Progettazione ed implementazione di una base di dati per la gestione della prevendita di biglietti per concerti

Basi di Dati e Laboratorio

Gruppo 7

Bregant Jodie
162799

Franco Enrico
144237

Mancin Rocco
162506

Remorini Matteo
161561

Anno accademico 2024-2025

Indice

1. TRACCIA

Un'azienda che si occupa della prenotazione dei biglietti per eventi musicali (concerti) organizzati in diverse città italiane vuole commissionare la realizzazione di una base di dati per la gestione della prevendita dei biglietti attraverso le diverse agenzie pubblicitarie cui è associata.
Le informazioni da memorizzare sono le seguenti.

I concerti sono tenuti da solisti o gruppi musicali in diverse città italiane, in ambienti diversi (teatri, palazzetti, stadi, etc.) e in date diverse. In un dato giorno, uno stesso solista/gruppo svolge al più un concerto. Ogni solista/gruppo può svolgere più concerti nella stessa città, nello stesso ambiente o in ambienti diversi. Il numero di posti disponibili per un dato concerto può ovviamente variare da ambiente ad ambiente, ma anche in uno stesso ambiente può variare da concerto a concerto (se l'evento è un concerto di musica classica si sistemano delle sedie e i posti diminuiscono, se l'evento è un concerto rock non vengono utilizzate le sedie e i posti aumentano). Analogamente il prezzo del biglietto varia da concerto a concerto (per semplicità, assumiamo che il prezzo del biglietto sia unico per ogni concerto).
Ad ogni agenzia è associato un codice identificativo e deve essere possibile stabilire quali prenotazioni sono state effettuate da ogni singola agenzia per ogni singolo concerto. Di ogni concerto, deve essere possibile controllare in ogni istante quanti posti sono già stati prenotati e quanti sono ancora liberi. Occorre, inoltre, poter determinare quanti biglietti ogni agenzia ha venduto per un determinato concerto (per poter dividere tra le varie agenzie ì diritti di prevendita). Infine, bisogna poter stabilire per ogni ambiente quanti biglietti sono stati venduti per un determinato concerto di un dato solista/gruppo musicale.

Sì definisca uno schema Entità-Relazioni che descriva il contenuto informativo del sistema, illustrando con chiarezza le eventuali assunzioni fatte. Lo schema dovrà essere completato con attributi ragionevoli per ciascuna entità (identificando le possibili chiavi) e relazione. Vanno specificati accuratamente i vincoli di cardinalità e partecipazione di ciascuna relazione.

2. ANALISI DEI REQUISITI

La fase di acquisizione dei requisiti si è basata su una lettura approfondita del testo fornito, che, nel nostro caso, ne era la fonte principale.
Essendo scritto in linguaggio naturale, si è dovuto prestare attenzione ad eventuali ambiguità e dubbi che potevano scaturire dalle diverse interpretazioni date alla traccia.
Tramite la formattazione del testo, costruzione di glossari e individuazione delle operazioni richieste al sistema, si è potuto raccogliere diversi requisiti funzionali utente e di dominio.
La fase di ingegneria dei requisiti si è svolta in modo dinamico ed interconnesso con la progettazione concettuale, e si è reso necessario iterare il procedimento più volte per rappresentare al meglio il dominio di interesse.

2.1. Analisi della traccia

Come prima operazione, si è evidenziato i termini nel testo importanti nel contesto della progettazione.

I concerti sono tenuti da solisti o gruppi musicali in diverse città italiane, in ambienti diversi (teatri, palazzetti, stadi, etc.) e in date diverse. In un dato giorno, uno stesso solista/gruppo svolge al più un concerto. Ogni solista/gruppo può svolgere più concerti nella stessa città, nello stesso ambiente o in ambienti diversi. Il numero di posti disponibili per un dato concerto può ovviamente variare da ambiente ad ambiente, ma anche in uno stesso ambiente può variare da concerto a concerto (se l'evento è un concerto di musica classica si sistemano delle sedie e i posti diminuiscono, se l'evento è un concerto rock non vengono utilizzate le sedie e i posti aumentano). Analogamente il prezzo del biglietto varia da concerto a concerto (per semplicità, assumiamo che il prezzo del biglietto sia unico per ogni concerto). Ad ogni agenzia è associato un codice identificativo e deve essere possibile stabilire quali prenotazioni sono state effettuate da ogni singola agenzia per ogni singolo concerto. Di ogni concerto, deve essere possibile controllare in ogni istante quanti posti sono già stati prenotati e quanti sono ancora liberi. Occorre, inoltre, poter determinare quanti biglietti ogni agenzia ha venduto per un determinato concerto (per poter dividere tra le varie agenzie ì diritti di prevendita). Infine, bisogna poter stabilire per ogni ambiente quanti biglietti sono stati venduti per un determinato concerto di un dato solista/gruppo musicale.

Legenda

- Entità
- Vincoli
- Informazioni utili
- Operazioni richieste

2.2. Glossario

Grazie alla formattazione precedente, si è potuto procedere alla costruzione di un glossario più preciso.

Termine	Descrizione	Sinonimi	Collegamenti
Concerto	Evento musicale tenuto da un solista o un gruppo.		Biglietto, Ambiente, Artista
Artista	Entità che tiene un concerto.	Solista, Gruppo musicale	Concerto
Ambiente	Edificio fisico in cui si tiene un concerto.	Teatro, Palazzetto, Stadio	Concerto
Biglietto	Entità astratta che fornisce l'autorizzazione di partecipare ad un concerto.	Prenotazione	Concerto, Agenzia
Agenzia	Compagnia responsabile della vendita dei biglietti.		Biglietto

Table 1: *La tabella riporta un glossario dei termini incontrati durante l'analisi della traccia.*

2.3. Ristrutturazione dei requisiti

Seguendo il glossario definito in precedenza, possiamo partizionare il testo in frasi relative al medesimo concetto.

Frasi relative a concerto
I concerti sono tenuti da solisti o gruppi musicali in diverse città italiane, in ambienti diversi (teatri, palazzetti, stadi, etc.) e in date diverse. Il numero di posti disponibili per un dato concerto può ovviamente variare da ambiente ad ambiente. Di ogni concerto, deve essere possibile controllare in ogni istante quanti posti sono già stati prenotati e quanti sono ancora liberi.
Frasi relative a artista
In un dato giorno, uno stesso solista/gruppo svolge al più un concerto. Ogni solista/gruppo può svolgere più concerti nella stessa città, nello stesso ambiente o in ambienti diversi. In un dato giorno, uno stesso solista/gruppo svolge al più un concerto.
Frasi relative a ambiente
In uno stesso ambiente (il numero di posti disponibili) può variare da concerto a concerto (se l'evento è un concerto di musica classica si sistemano delle sedie e i posti diminuiscono, se l'evento è un concerto rock non vengono utilizzate le sedie e i posti aumentano).
Frasi relative a biglietto
Analogamente il prezzo del biglietto varia da concerto a concerto (per semplicità, assumiamo che il prezzo del biglietto sia unico per ogni concerto). Infine, bisogna poter stabilire per ogni ambiente quanti biglietti sono stati venduti per un determinato concerto di un dato solista/gruppo musicale.
Frasi relative ad agenzia
Ad ogni agenzia è associato un codice identificativo e deve essere possibile stabilire quali prenotazioni sono state effettuate da ogni singola agenzia per ogni singolo concerto. Occorre, inoltre, poter determinare quanti biglietti ogni agenzia ha venduto per un determinato concerto (per poter dividere tra le varie agenzie ì diritti di prevendita).

Table 2: *La tabella riporta le frasi prese dalla traccia suddivise per tema.*

Si è notato come questa suddivisione del testo sia poco utile ai fini di una comprensione migliore dei requisiti: le frasi sono difficilmente associabili ad un solo termine di raggruppamento, in quanto nel testo la caratterizzazione avviene mettendo in relazione elementi diversi, ad esempio: "Bisogna poter stabilire per ogni *ambiente* quanti *biglietti* sono stati venduti per un determinato *concerto* di un dato *solista/gruppo musicale*."

2.4. Operazioni esplicitamente richieste

1. *Ottenimento del numero di biglietti venduti da ogni singola agenzia per ogni singolo concerto*
2. *Per ogni concerto, quanti dei posti sono già stati occupati (il biglietto è già stato venduto) e quanti ancora sono liberi*
3. *Determinare quanti biglietti sono stati venduti da ogni agenzia per uno specifico concerto*
4. *Per ogni ambiente, capire quanti biglietti sono stati venduti per un determinato concerto di un dato artista*

3. PROGETTAZIONE CONCETTUALE

L'insieme dei requisiti discusso e identificato nel precedente punto, viene ora rappresentato con un diagramma Entità-Relazioni.
Inizialmente la strategia *Inside-Out* si è dimostrata particolarmente utile per la scomposizione del problema e la coordinazione all'interno del team, per poi fare spazio ad elementi delle più rigide *Top-Down* e *Bottom-Up*.
Per questo il lavoro di progettazione e costruzione dello schema possiamo dire sia stato guidato da una strategia *Mista*.
Indubbiamente, questa fase si è dimostrata la più critica nella progettazione del sistema, avendo i membri del gruppo differenti opinioni riguardo le scelte migliori. Tuttavia, questo ha portato ad una estensiva discussione e cura ai dettagli.

