire la sicurezza e impedire il sovraffollamento.

Gli attori che interagiscono col sistema sono:

- **Organizzatori**: Gestiscono il calendario, le location e i contratti.
- **Partecipanti**: I fruitori finali che accedono all'evento.
- **Sponsor**: Enti esterni che finanziano gli eventi.

Individuazione delle Entità e Relazioni

In una prima fase di analisi del testo, sono state individuate le **4 entità fondamentali** che compongono il dominio:

1. **Evento**: L'oggetto centrale del sistema, caratterizzato da titolo, descrizione e collocazione temporale.

2. **Location**: Il luogo fisico che ospita l'evento, caratterizzato da indirizzo e capienza.

3. **Partecipante**: La persona fisica interessata a prendere parte all'evento.

4. **Sponsor**: L'azienda partner che contribuisce economicamente.

Le interazioni tra queste entità sono state inizialmente modellate come semplici relazioni:

- **Iscrizione:** Relazione tra *Partecipante* ed *Evento*.
- **Sponsorizzazione:** Relazione tra *Sponsor* ed *Evento*.

Raffinamento dei Requisiti ed Evoluzione del Modello

Approfondendo l'analisi per adattarla a uno scenario reale e professionale, il modello ha subito un'evoluzione in tre fasi logiche successive:

1. Evoluzione delle Relazioni in Entità (Promozione) Le relazioni iniziali *Iscrizione* e *Sponsorizzazione* si sono rivelate insufficienti se trattate come semplici collegamenti.

- La **Sponsorizzazione** necessita di attributi propri quali il *Valore del Contratto*, la *Data Inizio* e *Data Fine* validità, e il *Livello* (es. Gold, Silver). È diventata quindi un'entità (o associazione forte) denominata **Sponsorship**.
- L'**Iscrizione** necessita di storicizzare la *Data di Registrazione* e il *Prezzo d'acquisto* (che potrebbe variare nel tempo rispetto al prezzo di listino). È diventata quindi l'entità **Registration**.

2. Distinzione tra "Chi Paga" e "Chi Partecipa" Nel modello iniziale, il Partecipante era l'unico attore. Tuttavia, in un sistema di biglietteria reale, spesso chi acquista non coincide con chi usufruisce del servizio (es. un genitore che iscrive un figlio, o un capogruppo che iscrive amici). È stato quindi introdotto l'**User Account**: l'utente registrato con credenziali (Email/Password) che effettua l'accesso e il pagamento, separandolo dal **Partecipante** (l'intestatario del biglietto).

3. Gestione degli Ordini (Booking) Per permettere a un User Account di acquistare più biglietti in un'unica transazione, è stata introdotta l'entità **Booking**. Il Booking funge da contenitore per una o più *Registrations*, semplificando la gestione dei pagamenti e lo storico ordini.

Glossario dei termini

A seguito del raffinamento, il glossario completo del sistema è il seguente:

Termine	Descrizione
Event	Evento programmato in data e luogo specifici.
Location	Luogo fisico con capienza limitata che ospita l'evento.
Sponsor	Azienda o ente che contribuisce economicamente.
User Account	L'utente che possiede le credenziali, effettua l'accesso e paga.
Participant	La persona fisica titolare del biglietto (nome, cognome, età).
Booking	L'ordine d'acquisto complessivo.
Registration	Il singolo biglietto emesso per un partecipante.
Sponsorship	Il contratto economico tra uno Sponsor e un Evento.

Tabella 1.1: Glossario dei termini definitivo

Dati di carattere generale

Il sistema dovrà gestire le seguenti categorie di dati:

- **Anagrafica Utenti:** Email (login univoco), Password (memorizzata come Hash per sicurezza), Data registrazione.
- **Anagrafica Partecipanti:** Nome, Cognome, Data di Nascita (necessaria per il calcolo dell'età e verifica requisiti), Email di contatto.
- **Dettagli Evento:** Titolo, Descrizione, Data, Orari (Inizio/Fine), Prezzo del biglietto, Età minima richiesta (*Min_ Age*).
- **Dati Finanziari:** Valore dei contratti di sponsorizzazione e importi pagati per le iscrizioni (bloccati al momento dell'acquisto).
- **Identificativi:** Ogni registrazione dovrà produrre un **Ticket Code univoco** per la validazione all'ingresso.

Vincoli di Integrità e di Dominio

Per garantire la consistenza, l'integrità e la qualità dei dati, il sistema implementa i seguenti vincoli (gestiti tramite DDL e logica procedurale):

1. **Capienza Location (Sold Out):** Il numero di registrazioni attive non può superare la *Max_ Seats* della Location. Questo controllo è bloccante (Trigger).
2. **Requisito Età:** L'iscrizione è inibita se l'età calcolata del partecipante è inferiore al limite *Min_ Age* dell'evento.

3. **Atomicità delle Transazioni (ACID):** Il processo di registrazione è atomico. In caso di iscrizioni di gruppo, o tutti i partecipanti vengono registrati con successo, o l'intera operazione viene annullata (*Rollback*), evitando la memorizzazione di dati parziali o orfani.

4. **Gestione Gerarchica delle Cancellazioni:** Il sistema adotta una politica ibrida per l'integrità referenziale. Tra *User Account* e *Partecipante* è attivo un vincolo *Cascade*: l'eliminazione di un utente rimuove i profili anagrafici collegati inutilizzati. Tuttavia, se un partecipante possiede biglietti attivi (*Registration*), il vincolo `RESTRICT` a valle blocca l'intera catena di cancellazione, impedendo la perdita di dati contabili e garantendo che un utente con ordini attivi non possa essere rimosso accidentalmente.

5. **Unicità Spazio-Temporale:** È fisicamente impossibile che due eventi diversi occupino la stessa Location nello stesso momento. Tale vincolo è garantito a livello di schema relazionale tramite un indice `UNIQUE` **composto** sugli attributi `(ID_Location, Event_Date, Event_Start_Time)`. Questo impedisce strutturalmente al DBMS di accettare due record che condividano lo stesso luogo, data e ora di inizio.

6. **Coerenza Temporale Sponsor:** La data di fine contratto deve essere successiva a quella di inizio, e il periodo deve includere la data dell'evento.

7. **Storicizzazione del Prezzo:** Il prezzo del biglietto viene "congelato" al momento dell'acquisto per ottenere dati statistici validi, così da non essere influenzati da possibili future variazioni di prezzo del biglietto.

8. **Validità Formale dei Dati:** Sono imposti vincoli di formato, come la presenza del carattere '@' nelle email e l'obbligo di valori positivi per prezzi e capienze.

Capitolo 2

Progettazione

2.1 Progettazione Concettuale

La progettazione concettuale rappresenta il passaggio dai requisiti una descrizione formale e schematica della base di dati. Per gestire la complessità del sistema, la modellazione è avvenuta in tre fasi evolutive: Scheletro, Intermedio e Finale.

Schema Scheletro

In una prima fase di modellazione, basata sulla lettura preliminare dei requisiti, è stato elaborato uno schema essenziale che evidenzia solo le entità macroscopiche del dominio.

In questo schema semplificato:

- L'**Evento** è l'entità centrale.
- Il **Partecipante** interagisce direttamente con l'evento.
- La **Location** e lo **Sponsor** sono entità anch'esse.

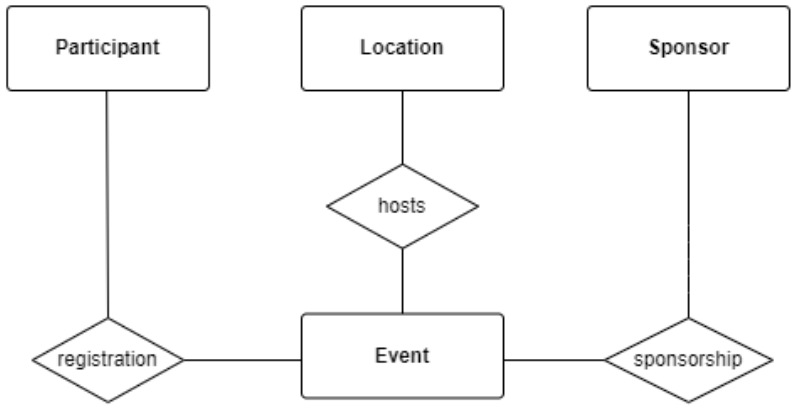

Figura 2.1: Schema E-R Scheletro: Modellazione iniziale

Schema Intermedio

In questa fase di raffinamento, il modello ha subito due cambiamenti strutturali per superare i limiti dello schema scheletro.

Innanzitutto, le associazioni *Iscrizione* e *Sponsorizzazione* sono state convertite a entità (rispettivamente **Registration** e **Sponsorship**). Questa trasformazione si è resa necessaria prima di ogni altra modifica per poter memorizzare attributi storici fondamentali come il prezzo d'acquisto, la data di registrazione e i dettagli contrattuali, che una semplice relazione non avrebbe potuto ospitare.

Successivamente, sulla base di queste nuove entità, è stata introdotta la distinzione tra i ruoli operativi:

- **User Account**: Introdotto per separare il soggetto pagante (titolare delle credenziali) dal fruitore del servizio (Partecipante).
- **Booking**: Introdotto come entità aggregatrice per raggruppare più *Registrations* in un'unica transazione economica.

Lo schema intermedio presenta quindi già le relazioni e la nuova struttura gerarchica degli utenti, ponendo le basi per la definizione finale dei vincoli.

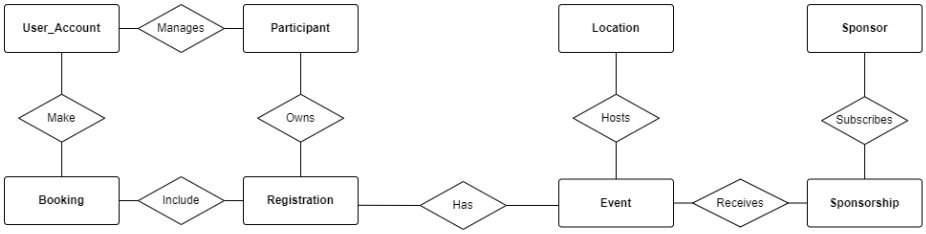

Figura 2.2: Schema E-R Intermedio

Schema E-R Finale

Lo schema E-R finale rappresenta il punto di arrivo del processo di modellazione, integrando tutte le entità e le relazioni emerse durante l'analisi in un'unica visione coerente.

A differenza delle versioni precedenti, questo diagramma fornisce una visione consolidata, in cui:

- Tutte le entità sono arricchite con il set completo dei propri attributi (es. prezzi, date di validità, credenziali, indirizzi).
- Sono state formalizzate le cardinalità precise per ogni associazione (es. la dipendenza gerarchica tra *User Account* e *Booking*, o il vincolo di capienza tra *Location* ed *Evento*).
- Sono stati definiti gli identificatori primari per ogni entità, per la traduzione verso il modello logico relazionale (tabelle e chiavi esterne).

Questo schema costituisce il riferimento per l'implementazione dei vincoli di integrità.

