Sistemi Informativi Avanzati
Anno Accademico 2015/2016
Prof. Domenico Beneventano

Progettazione Logica

Dal Capitolo 8 e 9 del libro

Data Warehouse - teoria e pratica della Progettazione
Autori: Matteo Golfarelli, Stefano Rizzi;

Editore: McGraw-Hill

Sviluppo di un Database/DataWarehouse

Fasi dello sviluppo	**Obiettivi**	**Modelli per Database**	**Modelli per DataWarehouse**
Progettazione **Concettuale**	Rappresentazione astratta, in forma grafica, ed indipendente dalla implementazione. Schema dei dati comprensibile anche per l'utente finale.	Modello concettuale: **E/R**	Modello concettuale: **DFM**
Progettazione **Logica**	Rappresentazione strettamente legata al sistema scelto per l'implementazione. Schema dei dati utile per semplificare ed ottimizzare le operazioni di interrogazione e manipolazione dei dati.	Modello logico: **Relazionale**	Modello logico: **ROLAP**) Relazionale (**star/snowflake** schema) **MOLAP** strutture multidimensionali

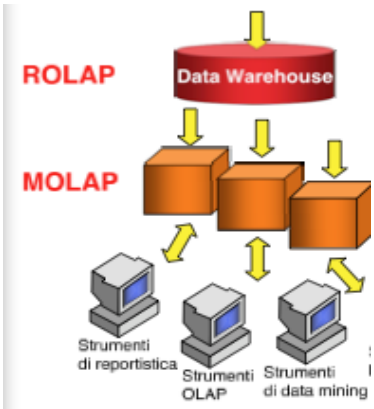
Modelli logici per il DataWarehouse

- La struttura multidimensionale dei dati può essere rappresentata utilizzando **due distinti modelli logici**:
  - ✓ **MOLAP** (*Multidimensional On-Line Analytical Processing*) memorizzano i dati utilizzando strutture intrinsecamente multidimensionali (es. vettori multidimensionali).
  - ✓ **ROLAP** (*Relational On-Line Analytical Processing*) utilizza il ben noto modello relazionale per la rappresentazione dei dati multidimensionali.

- L'utilizzo di soluzioni MOLAP:
  - ✓ È frenato dalla mancanza di strutture dati standard: i diversi sistemi usano strutture proprietarie che li rendono difficilmente sostituibili e accessibili mediante strumenti di terze parti.
  - ✓ Progettisti e sistemisti sono riluttanti a rinunciare alla loro ormai ventennale esperienza sui sistemi relazionali.

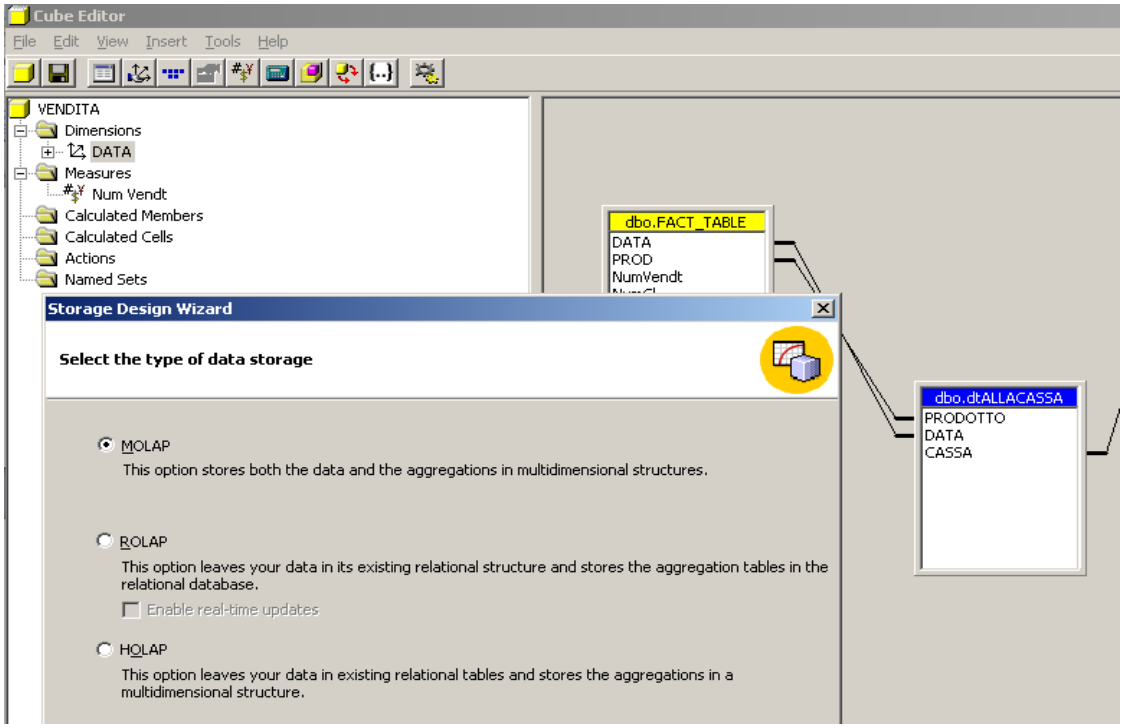
ROLAP e MOLAP

- I sistemi commerciali si differenziano in base al modello logico adottato
- Sebbene la maggior parte dei sistemi, soprattutto di grandi dimensioni, sia realizzato con soluzioni ROLAP, sono proposte anche alcune soluzioni ibride (HOLAP), che sfruttano le proprietà di entrambi i modelli:
  - ✓ Il **DW ROLAP** è ottimale per memorizzare enormi quantità di dati
  - ✓ I **DM MOLAP** massimizzano la velocità di accesso ai dati
  - ✓ I cubi MOLAP possono anche essere creati 'al volo' per svolgere specifiche sessioni di analisi