3.1. Diagramma Entità-Relazioni

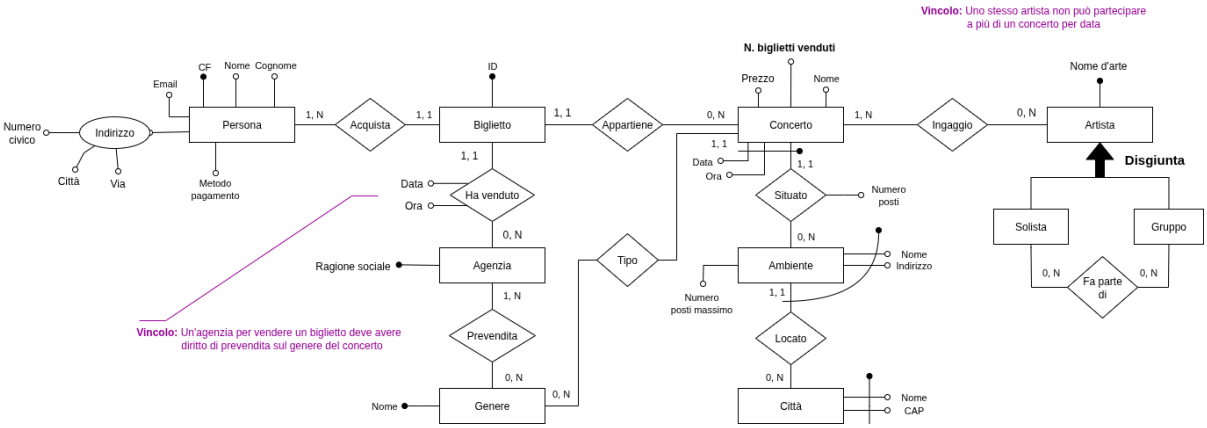


Figure 1: *L'iterazione finale e definitiva del diagramma E-R rappresentante i requisiti.*

Come si può vedere, il diagramma fa usa di diversi *design pattern* atti a rappresentare situazioni specifiche, come il pattern di *Generalizzazione* per indicare che la generica entità `Artista` si può andare a specializzare in `Solista` oppure `Gruppo`.
Per arrivare a questo risultato, si è sicuramente reso necessario un approccio *iterativo-incrementale*, che ha prodotto alcune delle scelte implementative riportate di seguito e ha permesso di raffinare ed aggiungere attributi rilevanti.

3.2. Scelte implementative e assunzioni sul dominio

- Si è deciso di rendere l'acquirente e l'utilizzatore del biglietto la stessa persona. Si sarebbe anche potuto tracciare separatamente, ma, al fine del dominio di interesse, si è dimostrato poco rilevante in quanto la parte dei pagamenti e tracciamenti associati non era trattata nel dettaglio, al costo di un'implementazione più difficoltosa. Nonostante ciò, sarebbe stata una modellazione più precisa del contesto reale.
- Si è deciso di avere un singolo tipo di biglietto.
- Si assume un limite temporale alle operazioni su un biglietto, non permettendo l'inserimento, la cancellazione o la modifica di esso se il concerto è in corso o appartenente al passato. Così facendo si escludono operazioni illogiche e si salvaguarda il mantenimento storico, che altrimenti falserebbe le statistiche.
- L'attributo *Prezzo* (di un biglietto) è memorizzato nell'entità `Concerto` e non in `Biglietto` perchè, come scritto nel testo, assumiamo che esso sia univoco per tutti i biglietti di un concerto. Evitando la conseguente ripetizione e spreco di memoria.

- Un `Concerto` può essere gratuito, ma anche in quel caso viene "venduto" un biglietto come modo per riservare un posto e/o fare da identificativo.
- L'attributo dell'entità `Concerto`, *N. biglietti venduti*, è un attributo derivato, sul quale nei passaggi seguenti viene fatta un'analisi delle ridondanze per verificarne l'efficacia. L'eventuale implementazione dipende da essa.
  Sarebbe un'informazione ricavabile contando il numero di istanze dell'entità `Biglietto` associate a un determinato `Concerto`.
- Assumiamo, inoltre, che gli attributi *Numero posti massimo* (`Ambiente`), *Numero posti* (relazione Situato) e il genere del `Concerto` siano invariabili, una volta definiti.
- Si è fatta l'assunzione che il nome di un *Artista solista* o di un *Artista gruppo musicale* fosse da considerarsi univoco.
- Si è voluta aggiungere anche una relazione *Fa parte di* alle due specializzazioni in quanto un artista solista (che tiene concerti da solista), può anche far parte di un gruppo (che tiene concerti come entità complessiva gruppo). Un riferimento reale può essere il caso del cantante del gruppo "Linkin Park", Mike Shinoda, che ha intrapreso anche una carriera in solo. La cardinalità minima di artisti in un gruppo è zero, perchè non è detto che ci sia tra loro un componente che performa al di fuori del gruppo.
- Si è pensato di suddividere le entità in tre sottogruppi, che ne caratterizzano l'esistenza:
  - Le entità quali `Genere`, `Citta` e `Agenzia` possono considerarsi "pre-caricate" e quindi *read-only*. Sono istanze che esistono indipendentemente dal resto.
  - Entità come `Ambiente`, `Artista`/`Gruppo` e `Concerto` sono da considerarsi modificabili, nel senso che è possibile eliminare o inserire istanze, ma al contempo sono elementi che hanno senso di esistere anche se non collegati a nessun'altra entità, ad esempio un artista può non aver tenuto nessun concerto ma essere memorizzato nella base di dati. In particolare, riguardo `Concerto`, è chiaro che sia inverosimile che esistano dei concerti nel database che non hanno venduto nemmeno un biglietto, ma al contrario è plausibile pensare che ci sia un certo periodo di tempo nel quale questa situazione potrebbe verificarsi.
  - L'ultima tipologia interessa entità come `Persona` e `Biglietto`. Questi sono elementi che consideriamo solo se collegati ad altre entità tramite relazioni. Ad esempio, nella nostra base di dati non ha senso considerare una persona che non ha comprato alcun biglietto, in quanto fuori dal dominio di interesse.
- Per l'entità `Agenzia` si utilizza la *ragione sociale* come *codice identificativo* indicato dalla traccia.
- Per "diritti di prevendita" di un'agenzia, si intende che ogni agenzia è autorizzata a vendere solo determinati generi di concerti (da questo l'aggiunta della relativa entità).

4. PROGETTAZIONE LOGICA

Questa fase produce uno schema logico partendo dalla documentazione delle fasi precedenti.

Si suddivide in due ulteriori passaggi:

1. **Ristrutturazione del *modello E/R*, in particolare**
   - Analisi delle ridondanze
   - Eliminazione dei costrutti non esprimibili nello schema relazionale
2. **Traduzione del *modello E/R* in schema relazionale**

Non si tratta solamente di un procedimento meccanico, è necessario considerare le prestazioni ed effettuare modifiche, dove possibile, per rendere il tutto più performante.

4.1. Ristrutturazione del modello E/R

4.1.1. Analisi delle ridondanze

Nel diagramma E/R prodotto nella fase di progettazione concettuale era presente un attributo derivato sull'entità `Concerto`, l'attributo *N. biglietti venduti*.
E' facile intuire che si tratta di un'aggiunta superflua, a livello puramente operativo, in quanto fornisce un'informazione ottenibile sfruttando i costrutti già presenti, immaginando la base di dati ultimata secondo lo schema. Ciononostante, non è scontato che la scelta appropriata sia disfarsene, in quanto in base alla frequenza e ai volumi dei dati relativi a certe operazioni, l'attributo aggiuntivo potrebbe garantirci un miglioramento nella *performance.*

Le operazioni considerate nell'analisi, che coinvolgono l'attributo ridondante, saranno:
1. *Inserimento* di un nuovo biglietto per il rispettivo concerto
2. *Ottenimento* dei biglietti venduti per un rispettivo concerto
3. *Eliminazione* di un biglietto già esistente

Di seguito le tabelle con cui è stata effettuata l'analisi e la legenda di riferimento per i simboli.

Legenda simboli usata nelle operazioni
$\boldsymbol{n}$ numero
$\boldsymbol{c}$ costo
$\boldsymbol{f}$ frequenza

Tipo	Definizione
E	Entità
R	Relazione

Table 3: *Legenda tipo considerato*

Tipo accesso	Definizione	Costo
W	Scrittura	2
R	Lettura	1

Table 4: *Legenda tipo accesso*

4.1.1.1. Frequenza delle operazioni giornaliere

Operazione	Frequenza
Inserimento	3685
Ottenimento	50
Eliminazione	73

Table 5: *La tabella riporta le frequenze medie delle operazioni*

La frequenza delle operazioni è stato scelta prendendo come caso di studio un episodio reale[1] e pensando ad un'agenzia di medie dimensioni che ha in carico il 10% dei biglietti nazionali.

Si arriva quindi a dire che

$$\left(\frac{n_{\text{ biglietti venduti}}}{n_{\text{giorni periodo considerato}}} \right) * 0.10 = \frac{8000000}{7 * 31} * 0.10 \cong 3685$$

[1]Nel 2022 in Italia nei primi 7 mesi sono stati venduti 8 milioni di biglietti: repubblica.it

biglietti inseriti al giorno

Nel caso dell'ottenimento delle informazioni, si è ipotizzato che l'operazione venga effettuata solamente dagli organizzatori del concerto e le agenzie con diritto di prevendita, per cui è stato scelto il valore di 50.
Riguardo l'eliminazione dei biglietti, invece, si è ipotizzato un tasso di cancellazione di biglietti acquistati del 2%, stabilendo quindi

$$n_{\text{ biglietti inseriti giornamente}} * 0.02 = 3685 * 0.02 \cong 73$$

biglietti eliminati al giorno

4.1.1.2. Volume dei dati

Entità / Relazione	Tipo	Volume
Concerto	E	1
Appartiene	R	433
Biglietto	E	433

Table 6: *La tabella riporta i volumi medi per un singolo concerto*

I volumi di `Appartiene` e `Biglietto` derivano sempre dai dati sopra descritti. Si tratta dei 3685 biglietti inseriti estesi ad un anno intero, divisi per il numero di concerti gestiti in un anno:

$$\frac{n_{\text{ biglietti}} * n_{\text{ giorni anno}}}{n_{\text{ concerti gestiti in un anno}}} = \frac{3685 * 365}{3100} \cong 433$$

biglietti giornalieri per ogni concerto.