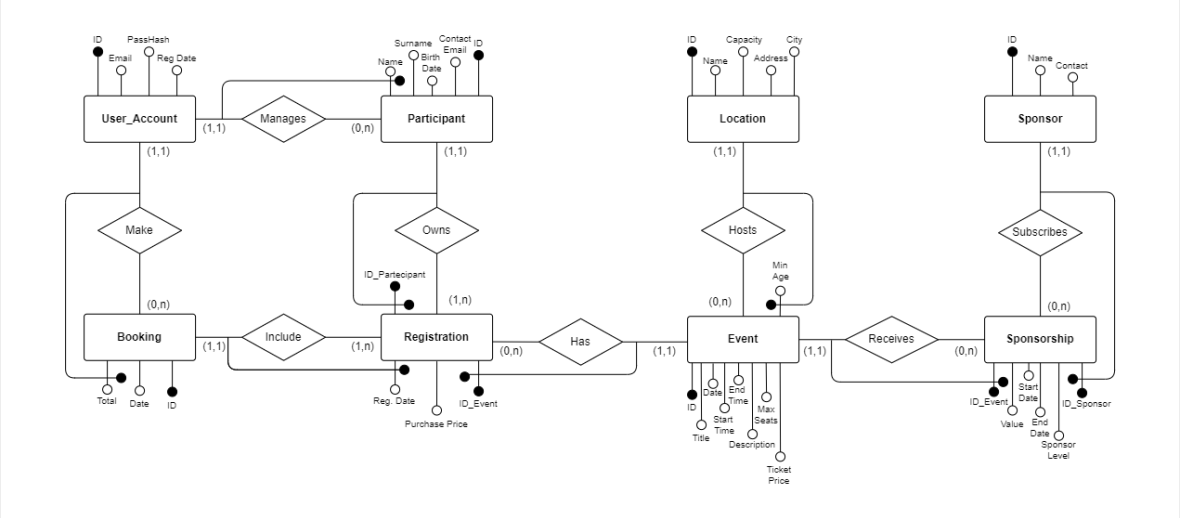

Figura 2.3: Schema E-R: Modellazione completa

Vincoli non esprimibili dallo schema E-R

Sebbene il diagramma E-R rappresenti efficacemente la struttura dei dati, esistono regole e vincoli di integrità che non possono essere espressi graficamente e che richiedono un'implementazione logica (Trigger o Check).

- **Vincolo di Capienza Dinamica:** La cardinalità massima della relazione tra *Partecipanti* ed *Evento* non è fissa (es. 1:1000), ma dipende dinamicamente dall'attributo `Max_Seats` della Location associata.
- **Coerenza Temporale (Contratti):** È imposto un vincolo di validità temporale sull'entità *Sponsorship*: la data di termine del contratto (`Contract_End_Date`) deve essere temporalmente successiva o uguale alla data di inizio (`Contract_Start_Date`). L'intervallo [`Start_Date`, `End_Date`] del contratto deve essere valido ed includere la data dell'evento.
- **Vincolo Età Calcolata:** L'associazione tra *Partecipante* ed *Evento* è permessa solo se:
$$(Data\_Odierna - Data\_Nascita) \geq Min\_Age$$
- **Validità Formale:** Gli attributi come Email devono rispettare un pattern specifico, e gli importi monetari (Prezzi, Contratti) non possono essere negativi.

Dizionario dei Dati - Entità

Descrizione dettagliata delle entità dello schema finale.

Entità	Descrizione	Identificatore
Evento	Evento in programma.	ID_Event
Location	Luogo fisico dell'evento.	ID_Location
User Account	Account di accesso al sistema.	ID_User
Partecipante	Persona fisica iscritta.	ID_Participant
Booking	Transazione di acquisto.	ID_Booking
Sponsor	Ente finanziatore.	ID_Sponsor
Sponsorship	Contratto di sponsorizzazione.	(ID_Sponsor, ID_Event)
Registration	Biglietto emesso (Iscrizione).	(ID_Part, ID_Event)

Tabella 2.1: Dizionario delle Entità

Dizionario dei Dati - Relazioni

Relazione	Entità Coinvolte	Card.
Ospita	Location (1) ↔ Evento (N)	1:N
Effettua	User (1) ↔ Booking (N)	1:N
Gestisce	User (1) ↔ Partecipante (N)	1:N
Include	Booking (1) ↔ Registration (N)	1:N
Riferisce	Registration (N) ↔ Evento (1)	N:1

Tabella 2.2: Dizionario delle Relazioni

Analisi Quantitativa (Volumi e Frequenze)

Stima del carico di lavoro previsto a regime per il dimensionamento del sistema.

Tavola dei Volumi

Concetto	Tipo	Volume Stimato
User Account	E	2.000
Partecipanti	E	5.000
Eventi	E	50
Registrazioni (Biglietti)	E	15.000
Booking (Ordini)	E	6.000

Tabella 2.3: Volumi stimati dopo 1 anno di attività

Tavola delle Operazioni e Frequenze

Op	Descrizione	Freq.	Tipo
OP1	**Inserimento Location** (Definizione nuova sede e capienza)	1/mese	I
OP2	**Inserimento Nuovo Evento** (con controllo vincolo *Unique*)	1/sett	I
OP3	**Modifica Dettagli Evento** (Es. cambio orario, descriz. o rinvio)	2/sett	I
OP4	**Registrazione Utente** (Creazione account con hash password)	10/gg	I
OP5	**Login Utente** (Verifica credenziali Email/Password)	100/gg	I
OP6	**Ricerca Eventi** (Query con filtri per Data, Luogo o Titolo)	200/gg	I
OP7	**Acquisto Singolo** (Transazione con creazione partecipante)	40/gg	I
OP8	**Acquisto di Gruppo** (Bulk Insert via **JSON**)	5/gg	I
OP9	**Cancellazione Account/Ordine** (Verifica vincoli *Cascade/Restrict*)	1/gg	I
OP10	**Verifica Capienza (Sold Out)** (Trigger automatico)	45/gg	Sys
OP11	**Verifica Età Minima** (Trigger automatico su data di nascita)	45/gg	Sys
OP12	**Stipula Contratto Sponsor** (Inserimento in *Sponsorship*)	2/mese	I
OP13	**Verifica Stato Sponsor** (Query su date contratto attive)	10/gg	I
OP14	**Visualizzazione Ticket** (Query su Vista `View_User_Tickets`)	500/gg	I
OP15	**Lista Partecipanti** (Query per l'organizzazione dell'evento)	5/sett	I
OP16	**Aggiornamento Prezzi** (Modifica listino eventi futuri)	1/mese	B
OP17	**Report Finanziario** (Totale incassi per evento/periodo)	1/mese	B

Tabella 2.4: Tavola estesa delle operazioni previste (I=Interattivo, B=Batch, Sys=System)

Analisi degli Accessi e Carico di Lavoro Per dimensionare correttamente il sistema, è stata effettuata un'analisi dettagliata degli accessi logici per ciascuna delle 17 operazioni previste. Si assume come unità di misura temporale il **giorno** e si definiscono:

- **L (Lettura):** Accesso in lettura a una tabella o indice.
- **S (Scrittura):** Operazione di inserimento, modifica o cancellazione.

Op	Freq/gg	Analisi Accessi (Unitari)	Carico Totale/gg
OP1 (Ins. Loc)	0.03	1L (Check Name) + 1S (Insert)	≈ 0
OP2 (Ins. Event)	0.14	2L (Check Loc/Time) + 1S (Insert)	≈ 1
OP3 (Mod. Event)	0.3	1L (Find ID) + 1S (Update)	≈ 1
OP4 (Reg. User)	10	1L (Check Email) + 1S (Insert)	10 L + 10 S
OP5 (Login)	**800**	1L (Select User by Email)	**800 L**
OP6 (Ricerca)	200	1L (Select Event) + 1L (Join Loc)	400 L
OP7 (Acq. Singolo)	40	2L (Event+User) + 2S (Book+Reg)	80 L + 80 S
OP8 (Acq. Json)	5	1L (Event) + 1S (Book) + 3S (Regs)	5 L + 20 S
OP9 (Cancel)	1	2L (Check Constraints) + 1S (Delete)	2 L + 1 S
OP10 (SoldOut)	45	1L (MaxSeats) + 1L (Count Reg)	90 L
OP11 (MinAge)	45	1L (Event Age) + 1L (User Dob)	90 L
OP12 (Sponsor)	0.1	2L (Check ID) + 1S (Insert)	≈ 0
OP13 (Check Spon)	10	1L (Select Status)	10 L
OP14 (View Tkt)	500	3L (Join Reg + Event + Part)	1.500 L
OP15 (List Part)	1	2L (Join Reg + User)	2 L
OP16 (Upd Price)	0.03	1S (Update)	≈ 0
OP17 (Report)	0.03	$N \times L$ (Full Scan - Batch)	(Batch)
TOTALE			$\approx$ **3.000 L + 115 S**

Tabella 2.5: Tavola degli Accessi: Dettaglio operazioni e stima del carico giornaliero

Conclusioni sull'Analisi: L'analisi quantitativa del carico di lavoro rivela un volume elevato di accessi in lettura (L) (oltre 3.000 accessi giornalieri, che riguardano le procedure di autenticazione e dalla consultazione dei titoli degli eventi) impone l'adozione di strategie di ottimizzazione come indici mirati e viste, le operazioni di scrittura, seppur quantitativamente inferiori (circa 115/giorno), riguardano le transazioni. Di conseguenza, l'architettura è stata progettata per massimizzare la velocità di risposta in lettura, garantendo comunqu una consistenza transazionale (ACID) per le cruciali operazioni di inserimento e modifica o eliminazione.

2.2 Analisi delle Ridondanze e Scelte Progettuali

Durante la fase di raffinamento, sono stati individuati attributi potenzialmente ridondanti. Per decidere se mantenere la ridondanza o derivare l'informazione, è stata effettuata un'analisi quantitativa basata sui volumi previsti e sul costo delle operazioni.

Attributo Derivato `Ticket_Code`

Ogni registrazione necessita di un codice univoco per la validazione.

- **Scenario A (Ridondanza):** Memorizzare il codice come `VARCHAR(32)`.
- **Scenario B (Calcolato):** Calcolare il codice a runtime (Viste).

Analisi dei Volumi

Considerando il volume annuo di registrazioni stimato (15.000 record), lo spazio aggiuntivo richiesto dalla ridondanza è calcolato nella Tabella 2.6.

Entità	Volume Annuo	Dimensione Attributo	Spazio Totale
Registration	15.000	33 Byte (VARCHAR 32)	$\approx$ 495 KB

Tabella 2.6: Stima dello spazio occupato dalla ridondanza del Ticket Code

Sebbene 495 KB siano trascurabili su disco moderno, il costo maggiore è nel mantenimento della consistenza (Analisi degli Accessi).

Analisi degli Accessi

Si confrontano le operazioni di Scrittura (generazione biglietto) e Lettura (controllo accessi) nei due scenari.

