Progettazione Data warehouse (SCHEMA)

Carlo Migliaccio

Novembre 2023

1 Progettazione concettuale (Dimensional Fact Model)

1. Identificazione del **fatto** (o dei fatti) di interesse

   → misure (attenzione all'additività!)

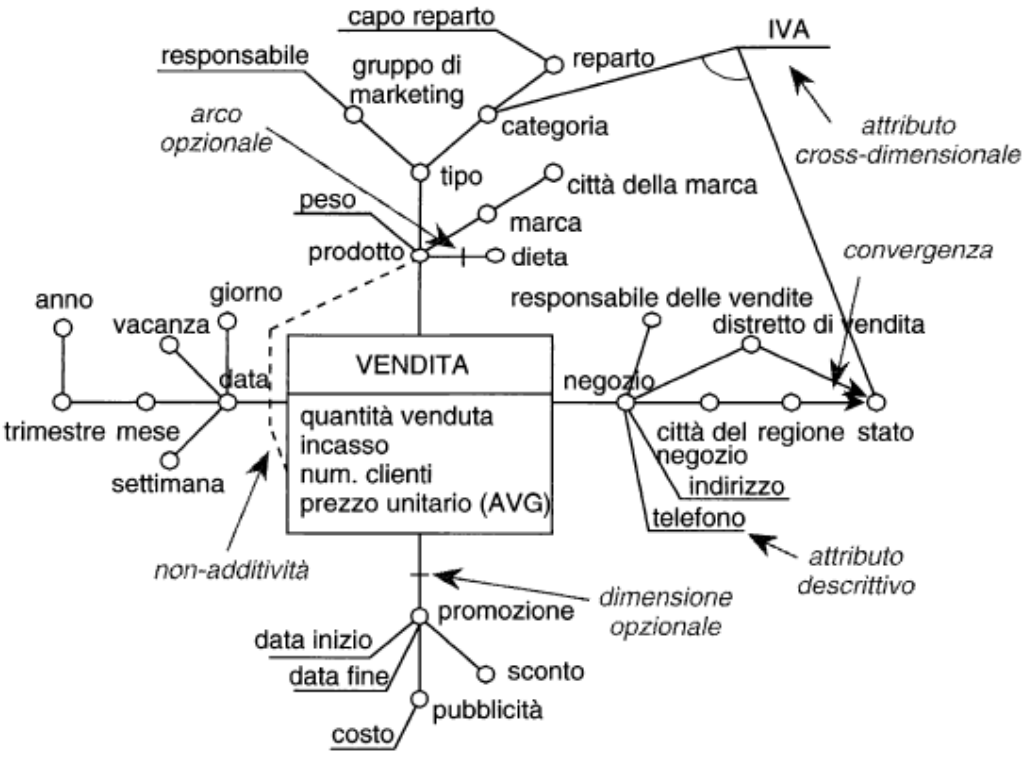
2. Identificazione delle **dimensioni di analisi**

   → definisce la granularità del fatto che spesso non è la stessa della fonte primaria di dati (OLTP ad esempio). Ad esempio:

   Data singola (OLTP) → Mese (OLAP)

3. **gerarchie** (verificare la relazione 1:n). Prestare attenzione a:

   - *archi multipli/attributi configurazione* (valore limite per la scelta: 10)
   - *gerarchia condivisa*, non sdoppiare gerarchie che potrebbero essere condivise
   - *punti di convergenza*
   - *opzionalità delle dimensioni, degli attributi*



Esempio di DFM completo

2 Progettazione logica (Schema a stella)

1. Per ogni dimensione si crea una `tabella dimensionale` che contenga tutti gli attributi della gerarchia (qui si appiattiscono tutti i livelli di gerarchia), la chiave primaria è una `chiave surrogata` (autogenerata) che non ha nessun significato ai fini dell'analisi.