Passiamo quindi a calcolare i costi che dovremmo sostenere per effettuare queste operazioni sia nel *caso con ridondanza* che nel *caso senza ridondanza*.

4.1.1.3. Caso con ridondanza

Tabella degli accessi con INSERIMENTO

	Tipo	Numero accessi	Tipo accesso
Biglietto	E	1	W
Appartiene	R	1	W
Ha venduto	R	1	W
Acquista	R	1	W
Concerto	E	1	R
Concerto	E	1	W

Table 7: *La tabella riporta il tipo di accesso alla base di dati rispetto l'operazione*

Complessivamente, sommando ogni entità o relazione incontrata effettuando l'operazione di INSERIMENTO e tenendo conto del costo di ogni singolo tipo di accesso si trova:

$$f_{\text{operazione}} * \sum(n_{\text{accesso}} * c_{\text{accesso}}) = 3685 * 5 * 2 + 3685 = 40535$$

accessi.

Tabella degli accessi con OTTENIMENTO

	Tipo	Numero accessi	Tipo accesso
Concerto	E	1	R

Table 8: *La tabella riporta il tipo di accesso alla base di dati rispetto l'operazione*

Si ha quindi:

$$f_{\text{operazione}} * \sum(n_{\text{accesso}} * c_{\text{accesso}}) = 50$$

accessi.

Tabella degli accessi con ELIMINAZIONE

	Tipo	Numero accessi	Tipo accesso
Biglietto	E	1	W
Acquista	R	1	W
Ha venduto	R	1	W
Appartiene	R	1	W
Concerto	E	1	R
Concerto	E	1	W

Table 9: *La tabella riporta il tipo di accesso alla base di dati rispetto l'operazione*

Otteniamo:

$$f_{\text{operazione}} * \sum(n_{\text{accesso}} * c_{\text{accesso}}) = 73 * 5 * 2 + 73 = 803$$

accessi.

Complessivamente, mantenendo l'attributo ridondante, si raggiungono:

$$n_{\text{accessi inserimento}} + n_{\text{accessi ottenimento}} + n_{\text{accessi cancellazione}} = 40535 + 50 + 803 = 41388$$

accessi alla base di dati giornalieri (per le operazioni considerate).

4.1.1.4. Caso senza ridondanza

Tabella degli accessi con INSERIMENTO

	Tipo	Numero accessi	Tipo accesso
Biglietto	E	1	W
Appartiene	R	1	W
Ha venduto	R	1	W
Acquista	R	1	W

Table 10: *La tabella riporta il tipo di accesso alla base di dati rispetto l'operazione*

Complessivamente, sommando ogni entità o relazione incontrata effettuando l'operazione di INSERIMENTO e tenendo conto del costo di ogni singolo tipo di accesso si trova:

$$f_{\text{operazione}}(n_{\text{accesso}} * c_{\text{accesso}}) = 3685 * 4 * 2 = 29480$$

accessi.

Tabella degli accessi con OTTENIMENTO

	Tipo	Numero accessi	Tipo accesso
Biglietto	E	433	R

Table 11: *La tabella riporta il tipo di accesso alla base di dati rispetto l'operazione*

Si conclude quindi che:

$$f_{\text{operazione}} * \sum_{1}^{n}(n_{\text{accesso}} * c_{\text{accesso}}) = 433 * 50 = 21650$$

accessi.

Tabella degli accessi con ELIMINAZIONE

	Tipo	Numero accessi	Tipo accesso
Biglietto	E	1	W
Acquista	R	1	W
Ha venduto	R	1	W
Appartiene	R	1	W

Table 12: *La tabella riporta il tipo di accesso alla base di dati rispetto l'operazione*

Abbiamo quindi:

$$f_{\text{operazione}} * \sum_{1}^{n}(n_{\text{accesso}} * c_{\text{accesso}}) = 73 * 4 * 2 = 584$$

accessi.

Complessivamente, senza mantenere l'attributo ridondante, si raggiungono:

$$n_{\text{accessi inserimento}} + n_{\text{accessi ottenimento}} + n_{\text{accessi cancellazione}} =$$
$$29480 + 21650 + 584 = 51714$$
accessi alla base di dati giornalieri.

4.1.1.5. Conclusioni

Dopo aver esaminato i due possibili casi (Section 4.1.1.3, Section 4.1.1.4) si può giungere alla conclusione che l'attributo ridondante è utile e riduce il numero di accessi necessari al database.
Si risparmiano in media:

$$n_{\text{accessi senza ridondanza}} - n_{\text{accessi con ridondanza}} = 51714 - 41388 = 10326$$

accessi alla base di dati giornalieri.

4.1.1.6. Analisi del punto di rottura

Si è provato ad analizzare fino a che punto mantenere l'attributo derivato sarebbe stato di beneficio per l'efficienza della base di dati.
Nel grafico seguente è stata considerata la frequenza dell'operazione di *Inserimento*, in quanto, come si può vedere nella sezione precedente (Section 4.1.1.3) è quella che contribuisce di più all'innalzamento dei costi, mantenendo invariate quelle delle altre operazioni.

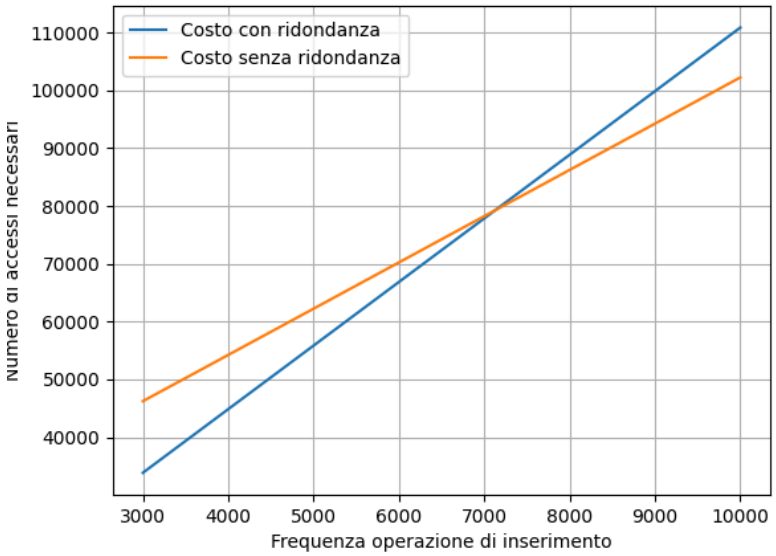


Figure 2: *Il grafico rappresentante l'andamento dei costi al variare della frequenza con cui vengono effettuati gli inserimenti.*

Il grafico rende chiara la conclusione che mantenere l'attributo ridondante sembra essere la scelta migliore, in questo scenario, fintanto che la frequenza degli inserimenti si mantiene all'incirca al di sotto delle 7100 unità (7127).

4.1.2. Eliminazione dei costrutti non esprimibili nello schema relazionale

Nel caso del nostro diagramma E/R, in questa fase si è reso necessario eliminare i costrutti di generalizzazione e attributi composti, che non sono esprimibili in alcun modo nel successivo schema relazionale.
Si è quindi ottenuto il seguente diagramma modificato:

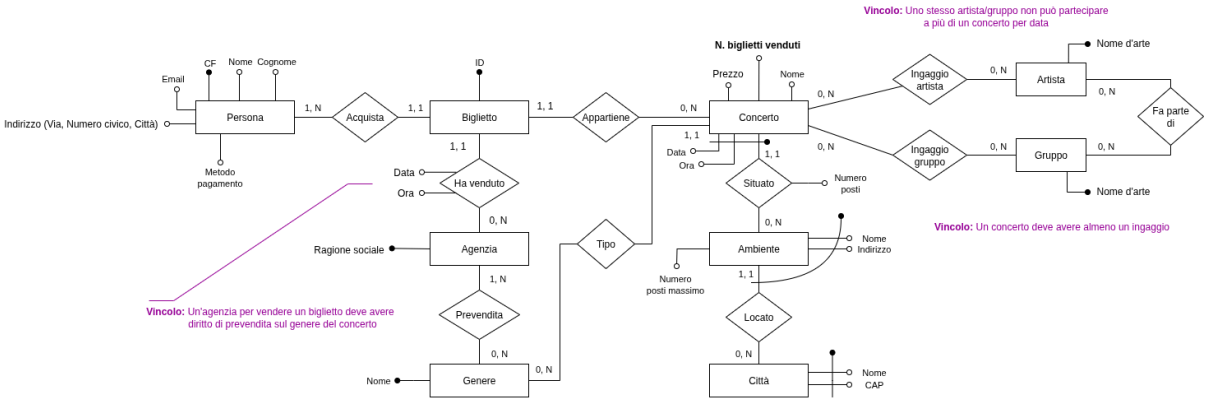


Figure 3: *Il diagramma E/R dopo la ristrutturazione nella fase di progettazione logica.*

Come si può notare dallo schema, la scelta progettuale è stata di mantenere i *figli* della generalizzazione ed eliminare il genitore, collegando direttamente l'entità `Concerto` alle due specializzazione.
Notiamo come questa scelta ci porti a dover definire un vincolo esterno per fare in modo di garantire che ogni istanza di `Concerto` comprenda almeno uno tra `Artista` e `Gruppo`.

Inoltre, è stato eliminato l'attributo composto *Indirizzo* rendendolo un attributo semplice, una stringa rappresentante le informazioni di Via, Numero civico e Città, immaginando che per le operazioni di nostro interesse non fosse fondamentale mantenere separate queste specifiche.