Esempio di sistema commerciale

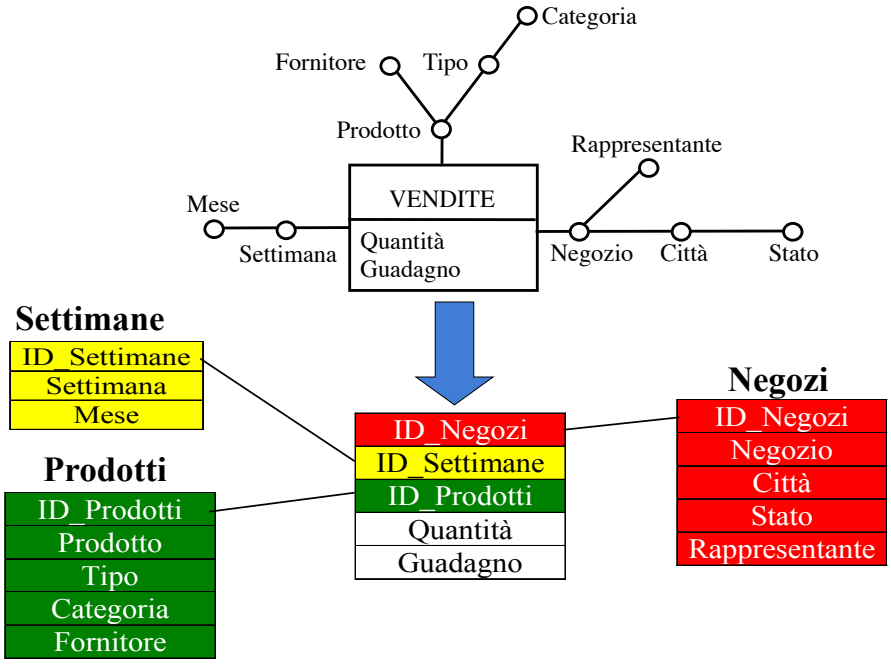
- Sql Server 2000 -Analysis Services:



ROLAP: lo schema a stella

- La modellazione multidimensionale su sistemi relazionali è basata sullo *schema a stella* (*star schema*) e sue varianti.
- Uno schema a stella è composto da:
  - ✓ Un insieme di relazioni DT_1…, DT_n, chiamate *dimension table*, ciascuna corrispondente a una dimensione.
    La DT_i corrispondente alla dimensione *Di* è caratterizzata dalla chiave primaria *Di* (in genere surrogata) e dall'insieme di attributi dimensionali della gerarchia della dimensione Di.
  - ✓ Una relazione *FT*, chiamata *fact table*, che importa le chiavi di tutte le dimension table. La chiave primaria di *FT* è data dall'insieme delle chiavi esterne dalle dimension table, d_1…, d_n (a meno di dipendenze funzionali tra le dimensioni)
    La fact table *FT* contiene gli attributi corrispondenti alle misure.
- Le Dimension Table sono completamente denormalizzate in quanto contengono tutte le dipendenze funzionali della gerarchia della dimensione

Lo schema a stella



Lo schema a stella

ID_Negozi	Negozio	Città	Stato	Rappresentante
1	N1	RM	I	R1
2	N2	RM	I	R1
3	N3	MI	I	R2
4	N4	MI	I	R2

Dimension Table

ID_Negozi	ID_Sett	ID_Prodotti	Quantità	Guadagno
1	1	1	100	100
1	2	1	150	150
3	3	4	350	350
4	4	4	200	200

Fact Table

ID_Sett.	Settimana	Mese
1	Gen1	Gen.
2	Gen2	Gen.
3	Feb1	Feb.
4	Feb2	Feb.

Dimension Table

ID_Prodotti	Prodotto	Tipo	Categoria	Fornitore
1	P1	A	X	F1
2	P2	A	X	F1
3	P3	B	X	F2
4	P4	B	X	F2

Lo snowflake schema

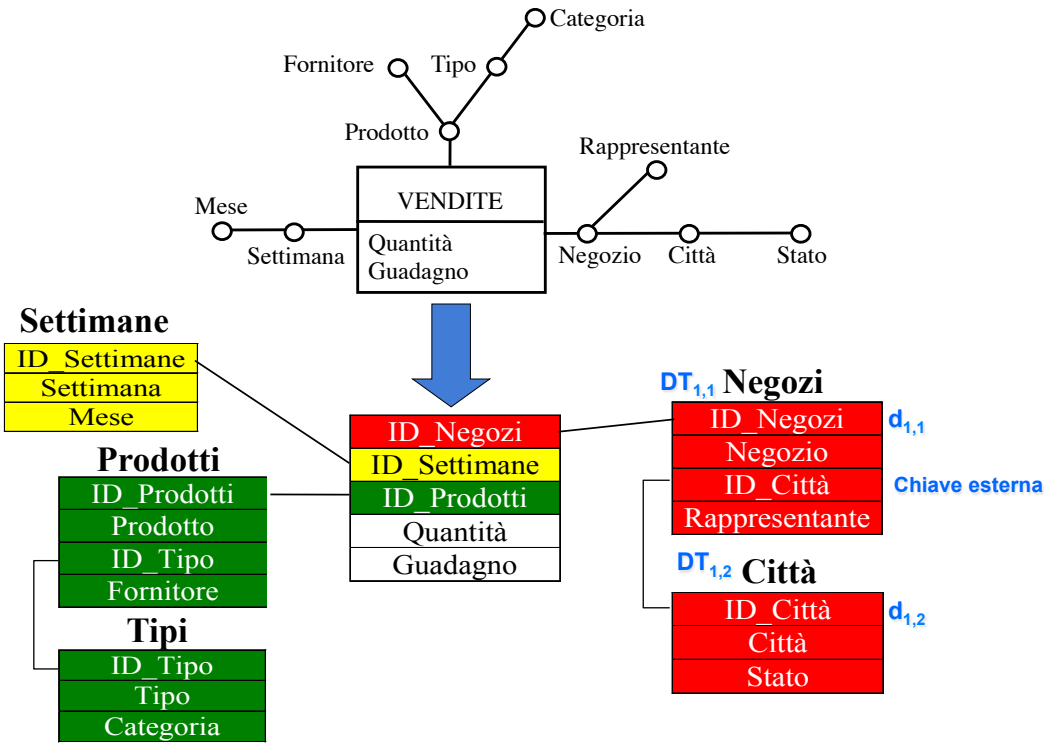
- Lo schema a fiocco di neve (*snowflake schema*) riduce la denormalizzazione delle dimension table DT_i degli schemi a stella eliminando le FD da attributi non chiave

- Lo *snowflake schema* si può ottenere
1) dallo star schema attraverso un processo di normalizzazione
2) direttamente dallo schema di fatto tramite le usuali regole di traduzione logico-relazionale

