Modello Fisico in MySQL: Esercizi con MySQL

Francesco Gobbi

I.I.S.S. Galileo Galilei, Ostiglia

4 giugno 2025

Modello Logico della Realtà

Schema Relazionale

- Cliente(id_cliente, nome, cognome, città)
- Prodotto(id_prodotto, nome, prezzo)
- Ordine(id_ordine, data_ordine, id_cliente)
- Ordine_Item(id_ordine_item, id_ordine, id_prodotto, quantita)

Descrizione

Gestione di un sistema di e-commerce con clienti che effettuano ordini composti da più prodotti.

Esercizio 1: Elenco Ordini con Cliente

Testo: Recuperare la lista di tutti gli ordini indicando per ciascuno: *id_ordine*, data ordine, nome e cognome del cliente.

Soluzione MySQL:

```sql
SELECT o.id_ordine, o.data_ordine, c.nome, c.cognome
FROM Ordine AS o, Cliente AS c
WHERE o.id_cliente = c.id_cliente;
```

Listing 1: Ordini con dati cliente usando equi-join

Spiegazione: Uso dell'equi-join nel `WHERE` tra le tabelle `Ordine` (alias o) e `Cliente` (alias c).

Esercizio 2: Top 3 Clienti per Spesa

Testo: Trovare i primi 3 clienti che hanno speso di più, calcolando per ciascuno la spesa totale. Considerare solo i clienti con spesa superiore a 500€.

Soluzione MySQL:

```
1 SELECT c.nome, c.cognome, SUM(p.prezzo * i.quantita)
      AS spesa_totale
2 FROM Cliente AS c, Ordine AS o, Ordine_Item AS i,
      Prodotto AS p
3 WHERE c.id_cliente = o.id_cliente AND o.id_ordine = i.
      id_ordine AND i.id_prodotto = p.id_prodotto
4 GROUP BY c.id_cliente
5 HAVING spesa_totale > 500
6 ORDER BY spesa_totale DESC
7 LIMIT 3;
```

Listing 2: Top 3 clienti per spesa usando equi-join

Esercizio 2: Top 3 Clienti per Spesa

Spiegazione:

- Uso dell'equi-join per collegare le varie tabelle coinvolte: `Cliente`, `Ordine`, `Ordine_Item`, `Prodotto`.
- `GROUP BY` raggruppa le righe per cliente.
- `SUM()` calcola la spesa totale come prodotto prezzo-quantità.
- `HAVING` filtra i clienti con spesa $>$ 500€.
- `ORDER BY ... DESC LIMIT 3` restituisce le prime 3 tuple, quindi i primi 3 clienti ordinati in ordine decrescente per quanto riguarda la spesa eseguita nell'ordine.

Esercizio 3: Numero di Prodotti per Ordine

Testo: Per ogni ordine, indicare il numero totale di prodotti presenti.

Soluzione MySQL:

```
SELECT o.id_ordine, COUNT(i.id_prodotto) AS
    numero_prodotti
FROM Ordine AS o, Ordine_Item AS i
WHERE o.id_ordine = i.id_ordine
GROUP BY o.id_ordine;
```

Listing 3: Conteggio prodotti per ordine con equi-join

Spiegazione: Uso dell'equi-join nel `WHERE`. Utilizzo di `COUNT()` e `GROUP BY` per contarei prodotti per ciascun ordine.

Esercizio 4: Prodotti Mai Ordinati

Testo: Elencare i prodotti presenti nel catalogo che non sono mai stati ordinati.

Soluzione MySQL:

```sql
SELECT nome
FROM Prodotto AS p
WHERE NOT EXISTS ( SELECT 1 FROM Ordine_Item i
                   WHERE i.id_prodotto = p.id_prodotto
                   );
```

Listing 4: Prodotti mai ordinati

Spiegazione:

- Utilizzo di una subquery correlata con `NOT EXISTS`.
- Per ogni prodotto verifichiamo se non esiste alcun record in `Ordine_Item`, così possiamo trovare i prodotti mai ordinati.

Esercizio 5: Clienti nella Stessa Città

Testo: Individuare i clienti che condividono la stessa città con almeno un altro cliente.

Soluzione MySQL:

```
1 SELECT DISTINCT c1.nome, c1.cognome, c1.citta
2 FROM Cliente AS c1, Cliente AS c2
3 WHERE c1.citta = c2.citta AND
4         c1.id_cliente <> c2.id_cliente;
```

Listing 5: Self-join clienti stessa città

Spiegazione:

- Utilizzo di Equi-join, in questo caso self-join, tra due tabelle uguali.
- La condizione `c1.id_cliente <> c2.id_cliente` evita l'abbinamento con sé stessi.
- La clausola `DISTINCT` rimuove duplicati.

Esercizio 6: Paginazione degli Ordini

Testo: Visualizzare la seconda pagina di ordini, 5 record per pagina, ordinati per data.