Operazione	Freq.	Tipo	Scenario A (Salvato)	Scenario B (Calcolato)
OP3: Acquisto (Insert)	45/gg	W	**2 Scritture** (Dati + Code)	**1 Scrittura**
OP14: Validazione (Read)	500/gg	R	1 Lettura Semplice	1 Lettura + Calcolo CPU
Aggiornamento	-	-	Trigger di update necessario	Nessuno

Tabella 2.7: Confronto del carico di lavoro tra scenario ridondante e calcolato

Decisione: Scenario B. Il tempo di calcolo esadecimale è trascurabile rispetto al vantaggio di non dover gestire allineamenti dati in fase di scrittura, garantendo che il codice sia sempre matematicamente coerente con gli ID.

Attributo `Status` in Sponsorship

Lo stato di un contratto (Attivo/Scaduto) dipende dalla data odierna.

Analisi	**Soluzione con Ridondanza**	**Soluzione Calcolata**
Aggiornamento	Batch periodico richiesto (Update)	Nessuno
Rischio	Dati non aggiornati se il batch fallisce	Dati sempre coerenti
Costo Query	Basso (Lettura diretta colonna)	Medio (Confronto date `NOW()`)

Tabella 2.8: Matrice decisionale per lo stato Sponsorship

Decisione: Soluzione Calcolata. La criticità di mostrare uno sponsor "Attivo" quando in realtà è scaduto (o viceversa) rende inaccettabile il rischio legato all'aggiornamento batch.

2.3 Ristrutturazione dello Schema E-R

Prima della traduzione in tabelle, sono state applicate le seguenti trasformazioni per normalizzare lo schema:

1. **Eliminazione delle relazioni N:M:**
   - La relazione *Iscrizione* è stata reificata nell'entità **Registration**, che eredita le chiavi primarie di Partecipante ed Evento.
   - La relazione *Sponsorizzazione* è stata reificata nell'entità **Sponsorship**.
2. **Gestione Identificatori:** Tutte le chiavi primarie "naturali" (es. Nome Location, Email Utente) sono state sostituite da chiavi artificiali numeriche (`AUTO_INCREMENT`) per efficienza nelle join, mantenendo vincoli `UNIQUE` sugli attributi naturali.
3. **Normalizzazione Indirizzi:** Per semplicità progettuale, gli indirizzi sono mantenuti come attributi della Location, ma separando la Città per eventuali future analisi geografiche (rispettando la 1NF).

2.4 Schema Logico Relazionale

Di seguito è riportato lo schema logico finale. Si adotta la seguente notazione grafica:

- Sottolineato: Chiave Primaria (PK).
- Tratteggiato: Chiave Esterna (FK).
- **Grassetto Sottolineato**: Chiave parte di una PK Composta (e FK).

- **Location**(ID_Location, Name, Capacity, Address, City)
- **Event**(ID_Event, Title, Event_Date, Start_Time, End_Time, Description, Max_Seats, Ticket_Price, Min_Age, ID_Location)
- **User_Account**(ID_User, Login_Email, Password_Hash, Registration_Date)
- **Participant**(ID_Participant, Name, Surname, Birth_Date, Contact_Email, ID_User)
- **Booking**(ID_Booking, Booking_Date, Total_Amount, ID_User)
- **Registration**(**ID_Participant**, **ID_Event**, ID_Booking, Registration_Date, Purchase_Price)
- **Sponsor**(ID_Sponsor, Sponsor_Name, Contact)
- **Sponsorship**(**ID_Sponsor**, **ID_Event**, Contract_Value, Start_Date, End_Date, Level)

2.5 Schema Logico Relazionale

Di seguito si riporta lo schema logico finale in notazione "Crow's Foot".

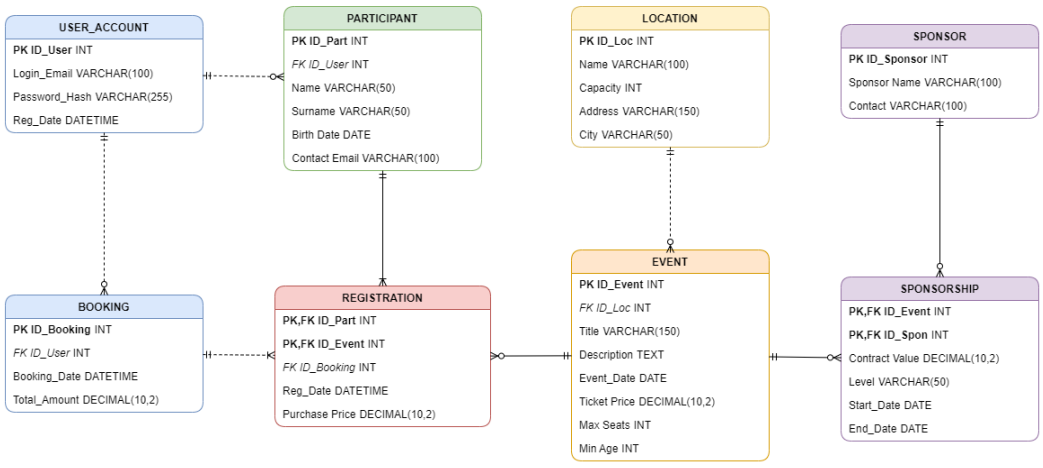

Figura 2.4: Schema Logico Relazionale: Diagramma delle tabelle e delle relazioni

Le chiavi primarie (PK) sono evidenziate in grassetto/sottolineate all'interno dello schema, mentre le relazioni (FK) sono rappresentate dalle linee di connessione.

Capitolo 3

Progettazione Fisica

In questa fase, lo schema logico è stato tradotto nel linguaggio di definizione dei dati (DDL) specifico per il DBMS scelto (MySQL/MariaDB). Sono state definite le strutture fisiche delle tabelle, i domini degli attributi e la logica attiva (Trigger) per garantire i vincoli di integrità.

3.1 Creazione dello Schema e delle Tabelle

Di seguito viene riportato il codice SQL per la creazione delle tabelle principali.

Tabelle Location e Evento

Le tabelle indipendenti sono state create per prime per soddisfare le dipendenze delle chiavi esterne.

```sql
-- Tabella Location
CREATE TABLE Location (
    ID_Location INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(100) NOT NULL UNIQUE,
    Capacity INT UNSIGNED NOT NULL,
    Address VARCHAR(150) NOT NULL,
    City VARCHAR(50) NOT NULL
) ENGINE=InnoDB;

-- Tabella Evento
CREATE TABLE Event (
    ID_Event INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    Title VARCHAR(150) NOT NULL,
    Event_Date DATE NOT NULL,
    Event_Start_Time TIME NOT NULL,
    Event_End_Time TIME NOT NULL,
    Description TEXT NOT NULL,
    Max_Seats INT UNSIGNED NOT NULL,
    Ticket_Price NUMERIC(10,2) NOT NULL DEFAULT 0.00,
    Min_Age INT UNSIGNED NOT NULL DEFAULT 0,
    ID_Location INT UNSIGNED NOT NULL,

    UNIQUE (Title, Event_Date, Event_Start_Time, ID_Location),
    UNIQUE (ID_Location, Event_Date, Event_Start_Time),

```

```
    CHECK (Ticket_Price >= 0),
    FOREIGN KEY (ID_Location) REFERENCES Location(ID_Location)
        ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE=InnoDB;
```

Listing 3.1: Creazione Tabelle Location ed Evento

Tabelle Utenti, Partecipanti e Ordini

Implementazione della struttura per la gestione degli acquisti e dei partecipanti.

```
-- Tabella Account Utente
CREATE TABLE User_Account (
    ID_User INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    Login_Email VARCHAR(100) NOT NULL UNIQUE,
    Password_Hash VARCHAR(255) NOT NULL,
    Registration_Date DATETIME DEFAULT CURRENT_TIMESTAMP,

    CHECK (Login_Email LIKE '%@%')
) ENGINE=InnoDB;

-- Tabella Partecipante
CREATE TABLE Participant (
    ID_Participant INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(50) NOT NULL,
    Surname VARCHAR(50) NOT NULL,
    Birth_Date DATE NOT NULL,
    Contact_Email VARCHAR(100) NOT NULL,
    ID_User INT UNSIGNED NOT NULL,

    CHECK (Contact_Email LIKE '%@%'),
    FOREIGN KEY (ID_User) REFERENCES User_Account(ID_User)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=InnoDB;

-- Tabella Booking
CREATE TABLE Booking (
    ID_Booking INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    Booking_Date DATETIME DEFAULT CURRENT_TIMESTAMP,
    Total_Amount NUMERIC(10,2) NOT NULL DEFAULT 0.00,
    ID_User INT UNSIGNED NOT NULL,

    FOREIGN KEY (ID_User) REFERENCES User_Account(ID_User)
        ON DELETE CASCADE
        ON UPDATE CASCADE
) ENGINE=InnoDB;
```

Listing 3.2: Creazione Tabelle User

Tabella Sponsor

Implementazione della struttura per la gestione degli sponsor.

```
-- Tabella Sponsor
```

```sql
CREATE TABLE Sponsor (
    ID_Sponsor INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    Sponsor_Name VARCHAR(100) NOT NULL UNIQUE,
    Contact VARCHAR(100) NOT NULL
) ENGINE=InnoDB;
```

Listing 3.3: Tabella Registration (Biglietti)

3.2 Implementazione delle Relazioni

Le relazioni molti a molti identificate nell'analisi concettuale sono state implementate come tabelle associative con attributi specifici.

```sql
-- Tabella Iscrizione
CREATE TABLE Registration (
    ID_Participant INT UNSIGNED NOT NULL,
    ID_Event INT UNSIGNED NOT NULL,
    ID_Booking INT UNSIGNED NOT NULL,

    Registration_Date DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    Purchase_Price NUMERIC(10,2) NOT NULL DEFAULT 0.00,
    -- PK composta cosi un partecipante non può iscriversi 2 volte
    PRIMARY KEY (ID_Participant, ID_Event),

    FOREIGN KEY (ID_Participant) REFERENCES Participant(ID_Participant)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    FOREIGN KEY (ID_Event) REFERENCES Event(ID_Event)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    FOREIGN KEY (ID_Booking) REFERENCES Booking(ID_Booking)
        ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE=InnoDB;
```

Listing 3.4: Tabella Registration (Biglietti)

```sql
-- Tabella Sponsorizzazione
CREATE TABLE Sponsorship (
    ID_Sponsor INT UNSIGNED NOT NULL,
    ID_Event INT UNSIGNED NOT NULL,
    Contract_Value NUMERIC(10,2) NOT NULL DEFAULT 0.00,
    Contract_Start_Date DATE NOT NULL,
    Contract_End_Date DATE NOT NULL,
    Sponsor_Level VARCHAR(50),
    Status ENUM('Active', 'Cancelled', 'Expired') NOT NULL DEFAULT 'Active',
    PRIMARY KEY (ID_Sponsor, ID_Event),

    FOREIGN KEY (ID_Sponsor) REFERENCES Sponsor(ID_Sponsor)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    FOREIGN KEY (ID_Event) REFERENCES Event(ID_Event)
        ON DELETE RESTRICT ON UPDATE CASCADE,

    CHECK (Contract_End_Date >= Contract_Start_Date)
) ENGINE=InnoDB;
```

Listing 3.5: Tabella Sponsorship (Contratti)

3.3 Implementazione della Logica Attiva (Triggers)

Per garantire i vincoli di business definiti nel Cap. 1.7, sono stati implementati dei Trigger che intervengono prima dell'inserimento (`BEFORE INSERT`) per validare o bloccare l'operazione.