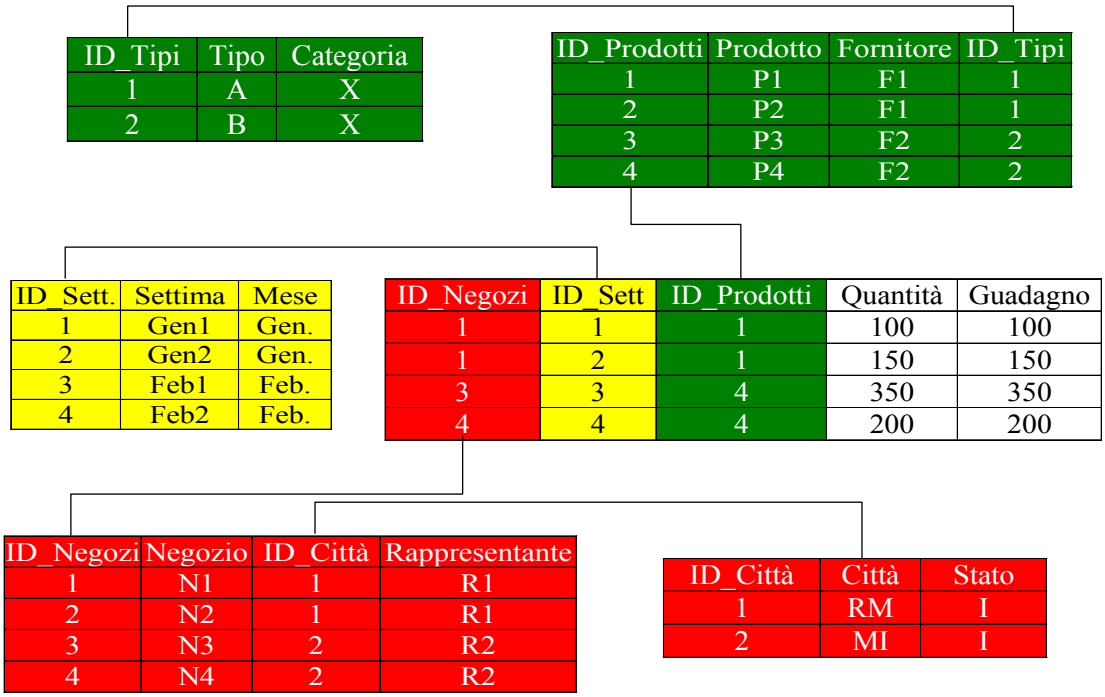
- Un arco da A a B, si traduce riportando nella Dimension Table di A, DT_A,
  - B, se B è una foglia
  - la foreign key riferita alla Dimension Table di B, DT_B, se B non è una foglia (normalmente si usano chiavi surrogate)

- Denominiamo *primarie* le dimension table le cui chiavi sono importate nella fact table, *secondarie* le rimanenti.

Lo snowflake schema



Lo snowflake schema



ID_Tipi	Tipo	Categoria
1	A	X
2	B	X

ID_Prodotti	Prodotto	Fornitore	ID_Tipi
1	P1	F1	1
2	P2	F1	1
3	P3	F2	2
4	P4	F2	2

ID_Sett.	Settima	Mese
1	Gen1	Gen.
2	Gen2	Gen.
3	Feb1	Feb.
4	Feb2	Feb.

ID_Negozi	ID_Sett	ID_Prodotti	Quantità	Guadagno
1	1	1	100	100
1	2	1	150	150
3	3	4	350	350
4	4	4	200	200

ID_Negozi	Negozio	ID_Città	Rappresentante
1	N1	1	R1
2	N2	1	R1
3	N3	2	R2
4	N4	2	R2

ID_Città	Città	Stato
1	RM	I
2	MI	I

Chiavi Surrogate

- Snowflake schema **con** chiavi surrogate

```
FT_VENDITE(IDProdotto:DT_PRODOTTO,IDNegozio:DT_NEGOZIO,
                IDSettimana:DT_SETTIMANA,Quantità,Quadagno)

    DT_SETTIMANA(IDSettimana,Settimana,Mese)
    DT_PRODOTTO(IDProdotto,Prodotto,Fornitore,IDTipo:DT_TIPO)
        DT_TIPO(IDTipo,Tipo,Categoria)
    DT_NEGOZIO(IDNegozio,NegozioRappresentante,IDCitta:DT_CITTA)
        DT_CITTA(IDCitta,Citta,Stato)
```

- Snowflake schema **senza** chiavi surrogate

```
FT_VENDITE(Prodotto:DT_PRODOTTO,Negozio:DT_NEGOZIO,
                Settimana:DT_SETTIMANA,Quantità,Quadagno)

    DT_SETTIMANA(Settimana,Mese)
    DT_PRODOTTO(Prodotto,Fornitore,Tipo:DT_TIPO)
        DT_TIPO(Tipo,Categoria)
    DT_NEGOZIO(Negozio,Rappresentante,Citta:DT_CITTA)
        DT_CITTA(Citta,Stato)
```

Chiavi Surrogate

- Senza chiavi surrogate si usa direttamente la chiave *semantica,* quale Citta*,* Negozio, …
- Come in un generico database, la decisione sull'uso o meno di chiavi surrogate si basa anche su considerazioni di **efficienza**
  - ✓ Anche se si ha un attributo in più, la chiave surrogata può ridurre lo spazio occupato in quanto è un *codice corto* quindi risparmio spazio quando si usa come foreign key
  - ✓ Le chiavi surrogate possono richiedere i join necessari a recuperare le informazioni (ad esempio per ricavare il negozio e la sua città)
- In un Data Mart la soluzione con chiavi surrogate è comunque indispensabile per l'implementazione di ***Scenari Temporali***.

❖ L'uso delle chiavi surrogate non cambia la *logica* dello schema: per semplicità, negli esercizi di progettazione logica normalmente non useremo chiavi surrogate

Progettazione logica

- Include l'insieme dei passi che, a partire dallo schema concettuale, permettono di determinare lo schema logico del data mart

- Le principali operazioni da svolgere durante la progettazione logica sono:
  1. Scelta dello schema logico da utilizzare (es. star/snowflake schema)
  2. Traduzione degli schemi di fatto

- Altre operazioni che possono essere svolte durante la progettazione logica riguardano l'ottimizzazione del sistema (ad esempio, scelta delle viste da materializzare); noi ~~**non**~~ tratteremo questo aspetto in **[VISTE MATERIALIZZATE]**

Star *VS* Snowflake

- Esistono pareri contrastanti sull'utilità dello snowflaking, in quanto esso contrasta con la filosofia del data warehousing di avere tabelle completamente denormalizzate.

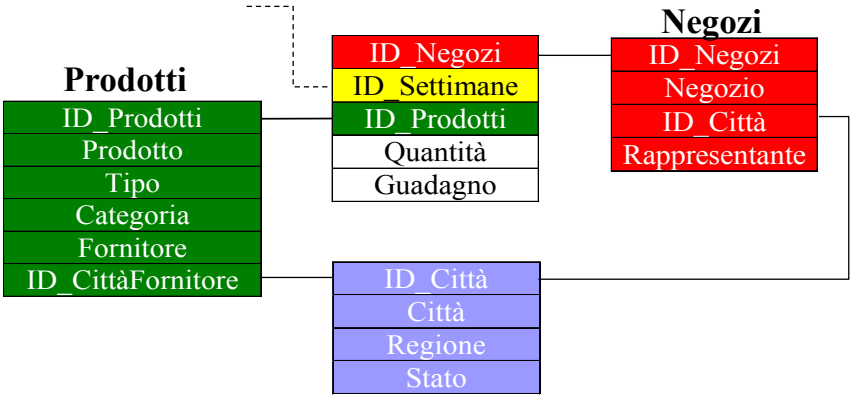
- Nello **star schema** le Dimension Table sono **denormalizzate**
  - 👍 Basta un join per recuperare tutti i dati relativi a una dimensione
  - 👎 La denormalizzazione introduce una forte ridondanza nei dati

- Nello **Snowflake** schema le Dimension Table sono **normalizzate** (eventualmente solo alcune di esse)
  - 👍 La normalizzazione elimina la ridondanza nei dati e riduce quindi lo spazio richiesto per la memorizzazione
  - 👎 E' necessario un join tra tutte le tabelle secondarie per recuperare tutti i dati relativi a una dimensione

- ❖ Il confronto sull'**efficienza** delle due soluzioni (spazio occupato, velocità nelle query …) verrà fatto quando si parlerà di **viste materializzate**.