   Casi particolari:

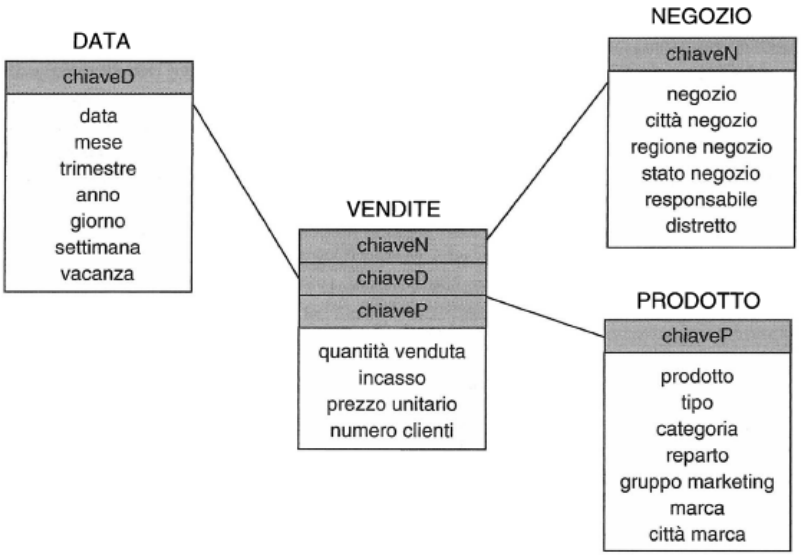
   - *arco multiplo*:
     - *Tabella BRIDGE* (eventualmente con Peso: pensa alla percentuale di contributo nella scrittura di un libro);
     - *push down* nella tabella dei fatti (da evitare)
     - *attributo configurazione*, se il dominio dell'attributo è limitato ad essere ≤ 10
   - *dimensioni degeneri* (dimensioni con un solo attributo):
     - *tabella dimensionale* che abbia solo quell'attributo oltre la chiave surrogata;
     - *push down* nella tabella dei fatti $\rightarrow$ l'attributo entra a far parte della chiave primaria;
     - *junk dimension* che inglobi tutte le dimensioni degeneri (Attenzione alle cardinalità dei domini delle singole dimensioni degeneri, dal momento che la junk dimension contiene in modo combinatorio i valori delle dimensioni);
     - *soluzioni miste* che stiano a metà delle precedenti, ad esempio: alcune le metto nella tabella dei fatti mentre per altre posso scegliere un push down o una dimension table.

2. Il fatto diventa la `tabella dei fatti` (centro stella), è quella che ha dimensione maggiore di tutto lo schema a stella. Lo schema della tabella dei fatti è costituito da:

   - Le **chiavi delle dimensional table** che messe insieme costituiscono la chiave primaria;
   - Le **misure** che diventano attributi.

NOTE:

- le dimensioni di analisi condivise non vanno duplicate!
- è bene analizzare le query principali prima di congelare lo schema logico derivante da questa fase
- Gli attributi temporali sono da intendersi come:
  - Data $\Rightarrow$ GG/MM/AAAA
  - Mese $\Rightarrow$ MM-AAAA
  - Semestre $\Rightarrow$ S-AAAA, dove S è 1 o 2



Schema a stella

3 Query in SQL esteso

1. Analisi attributi nella clausola `SELECT`;
2. Costruzione del corpo della query con:
   - `FROM` ⇒ scelta delle tabelle
   - `WHERE` ⇒ condizioni di join e predicati di selezione
   - Attributi di `GROUP BY`. **Nota che...** se nelle `PARTITION BY` ci sono attributi non previsti nella `GROUP BY`, questi vi vanno inclusi, se non si può a causa di assenza di dipendenza funzionale del tipo A→B (A determina B), c'è un problema di formulazione della query
   - Funzioni aggregate nella `SELECT`

NOTA CHE...