4.2. Traduzione del modello E/R in schema relazionale

Legenda:

- **Tabella**
- Attributo
- Chiave primaria
- FK: Chiave esterna ⟶ Attributo di riferimento

Persona (CF, nome, cognome, email, indirizzo, metodo_pagamento)

Agenzia (ragione_sociale)

Genere (nome)

Diritto_Prevendita (agenzia, genere)

FK: agenzia ⟶ Agenzia.ragione_sociale

FK: genere ⟶ Genere.nome

Citta (nome, CAP)

Ambiente (nome, indirizzo, *locato*, numero_posti_massimo)

FK: *locato* ⟶
nome_citta → Citta.nome
cap_citta → Citta.cap

Concerto (data, ora, *luogo*, genere, prezzo, numero_biglietti_venduti, numero_posti, nome)

FK: *luogo* ⟶
nome_ambiente → Ambiente.nome

indirizzo → Ambiente.indirizzo
citta_nome → Ambiente.citta_nome
cap_citta → Ambiente.citta_cap

FK: genere ⟶ Genere.nome

Biglietto (id, venditore, proprietario, *concerto*, data_vendita, ora_vendita)

FK: venditore ⟶ Agenzia.ragione_sociale

FK: proprietario ⟶ Persona.CF

FK: *concerto* ⟶
data_concerto → Concerto.data
ora_concerto → Concerto.ora
nome_ambiente → Concerto.ambiente_nome
indirizzo_ambiente → Concerto.indirizzo
nome_citta → Concerto.citta_nome
cap_citta → Concerto.citta_cap

Artista (nome_arte)

Gruppo (nome_arte)

Artista_In_Gruppo (artista, gruppo)

FK: artista ⟶ Artista.nome_arte

FK: gruppo ⟶ Gruppo.nome_arte

Ingaggio_Artista (concerto, artista)

FK: *concerto* ⟶
data_concerto → Concerto.data
ora_concerto → Concerto.ora
nome_ambiente → Concerto.ambiente_nome
indirizzo_ambiente → Concerto.indirizzo
nome_citta → Concerto.citta_nome
cap_citta → Concerto.citta_cap

FK: artista ⟶ Artista.nome_arte

Ingaggio_Gruppo (*concerto*, gruppo)

FK: *concerto* ⟶
data_concerto → Concerto.data
ora_concerto → Concerto.ora
nome_ambiente → Concerto.ambiente_nome
indirizzo_ambiente → Concerto.indirizzo
nome_citta → Concerto.citta_nome
cap_citta → Concerto.citta_cap

FK: gruppo ⟶ Gruppo.nome_arte

4.2.1. Vincoli

Nelle due fasi precedenti di progettazione ci sono dei vincoli di dominio e integrità che non possono essere garantiti con i costrutti della progettazione concettuale e logica, ma solamente nella fase dell'implementazione vera e propria.

4.2.1.1. Vincoli di dominio

Persona

- *metodo_pagamento* : ha tre valori possibili (contanti, carta_di_credito e bitcoin)
- *CF* : deve rispettare la composizione standard del Codice Fiscale
- *Email* : deve essere di un formato accettabile, quindi presentare una chiocciola e un dominio accettato dopo il punto

Citta

- *cap* : deve rispettare la composizione standard di cinque numeri decimali

Ambiente

- *numero_posti_massimo* : deve appartenere all'insieme $\mathbb{N}^+$

Concerto

- *numero_posti* : deve appartenere all'insieme $\mathbb{N}^+$
- *numero_biglietti_venduti* : deve essere un numero intero positivo $<$ di *numero_posti*
- *prezzo* : deve essere un numero reale positivo o nullo

4.2.1.2. Vincoli d'integrità

Derivanti dalle relazioni con cardinalità 1:N nello schema E/R, dalla parte contrassegnata dall'1.
VI-1: Persona → Biglietto (1:N) — Una persona deve aver acquistato almeno un biglietto.
VI-2: Agenzia → Genere (1:N) — Un'agenzia deve aver diritto di prevendita su almeno un genere.

Derivanti dai requisiti e/o scelte implementative.
VI-3: Un'agenzia deve avere diritto di prevendita sul genere del concerto per vendere biglietti.
VI-4: Uno stesso artista/gruppo non può partecipare a più di un concerto nello stesso giorno.
VI-5: Un concerto deve avere almeno un ingaggio.
VI-6: Il numero di posti di un concerto deve essere inferiore al numero di posti massimo dell'ambiente in cui si svolge.

5. PROGETTAZIONE FISICA

In questa fase si traduce lo schema logico nella struttura fisica effettiva che verrà implementata nel DBMS PostgreSQL.
Si produce il codice (in questo caso in *SQL*) per la definizione delle tabelle e degli indici.

5.1. Definizione delle tabelle

Di seguito sono stati selezionati due *snippet di codice* SQL per la parte di definizione delle tabelle che si ritengono essere le più complesse e rappresentative.

```sql
CREATE TABLE Concerto (
  nome VARCHAR(200) NOT NULL,
  numero_posti INTEGER NOT NULL,
  numero_biglietti_venduti INTEGER NOT NULL DEFAULT 0,
  prezzo REAL NOT NULL CHECK (prezzo >= 0), -- Potrebbe essere gratuito
  data DATE NOT NULL,
  ora TIME NOT NULL,
  -- Ambiente
  nome_ambiente VARCHAR(50),
```

```
  indirizzo VARCHAR(200),
  -- Città
  nome_citta VARCHAR(50),
  cap_citta CHAR(5),

  genere TEXT NOT NULL REFERENCES Genere (nome) ON UPDATE CASCADE ON DELETE RESTRICT,

  PRIMARY KEY (data, ora, nome_ambiente, indirizzo, nome_citta, cap_citta),
  FOREIGN KEY (nome_ambiente, indirizzo, nome_citta, cap_citta)
    REFERENCES Ambiente (nome, indirizzo, nome_citta, cap_citta)
    ON UPDATE CASCADE ON DELETE RESTRICT
);

CREATE TABLE Biglietto (
  id BIGSERIAL PRIMARY KEY, -- BIGSERIAL ha l'autoincrement di default
  venditore VARCHAR(100) NOT NULL REFERENCES Agenzia (ragione_sociale)
                                  ON UPDATE CASCADE ON DELETE RESTRICT,
  proprietario CHAR(16) NOT NULL REFERENCES Persona (CF)
                                  ON UPDATE CASCADE ON DELETE RESTRICT,

  data_vendita DATE NOT NULL DEFAULT CURRENT_DATE,
  ora_vendita TIME NOT NULL DEFAULT CURRENT_TIME,
  data_concerto DATE NOT NULL,
  ora_concerto TIME NOT NULL,
  nome_ambiente VARCHAR(50) NOT NULL,
  indirizzo_ambiente VARCHAR(200) NOT NULL,
  nome_citta VARCHAR(50) NOT NULL,
  cap_citta CHAR(5) NOT NULL,

    FOREIGN KEY (data_concerto, ora_concerto, nome_ambiente, indirizzo_ambiente,
nome_citta, cap_citta)
    REFERENCES Concerto (data, ora, nome_ambiente, indirizzo, nome_citta, cap_citta)
    ON UPDATE CASCADE ON DELETE RESTRICT
);
```

Si è optato per l'utilizzo di `RESTRICT` in caso di cancellazione per evitare perdite di informazioni. La cancellazione ad esempio di un ambiente è subordinata alla non esistenza di un concerto svolto in esso.

5.2. Definizione degli indici

La scelta degli indici da implementare per il miglioramento delle performance è stata effettuata considerando le operazioni più frequenti e di particolare complessità computazionale. La tipologià di un indice ha un impatto considerevole nel livello di ottimizzazione, perciò le scelte sono state supportate anche da confronti pratici sul database popolato (popolamento successivamente descritto). Confronti basati sulle statistiche e i piani di esecuzione per la medesima query, tramite il comando `EXPLAIN ANALYZE` di PostgreSQL.

5.2.1. Approfondimento teorico sui principali tipi di indici

Le due principali tipologie utili per questo progetto sono `Hash` e `B-Tree`.

- `Hash`:
  Particolarmente vantaggiosi in situazioni di cardinalità molto alta (molti valori unici, con poche ripetizioni) e su query con condizioni di uguaglianza; per questo in rari casi applicabili efficacemente. Inoltre, non sono adatti per query che comportano ordinamenti o range.
- `B-tree`:
  Più versatili, si comportano in maniera ottimale sulle *range query* e anche in caso di query ordinate.

In Postgresql sono talmente ben ottimizzati che si dimostrano la soluzione de facto per quasi ogni situazione. Sono però più lenti in lookup rispetto al tipo hash.

5.2.2. Indici implementati

Si è iniziato determinando gli indici utili per le query descritte in questa relazione (sia di esempio che per l'analisi dei dati in R).

`Biglietto`

`venditore`: velocizzare l'ottenimento di biglietti venduti da una certa agenzia. Viene così ottimizzata l'operazione richiesta dal testo.

```
CREATE INDEX biglietto_venditore_idx ON Biglietto (venditore);
```

Il confronto pratico mostra differenze trascurabili sia su selezione (vedere query sezione successiva) che inserimento di un biglietto, mentre la creazione dell'indice da parte del tipo Hash impiega notevolmente di più (più di 10x tempo). Si è perciò deciso di seguire lo standard de facto.

`proprietario`: per trovare più in fretta i biglietti di una persona e rendere più efficiente la query di analisi.

```
CREATE INDEX biglietto_proprietario_idx ON Biglietto USING HASH (proprietario);
```

La tipologia di indice da utilizzare in questo caso varia a seconda della query di selezione. Per la selezione dei biglietti di una certa persona il tipo `Hash` è particolarmente convieniente, mentre per la selezione del numero di biglietti acquistati da ogni persona registrata (query utilizzata in analisi dei dati), il tipo `B-Tree` è migliore.
Si è pensato di impiegare il tipo Hash poichè si ritiene la prima query molto più di frequente uso.