Star *VS* Snowflake

- Oltre all'efficienza delle due soluzioni, si può considerare anche la *semplicità* dello schema logico: uno snowflak ha sì più tabelle, ma esse possono essere usate in più schemi
- Lo SnowFlake può essere utile quando una parte di una gerarchia è comune a più dimensioni (dello stesso schema o di schemi diversi) . Nell'esempio la dimension table secondaria è riutilizzata per più gerarchie



Star *VS* Snowflake

- Un'altra considerazione nella scelta tra star o snowflake riguarda l'**alimentazione** del Data Mart, ovvero come progettare l'alimentazione delle dimension table *condivise*
- Esempio:
  - Star schema

    ```
    DT_PRODOTTO(IDProdotto,Prodotto,Tipo,Categoria,Fornitore,
            CittaFornitore,RegioneFornitore,StatoFornitore)
    ```
  - ```
    DT_NEGOZIO(IDNegozio,Negozio,
            CittaNegozio,RegioneNegozio,StatoNegozio)
    ```
  - Snowflake schema

    ```
    DT_PRODOTTO(IDProdotto,Prodotto,Tipo,Categoria,Fornitore,
            IdCittaFornitore:DT_CITTA)
    DT_NEGOZIO(IDNegozio,Negozio, IDCittaNegozio:DT_CITTA)

    DT_CITTA(IDCitta,Citta,Regione,Stato)
    ```
- Nello snowflake schema, la dimension table `DT_CITTA` deve contenere sia le città dei fornitori sia le città dei negozi

Dagli schemi di fatto agli star schema

- La regola di base per la traduzione di uno schema di fatto in schema a stella prevede di:

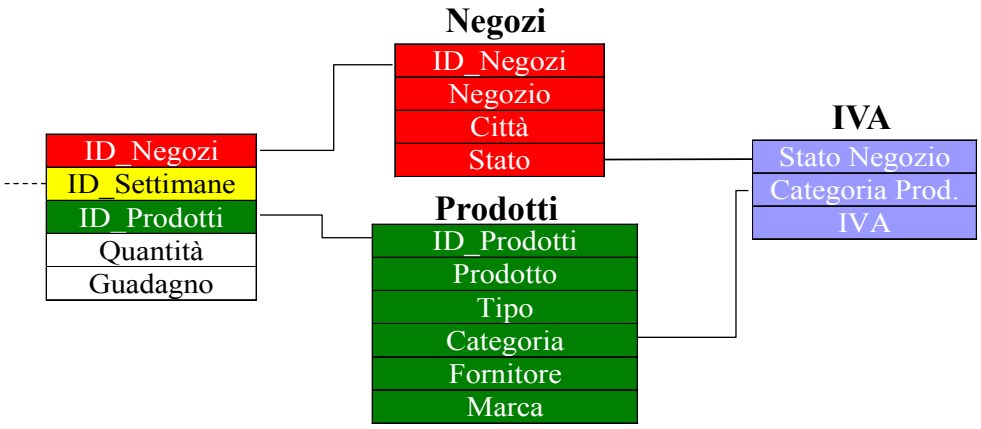
  Creare una fact table contenente tutte le misure e gli attributi descrittivi direttamente collegati con il fatto e, per ogni gerarchia, creare una dimension table che ne contiene tutti gli attributi.

- In aggiunta a questa semplice regola, la corretta traduzione di uno schema di fatto richiede una trattazione approfondita dei costrutti avanzati del DFM

- Attributi descrittivi
  - Se collegato a un attributo dimensionale, va incluso nella dimension table che contiene l'attributo.
  - Se collegato direttamente al fatto deve essere incluso nella fact table.
    ```

Attributi cross-dimensionali

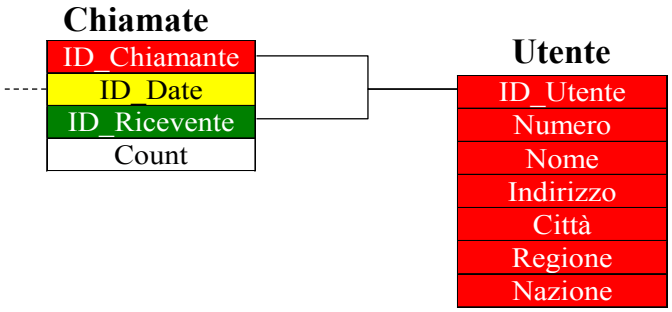
- Dal punto di vista concettuale, un attributo cross-dimensionale b definisce un'associazione molti-a-molti tra due o più attributi dimensionali a_1..., a_m.
- La sua traduzione a livello logico richiede l'inserimento di una nuova tabella che includa b e abbia come chiave gli attributi a_1..., a_m.



Gerarchie condivise

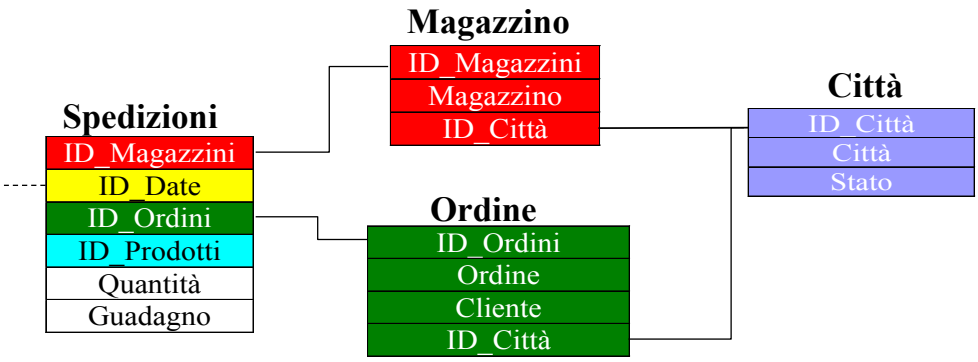
- Se una gerarchia si presenta più volte nello stesso fatto (o in due schemi di fatto diversi) non conviene introdurre copie ridondanti delle relative dimension table.

- Se le due gerarchie contengono esattamente gli stessi attributi sarà sufficiente importare due volte la chiave della medesima dimesion table



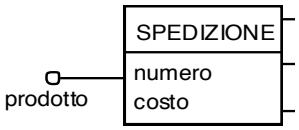
Gerarchie condivise

- Se le due gerarchie condividono solo una parte degli attributi è necessario decidere se:
  I. Introdurre ulteriore ridondanza nello schema duplicando le gerarchie e replicando i campi comuni.
  II. Eseguire uno snowflake sul primo attributo condiviso introducendo una terza tabella comune a entrambe le dimension table.



Dimensioni degeneri

- Questo termine indica una dimensione la cui gerarchia contiene un solo attributo.



- Se la lunghezza dell'attributo non è eccessiva può convenire evitare la creazione di una specifica dimension table importando direttamente i valori dell'attributo nella fact table.