Controllo Capienza e Sold Out

Questi trigger impediscono l'overbooking bloccando l'inserimento se le registrazioni superano i posti disponibili, sono stati anche gestiti eventuali aggiornamenti della capienza.

```
DELIMITER //
-- Controllo Sold Out (Max_Seats dell'evento)
CREATE TRIGGER Check_Max_Seats
BEFORE INSERT ON Registration
FOR EACH ROW
BEGIN
    DECLARE current_count INT;
    DECLARE max_capacity INT;
    SELECT COUNT(*) INTO current_count FROM Registration WHERE
    ID_Event = NEW.ID_Event;
    SELECT Max_Seats INTO max_capacity FROM Event WHERE ID_Event = NEW
    .ID_Event;
    IF current_count >= max_capacity THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Evento SOLD OUT.';
    END IF;
END //

-- Controllo posti in Location, INSERT
CREATE TRIGGER Check_Event_Location_Limit
BEFORE INSERT ON Event
FOR EACH ROW
BEGIN
    DECLARE loc_cap INT;
    -- controllo in base alla capieza della location scelta per l'
    evento
    SELECT Capacity INTO loc_cap FROM Location WHERE ID_Location = NEW
    .ID_Location;
    IF NEW.Max_Seats > loc_cap THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Max_Seats supera
    la capienza fisica della Location!';
    END IF;
END //

-- Controllo posti in Location, UPDATE
CREATE TRIGGER Check_Event_Location_Limit_Update
BEFORE UPDATE ON Event
FOR EACH ROW
BEGIN
    DECLARE loc_cap INT;
    -- controllo in base alla capieza della location scelta per l'
    aggiornamento dell'evento
    SELECT Capacity INTO loc_cap FROM Location WHERE ID_Location = NEW
    .ID_Location;

```

```
    IF NEW.Max_Seats > loc_cap THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'ERRORE UPDATE: Max_Seats supera la capienza della Location!';
    END IF;
END //

-- Controllo (inverso) capienza della Location, se ci sono già eventi programmati che richiedono più posti.
CREATE TRIGGER Check_Location_Capacity_Update
BEFORE UPDATE ON Location
FOR EACH ROW
BEGIN
    DECLARE conflict_count INT;
    IF NEW.Capacity < OLD.Capacity THEN
        SELECT COUNT(*) INTO conflict_count
        FROM Event
        WHERE ID_Location = NEW.ID_Location
          AND Max_Seats > NEW.Capacity
          AND Event_Date >= CURDATE(); -- per ignorare eventi passati
        IF conflict_count > 0 THEN
            SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Impossibile ridurre la capienza. Ci sono eventi associati che richiedono più posti!';
        END IF;
    END IF;
END //
```

Controllo Età Minima

Verifica che l'età del partecipante sia compatibile con i requisiti dell'evento.

```
DELIMITER //
CREATE TRIGGER Check_Participant_Age
BEFORE INSERT ON Registration
FOR EACH ROW
BEGIN
    DECLARE required_age INT;
    DECLARE participant_dob DATE;
    DECLARE current_age INT;

    SELECT Min_Age INTO required_age FROM Event WHERE ID_Event = NEW.ID_Event;
    SELECT Birth_Date INTO participant_dob FROM Participant
    WHERE ID_Participant = NEW.ID_Participant;

    SET current_age = TIMESTAMPDIFF(YEAR, participant_dob, CURDATE());

    IF current_age < required_age THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'ERRORE: Età insufficiente per questo evento.';
    END IF;
END //
DELIMITER ;
```

Controllo data di nascita

I seguenti trigger si occupano del controllo della data di nascita, e del controllo dell'aggiornamento del partecipante nel caso abbia iscrizioni con un'età maggiore rispetto a quella modificata.

```
DELIMITER //

-- controllo data di nascita
CREATE TRIGGER Check_Participant_DOB
BEFORE INSERT ON Participant
FOR EACH ROW
BEGIN
    IF NEW.Birth_Date > CURDATE() THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'La data di nascita non può essere nel futuro!';
    END IF;
END //

-- controllo aggiornamento data di nascita
CREATE TRIGGER Check_Participant_Update_Logic
BEFORE UPDATE ON Participant
FOR EACH ROW
BEGIN
    DECLARE invalid_registrations INT;
    DECLARE actual_age INT;
    IF NEW.Birth_Date > CURDATE() THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La data di nascita non può essere nel futuro!';
    END IF;

    SET actual_age = TIMESTAMPDIFF(YEAR, NEW.Birth_Date, CURDATE());

    SELECT COUNT(*) INTO invalid_registrations
    FROM Registration R
    JOIN Event E ON R.ID_Event = E.ID_Event
    WHERE R.ID_Participant = NEW.ID_Participant
      AND E.Min_Age > actual_age;

    IF invalid_registrations > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Impossibile cambiare data di nascita. L\'utente è iscritto a eventi che richiedono un\'età maggiore!';
    END IF;

END //
```

Gestione del Prezzo

Questi trigger si occupano della gestione dei costi dei biglietti e dell'intero ordine.

```
DELIMITER //

-- Salva Prezzo Automatico in Registration
CREATE TRIGGER Set_Registration_Price
BEFORE INSERT ON Registration
FOR EACH ROW
BEGIN
    DECLARE current_price NUMERIC(10,2);
    SELECT Ticket_Price INTO current_price FROM Event WHERE ID_Event =
    NEW.ID_Event;
    SET NEW.Purchase_Price = current_price;
END //

-- Aggiornamento Totale Ordine in Booking
CREATE TRIGGER Update_Booking_Total_Insert
AFTER INSERT ON Registration
FOR EACH ROW
BEGIN
    UPDATE Booking
    SET Total_Amount = Total_Amount + NEW.Purchase_Price
    WHERE ID_Booking = NEW.ID_Booking;
END //

-- aggiornamento Booking
CREATE TRIGGER Update_Booking_Total_Delete
AFTER DELETE ON Registration
FOR EACH ROW
BEGIN
    UPDATE Booking
    SET Total_Amount = Total_Amount - OLD.Purchase_Price
    WHERE ID_Booking = OLD.ID_Booking;
END //
```

Gestione Data Sponsorship

Attraverso questi trigger vengono gestite le date di scadenza dei contratti tra gli sponsor e gli eventi, controllando che non scadano prima della data dell'evento.

```
DELIMITER //

-- Controllo Date Sponsor INSERT
CREATE TRIGGER Check_Sponsorship_Dates
BEFORE INSERT ON Sponsorship
FOR EACH ROW
BEGIN
    DECLARE event_day DATE;
    SELECT Event_Date INTO event_day FROM Event WHERE ID_Event = NEW.
   ID_Event;
    IF NEW.Contract_End_Date < event_day THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Contratto scade
   prima dell\evento';
    END IF;
END //
```

```
14
15 -- Controllo Date Sponsor UPDATE
16 CREATE TRIGGER Check_Sponsorship_Dates_Update
17 BEFORE UPDATE ON Sponsorship
18 FOR EACH ROW
19 BEGIN
20     DECLARE event_day DATE;
21     SELECT Event_Date INTO event_day FROM Event WHERE ID_Event = NEW.
       ID_Event;
22     IF NEW.Contract_End_Date < event_day THEN
23         SIGNAL SQLSTATE '45000'
24         SET MESSAGE_TEXT = 'Contratto scade prima dell\evento';
25     END IF;
26 END //
```

3.4 Viste (Views)

Per semplificare l'accesso ai dati aggregati e nascondere la complessità delle join, sono state definite le seguenti viste.

View_User_Tickets

Questa vista risolve la ridondanza del `Ticket_Code` discusso nel Cap. 2.2, calcolandolo dinamicamente solo quando richiesto.

```
1  CREATE OR REPLACE VIEW View_User_Tickets AS
2  SELECT
3      P.ID_User,
4      R.ID_Event,
5      R.ID_Participant,
6      R.ID_Booking,
7      E.Title AS Event_Name,
8      L.Name AS Location_Name,
9      L.Address,
10     L.City,
11     E.Event_Date,
12     E.Event_Start_Time,
13     P.Name,
14     P.Surname,
15     CONCAT(HEX(R.ID_Event), '-', HEX(R.ID_Participant), '-', HEX(
       UNIX_TIMESTAMP(R.Registration_Date))) AS Ticket_Code,
16     R.Purchase_Price,
17     R.Registration_Date
18 FROM Registration R
19 JOIN Event E ON R.ID_Event = E.ID_Event
20 JOIN Location L ON E.ID_Location = L.ID_Location
21 JOIN Participant P ON R.ID_Participant = P.ID_Participant;
```

View_Sponsorship_Status

Questa vista risolve la ridondanza dello `Status` discusso nel Cap. 2.2, calcolandolo dinamicamente attraverso la data di scadenza, solo quando richiesto.

```
-- visualizzazione di tutti i contratti
CREATE OR REPLACE VIEW View_Sponsorship_Status AS
SELECT
    S.ID_Sponsor,
    Sp.Sponsor_Name,
    S.ID_Event,
    E.Title AS Event_Title,
    S.Contract_Start_Date,
    S.Contract_End_Date,
    S.Contract_Value,
    S.Sponsor_Level,
    CASE
        WHEN S.Contract_End_Date < CURDATE() THEN 'Expired'
        WHEN S.Contract_End_Date = CURDATE() THEN 'Expiring today'
        WHEN S.Contract_Start_Date > CURDATE() THEN 'Scheduled'
        ELSE 'Active'
    END AS Current_Status
FROM Sponsorship S
JOIN Sponsor Sp ON S.ID_Sponsor = Sp.ID_Sponsor
JOIN Event E ON S.ID_Event = E.ID_Event;
```

View_Event_Stats

Questa vista è utile per la visualizzazione delle statistiche che riguardano tutti gli eventi, attraverso l'uso delle condizioni possono essere mirate a eventi specifici.

```
-- STATISTICHE eventi utilizzabile con le condizioni
CREATE OR REPLACE VIEW View_Event_Stats AS
SELECT
    E.ID_Event,
    E.Title,
    E.Event_Date,
    E.Max_Seats,
    COUNT(R.ID_Participant) AS Tickets_Sold,
    (E.Max_Seats - COUNT(R.ID_Participant)) AS Seats_Remaining,
    COALESCE(SUM(R.Purchase_Price), 0) AS Total_Revenue,
    CONCAT(ROUND((COUNT(R.ID_Participant) / E.Max_Seats * 100), 1), '%
   ') AS Occupancy_Rate
FROM Event E
LEFT JOIN Registration R ON E.ID_Event = R.ID_Event
GROUP BY E.ID_Event, E.Title, E.Event_Date, E.Max_Seats;
```

Capitolo 4

Implementazione delle Operazioni e Logica Applicativa

In questo capitolo vengono descritte le procedure memorizzate (*Stored Procedures*) sviluppate per gestire la logica applicativa direttamente a livello di database.