`Diritto_Prevendita`

`agenzia`: per verificare rapidamente qualora un'agenzia abbia il diritto di prevendita.
Operazione effettuata dai trigger dopo ogni inserimento di biglietto.

```
CREATE INDEX diritto_prevendita_agenzia_idx ON Diritto_Prevendita (agenzia);
```

In questo caso non è possibile effettuare un confronto a causa della ridotta cardinalità della tabella popolata; si opta per il tipo `B-Tree` secondo teoria.

Successivamente, immaginando casi d'uso reali si sono definiti i seguenti indici.

`Concerto`

`prezzo` e **`data`**: si considerano indici utili in quanto query del tipo "Quanti concerti costavano più di/nel periodo ?" si ritengono operazioni che un'agenzia o l'azienda commissionatrice della base di dati potrebbero eseguire di frequente.

```
CREATE INDEX concerto_prezzo_idx ON Concerto (prezzo);
CREATE INDEX concerto_data_idx ON Concerto (data);
```

La presenza di query di tipo range rende l'utilizzo di indici `B-Tree` inevitabile.

5.2.3. Considerazioni ulteriori

Una ulteriore discussione ha evidenziato il fatto che l'inserimento dei biglietti probabilmente sarà un'operazione massiva (ripetuta molte volte in span brevi di tempo) e che quindi avrebbe coinvolto sistemi di caching (a livello hardware e software).
Essendo essa molto efficace nell'ottimizzazione, si è valutato di togliere l'indice su *agenzia* di `Diritto_prevendita` in quanto avrebbe addirittura potuto rallentare l'operazione.
Alla fine si è deciso di mantenerlo rimandando ad una più approfondita analisi.

6. IMPLEMENTAZIONE

6.1. Vincoli di integrità con trigger e check

Si implementano i controlli per far rispettare tre dei vincoli posti, scelti a titolo esemplificativo e rappresentativo.

6.1.1. Vincoli di dominio

Per il metodo di pagamento descritto nella tabella `Persona` si crea un tipo di dato ad hoc, utilizzando il costrutto *enum*.

```
CREATE TYPE metodo_di_pagamento AS ENUM (
  'contanti',
  'carta_di_credito',
  'bitcoin'
);
```

I restanti vincoli di dominio sono implementati tramite *CHECK*.

- Composizione del codice attivazione postale.
  `cap CHAR(5) CHECK (cap ~ '[0-9]{5}')`
- Massimo numero di posti per un ambiente.
  `numero_posti_massimo INTEGER CHECK(numero_posti_massimo > 0)`
- Numero posti per un concerto.
  `numero_posti INTEGER CHECK (numero_posti > 0)`
- Numero biglietti venduti per un concerto.

  ```
  numero_biglietti_venduti INTEGER CHECK (
    numero_biglietti_venduti >= 0 AND numero_biglietti_venduti <= numero_posti
  )
  ```

- Prezzo dei biglietti per un concerto.
  `prezzo REAL CHECK (prezzo >= 0)`

6.1.2. Vincoli d'integrità - Trigger

Sono stati sviluppati tre vincoli in particolare.
Di seguito due funzioni di particolare interesse, sia perchè condivise tra più trigger, sia per la possibilità di utilizzo in query di interrogazione.

```
--Controlla se un concerto ha almeno un ingaggio (artista e/o gruppo) collegato
-- PARAMS : attributi che compongono la chiave primaria di Concerto
-- RETURN : booleano se ha almeno un Ingaggio_*
CREATE OR REPLACE FUNCTION Check_Almeno_Un_Ingaggio_Concerto(
  conc_data DATE, conc_ora TIME, conc_nome_ambiente VARCHAR,
  conc_indirizzo VARCHAR, conc_nome_citta VARCHAR, conc_cap_citta CHAR
) RETURNS BOOLEAN LANGUAGE PLPGSQL AS
$$
  BEGIN
    RETURN (
      EXISTS (
        SELECT 1 FROM Ingaggio_Artista
        WHERE data_concerto = conc_data AND
              ora_concerto = conc_ora AND
              nome_ambiente = conc_nome_ambiente AND
              indirizzo_ambiente = conc_indirizzo AND
              nome_citta = conc_nome_citta AND
```

```
            cap_citta = conc_cap_citta
      ) OR EXISTS (
        SELECT 1 FROM Ingaggio_Gruppo
        WHERE data_concerto = conc_data AND
              ora_concerto = conc_ora AND
              nome_ambiente = conc_nome_ambiente AND
              indirizzo_ambiente = conc_indirizzo AND
              nome_citta = conc_nome_citta AND
              cap_citta = conc_cap_citta
      )
    );
  END;
$$;

-- Controlla se una persona ha almeno un biglietto registrato a suo nome
-- PARAMS : attributo chiave primaria di Persona
-- RETURN : booleano se ha almeno un Biglietto
CREATE OR REPLACE FUNCTION Check_Persona_Almeno_Un_Biglietto(pers CHAR)
  RETURNS BOOLEAN LANGUAGE PLPGSQL AS
$$
  BEGIN
    RETURN EXISTS (SELECT 1 FROM Biglietto WHERE proprietario = pers);
  END;
$$;
```

6.1.2.1. Cardinalità 1:N della relazione `Acquista` tra Persona e Biglietto

#VI-1 — Si utilizzano due trigger per gestire la cardinalità minima di `Persona`, ovvero la necessità di avere almeno un biglietto a lui intestato:

- alla fine della transazione in cui avviene l'inserimento di una nuova persona viene controllato che abbia almeno un biglietto collegato, altrimenti fallisce l'operazione;
- alla fine della transazione in cui avviene il cancellamento di un biglietto viene controllato che non fosse l'ultimo della persona, altrimenti fallisce l'operazione.

```
-- Trigger per inserimento
CREATE CONSTRAINT TRIGGER Persona_ha_Biglietto
  AFTER INSERT ON Persona
  DEFERRABLE INITIALLY DEFERRED
  FOR EACH ROW
  EXECUTE FUNCTION Check_Persona_Biglietto_trigger();

-- Trigger per cancellazione
CREATE CONSTRAINT TRIGGER Ultimo_Biglietto_Persona
  AFTER DELETE ON Biglietto
  DEFERRABLE INITIALLY DEFERRED
  FOR EACH ROW
  EXECUTE FUNCTION Check_Persona_Biglietto_trigger();

-- Funzione condivisa
CREATE OR REPLACE FUNCTION Check_Persona_Biglietto_trigger()
  RETURNS TRIGGER LANGUAGE PLPGSQL AS
$$
  DECLARE
    persona_esiste boolean;
    almeno_biglietto boolean;
  BEGIN
    -- almeno_biglietto   persona_esiste
```

```
    -- false              false             OK
    -- false              true              FAIL
    -- true               false             impossibile
    -- true               true              OK

    persona_esiste = true;
    IF NEW IS NULL THEN
      persona_esiste = EXISTS (SELECT 1 FROM Persona WHERE CF = OLD.proprietario);
      almeno_biglietto = Check_Persona_Almeno_Un_Biglietto(OLD.proprietario);
    ELSE
      almeno_biglietto = Check_Persona_Almeno_Un_Biglietto(NEW.CF);
    END IF;

    IF persona_esiste AND NOT almeno_biglietto THEN
      RAISE EXCEPTION 'La persona deve avere almeno un biglietto';
      RETURN null;
    END IF;
    RETURN COALESCE(NEW, OLD);
  END;
$$;
```

6.1.2.2. Diritti di prevendita

#VI-5 — Un'agenzia può vendere un biglietto solo se ha il diritto di prevendita sul genere del concerto. Questo vincolo è controllato da un trigger valutato prima dell'inserimento del nuovo biglietto.

```
CREATE TRIGGER Agenzia_ha_Diritto_Prevendita
  BEFORE INSERT ON Biglietto
  FOR EACH ROW
  EXECUTE FUNCTION Check_Diritto_Prevendita_Agenzia_trigger();

CREATE OR REPLACE FUNCTION Check_Diritto_Prevendita_Agenzia_trigger()
  RETURNS TRIGGER LANGUAGE PLPGSQL AS
$$
  DECLARE
    genere_concerto TEXT;
  BEGIN
    -- Dal biglietto ricava il genere del concerto
    genere_concerto = (
      SELECT Concerto.genere
      FROM Concerto
      WHERE data = NEW.data_concerto AND
            ora = NEW.ora_concerto AND
            nome_ambiente = NEW.nome_ambiente AND
            indirizzo = NEW.indirizzo_ambiente AND
            nome_citta = NEW.nome_citta AND
            cap_citta = NEW.cap_citta
    );
    -- Controllo esistenza diritto di prevendita da parte dell'agenzia venditrice
    IF NOT EXISTS (
      SELECT 1 FROM Diritto_Prevendita
      WHERE agenzia = NEW.venditore AND
            genere = genere_concerto
    ) THEN
      RAISE EXCEPTION '% non ha diritto di prevendita per il genere %', NEW.venditore,
genere_concerto;
      RETURN null;
    END IF;
```

```
    RETURN NEW;
  END;
$$;
```

6.1.2.3. Ingaggi in un concerto

#VI-7 — Un concerto deve avere almeno un ingaggio di un'artista o gruppo. Il vincolo si rende necessario dalla ristrutturazione della generalizzazione.