```
FT_SPEDIZIONE(ID_Prodotto, ID_Dim_1, … , ID_Dim_n,numero,costo)
DT_Prodotto(ID_Prodotto,Prodotto)
```

```
FT_SPEDIZIONE(Prodotto, ID_Dim_1, … , ID_Dim_n,numero,costo)
```

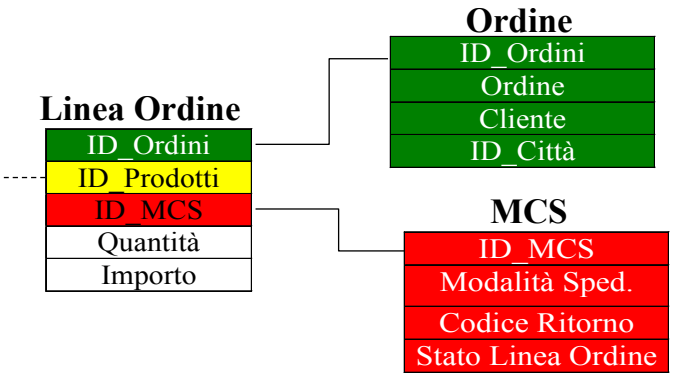
- Si noti che una dimension table per prodotto senza chiave surrogata non ha alcun senso:

```
FT_SPEDIZIONE(Prodotto, ID_Dim_1, … , ID_Dim_n,numero,costo)
DT_Prodotto(Prodotto)
```

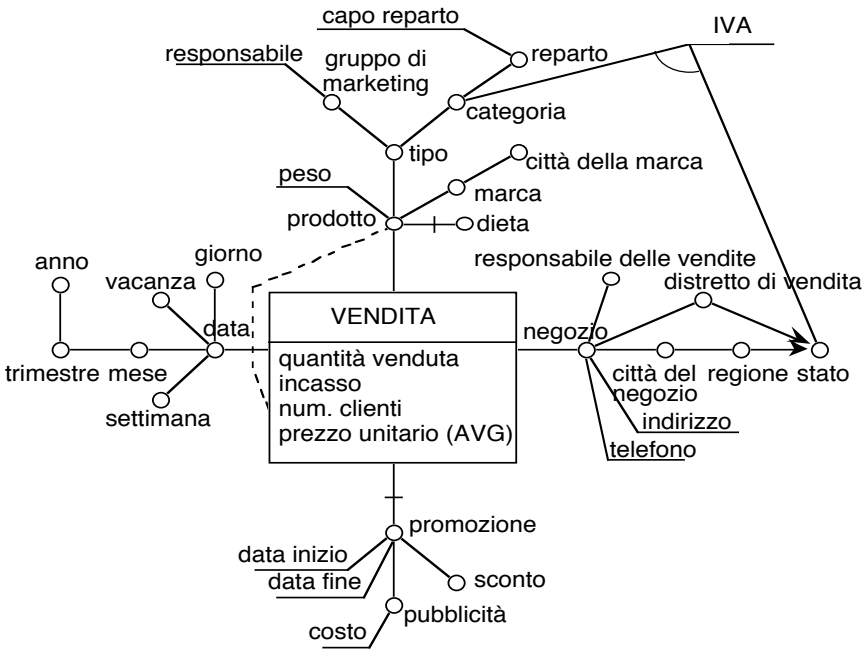
Dimensioni degeneri: Junk dimension

- Una soluzione alternativa è quella di utilizzare un'unica dimension table per modellare più dimensioni degeneri (*junk dimension*)
  - ✓ In una junk dimension non esiste alcuna dipendenza funzionale tra gli attributi per cui risultano valide tutte le possibili combinazioni di valori.
  - ✓ Questa soluzione risulta attuabile solo quando il numero di valori distinti per gli attributi coinvolti è limitato.

ESEMPIO:
Fatto Linea_Ordine con dimensioni degeneri
1. Modalità Spedizione
2. Codice Ritorno
3. Stato Linea Ordine



Progettazione Logica del fatto VENDITA



Star Schema: FT_VENDITA

- Non si usano chiavi surrogate
- Per la dimensione opzionale Promozione: si userà una opportuna codifica delle vendite senza promozione, ovvero il relativo valore nullo sarà opportunamente codificato. Quindi nella fact table Promozione è chiave al pari delle altre dimensioni

```
FT_VENDITA (   Prodotto:DT_PRODOTTO,
               Negozio:DT_NEGOZIO,
               Data:DT_DATA,
               Promozione:DT_PROMOZIONE,
               QuantitàVenduta,
               Incasso,
               NumeroClienti,
               PrezzoUnitario_SUM,
               PrezzoUnitario_COUNT)
```

Schema Temporale:
La misura PrezzoUnitaria (AVG) è calcolata come rapporto tra PrezzoUnitario_SUM
e
PrezzoUnitario_COUNT

Star Schema: DT_NEGOZIO

- La Dimension Table deve contenere tutti gli attributi della gerarchia: essendoci una convergenza su Stato, tale attributo dimensionale è riportato una sola volta nella DT
  - ✓ Con una condivisione su Stato: due attributi distinti nella DT, `DistrVendita_Stato, Città_Stato`

- Semplice corrispondenza uno-a-uno della DT con gli attributi dimensionali della gerarchia; quindi: `DistrVendita` è un attributo semplice di DT_NEGOZIO anche se nello schema del DB operazionale era un attributo composto da NumDistretto + Stato!