1. L'operatore di **media aritmetica** è particolare: non sempre `AVG(Attributo)` produce un risultato che sia semanticamente corretto. **Esempio** Se mi viene chiesto di calcolare una media giornaliera e nella partizione individuata dalla `GROUP BY` nonsono sicuro di avere una sola tupla per ogni giorno, allora AVG() non produce il risultato desiderato! Le soluzioni possibili a questo punto sono tre:
   (a) Applico la definizione di media aritmetica: `SUM(Attributo)/COUNT(DISTINCT Data)`
   (b) uso una table function "customizzata" nella FROM
   (c) uso `OVER` ricordandomi di mettere il `DISTINCT`
2. Nelle `WINDOW` nel caso di calcolo di cumulativi, Top N, medie mobili e rank, l'`ORDER BY` è obbligatorio mentre non lo è per altri aggregati (es. somma);
3. Le finestre di calcolo definite dopo la keyword `OVER` possono avere un'"apertura" più ampia rispetto a quella definita dalla `GROUP BY`. Si ricordi inoltre che nel caso di compresenza di OVER e GROUP BY, la OVER viene applicata al risultato della GB.

4 Viste materializzate

```
CREATE MATERIALIZED VIEW NomeVista
BUILD [IMMEDIATE | DEFERRED]
REFRESH [COMPLETE | FAST] [ON COMMIT | ON DEMAND ]
ENABLE QUERY REWRITE
AS QueryVista
```

- Osservare il **carico di lavoro (query)**, analizzando:
  - *predicati di selezione*, non applico alla vista i predicati di selezione
  - *attributi di Group By*
  - *funzioni aggregate*
- Determinare la vista che rappresenta il **denominatore comune**. Il fattore di riduzione deve essere almeno $\geq 10^3$, altrimenti non ha senso creare una vista materializzata
- Individuare correttamente la granularità della vista (attributo "meno profondo" in ogni gerarchia) ⇒ rappresenta un vero e proprio **identificatore** della vista.

4.1 Manutenzione di viste tramite TRIGGER

Talvolta il prodotto DBMS che si utilizza potrebbe non avere a disposizione il comando `CREATE MATERIALIZED VIEW` e di conseguenza la gestione del build e refresh delle viste materializzate. In questo caso le viste diventano vere e proprie tabelle definite con l'istruzione `CREATE TABLE`, ma la loro gestione deve essere affidata alla presenza di uno o più **trigger** che ne gestiscano modifiche eventuali.

I trigger che tratteremo avranno sempre la stessa semantica e in particolare:

- modo di esecuzione: `AFTER`. Il trigger viene eseguito appena **dopo** l'esecuzione dell'evento innescante (triggering event)
- granularità: `FOR EACH ROW`. Il trigger è eseguito una volta per ogni riga modificata dall'event innescante).

Gli altri due aspetti semantici possibili (BEFORE e FOR EACH STATEMENT) hanno alle spalle meccanismi spinosi che coinvolgono le proprietà ACID delle transazioni.

Operazioni da svolgere:

1. Lettura dei valori degli attributi necessari dalle dimensioni
2. Verifica esistenza tupla nella vista (nel caso di operazioni di tipo `INSERT`)
3. IF (tupla esiste) THEN ... `UPDATE` ELSE `INSERT`

4.1.1 Creazione di `MATERIALIZED VIEW LOG`

Nel caso si scelga di scegliere per la vista materializzata un REFRESH di tipo FAST, c'è bisogno di creare prima della vista dei log materializzati che tengano traccia delle modifiche delle tabelle coinvolte dalla vista. Di seguito si riporta la sintassi di ORACLE:

```
CREATE MATERIALIZED VIEW LOG ON  Tabella
WITH SEQUENCE, ROWID
(Id..., Campo1, Campo2, ..., CampoN)
INCLUDING NEW VALUES;
```

Lo schema di **Tabella** deve coinvolgere attributi che sono stati materializzati nella vista.

4.2 Note conclusive

1. Nella stesura del Trigger attenzione a non dimenticare la clausola `WHERE` nelle UPDATE, altrimenti si richia di modificare oltre alle tuple interessate anche quelle che prima erano a posto!
2. Nella vista vanno memorizzate **funzioni aggregate distributive** (SUM, COUNT, MIN, MAX) in modo che posso sempre ricostruire a livelli diversi di profondità altri operatori che distributivi non sono (es. AVG);
3. Normalmente, ad esclusione di situazioni particolari, non vanno messi predicati di selezione nella vista $\rightarrow$ la maggior parte delle volte questo può portare a perdita di generalità