Il rispetto di questo vincolo è assicurato da tre trigger complessivi, uno per l'inserimento di un concerto e due per la cancellazione di ingaggi (artista e gruppo).
La funzione di gestione delle trigger è condivisa, minimizzando la ripetizione di codice.

```
CREATE OR REPLACE FUNCTION Check_Ingaggio_Concerto_trigger()
  RETURNS TRIGGER LANGUAGE PLPGSQL AS
$$
  DECLARE
    ok boolean;
  BEGIN
    IF NEW IS NULL THEN
      ok = Check_Almeno_Un_Ingaggio_Concerto(
        OLD.data_concerto, OLD.ora_concerto, OLD.nome_ambiente,
        OLD.indirizzo_ambiente, OLD.nome_citta, OLD.cap_citta
      );
    ELSE
      ok = Check_Almeno_Un_Ingaggio_Concerto(
        NEW.data, NEW.ora, NEW.nome_ambiente,
        NEW.indirizzo, NEW.nome_citta, NEW.cap_citta
      );
    END IF;

    IF NOT ok THEN
      RAISE EXCEPTION 'Concerto deve avere almeno un ingaggio';
      RETURN null;
    END IF;
    RETURN COALESCE(NEW, OLD);
  END;
$$;
```

È stato possibile non utilizzare una funzione distinta tra inserimento e cancellazione (necessaria a causa della differenza di keyword `NEW`-`OLD`) grazie ad un controllo sul valore `NULL` e alla funzione `COALESCE()`, sfruttando la proprietà che assumono valore nullo a seconda dell'operazione svolta.

6.1.2.3.1. Inserimento di un nuovo concerto

Nella stessa transazione in cui viene aggiunto un nuovo concerto occorre includere almeno un ingaggio. Dato che le tabelle `Ingaggio_*` referenziano il concerto, il controllo da parte del trigger avviene a fine transazione.

```
CREATE CONSTRAINT TRIGGER Inserimento_Concerto
  AFTER INSERT ON Concerto
  DEFERRABLE INITIALLY DEFERRED
  FOR EACH ROW
  EXECUTE FUNCTION Check_Ingaggio_Concerto_trigger();
```

6.1.2.3.2. Cancellazione di un ingaggio

Un ingaggio può essere eliminato sole se non è l'unico per il concerto a cui fa riferimento. Il controllo da parte del trigger avviene a fine transazione.

```
CREATE CONSTRAINT TRIGGER Cancellazione_Ingaggio_Artista
  AFTER DELETE ON Ingaggio_Artista
  DEFERRABLE INITIALLY DEFERRED
  FOR EACH ROW
  EXECUTE FUNCTION Check_Ingaggio_Concerto_trigger();
CREATE CONSTRAINT TRIGGER Cancellazione_Ingaggio_Gruppo
  AFTER DELETE ON Ingaggio_Gruppo
  DEFERRABLE INITIALLY DEFERRED
  FOR EACH ROW
  EXECUTE FUNCTION Check_Ingaggio_Concerto_trigger();
```

6.2. Popolamento della base di dati

Per il popolamento della base di dati, non avendo a disposizione dei dati reali da utilizzare, si è scelto di creare dati fittizi, ma plausibili, usando uno script in Golang.

Dopo un primo approccio non soddisfacente usando il tool online *Mockaroo*, per via di problemi nel mantenimento di vincoli tra i dati delle diverse tabelle, si è preferito scrivere un programma apposito. Il linguaggio Python si è dimostrato problematico per via della quantità dei dati da generare, quindi la scelta è ricaduta su un linguaggio compilato.

Per aiutarsi con la creazione dei dati, oltre alle librerie standard di Golang si è utilizzata la libreria **gofakeit**[2] che fornisce determinate funzioni per la creazione di dati finti come nomi, cognomi, città e luoghi.

Lo script in Golang, reperibile alla seguente pagina GitHub: ProgettoBasiDiDati genera un file `csv` per ogni tabella della base di dati assieme ad un file `SQL`, e uno in `PSQL`, per effettuarne l'inserimento in blocco tramite lo statement `COPY`. È stato deciso per questa opzione invece di inserirli durante l'esecuzione, per avere gli stessi dati tra tutti i membri del gruppo e permettere di lavorare anche a chi non ha installato Golang.

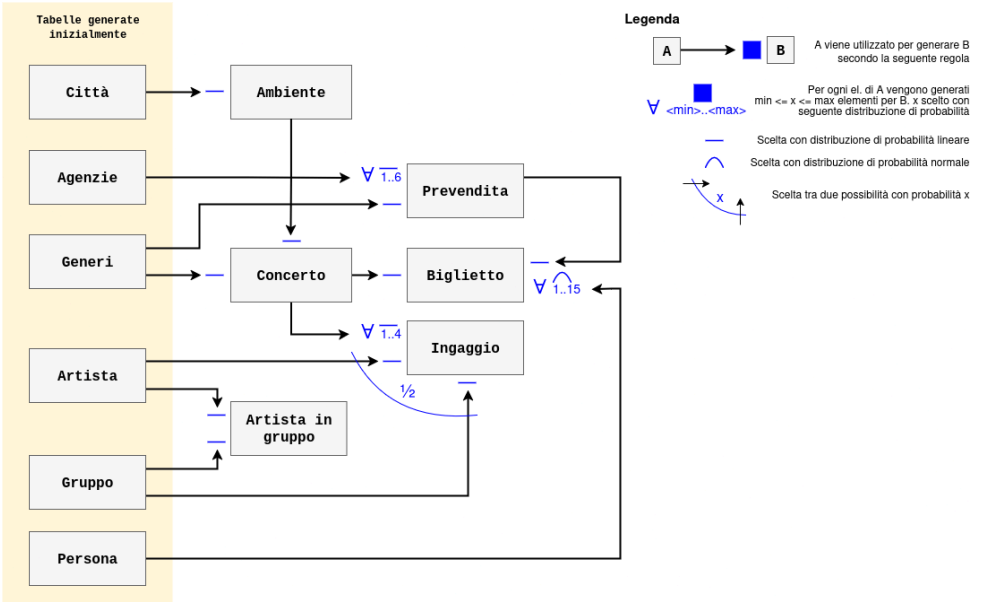


Figure 4: *Schematizzazione del processo di generazione dei dati.*

[2] Random fake data generator written in go: https://github.com/brianvoe/gofakeit

6.3. Operazioni implementate - Query SQL

Una volta completata la popolazione della base di dati, sono state scritte delle query di esempio per dimostrare le funzionalità della base di dati.

6.3.1. Inserimento

1. Aggiunta di un concerto

```
INSERT INTO Concerto (
  nome, numero_posti, prezzo, data, ora, nome_ambiente,
  indirizzo, nome_citta, cap_citta, genere
) VALUES (
  'J.Fisher - The return', 100, 20.00, '2023-10-15', '20:00:00', 'us Arena',
  '60242 Plainschester, Norfolk, Texas 72264', 'Plano', '18016', 'Pop'
);

INSERT INTO Ingaggio_Artista (
  data_concerto, ora_concerto, nome_ambiente, indirizzo_ambiente,
  nome_citta, cap_citta, artista
) VALUES (
  '2023-10-15', '20:00:00', 'us Arena', '60242 Plainschester, Norfolk, Texas 72264',
  'Plano', '18016', 'Jaquelin Fisher'
);
```

La query dimostra l'inserimento di un nuovo concerto e la connessione del concerto all'artista.

2. Acquisto di un biglietto

```
INSERT INTO Persona (
  CF, nome, cognome, email, indirizzo,
  metodo_pagamento
) VALUES (
  'ABCDEF12G34H567I', 'Mario', 'Rossi', 'mario.rossi@gmail.com', 'Via Udine 1,
Milano',
  '5338 5689 5214 4568'
);

INSERT INTO Biglietto (
  proprietario, venditore, data_vendita, ora_vendita, data_concerto,
  ora_concerto, nome_ambiente, indirizzo_ambiente, nome_citta, cap_citta
) VALUES (
  'ABCDEF12G34H567I', 'Equifax SpA', '2023-10-01', '10:00:00', '2023-10-15',
  '20:00:00', 'us Arena', '60242 Plainschester, Norfolk, Texas 72264', 'Plano',
'18016'
);
```

La query dimostra l'inserimento di un nuovo biglietto e relativa persona (assumendo che essa non esista già).

6.3.2. Aggiornamento / cancellazione

1. Cancellazione di un biglietto

```
DELETE FROM Biglietto WHERE id = 1;
```

La query dimostra l'eliminazione di un biglietto con successo, nel caso in cui a quella persona siano associati altri bigilietti; in caso contrario la cancellazione fallisce a causa del trigger `Ultimo_Biglietto_Persona` e si rende necessario cancellare anche la persona proprietaria.

6.3.3. Query / interrogazioni

1. Stabilire il numero di biglietti venduti e disponibili

```
SELECT
  numero_biglietti_venduti,
  (numero_posti - numero_biglietti_venduti) AS biglietti_disponibili
FROM Concerto
WHERE     data = '2023-10-15'
      AND ora = '20:00:00'
      AND nome_ambiente = 'us Arena'
      AND indirizzo = '60242 Plainschester, Norfolk, Texas 72264'
      AND nome_citta = 'Plano'
      AND cap_citta = '18016';
```

La query trova il numero di biglietti venduti e disponibili per un concerto. Fa uso di un attributo ridonandate per accedere rapidamente al numero di biglietti venduti.

2. Stabilire quanti biglietti sono stati venduti da una specifica agenzia per un concerto

```
SELECT COUNT(*) AS biglietti_venduti
FROM Biglietto
WHERE     venditore = 'Equifax SpA'
      AND data_concerto = '2023-10-15'
      AND ora_concerto = '20:00:00'
      AND nome_ambiente = 'us Arena'
      AND indirizzo_ambiente = '60242 Plainschester, Norfolk, Texas 72264'
      AND nome_citta = 'Plano'
      AND cap_citta = '18016';
```

La query trova il numero di biglietti venduti da un agenzia per uno specifico concerto.