```
DT_NEGOZIO (Negozio, RespVendite, indirizzo, telefono,
            DistrVendita,
            Citta,Regione,Stato)
```

Star Schema: DT_PRODOTTO

```
DT_PRODOTTO(Prodotto, Dieta, Marca, CittàMarca, Tipo,
            GruppoMarketing, Categoria, Reparto,
            Peso,Responsabile,CapoReparto)
```

- E' sottointeso che tutti gli attributi dimensionali, anche se opzionali, non hanno valori nulli, in quanto i valori nulli sono opportunamente codificati

- L'attributo **cross-dimensionale** IVA non può essere inserito in DT_PRODOTTO (e neanche in DT_NEGOZIO) ma richiede una nuova tabella (senza foreign key perché nello schema non sono previste le relative tabelle):

```
DT_IVA(Categoria,Stato, Iva)
```


Star Schema: DT_DATA e DT_PROMOZIONE

- DT_DATA

```
DT_DATA(Data, Giorno,Vacanza, Settimana,
        Mese, Trimestre, Anno)
```

- I sistemi OLAP gestiscono direttamente varie gerarchie su un attributo su un attributo di tipo *datatime*:
  → se si definisce Data come *datatime* non è necessario introdurre la relativa dimension table
- E' come se si considerasse Data dimensione degenere.

- DT_PROMOZIONE è relativa ad una dimensione opzionale
  - ✓ deve essere contenere una tupla per rappresentare assenza di promozione: tale tupla avrà opportuni valori anche per gli altri attributi, quali ad esempio Sconto=0.

```
DT_PROMOZIONE(Promozione, Sconto, Pubblicità,
              Costo,DataInizio,DataFine)
```

Snowflake Schema

- La fact table non varia rispetto allo star-schema, possono variare solo le dimension table

- Riportiamo DT_NEGOZIO

```
DT_NEGOZIO(Negozio, RespVendite, indirizzo, telefono
                DistrVendita:DT_DISTRETTO_VENDITA,
                Città:DT_CITTA)

      DT_DISTRETTO_VENDITA(DistrVendita, Stato)
      DT_CITTA(Città, Regione:DT_REGIONE)
            DT_REGIONE(Regione,Stato)
```


Snowflake Schema: DT_NEGOZIO (varianti)

- Se c'è la convergenza su Stato allora

  `DT_DISTRETTO_VENDITA(DistrVendita, Stato)`

  è ridondante: se viene eliminata, il legame tra il distrVendita e lo Stato si ottiene facendo il join tra le altre tabelle
  DT_NEGOZIO, DT_CITTA e DT_REGIONE.

  - Si può tenere `DT_DISTRETTO_VENDITA` per facilitare la costruzione dei cubi nel sistema OLAP

- E' possibile lasciare alcune dimension table secondarie non normalizzate (cioè denormalizzate).

- Ad esempio, si può non introdurre DT_REGIONE e usare

  `DT_CITTA(Città, Regione,Stato)`

Snowflake Schema: DT_PRODOTTO

- A titolo di esempio, otteniamo DT_PRODOTTO come normalizzazione a partire dalla relativa dimension table dello star schema, sulla base delle FD presenti nella gerarchia:

```
DT_PRODOTTO(Prodotto, Dieta, Marca, CittàMarca, Tipo,
              GruppoMarketing, Categoria, Reparto,
              Peso,Responsabile,CapoReparto)

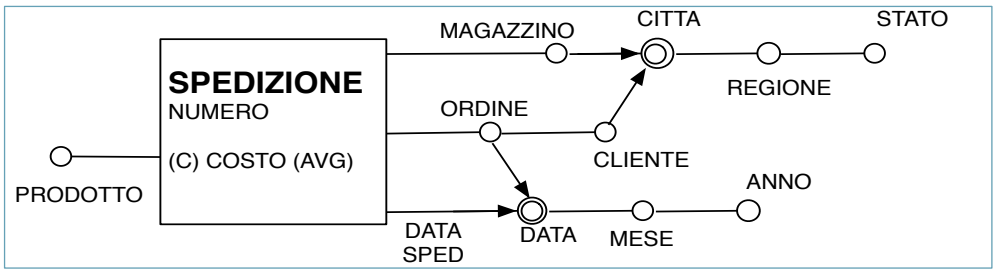
        Marca → CittàMarca
        Tipo → Categoria
        Tipo → GruppoMarketing
        Categoria→ Reparto
```

- Normalizzando si ottiene (le FK sono omesse) :

```
DT_PRODOTTO(Prodotto, Dieta, Marca, Tipo, Peso)
    DT_Marca(Marca, CittàMarca)
    DT_Tipo(Tipo,Categoria, GruppoMarketing, Responsabile)
    DT_Categoria(Categoria,Reparto, CapoReparto)
```

- Si noti che gli attributi descrittivi non entrano in gioco durante la normalizzazione: essi vengono collocati nella tabella che contiene il relativo attributo dimensionale

Esempio di Star schema




```
FT_SPEDIZIONE(ORDINE:DT_ORDINE, MAGAZZINO:DT_MAGAZZINO,
                DATASPED:DT_DATA,PRODOTTO,NUMERO,COSTO_SUM,COSTO_COUNT)

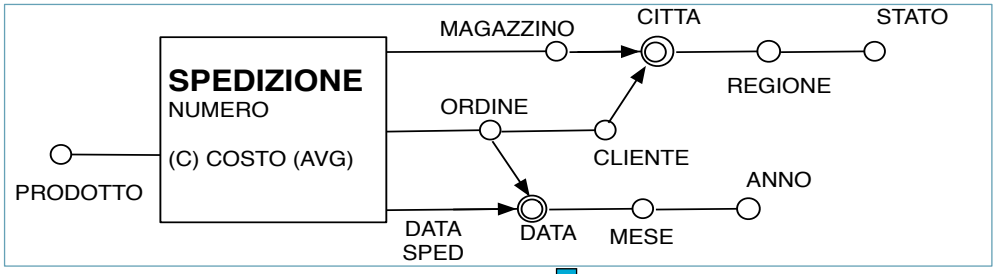
        DT_DATA(DATA,MESE,ANNO)

        DT_ORDINE(ORDINE,DATA,MESE,ANNO,CLIENTE,
                    CITTA,REGIONE,STATO)

        DT_MAGAZZINO(MAGAZZINO,CITTA,REGIONE,STATO)
```

- Convergenza su REGIONE: da un punto di vista logico si potrebbe togliere REGIONE,STATO in una delle due dimension table, ad esempio in DT_MAGAZZINO, ma in pratica tale semplificazione non viene mai effettuata.

Esempio di SnowFlake schema




```
FT_SPEDIZIONE(ORDINE:DT_ORDINE, MAGAZZINO:DT_MAGAZZINO,
                DATASPED:DT_DATA,PRODOTTO,NUMERO,COSTO_SUM,COSTO_COUNT)

        DT_DATA(DATA,MESE:DT_MESE)

                DT_MESE(MESE,ANNO)

        DT_ORDINE(ORDINE,DATA: DT_DATA,CLIENTE:DT_CLIENTE)

                DT_CLIENTE(CLIENTE,CITTA:DT_CITTA)