Soluzione MySQL:

```sql
SELECT id_ordine, data_ordine
FROM Ordine
ORDER BY data_ordine
LIMIT 5 OFFSET 5;
```

Listing 6: Paginazione ordini

Spiegazione:

- `LIMIT 5 OFFSET 5` salta i primi 5 record e ne restituisce i successivi 5. Il comando `OFFSET 5` fa lo scostamento dei primi 5, da non considerare.

Esercizio 7: Andamento Mensile delle Vendite

Testo: Calcolare per ciascun mese dell'anno in corso il totale delle vendite.

Soluzione MySQL:

```sql
WITH ordini_mese AS (
  SELECT DATE_FORMAT(data_ordine, '%Y-%m') AS mese,
      SUM(p.prezzo * i.quantita) AS fatturato
  FROM Ordine AS o, Ordine_Item AS i, Prodotto AS p
  WHERE o.id_ordine = i.id_ordine AND i.id_prodotto =
      p.id_prodotto
    AND YEAR(o.data_ordine) = YEAR(CURDATE())
  GROUP BY mese
)
SELECT mese, fatturato
FROM ordini_mese
ORDER BY mese;
```

Listing 7: Vendite mensili con CTE

Spiegazione: DATE_FORMAT e filtro anno corrente.

CTE: Common Table Expression

Cos'è una CTE?

Una **CTE** (Common Table Expression) è una subquery temporanea con un nome, definita tramite `WITH`, che può essere richiamata subito dopo nella query principale.

Vantaggi delle CTE:

- Migliora la leggibilità del codice SQL, per quanto si potrebbe fare tutto nella medesima query.
- Permette di suddividere query complesse in blocchi logici.
- Può essere riutilizzata più volte nella stessa query.

Nel nostro esercizio:

- La CTE `ordini_mese` calcola il fatturato per ciascun mese dell'anno corrente.
- La query principale seleziona i dati aggregati ordinandoli per mese.

Esercizio 8: Clienti Top dell'anno

Testo: Trovare i clienti che hanno speso più di 1000€ nell'anno in corso.

Soluzione MySQL con CTE:

```
1  WITH spesa_clienti AS (
2    SELECT c.id_cliente, c.nome, c.cognome, SUM(p.prezzo
         * i.quantita) AS totale_speso
3    FROM Cliente AS c, Ordine AS o, Ordine_Item AS i,
       Prodotto AS p
4    WHERE c.id_cliente = o.id_cliente AND o.id_ordine =
       i.id_ordine AND i.id_prodotto = p.id_prodotto
       AND YEAR(o.data_ordine) = YEAR(CURDATE())
5    GROUP BY c.id_cliente, c.nome, c.cognome
6  )
7  SELECT *
8  FROM spesa_clienti
9  WHERE totale_speso > 1000
10 ORDER BY totale_speso DESC;
```

Listing 8: Clienti top con spesa annua

Spiegazione: La CTE calcola la spesa totale per ciascun cliente

Esercizio 9: Prodotti mai ordinati

Testo: Elencare tutti i prodotti che non sono mai stati ordinati.

Soluzione MySQL con CTE:

```sql
WITH prodotti_ordinati AS (
  SELECT DISTINCT id_prodotto
  FROM Ordine_Item
)
SELECT p.id_prodotto, p.nome, p.prezzo
FROM Prodotto AS p
LEFT JOIN prodotti_ordinati AS po ON p.id_prodotto =
    po.id_prodotto
WHERE po.id_prodotto IS NULL;
```

Listing 9: Prodotti mai ordinati

N.B.: In questo caso è necessario l'uso del `LEFT JOIN` per trovare i prodotti esclusi.

Esercizio 10: Prodotti ordinati solo da clienti di Verona

Testo: Trovare i prodotti che sono stati ordinati **almeno una volta** e **mai da clienti non di Verona**.

Soluzione MySQL con sottoquery nel WHERE

```
1 SELECT DISTINCT p.id_prodotto, p.nome
2 FROM Prodotto AS p, Ordine_Item AS i, Ordine AS o,
      Cliente AS c
3 WHERE p.id_prodotto = i.id_prodotto AND i.id_ordine =
      o.id_ordine AND o.id_cliente = c.id_cliente AND p.
      id_prodotto NOT IN (
4         SELECT i2.id_prodotto
5         FROM Ordine_Item AS i2, Ordine AS o2, Cliente
              AS c2
6         WHERE i2.id_ordine = o2.id_ordine AND o2.
              id_cliente = c2.id_cliente AND c2.citt
              <> 'Verona'
7   );
```

Listing 10: Prodotti ordinati solo da clienti veronesi

Spiegazione: La sottoquery seleziona i prodotti ordinati da clienti di altre città; la query esterna li esclude.