3. Percentuale di agenzie che hanno effettivamente venduto almeno un biglietto per un dato concerto

```
SELECT COUNT(DISTINCT b.venditore) / COUNT(DISTINCT dp.agenzia) * 100 AS
percentuale_agenzie_attive
FROM Diritto_Prevendita dp
JOIN Concerto c ON dp.genere = c.genere
LEFT JOIN Biglietto b ON
  b.data_concerto = c.data AND
  b.ora_concerto = c.ora AND
  b.nome_ambiente = c.nome_ambiente AND
  b.indirizzo_ambiente = c.indirizzo AND
  b.nome_citta = c.nome_citta AND
  b.cap_citta = c.cap_citta AND
  b.venditore = dp.agenzia
WHERE
  c.data = '2020-08-28' AND
  c.ora = '22:00:00' AND
  c.nome_ambiente = 'world Arena' AND
  c.indirizzo = '1529 Ovalchester, San Jose, New Mexico 79549' AND
  c.nome_citta = 'St. Paul' AND
  c.cap_citta = '86216';
```

La query, fissato un concerto, trova la percentuale di agenzie (con diritto di prevendita disponibile) che hanno effettivamente venduto almeno un biglietto per un dato concerto.

4. Lista delle città in cui si è esibito un artista

```
SELECT DISTINCT
  c.nome_citta,
  c.cap_citta
FROM Concerto c
-- Esibizioni da solista
WHERE EXISTS (
  SELECT *
  FROM Ingaggio_Artista ia
  WHERE
    ia.artista = 'NomeArtista' AND
    ia.data_concerto = c.data AND
    ia.ora_concerto = c.ora AND
    ia.nome_ambiente = c.nome_ambiente AND
    ia.indirizzo_ambiente = c.indirizzo AND
    ia.nome_citta = c.nome_citta AND
    ia.cap_citta = c.cap_citta
)
-- Oppure esibizioni tramite un gruppo
OR EXISTS (
  SELECT *
  FROM Ingaggio_Gruppo ig
  JOIN Artista_In_Gruppo aig ON ig.gruppo = aig.gruppo
  WHERE
    aig.artista = 'NomeArtista' AND
    ig.data_concerto = c.data AND
    ig.ora_concerto = c.ora AND
    ig.nome_ambiente = c.nome_ambiente AND
    ig.indirizzo_ambiente = c.indirizzo AND
    ig.nome_citta = c.nome_citta AND
    ig.cap_citta = c.cap_citta
);
```

La query trova le città in cui si è esibito un artista da solista o da componente di un gruppo.

7. ANALISI DEI DATI

Sono state scelte delle domande che si potrebbe porre l'azienda proprietaria della piattaforma, anche da un punto di vista di Business Intelligence. Per ogni domanda scelta è stata sviluppata una query SQL per ottenere i dati nel formato pronto, tramite l'utilizzo del linguaggio R, ad essere trasformati in grafici.

I dati utilizzati per riempire il database, essendo stati generati artificialmente, possono originare grafici con pattern non completamente realistici. Se ne effettua comunque una descrizione per mostrare le potenzialità.

7.1. Quale è la distribuzione per genere dell'adesione percentuale ai concerti?

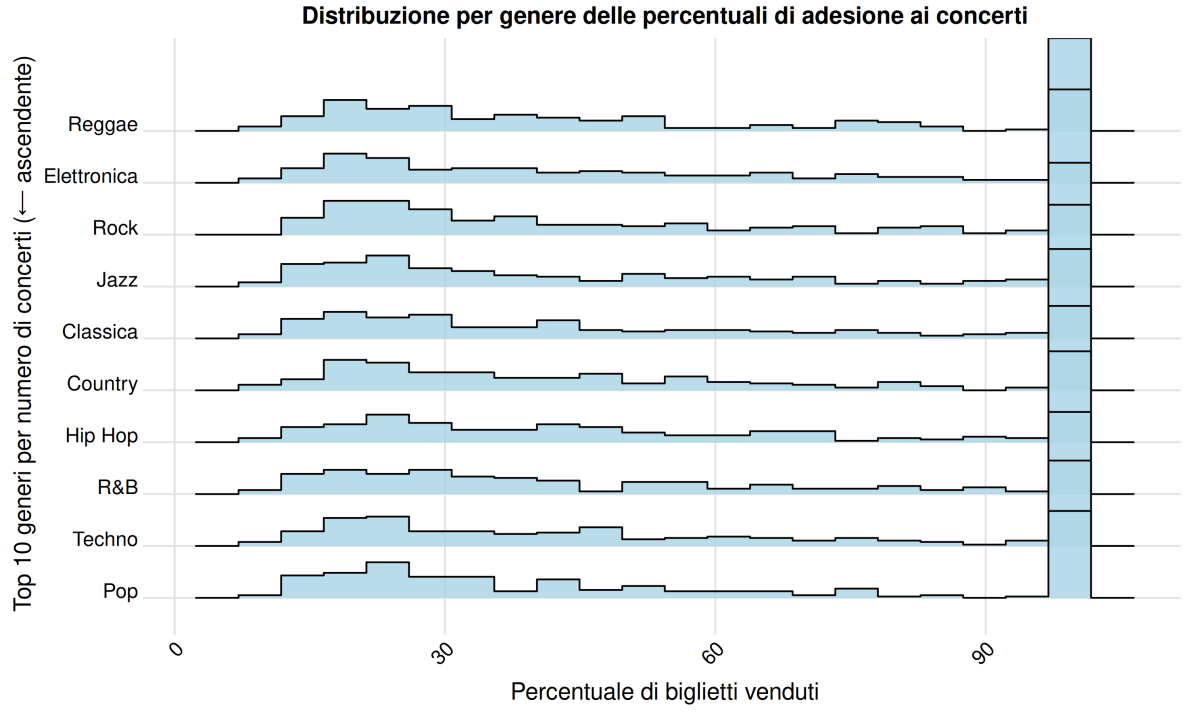
Presi i 10 generi per i quali si sono tenuti più concerti, e calcolato il numero di biglietti venduti sul totale dei posti disponibili (adesione percentuale), che forma ha la distribuzione delle percentuali di adesione per ogni concerto? C'è una differenza tra i vari generi? Ci sono caratteristiche comuni?

Query sviluppata

```
WITH percentuali_concerto AS (
    SELECT genere,
           nome AS concerto,
        (numero_biglietti_venduti::DECIMAL / numero_posti) * 100 AS percentuale_vendita
    FROM Concerto
    GROUP BY genere, nome, numero_biglietti_venduti, numero_posti
),
top_generi AS (
    SELECT genere
    FROM percentuali_concerto
    GROUP BY genere
    ORDER BY COUNT(*) DESC
    LIMIT 9
)

SELECT pc.genere,
       pc.concerto,
       pc.percentuale_vendita
FROM percentuali_concerto pc
JOIN top_generi tg
  ON pc.genere = tg.genere
ORDER BY pc.genere, pc.concerto;
```

Grafico



Per tutti i generi, le distribuzioni assumo forma bimodale. La forte concentrazione verso l'estremo destro (90–100%) segna la moda prevalente, ed è segno che molti concerti raggiungono una vendita quasi totale dei biglietti. La seconda moda (locale) alle percentuali basse indica la presenza di concerti con una partecipazione scarsa.

A parte i concerti che fanno sold-out o quasi, gli altri hanno spesso una percentuale di adesione nel range 10–30%. Questo riflette probabilmente l'elevata eterogeneità dell'offerta (dai grandi eventi alle esibizioni minori).

È possibile studiare e comparare le distribuzioni numericamente grazie alle apposite metriche. Visivamente si può notare come i generi Hip Hop, Jazz e Rock presentano code con valori più alti rispetto alla media.

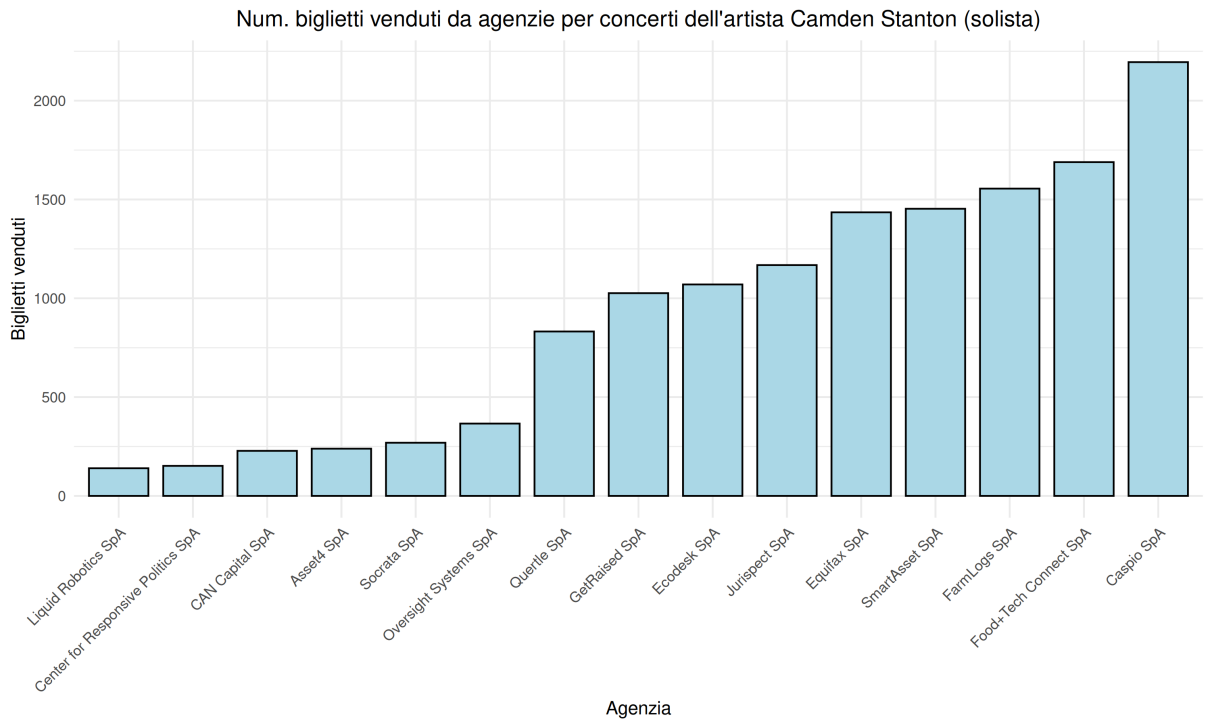
7.2. Dato uno specifico artista, quale agenzia ha venduto più biglietti?