4.1 Strategia Transazionale (ACID)

Le operazioni critiche, come l'acquisto di biglietti, coinvolgono scritture su più tabelle (creazione *Booking* $\rightarrow$ inserimento *Registration*). Per garantire l'integrità dei dati, queste operazioni sono racchiuse in blocchi transazionali:

- **Atomicità:** Se un solo inserimento fallisce (es. per Sold Out o errore età), l'intera transazione viene annullata (`ROLLBACK`), evitando ordini orfani o pagamenti senza biglietto.
- **Isolamento:** Le transazioni gestiscono la concorrenza per evitare che l'ultimo biglietto venga venduto contemporaneamente a due utenti.

4.2 Procedure di Iscrizione

Iscrizione Singola

Questa procedura gestisce l'iscrizione di un singolo partecipante. Include una logica "intelligente" che verifica preliminarmente tutti i vincoli (Capienza, Età, Esistenza Utente) prima di tentare la scrittura, restituendo messaggi di errore parlanti.

```
1
2 -- Atomicità Ordine + (Partecipante + Iscrizione)
3 CREATE PROCEDURE Transaction_Register_Single_New (IN p_EventID INT, IN
      p_UserID INT, IN p_Name VARCHAR(50), IN p_Surname VARCHAR(50), IN
     p_Birth_Date DATE, IN p_ContactEmail VARCHAR(100))
4 proc_label:BEGIN
5     DECLARE v_BookingID INT;
6     DECLARE v_ParticipantID INT DEFAULT NULL;
7
8     DECLARE EXIT HANDLER FOR SQLEXCEPTION
9     BEGIN
```

```
10          ROLLBACK;
11          RESIGNAL;
12      END;
13
14      START TRANSACTION;
15      -- cerco se esite l'identico partecipante
16      SELECT ID_Participant INTO v_ParticipantID
17      FROM Participant
18      WHERE ID_User = p_UserID
19        AND Name = p_Name
20        AND Surname = p_Surname
21        AND Birth_Date = p_Birth_Date;
22
23      INSERT INTO Booking (ID_User) VALUES (p_UserID);
24      SET v_BookingID = LAST_INSERT_ID();
25
26      -- controllo se esite
27      IF v_ParticipantID IS NOT NULL THEN
28          INSERT INTO Registration (ID_Participant, ID_Event, ID_Booking
    )
29          VALUES (v_ParticipantID, p_EventID, v_BookingID);
30          COMMIT;
31          SELECT CONCAT('Partecipante già presente iscrizione completata
    ') AS Esito;
32      ELSE
33          INSERT INTO Participant (Name, Surname, Birth_Date,
    Contact_Email, ID_User)
34          VALUES (p_Name, p_Surname, p_Birth_Date, p_ContactEmail,
    p_UserID);
35          SET v_ParticipantID = LAST_INSERT_ID();
36          INSERT INTO Registration (ID_Participant, ID_Event, ID_Booking
    )
37          VALUES (v_ParticipantID, p_EventID, v_BookingID);
38          COMMIT;
39          SELECT CONCAT('nuovo Partecipante creato, iscrizione
    completata.') AS Esito;
40      END IF;
41
42  END //
43
44  -- Atomicità Boking + Iscrizione per Partecipante esistente
45  CREATE PROCEDURE Transaction_Register_Existing (IN p_EventID INT, IN
    p_UserID INT, IN p_ParticipantID INT)
46  proc_label: BEGIN
47      DECLARE v_BookingID INT;
48      DECLARE v_OwnerID INT;
49      DECLARE EXIT HANDLER FOR SQLEXCEPTION
50      BEGIN
51          ROLLBACK;
52          RESIGNAL;
53      END;
54
55      START TRANSACTION;
56      SELECT ID_User INTO v_OwnerID FROM Participant WHERE
    ID_Participant = p_ParticipantID;
57      IF v_OwnerID != p_UserID THEN
58          ROLLBACK;
```

```
59          -- SELECT 'partecipante non collegato allo stesso account' AS
    Status; -- non potrebbe accadere
60          LEAVE proc_label;
61      END IF;
62
63      INSERT INTO Booking (ID_User) VALUES (p_UserID);
64      SET v_BookingID = LAST_INSERT_ID();
65      INSERT INTO Registration (ID_Participant, ID_Event, ID_Booking)
66      VALUES (p_ParticipantID, p_EventID, v_BookingID);
67      COMMIT;
68      SELECT 'Iscrizione effettuata.' AS Status;
69  END //
```

Listing 4.1: Procedura Register_Single_New con Transaction Control

Iscrizione di Gruppo con JSON

Per ottimizzare le performance nelle iscrizioni multiple (es. famiglie o scolaresche), è stata implementata una procedura in grado di ricevere una struttura JSON contenente la lista dei partecipanti. Questa tecnica riduce il numero di chiamate client-server (round-trip) da N a 1.

```
1
2   -- Atomicità Iscrizione Multipla
3   CREATE PROCEDURE Transaction_Register_Group_JSON(IN p_EventID INT, IN
        p_UserID INT, IN p_JSON_Data JSON)
4   proc_label: BEGIN
5       DECLARE i INT DEFAULT 0;
6       DECLARE prenotation_count INT;
7       DECLARE v_BookingID INT;
8       DECLARE v_Name VARCHAR(50);
9       DECLARE v_Surname VARCHAR(50);
10      DECLARE v_Birth_Date DATE;
11      DECLARE v_Email VARCHAR(100);
12      DECLARE new_participant_id INT;
13      DECLARE existing_participant_id INT;
14      DECLARE EXIT HANDLER FOR SQLEXCEPTION
15      BEGIN
16          ROLLBACK;
17          SELECT 'Transazione fallita. Rollback totale.' AS Status;
18      END;
19
20      START TRANSACTION;
21      INSERT INTO Booking (ID_User) VALUES (p_UserID);
22      SET v_BookingID = LAST_INSERT_ID();
23      SET prenotation_count = JSON_LENGTH(p_JSON_Data);
24
25      WHILE i < prenotation_count DO
26          SET v_Name = JSON_UNQUOTE(JSON_EXTRACT(p_JSON_Data, CONCAT('$[
        ', i, '].name')));
27          SET v_Surname = JSON_UNQUOTE(JSON_EXTRACT(p_JSON_Data, CONCAT(
        '$[', i, '].surname')));
28          SET v_Birth_Date = JSON_UNQUOTE(JSON_EXTRACT(p_JSON_Data,
        CONCAT('$[', i, '].dob')));
29          SET v_Email = JSON_UNQUOTE(JSON_EXTRACT(p_JSON_Data, CONCAT('$
        [', i, '].email')));
```

```
30          SET existing_participant_id = NULL;
31
32          -- cerco se esiste già
33          SELECT ID_Participant INTO existing_participant_id
34          FROM Participant
35          WHERE ID_User = p_UserID
36            AND Name = v_Name
37            AND Surname = v_Surname
38            AND Birth_Date = v_Birth_Date;
39
40          IF existing_participant_id IS NOT NULL THEN
41              SET new_participant_id = existing_participant_id;
42          ELSE
43              -- se non esiste
44              INSERT INTO Participant (Name, Surname, Birth_Date,
    Contact_Email, ID_User)
45              VALUES (v_Name, v_Surname, v_Birth_Date, v_Email, p_UserID
    );
46              SET new_participant_id = LAST_INSERT_ID();
47          END IF;
48          INSERT INTO Registration (ID_Participant, ID_Event, ID_Booking
    )
49          VALUES (new_participant_id, p_EventID, v_BookingID);
50          SET i = i + 1;
51      END WHILE;
52      COMMIT;
53      SELECT CONCAT('Gruppo iscritto, Booking ID: ', v_BookingID) AS
    Status;
54  END //
55
56  -- Atomicità Iscrizione Multipla per Partecipanti esistente
57
58  CREATE PROCEDURE Transaction_Register_Existing_Group_JSON(
59      IN p_EventID INT,
60      IN p_UserID INT,
61      IN p_JSON_ParticipantIDs JSON
62  )
63  proc_label: BEGIN
64      DECLARE i INT DEFAULT 0;
65      DECLARE count_ids INT;
66      DECLARE v_BookingID INT;
67      DECLARE v_ParticipantID INT;
68      DECLARE v_OwnerID INT;
69      DECLARE EXIT HANDLER FOR SQLEXCEPTION
70      BEGIN
71          ROLLBACK;
72          SELECT 'Rollback totale.' AS Status;
73      END;
74
75      START TRANSACTION;
76
77      INSERT INTO Booking (ID_User) VALUES (p_UserID);
78      SET v_BookingID = LAST_INSERT_ID();
79      SET count_ids = JSON_LENGTH(p_JSON_ParticipantIDs);
80      WHILE i < count_ids DO
81          SET v_ParticipantID = JSON_EXTRACT(p_JSON_ParticipantIDs,
    CONCAT('$[', i, ']'));
```

```
82
83          -- sontrollo che questo ID esiste, associato allo stesso
    utente
84          SELECT ID_User INTO v_OwnerID FROM Participant WHERE
    ID_Participant = v_ParticipantID;
85
86          IF v_OwnerID IS NULL OR v_OwnerID != p_UserID THEN
87              ROLLBACK;
88              -- SELECT CONCAT('Il partecipante ID ', v_ParticipantID, '
    non eiste') AS Status; -- non può succedere
89              LEAVE proc_label;
90          END IF;
91          INSERT INTO Registration (ID_Participant, ID_Event, ID_Booking
    )
92          VALUES (v_ParticipantID, p_EventID, v_BookingID);
93          SET i = i + 1;
94      END WHILE;
95      COMMIT;
96      SELECT CONCAT('Gruppo gis esistente iscritto. Booking ID: ',
    v_BookingID) AS Status;
97  END //
```

Listing 4.2: Gestione Insert con JSON

Gestione delle Iscrizioni

Oltre all'inserimento, il sistema deve gestire le modifiche post-vendita garantendo la consistenza contabile. Sono state implementate due procedure transazionali per gestire il cambio data e la cancellazione con pulizia automatica.

Cambio Data Evento (Swap)

La procedura `Transaction_Swap_Event_Date` permette a un partecipante di spostare la propria iscrizione da un evento a un altro. Per garantire la parità economica senza gestire rimborsi complessi in questa fase, la transazione applica una logica rigida:

- Verifica che il titolo dell'evento sia identico (stesso tipo di concerto/spettacolo).
- Verifica che il prezzo del biglietto sia invariato.
- Esegue atomicamente la creazione del nuovo biglietto e l'eliminazione del precedente.