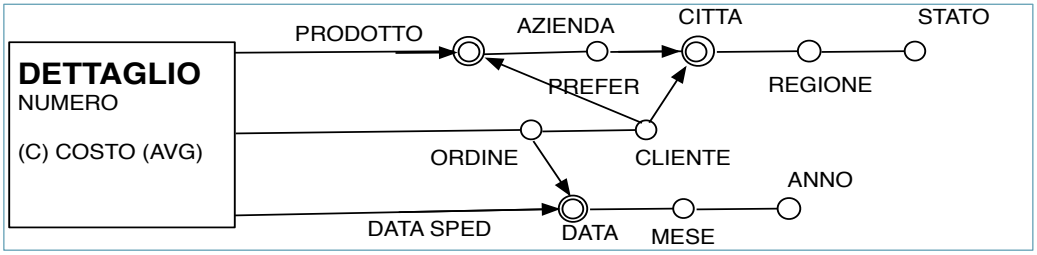
        DT_MAGAZZINO(MAGAZZINO,CITTA:DT_CITTA)

                DT_CITTA(CITTA,REGIONE:DT_REGIONE)

                        DT_REGIONE(REGIONE,STATO)
```

- Convergenza su REGIONE: quali semplificazioni ?

Esempio di Star schema



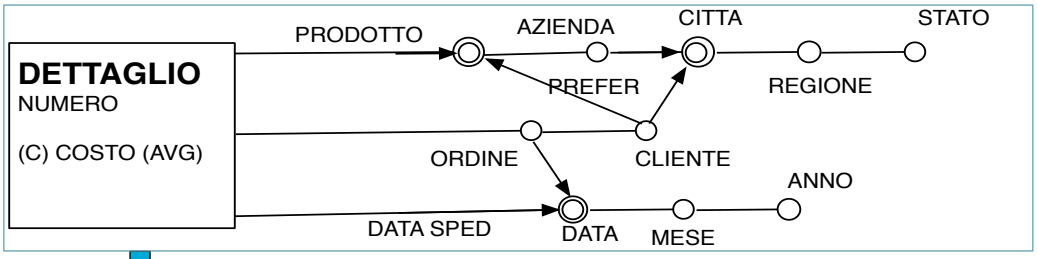
Rispetto a SPEDIZIONE cambia DT_ORDINE,
C'è DT_PRODOTTO, la fact_table è simile

```
DT_ORDINE(ORDINE,DATA,MESE,ANNO,
                CLIENTE,CLIENTE_CITTA,CLIENTE_REGIONE,CLIENTE_STATO,
                PRODOTTO, PRODOTTO_AZIENDA,
                PRODOTTO_CITTA, PRODOTTO_REGIONE, PRODOTTO_STATO)

DT_PRODOTTO(PRODOTTO,AZIENDA,CITTA,REGIONE,STATO
```

- Convergenza su REGIONE *all'interno della dimensione ORDINE*: si può togliere `PRODOTTO_REGIONE, PRODOTTO_STATO`
- Convergenza su REGIONE *tra le dimensioni ORDINE e PRODOTTO*: come in SPEDIZIONE - non si effettua alcuna semplificazione

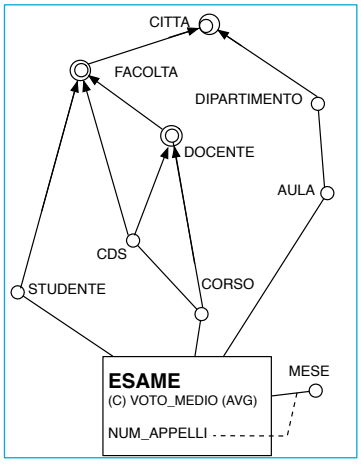
Esempio di SnowFlake schema




```
DT_ORDINE(ORDINE,DATA: DT_DATA,CLIENTE:DT_CLIENTE)
        DT_CLIENTE(CLIENTE,CITTA:DT_CITTA, PREFER:DT_PRODOTTO)
DT_PRODOTTO(PRODOTTO,AZIENDA:DT_AZIENDA)
DT_AZIENDA(AZIENDA,CITTA:DT_CITTA)
        DT_CITTA(CITTA,REGIONE:DT_REGIONE)
                DT_REGIONE(REGIONE,STATO)
```

- Convergenza su REGIONE *all'interno della dimensione ORDINE*: quali semplificazioni ?
- Convergenza su REGIONE *tra le dimensioni ORDINE e PRODOTTO*: quali semplificazioni ?

Esempio



Star Schema

```
FACT_TABLE(STUDENTE:DT_STUDENTE,CORSO:DT_CORSO,AULA:DT_AULA,MESE,
           VOTO_SUM,VOTO_COUNT,NUM_APPELLI)
DT_STUDENTE(STUDENTE, FACOLTA, FACOLTA_CITTA)
DT_AULA(AULA, DIPARTIMENTO, DIPARTIMENTO_CITTA)
DT_CORSO(CORSO,DOCENTE,DOCENTE_FACOLTA,DOCENTE_FACOLTA_CITTA,
         CDS, CDS_FACOLTA, CDS_FACOLTA_CITTA,
         CDS_PRESID, CDS_PRESID_FACOLTA, CDS_PRESID_FACOLTA_CITTA)
```

SnowFlake Schema