Viene fissato un artista (come solista) ed aggregati i dati di tutti i suoi concerti.

Query sviluppata

```
SELECT b.venditore AS agenzia,
       COUNT(*) AS biglietti_venduti
FROM Biglietto b
JOIN Ingaggio_Artista ia
  ON b.data_concerto = ia.data_concerto
 AND b.ora_concerto = ia.ora_concerto
 AND b.nome_ambiente = ia.nome_ambiente
 AND b.indirizzo_ambiente = ia.indirizzo_ambiente
 AND b.nome_citta = ia.nome_citta
 AND b.cap_citta = ia.cap_citta
JOIN Artista a
  ON ia.artista = a.nome_arte
WHERE a.nome_arte = 'Camden Stanton'
GROUP BY b.venditore
ORDER BY biglietti_venduti DESC;
```

Grafico



La presenza di un leader chiaramente dominante e di un gradino marcato tra fascia alta, media e bassa indica un mercato poco uniforme: potere distributivo concentrato, con probabilmente accordi o canali preferenziali per le prime agenzie.

Questo visualizzazione permette anche di scoprire agenzie interessanti dal punto di vista dell'artista, sia per la creazione di accordi che per l'individuazione di comportamenti scorretti da parte delle agenzie.

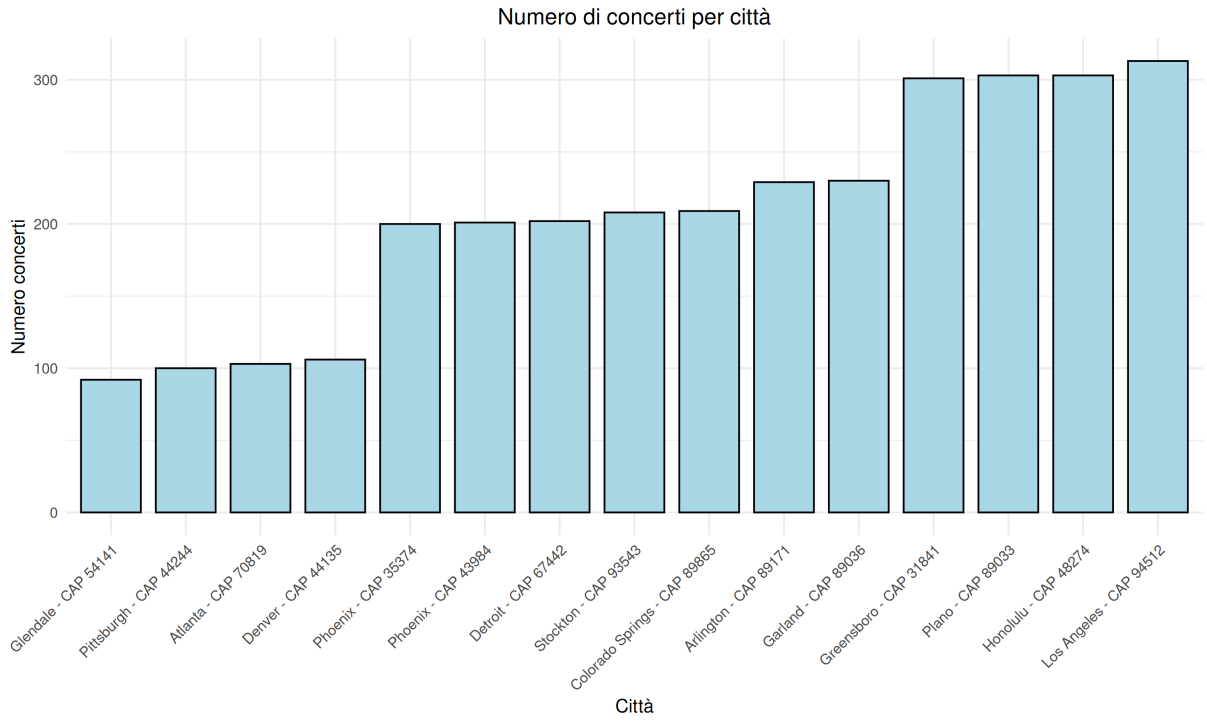
7.3. Come sono distribuiti i concerti per città?

Prese le 15 città che hanno ospitato più concerti, come sono distribuiti i suddetti concerti?

Query sviluppata

```
SELECT nome_citta,
       cap_citta,
       COUNT(*) AS numero_concerti
FROM Concerto
GROUP BY nome_citta, cap_citta
ORDER BY numero_concerti DESC;
```

Grafico



Il grafico mostra che sono 4 città principali che ospitano concerti: Los Angeles, Honolulu, Plano e Greensboro. Tutte superano quota 300, circa 3.3 volte il valore minimo (Glendale con 90 concerti). La distribuzione è a gradini, con una fascia bassa (90–110 concerti), fascia centrale (200–230 concerti) e fascia alta (300–310 concerti). Ci sono dei salti netti tra le varie fasce.
Tramite un'analisi più approfondita si possono scovare eventuali cause particolari, oltre alle popolarità dei luoghi.

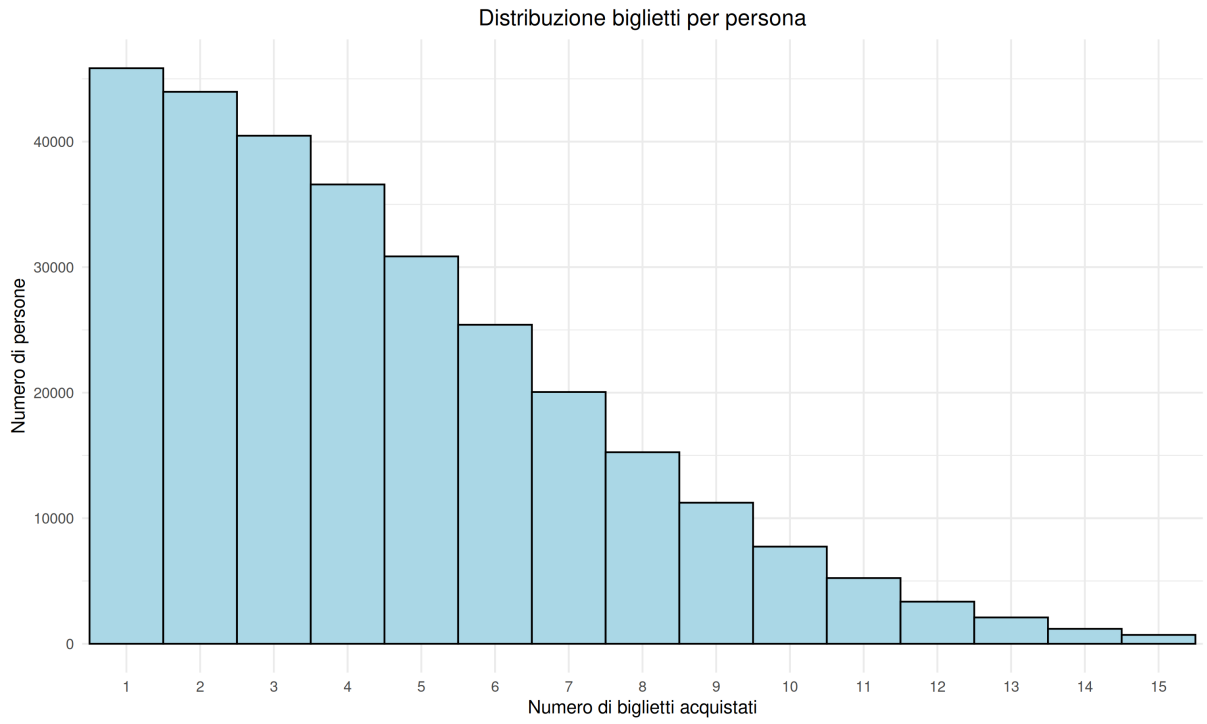
7.4. Come sono distribuiti i biglietti per persona?

Quanti biglietti tendono a venir acquistati da una singola persona?

Query sviluppata

```sql
-- 1. Contiamo quanti biglietti ha comprato ogni persona
WITH biglietti_per_persona AS (
    SELECT proprietario,
           COUNT(*) AS num_biglietti
    FROM Biglietto
    GROUP BY proprietario
)
-- 2. Calcoliamo quante persone hanno comprato esattamente quel numero di biglietti
SELECT num_biglietti,
       COUNT(*) AS numero_persone
FROM biglietti_per_persona
GROUP BY num_biglietti
ORDER BY num_biglietti;
```

Grafico



Il numero di persone decresce monotonicamente all'aumentare dei biglietti acquistati. La maggior parte delle persone acquista 1 biglietto, seguite da vicino con chi ne acquista 2 o 3, indicando che la grande maggioranza degli utenti effettua acquisti singoli o di pochi biglietti.

La diminuzione appare quasi esponenziale: ogni biglietto aggiuntivo riduce sensibilmente la numerosità del gruppo corrispondente, senza presenza di picchi secondari (segmenti distinti di "grandi acquirenti"). La coda oltre 12 biglietti diventa sottile estendendosi fino a 15, suggerendo che un piccolo numero di utenti sono assidui partecipanti di concerti.