In caso di discrepanza (es. prezzo diverso), la transazione esegue il *Rollback* per evitare perdite economiche.

```
1  CREATE PROCEDURE Transaction_Swap_Event_Date (IN p_ParticipantID INT,
       IN p_OldEventID INT, IN p_NewEventID INT)
2  proc_label: BEGIN
3      DECLARE v_OldTitle VARCHAR(150);
4      DECLARE v_NewTitle VARCHAR(150);
5      DECLARE v_OldPrice NUMERIC(10,2);
6      DECLARE v_NewPrice NUMERIC(10,2);
7
8      DECLARE v_UserID INT;
```

```
9       DECLARE v_NewBookingID INT;
10
11      DECLARE EXIT HANDLER FOR SQLEXCEPTION
12      BEGIN
13          ROLLBACK;
14          SELECT 'Transazione annullata.' AS StatusSwapEventDate;
15      END;
16
17      START TRANSACTION;
18
19      -- vecchio evento
20      SELECT Title, Ticket_Price INTO v_OldTitle, v_OldPrice FROM Event
        WHERE ID_Event = p_OldEventID;
21      -- nuovo evento
22      SELECT Title, Ticket_Price INTO v_NewTitle, v_NewPrice FROM Event
        WHERE ID_Event = p_NewEventID;
23
24      IF v_OldTitle != v_NewTitle THEN
25          ROLLBACK;
26          SELECT 'Evento diverso. (Titolo)' AS StatusSwapEventDate;
27          LEAVE proc_label;
28      END IF;
29
30      IF v_OldPrice != v_NewPrice THEN
31          ROLLBACK;
32          SELECT 'Prezzo diverso.' AS StatusSwapEventDate;
33          LEAVE proc_label;
34      END IF;
35
36      -- creazione ordine+iscrizione
37      SELECT ID_User INTO v_UserID FROM Participant WHERE ID_Participant
        = p_ParticipantID;
38
39      INSERT INTO Booking (ID_User) VALUES (v_UserID);
40      SET v_NewBookingID = LAST_INSERT_ID();
41
42      INSERT INTO Registration (ID_Participant, ID_Event, ID_Booking)
        VALUES (p_ParticipantID, p_NewEventID, v_NewBookingID);
43
44      -- cancella ordine vecchio
45      DELETE FROM Registration
46      WHERE ID_Participant = p_ParticipantID AND ID_Event = p_OldEventID
        ;
47
48      COMMIT;
49      SELECT 'SUCCESSO: Cambio data effettuato.' AS StatusSwapEventDate;
50
51  END //
```

Cancellazione con Rimozione Orfani (Unregister)

La procedura `Transaction_Unregister` gestisce la rinuncia al biglietto. Una particolarità di questa implementazione è la gestione della pulizia dei dati (*Garbage Collection*):

1. Elimina la registrazione specifica (`DELETE FROM Registration`).
2. Verifica se il `Participant` ha altre iscrizioni attive.
3. Se il partecipante non ha più biglietti associati, viene eliminato fisicamente dal database per evitare di conservare anagrafiche "orfane" inutilizzate.

```
-- Atomictà per eliminazione Register+Partecipant
CREATE PROCEDURE Transaction_Unregister(IN p_ParticipantID INT, IN p_EventID INT)
BEGIN
    DECLARE v_RemainingTickets INT;
    DECLARE v_Exists INT;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SELECT 'Errore durante la cancellazione. Rollback effettuato.' AS Status;
    END;

    START TRANSACTION;

    SELECT COUNT(*) INTO v_Exists
    FROM Registration
    WHERE ID_Participant = p_ParticipantID AND ID_Event = p_EventID;
    -- se esiste l'evento da da eliminare
    IF v_Exists > 0 THEN
        DELETE FROM Registration
        WHERE ID_Participant = p_ParticipantID AND ID_Event = p_EventID;
        -- se ci sono altre registrazioni lo stesso partecipante
        SELECT COUNT(*) INTO v_RemainingTickets
        FROM Registration
        WHERE ID_Participant = p_ParticipantID;

       -- cntrollo se il biglietto esistete
        IF v_RemainingTickets = 0 THEN
            DELETE FROM Participant WHERE ID_Participant = p_ParticipantID;
            SELECT 'Biglietto cancellato. Partecipante rimosso.' AS Status;
        ELSE
            SELECT 'Biglietto cancellato. Il partecipante ha ancora altri eventi.' AS Status;
        END IF;

    ELSE
        SELECT 'Biglietto non trovato.' AS Status;
    END IF;

    COMMIT;
END //
```

Gestione delle Sponsorizzazioni

Per la parte amministrativa, è stato sviluppato un set completo di procedure per gestire il contratto di sponsorizzazione (`Sponsorship`). A differenza delle semplici operazioni CRUD, queste transazioni implementano regole per la validità temporale.

- **Inserimento (`Transaction_Add_Sponsorship`):** Prima di inserire un nuovo contratto, verifica che non esista già una sponsorizzazione attiva per la stessa coppia Sponsor-Evento, prevenendo duplicati.

```
CREATE PROCEDURE Transaction_Add_Sponsorship(IN p_SponsorID INT,
    IN p_EventID INT,IN p_Value NUMERIC(10,2),IN p_StartDate DATE,
     IN p_EndDate DATE, IN p_SponsorLevel VARCHAR(50))
proc_label: BEGIN
    DECLARE v_Exists INT;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        RESIGNAL;
    END;

    START TRANSACTION;
    -- se esiste contratto evento-sponsor
    SELECT COUNT(*) INTO v_Exists
    FROM Sponsorship
    WHERE ID_Sponsor = p_SponsorID AND ID_Event = p_EventID;
    IF v_Exists > 0 THEN
        ROLLBACK;
        SELECT 'Lo Sponsor ha già un contratto attivo per questo
    evento' AS Status;
        LEAVE proc_label;
    END IF;
    INSERT INTO Sponsorship (ID_Sponsor, ID_Event, Contract_Value
    , Contract_Start_Date, Contract_End_Date, Sponsor_Level)
    VALUES (p_SponsorID, p_EventID, p_Value, p_StartDate,
    p_EndDate, p_SponsorLevel);
    COMMIT;
    SELECT 'Sponsorship registrata' AS Status;
END //


```

- **Aggiornamento (`Transaction_Update_Sponsorship`):** Permette di modificare i termini contrattuali (valore, date, livello) solo se esiste un contratto attivo, garantendo l'integrità referenziale applicativa.

```
CREATE PROCEDURE Transaction_Update_Sponsorship(IN p_SponsorID
    INT, IN p_EventID INT, IN p_NewValue NUMERIC(10,2), IN
    p_NewStartDate DATE, IN p_NewEndDate DATE, IN p_NewLevel
    VARCHAR(50))
BEGIN
    DECLARE v_Exists INT;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        RESIGNAL;
```

```
    END;
    START TRANSACTION;
    -- se esiste contratto evento-sponsor
    SELECT COUNT(*) INTO v_Exists
    FROM Sponsorship
    WHERE ID_Sponsor = p_SponsorID AND ID_Event = p_EventID;
    IF v_Exists = 0 THEN
        ROLLBACK;
        SELECT 'Lo Sponsor non ha un contratto attivo per questo
   evento' AS Status;
    ELSE
        UPDATE Sponsorship
        SET Contract_Value = p_NewValue,
            Contract_Start_Date = p_NewStartDate,
            Contract_End_Date = p_NewEndDate,
            Sponsor_Level = p_NewLevel
        WHERE ID_Sponsor = p_SponsorID AND ID_Event = p_EventID;

        COMMIT;
        SELECT 'Dati Sponsorship aggiornati' AS Status;
    END IF;
END //

```

- **Terminazione Anticipata (`Transaction_Cancel_Sponsorship`):** Questa procedura implementa una cancellazione logica ("Soft Delete") anziché fisica. Invece di rimuovere la riga (perdendo lo storico del fatto che lo sponsor ha pagato), la procedura imposta la `Contract_End_Date` alla data odierna (`CURDATE()`). Questo approccio permette di revocare i benefici dello sponsor (es. visibilità logo) mantenendo però traccia del contratto nei report finanziari storici.

```
CREATE PROCEDURE Transaction_Cancel_Sponsorship(IN p_SponsorID
   INT, IN p_EventID INT)
BEGIN
    DECLARE v_Exists INT;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SELECT 'Errore cacncellazione sponsorship.' AS Status;
    END;
    START TRANSACTION;
    SELECT COUNT(*) INTO v_Exists
    FROM Sponsorship
    WHERE ID_Sponsor = p_SponsorID
      AND ID_Event = p_EventID
      AND Contract_End_Date >= CURDATE();
    IF v_Exists > 0 THEN
        UPDATE Sponsorship
        SET Contract_End_Date = CURDATE()
        WHERE ID_Sponsor = p_SponsorID AND ID_Event = p_EventID;
        COMMIT;
        SELECT 'Contratto annulato in data odierna' AS Status;
    ELSE
        ROLLBACK;
        SELECT 'Contratto attivo non trovato' AS Status;
    END IF;
```

```
25 END //
```

4.3 Procedure di Supporto all'Area Utente

Per permettere un gestione semplificata sono state realizzate procedure di lettura che astraggono la complessità delle operazioni necessarie.

Consultazione Storico e Biglietti

Queste procedure permettono all'utente di recuperare i propri acquisti e i dettagli dei partecipanti collegati. In particolare, `Procedure_Get_Single_Ticket` sfrutta la vista `View_User_Tickets` per restituire un oggetto "Biglietto" completo di codice univoco e dati della location, pronto per la stampa o la visualizzazione QR.

```
1  -- - visualizza il ticket di un partecipante usando la viw
2  CREATE PROCEDURE Procedure_Get_Single_Ticket(IN p_EventID INT, IN
       p_ParticipantID INT)
3  BEGIN
4      SELECT
5          Event_Name,
6          Location_Name,
7          Address,
8          City,
9          Event_Date,
10         Event_Start_Time,
11         Name,
12         Surname,
13         Ticket_Code,
14         Purchase_Price,
15         Registration_Date
16     FROM View_User_Tickets
17     WHERE ID_Event = p_EventID
18       AND ID_Participant = p_ParticipantID;
19 END //
20
21 CREATE PROCEDURE Procedure_Get_User_Tickets_List(IN p_UserID INT)
22 BEGIN
23     SELECT
24         ID_Event,
25         ID_Participant,
26         Event_Name,
27         Event_Date,
28         -- Ticket_Code,
29         CONCAT(Name, ' ', Surname) AS Participant
30     FROM View_User_Tickets
31     WHERE ID_User = p_UserID
32     ORDER BY Event_Date DESC;
33 END //
34
35 -- - storico iscrizione di un utente specifico
36 CREATE PROCEDURE Procedure_Get_User_Registrations_History(IN p_UserID
       INT)
37 BEGIN
38     SELECT
```

```
        P.ID_Participant,
        P.Name AS Participant_Name,
        P.Surname AS Participant_Surname,
        E.ID_Event,
        E.Title AS Event_Title,
        E.Event_Date,
        E.Event_Start_Time
    FROM Registration R
    JOIN Participant P ON R.ID_Participant = P.ID_Participant
    JOIN Event E ON R.ID_Event = E.ID_Event
    WHERE P.ID_User = p_UserID
    ORDER BY
        P.Surname ASC,
        P.Name ASC,
        E.Event_Date DESC;
END //

-- - partecipanti collegati all'utente specificato
CREATE PROCEDURE Procedure_Get_User_Participants(IN p_UserID INT)
BEGIN
    SELECT
        ID_Participant,
        Name,
        Surname,
        Birth_Date,
        TIMESTAMPDIFF(YEAR, Birth_Date, CURDATE()) AS Age,
        Contact_Email
    FROM Participant
    WHERE ID_User = p_UserID
    ORDER BY Surname ASC, Name ASC;
END //
```

Gestione Sicurezza Account

La procedura di modifica password implementa la verifica della corrispondenza della vecchia password prima di sovrascrivere l'hash memorizzato.

```
-- - cambio password user_account
CREATE PROCEDURE Procedure_Change_Password(IN p_UserID INT, IN
   p_OldPassword VARCHAR(255), IN p_NewPasswordHash VARCHAR(255))
proc_label: BEGIN
    DECLARE v_CurrentPassword VARCHAR(255);
    SELECT Password_Hash INTO v_CurrentPassword
    FROM User_Account
    WHERE ID_User = p_UserID;
    IF v_CurrentPassword IS NOT NULL AND v_CurrentPassword =
   p_OldPassword THEN
        UPDATE User_Account
        SET Password_Hash = p_NewPasswordHash
        WHERE ID_User = p_UserID;
        SELECT 'Password aggiornata.' AS Esito;
    ELSE
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'la vecchia
   password non è corretta.';
    END IF;
END //
```

Operazioni di Controllo e Gestione Evento

Il sistema fornisce strumenti per la gestione dell'evento.