```
FACT_TABLE(STUDENTE:DT_STUDENTE,CORSO:DT_CORSO,AULA:DT_AULA,MESE,
           VOTO_SUM,VOTO_COUNT,NUM_APPELLI)
DT_STUDENTE(STUDENTE, FACOLTA:DT_FACOLTA)
DT_FACOLTA(FACOLTA, CITTA)
DT_AULA(AULA, DIPARTIMENTO:DT_DIPARTIMENTO)
DT_DIPARTIMENTO(DIPARTIMENTO, CITTA)
DT_CORSO(CORSO,DOCENTE:DT_DOCENTE,CDS:DT_CDS)
DT_DOCENTE(DOCENTE, FACOLTA:DT_FACOLTA)
DT_CDS(CDS,PRESIDENTE:DT_DOCENTE,FACOLTA:DT_FACOLTA)
```

Progetto logico: considerazioni

- Lo schema logico del DW si ottiene con una (semplice) traduzione dello schema di fatto, ovvero in esso si deve riportare tutto e solo quello presente in uno schema di fatto
  - ✓ Nel costruire lo schema logico, non si considera più lo schema (E/R, relazionale) del DB operazionale

- Lo schema relazionale del DB operazionale deve essere considerato in fase di alimentazione del DW
  - ✓ Ad esempio, nel caso trattato, in fase di alimentazione del DW precedente, verrà stabilito che l'attributo dimensionale DistrVendita verrà alimentato tramite la concatenazione degli attributi Stato e NumDistretto!

Nota: vincoli di inclusione

- Consideriamo lo star-schema precedente e la tabella:

  `DT_IVA(Categoria,Stato, Iva)`

- Categoria e Stato sono due attributi (rispettivamente di DT_PRODOTTO e DT_NEGOZIO): non essendo chiavi **non si possono** definire le FK

- D'altra parte si dovrebbe vincolare i valori di Categoria (Stato) in IVA ad essere dei valori anche di Categoria (Stato) nella tabella DT_PRODOTTO (DT_NEGOZIO)

- In relazionale questi corrispondono a *vincoli di inclusione*

  DT_IVA(Categoria) ⊆ DT_PRODOTTO(Categoria)
  DT_IVA(Stato) ⊆ DT_NEGOZIO(Stato)

  che generalizzano il concetto di vincolo di integrità referenziale.

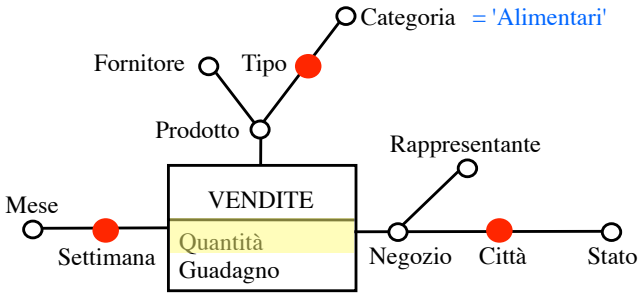
- Nei DBMS non sono gestiti vincoli di inclusione. Essi devono essere definiti e gestiti dal progettista attraverso l'uso di *trigger* .

Il carico di lavoro

- È necessario identificare in fase di progettazione logica un **carico di lavoro** preliminare, di riferimento: insieme delle principali interrogazioni cui il sistema viene sottoposto dagli utenti finali
  - ✓ Colloqui con gli utenti

- Il carico di lavoro di un sistema OLAP è per sua natura estemporaneo

- Le interrogazioni OLAP sono facilmente caratterizzabili
  - ✓ Pattern di aggregazione
  - ✓ Misure richieste
  - ✓ Clausole di selezione



Totale della quantità venduta per i diversi tipi di prodotto, in ogni settimana e città ma solo per i prodotti alimentari

Dinamicità del carico di lavoro

- Il carico di lavoro preliminare non è di per sé sufficiente a ottimizzare le prestazioni del sistema
  - ✓ L'interesse degli utenti cambia nel tempo
  - ✓ Il numero di interrogazioni aumenta al crescere della confidenza degli utenti con il sistema

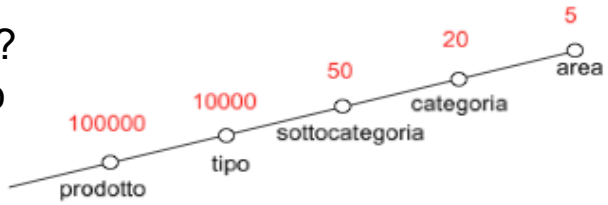
- Per ottimizzare la struttura logica del data mart è necessaria una fase di tuning attuabile solo dopo che il sistema è stato messo in funzione

- Il carico di lavoro reale può essere desunto dal log delle interrogazioni sottoposte al sistema

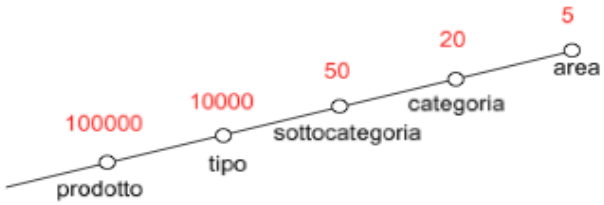
Il volume dati

- Consiste nelle informazioni necessarie a determinare/ stimare la dimensione del data mart.
  - ✓ Numero di valori distinti degli attributi nelle gerarchie
  - ✓ Lunghezza degli attributi
  - ✓ Numero di eventi di ogni fatto
- Deve essere calcolato considerando la quantità di dati necessari a coprire l'intervallo temporale del DM.
- È utilizzato sia durante la progettazione logica sia durante la progettazione fisica per determinare:
  - ✓ la dimensione di tabelle e indici, i costi di accesso

- Esempio: Star VS Snowflake ? È utile il rapporto tra il numero di valori distinti lungo le gerarchie



Il volume dati : esempio

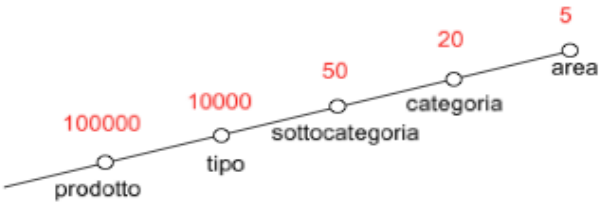


Attributi = 30 byte

- **Star Schema:**
  DT_Prodotto(P,T,S,C,A) 100.000 * 30 * 5 = **15.000.000**

- **SnowFlake Schema** (*snowflaking* su T) : Totale **7.200.000**
  DT_Prodotto(P,T) 100.000 * 30 * 2 = 6.000.000
  DT_Tipo(T,S,C,A) **10.000** * 30 * 4 = 1.200.000
  - ✓ **1.000** * 30 * 4 = 120.000 ➔ Totale **6.120.000**
  - ✓ **25.000** * 30 * 4 = 3.000.000 ➔ Totale **9.000.000**

Il volume dati : esempio



Attributi = 30 byte

Chiavi Surrogate = 3 byte

- **Star Schema:**
  DT_Prodotto(P,T,S,C,A) 100.000 * 30 * 5 = **15.000.000**

- **SnowFlake Schema** (*snowflaking* su T) : Totale **4.350.000**
  DT_Prodotto(P, kT) 100.000 * (30 + 3) = 3.300.000
  DT_Tipo(kT, T,S,C,A) **10.000** * (30 * 4 + 3) = 1.230.000
  - ✓ **1.000** * (30 * 4 + 3) = 123.000 ➔ Totale **3.423.000**
  - ✓ **25.000** * (30 * 4 + 3) = 3.075.000 ➔ Totale **6.375.000**

Il problema della sparsità

- La bontà delle stime è spesso compromessa a causa del problema della sparsità.
  - ✓ Nel modello multidimensionale, a un insieme di coordinate corrisponde un possibile evento anche se questo non è realmente avvenuto

- Normalmente il numero di eventi accaduti è di gran lunga inferiore a quelli possibili

- Tenere traccia degli eventi non accaduti comporta uno spreco di risorse e riduce le prestazioni del sistema
  - ✓ ROLAP: memorizza solo gli eventi accaduti
  - ✓ MOLAP: richiede tecniche complesse per ridurre al minimo lo spazio necessario a tenere traccia degli eventi non accaduti