Validazione Accessi

La procedura `Procedure_Validate_Ticket_Code` implementa un sistema di controllo accessi. Ricevendo in input il codice esadecimale, interrogando la vista per confermare la validità, restituisce anche i dati del partecipante.

```
-- _ validazione bigietto attraverso il codice
CREATE PROCEDURE Procedure_Validate_Ticket_Code(IN p_TicketCode
    VARCHAR(100))
BEGIN
    SELECT
        'VALID' AS Status,
        Event_Name,
        Event_Date,
        Name,
        Surname,
        Ticket_Code
    FROM View_User_Tickets
    WHERE Ticket_Code = p_TicketCode;
END //
```

Lista Partecipanti

Permette agli organizzatori di estrarre la lista completa degli iscritti per un evento, includendo l'email dell'acquirente (User).

```
--  _ lista partecipanti a un evtno specifico (Registration +
    Participant + User_Account)
--  _ Ticket_Code, Age, Buyer_Email
CREATE PROCEDURE Procedure_Get_Event_Participants_List(IN p_EventID
    INT)
BEGIN
    SELECT
        P.Name,
        P.Surname,
        P.Birth_Date,
        TIMESTAMPDIFF(YEAR, P.Birth_Date, CURDATE()) AS Age,
        CONCAT(HEX(R.ID_Event), '-', HEX(R.ID_Participant), '-', HEX(
    UNIX_TIMESTAMP(R.Registration_Date))) AS Ticket_Code,
        R.Registration_Date,
        U.Login_Email AS Buyer_Email
    FROM Registration R
    JOIN Participant P ON R.ID_Participant = P.ID_Participant
    JOIN User_Account U ON P.ID_User = U.ID_User
    WHERE R.ID_Event = p_EventID
    ORDER BY P.Surname ASC, P.Name ASC;
END //
```

Monitoraggio delle statistiche degli eventi

Il monitoraggio degli eventi è gestito tramite procedure che uniscono i ricavi (Vendita Biglietti e Contratti Sponsor) per fornire il guadagno unico dell'evento.

Monitoraggio finanziario e occupazione

La procedura `Report_Event_Revenue_Stats` rappresenta lo strumento di analisi principale. Essa calcola dinamicamente:

- Il tasso di riempimento della sala (*Occupancy Rate*).
- Il ricavo generato dai biglietti (*Ticket Revenue*).
- Il ricavo generato dalle sponsorizzazioni (*Sponsorship Revenue*).
- Il bilancio totale dell'evento.

La gestione dei `NULL` cin `COALESCE` garantisce che il report sia consistente anche per eventi che non hanno sponsor o iscrizioni.

```
1  --  _ STATISTICHE Event (Registration + Sponsorship)
2  --  _ BILANCIO TOTALE (contabilità)
3  CREATE PROCEDURE Report_Event_Revenue_Stats(IN p_EventID INT)
4  BEGIN
5      SELECT
6          E.ID_Event,
7          E.Title AS Event_Title,
8          E.Event_Date,
9          -- statistiche Partecipazione
10         COUNT(R.ID_Participant) AS Tickets_Sold,
11         E.Max_Seats AS Total_Capacity,
12         CONCAT(ROUND((COUNT(R.ID_Participant) / E.Max_Seats * 100), 1)
       , '%') AS Occupancy_Rate,
13         -- incasso totale biglietti
14         COALESCE(SUM(R.Purchase_Price), 0.00) AS Revenue_Tickets,
15         -- incasso totale Sponsor
16         COALESCE(
17             (SELECT SUM(S.Contract_Value)
18              FROM Sponsorship S
19              WHERE S.ID_Event = E.ID_Event), 0.00
20         ) AS Revenue_Sponsorship,
21         -- totale
22         (COALESCE(SUM(R.Purchase_Price), 0.00) + COALESCE(
23                 (SELECT SUM(S.Contract_Value)
24                  FROM Sponsorship S
25                  WHERE S.ID_Event = E.ID_Event), 0.00)
26         ) AS Total_Revenue
27     FROM Event E
28     LEFT JOIN Registration R ON E.ID_Event = R.ID_Event
29     -- se p_EventID passato è NULL mostra tutto, altrimenti lo
       specifico evento richiesto
30     WHERE (p_EventID IS NULL OR E.ID_Event = p_EventID)
31     GROUP BY E.ID_Event, E.Title, E.Event_Date, E.Max_Seats
32     ORDER BY Total_Revenue DESC;
33 END //
```

Monitoraggio Contratti Sponsor

Queste procedure permettono di filtrare lo stato delle sponsorizzazioni sia dal punto di vista dell'evento sia dal punto di vista dello sponsor, fornendo dati cruciali per il rinnovo dei contratti.

```
-- _ sponsorship di un evento sepicifico
CREATE PROCEDURE Procedure_Get_Event_Sponsorships(IN p_EventID INT)
BEGIN
    SELECT
        Sponsor_Name,
        Contract_Value,
        Contract_Start_Date,
        Contract_End_Date,
        Current_Status,
        Sponsor_Level
    FROM View_Sponsorship_Status
    WHERE ID_Event = p_EventID
    ORDER BY Contract_Value DESC;
END //

-- | sponsorship di uno sponsor specifico
CREATE PROCEDURE Procedure_Get_Sponsor_Sponsorships(IN p_SponsorID INT
    )
BEGIN
    SELECT
        Event_Title,
        Contract_Value,
        Contract_Start_Date,
        Contract_End_Date,
        Current_Status,
        Sponsor_Level
    FROM View_Sponsorship_Status
    WHERE ID_Sponsor = p_SponsorID
    ORDER BY Contract_End_Date DESC;
END //
```

Capitolo 5

Validazione delle Operazioni

5.1 Test

Di seguito vengono riportati il codice utilizzato per verificare il corretto funzionamento dei vincoli e delle procedure.

Violazione Età Minima

Tentativo di iscrizione di un minore a un evento vietato (Trigger `Check_Participant_Age`).

```
-- Evento ID 1 ha Min_Age = 18
CALL Transaction_Register_Single_New(1, 1, 'Mario', 'Rossi', '
    2015-01-01', 'mario@test.it');

-- Risultato Atteso:
-- Error Code: 1644. Partecipante non soddisfa l età minima richiesta
    per questo evento.
```

Violazione Capienza (Sold Out)

Tentativo di iscrizione oltre il limite `Max_Seats` (Trigger `Check_Event_Capacity`).

```
-- Evento ID 2 ha Max_Seats = 100.
-- Dopo aver inserito 100 biglietti:
CALL Transaction_Register_Single_New(2, 2, 'Luigi', 'Verdi', '
    1990-01-01', 'luigi@test.it');

-- Risultato Atteso:
-- Evento SOLD OUT
```

Generazione e validazione Biglietti

- Interrogazione della vista con la procedura per ottenere i partecipanti che un utente ha iscritto.

```
CALL Procedure_Get_User_Tickets_List(1);
```

Output:

ID_Event	ID_Participant	Event_Name	Event_Date	Participant
2	1	Concerto	2025-06-21	Alessandro Ferrante
2	2	Concerto	2025-06-21	Marco Ferrante
2	3	Concerto	2025-06-21	Giovanni Ferrante
2	4	Concerto	2025-06-21	Lucia Ferrante
1	1	Talk	2025-05-10	Alessandro Ferrante

- Interrogazione della vista con la procedura per ottenere il singolo biglietto di un partecipante con il ticket code.

```
CALL Procedure_Get_Single_Ticket(1, 1);
```

Event_Name	Location_Name	Address	City	Event_Date	Event_Start_Time	Name	Surname	Ticket_Code	Purchase_Price	Registration_Date
Talk	Sala A	via sql 1	roma	2025-05-10	10:00:00	Alessandro	Ferrante	1-1-6941DB6E	15.00	2025-12-16 23:21:34

- Interrogando la vista per confermare la validità del ticket code.

```
CALL Procedure_Validate_Ticket_Code('1-1-6941DB6E');
```

Status	Event_Name	Event_Date	Name	Surname	Ticket_Code
VALID	Talk	2025-05-10	Alessandro	Ferrante	1-1-6941DB6E

Cambio Password

Simulazione del cambio password. La procedura verifica la vecchia password ('pass') prima di aggiornare quella nuova ('pass_').

```
CALL Procedure_Change_Password(1, 'pass', 'pass_');
```

Esito
Password aggiornata.

Analisi Contratti per Sponsor

Restituisce tutti i contratti di sponsorizzazione legati allo sponsor con ID=1, ordinati per scadenza.

```
CALL Procedure_Get_Sponsor_Sponsorships(2);
```

Event_Title	Contract_Value	Contract_Start_Date	Contract_End_Date	Current_Status	Sponsor_Level
Concerto	2500.00	2025-03-01	2025-08-01	Expired	platinum

Analisi Contratti per Evento

Restituisce tutti i contratti di sponsorizzazione legati all'evento con ID=1, ordinati per scadenza.

```
CALL Procedure_Get_Event_Sponsorships(3);
```

Sponsor_Name	Contract_Value	Contract_Start_Date	Contract_End_Date	Current_Status	Sponsor_Level
Tech Store	500.00	2025-06-01	2025-08-01	Expired	silver

Report Monitoraggio Globale

Passando il valore `NULL` come parametro alla procedura. Invece di filtrare per un singolo evento, restituisce le statistiche finanziarie (biglietti venduti e sponsorizzazioni) di **tutti** gli eventi presenti a sistema, fornendo una visione d'insieme per l'amministrazione.

```
CALL Report_Event_Revenue_Stats(NULL);
```

ID_Event	Event_Title	Event_Date	Tickets_Sold	Total_Capacity	Occupancy_Rate	Revenue_Tickets	Revenue_Sponsorship	Total_Revenue
2	Concerto	2025-06-21	5	2000	0.3%	150.00	2500.00	2650.00
4	Festival	2025-08-10	3	500	0.6%	60.00	1000.00	1060.00
3	Corso SOL	2025-07-15	1	20	5.0%	50.00	500.00	550.00
1	Talk	2025-05-10	1	50	2.0%	15.00	300.00	315.00
5	Gara Auto	2025-09-01	2	2000	0.1%	50.00	0.00	50.00

Report Monitoraggio Singolo Evento

Genera il report per l'evento ID=1, aggregando incassi da biglietti e sponsorizzazioni.

```
CALL Report_Event_Revenue_Stats(2);
```

ID_Event	Event_Title	Event_Date	Tickets_Sold	Total_Capacity	Occupancy_Rate	Revenue_Tickets	Revenue_Sponsorship	Total_Revenue
2	Concerto	2025-06-21	5	2000	0.3%	150.00	2500.00	2650.00

Interrogazione Vista Biglietti Completi

Visualizzazione denormalizzata di tutti i biglietti emessi, con dettagli su location e codici univoci.

```
SELECT * FROM View_User_Tickets;
```

ID_User	ID_Event	ID_Participant	ID_Booking	Event_Name	Location_Name	Address	City	Event_Date	Event_Start_Time	Name	Surname	Ticket_Code	Purchase_Price	Registration_Date
1	2	1	3	Concerto	Arena B	via dei bit 10	modica	2025-06-21	21:00:00	Alessandro	Ferrante	2-1-6941DB6E	30.00	2025-12-16 23:21:34
1	2	2	2	Concerto	Arena B	via dei bit 10	modica	2025-06-21	21:00:00	Marco	Ferrante	2-2-6941DB6E	30.00	2025-12-16 23:21:34
1	2	3	2	Concerto	Arena B	via dei bit 10	modica	2025-06-21	21:00:00	Giovanni	Ferrante	2-3-6941DB6E	30.00	2025-12-16 23:21:34
1	2	4	2	Concerto	Arena B	via dei bit 10	modica	2025-06-21	21:00:00	Lucia	Ferrante	2-4-6941DB6E	30.00	2025-12-16 23:21:34
5	2	11	7	Concerto	Arena B	via dei bit 10	modica	2025-06-21	21:00:00	Elena	Rosa	2-B-6941DB6E	30.00	2025-12-16 23:21:34
3	3	7	5	Corso SOL	Saletta C	via casa 5	Napoli	2025-07-15	09:00:00	Carlo	Verdi	3-7-6941DB6E	50.00	2025-12-16 23:21:34
4	4	8	6	Festival	Parco D	piazza 3	torino	2025-08-10	18:00:00	Anna	Gialli	4-8-6941DB6E	20.00	2025-12-16 23:21:34
4	4	9	6	Festival	Parco D	piazza 3	torino	2025-08-10	18:00:00	Piero	Gialli	4-9-6941DB6E	20.00	2025-12-16 23:21:34
4	4	10	6	Festival	Parco D	piazza 3	torino	2025-08-10	18:00:00	Marta	Blu	4-A-6941DB6E	20.00	2025-12-16 23:21:34
2	5	5	4	Gara Auto	Arena B	via dei bit 10	modica	2025-09-01	10:00:00	Luca	Bianchi	5-5-6941DB6E	25.00	2025-12-16 23:21:34
2	5	6	4	Gara Auto	Arena B	via dei bit 10	modica	2025-09-01	10:00:00	Sara	Neri	5-6-6941DB6E	25.00	2025-12-16 23:21:34
1	1	1	1	Talk	Sala A	via sql 1	roma	2025-05-10	10:00:00	Alessandro	Ferrante	1-1-6941DB6E	15.00	2025-12-16 23:21:34

Interrogazione Vista Stato Sponsor

Monitoraggio dello stato (Attivo/Scaduto) di tutti i contratti di sponsorizzazione calcolato in tempo reale.

```
SELECT * FROM View_Sponsorship_Status;
```

ID_Sponsor	Sponsor_Name	ID_Event	Event_Title	Contract_Start_Date	Contract_End_Date	Contract_Value	Sponsor_Level	Current_Status
1	Bar Sport	1	Talk	2025-01-01	2025-12-16	300.00	bronze	Expired
2	Pizzeria Gino	2	Concerto	2025-03-01	2025-08-01	2500.00	platinum	Expired
3	Market Da Pino	4	Festival	2025-05-01	2025-09-01	1000.00	silver	Expired
4	Tech Store	3	Corso SQL	2025-06-01	2025-08-01	500.00	silver	Expired

Interrogazione Vista Statistiche Eventi

Panoramica globale su vendite, posti rimanenti e tasso di occupazione per ogni evento.

```
SELECT * FROM View_Event_Stats;
```

ID_Event	Title	Event_Date	Max_Seats	Tickets_Sold	Seats_Remaining	Total_Revenue	Occupancy_Rate
1	Talk	2025-05-10	50	1	49	15.00	2.0%
2	Concerto	2025-06-21	2000	5	1995	150.00	0.3%
3	Corso SQL	2025-07-15	20	1	19	50.00	5.0%
4	Festival	2025-08-10	500	3	497	60.00	0.6%
5	Gara Auto	2025-09-01	2000	2	1998	50.00	0.1%

Storico Iscrizioni Utente

Questa procedura permette di recuperare la cronologia completa delle iscrizioni effettuate dall'account specificato (ID=1). Il result set include non solo gli eventi a cui l'utente partecipa personalmente, ma anche quelli per cui ha effettuato l'iscrizione per conto terzi (es. familiari), ordinati cronologicamente.

```
CALL Procedure_Get_User_Registrations_History(1);
```

ID_Participant	Participant_Name	Participant_Surname	ID_Event	Event_Title	Event_Date	Event_Start_Time
1	Alessandro	Ferrante	2	Concerto	2025-06-21	21:00:00
1	Alessandro	Ferrante	1	Talk	2025-05-10	10:00:00
3	Giovanni	Ferrante	2	Concerto	2025-06-21	21:00:00
4	Lucia	Ferrante	2	Concerto	2025-06-21	21:00:00
2	Marco	Ferrante	2	Concerto	2025-06-21	21:00:00

Verifica Booking

Queste interrogazioni permettono di verificare la corretta persistenza delle transazioni economiche. Viene mostrata prima la visione globale dell'intera tabella e successivamente il filtraggio per uno specifico utente (ID=1).

```
-- Visione globale di tutte le transazioni
SELECT * FROM Booking;
```

ID_Booking	Booking_Date	Total_Amount	ID_User
1	2025-12-16 23:21:34	15.00	1
2	2025-12-16 23:21:34	90.00	1
3	2025-12-16 23:21:34	30.00	1
4	2025-12-16 23:21:34	50.00	2
5	2025-12-16 23:21:34	50.00	3
6	2025-12-16 23:21:34	60.00	4
7	2025-12-16 23:21:34	30.00	5

```sql
-- Verifica transazioni dello specifico utente
SELECT * FROM Booking WHERE ID_User=1;
```

ID_Booking	Booking_Date	Total_Amount	ID_User
1	2025-12-16 23:21:34	15.00	1
2	2025-12-16 23:21:34	90.00	1
3	2025-12-16 23:21:34	30.00	1

Lista Partecipanti

Verifica il recupero delle anagrafiche collegate all'utente 1. Questa procedura permette di visualizzare i partecipanti già presenti, evitando così di reinserirli.

```sql
CALL Procedure_Get_User_Participants(1);
```

ID_Participant	Name	Surname	Birth_Date	Age	Contact_Email
1	Alessandro	Ferrante	2003-04-28	22	af@alessandroferrante.net
3	Giovanni	Ferrante	2010-05-05	15	g@email.com
4	Lucia	Ferrante	1975-03-10	50	lucia@email.com
2	Marco	Ferrante	2005-01-01	20	m@email.com

Test Negativo: Fallimento Cambio Data (Swap)

Questo test verifica la robustezza della logica transazionale e dei controlli di coerenza. Si tenta di spostare un partecipante (ID 1) dall'Evento 1 all'Evento 3. Poiché i due eventi hanno titoli differenti (es. un "Concerto" e uno "Spettacolo Teatrale"), la procedura intercetta l'incongruenza, esegue il `ROLLBACK` e restituisce un messaggio di errore, impedendo lo scambio non valido.

```sql
-- Tentativo di swap tra eventi diversi (ID 1 -> ID 3)
CALL Transaction_Swap_Event_Date(1, 1, 3);

-- Risultato atteso: 'Evento diverso. (Titolo)'
```

StatusSwapEventDate
Evento diverso. (Titolo)

Lista Partecipanti per un determinato Evento

Questa funzionalità è fondamentale per la gestione dell'evento. Eseguendo la procedura per un evento specifico, il sistema restituisce l'elenco completo degli iscritti, combinando i dati anagrafici del partecipante con il codice univoco del biglietto e l'email dell'acquirente (User Account). Ideale per il controllo degli ingressi.

```
1 CALL Procedure_Get_Event_Participants_List(2);
```

Name	Surname	Birth_Date	Age	Ticket_Code	Registration_Date	Buyer_Email
Alessandro	Ferrante	2003-04-28	22	2-1-6941DB6E	2025-12-16 23:21:34	af@alessandroferrante.net
Giovanni	Ferrante	2010-05-05	15	2-3-6941DB6E	2025-12-16 23:21:34	af@alessandroferrante.net
Lucia	Ferrante	1975-03-10	50	2-4-6941DB6E	2025-12-16 23:21:34	af@alessandroferrante.net
Marco	Ferrante	2005-01-01	20	2-2-6941DB6E	2025-12-16 23:21:34	af@alessandroferrante.net
Elena	Rosa	1995-12-12	30	2-B-6941DB6E	2025-12-16 23:21:34	e@email.com

Capitolo 6

Conclusioni

Il progetto ha portato alla realizzazione di un sistema di gestione eventi completo, basato su un database relazionale. L'analisi iniziale dei requisiti ha permesso di identificare le criticità del dominio, in particolare la gestione della capienza delle location e la distinzione tra acquirente e partecipante.

6.1 Risultati Raggiunti

Gli obiettivi sono stati raggiunti con le seguenti implementazioni:

- **Integrità dei Dati:** L'uso estensivo di vincoli e Trigger garantisce che non sia possibile vendere biglietti oltre la capienza (Overbooking) o a partecipanti non idonei (Età minima).
- **Ottimizzazione:** La scelta di calcolare attributi derivati (come il `Ticket_Code` e lo stato delle sponsorizzazioni) ha ridotto la ridondanza senza sacrificare le performance di lettura.
- **Scalabilità delle Transazioni:** L'implementazione di procedure transazionali ACID e il supporto all'inserimento in gruppi tramite JSON permettono al sistema di gestire picchi di carico e ordini complessi in modo efficiente.