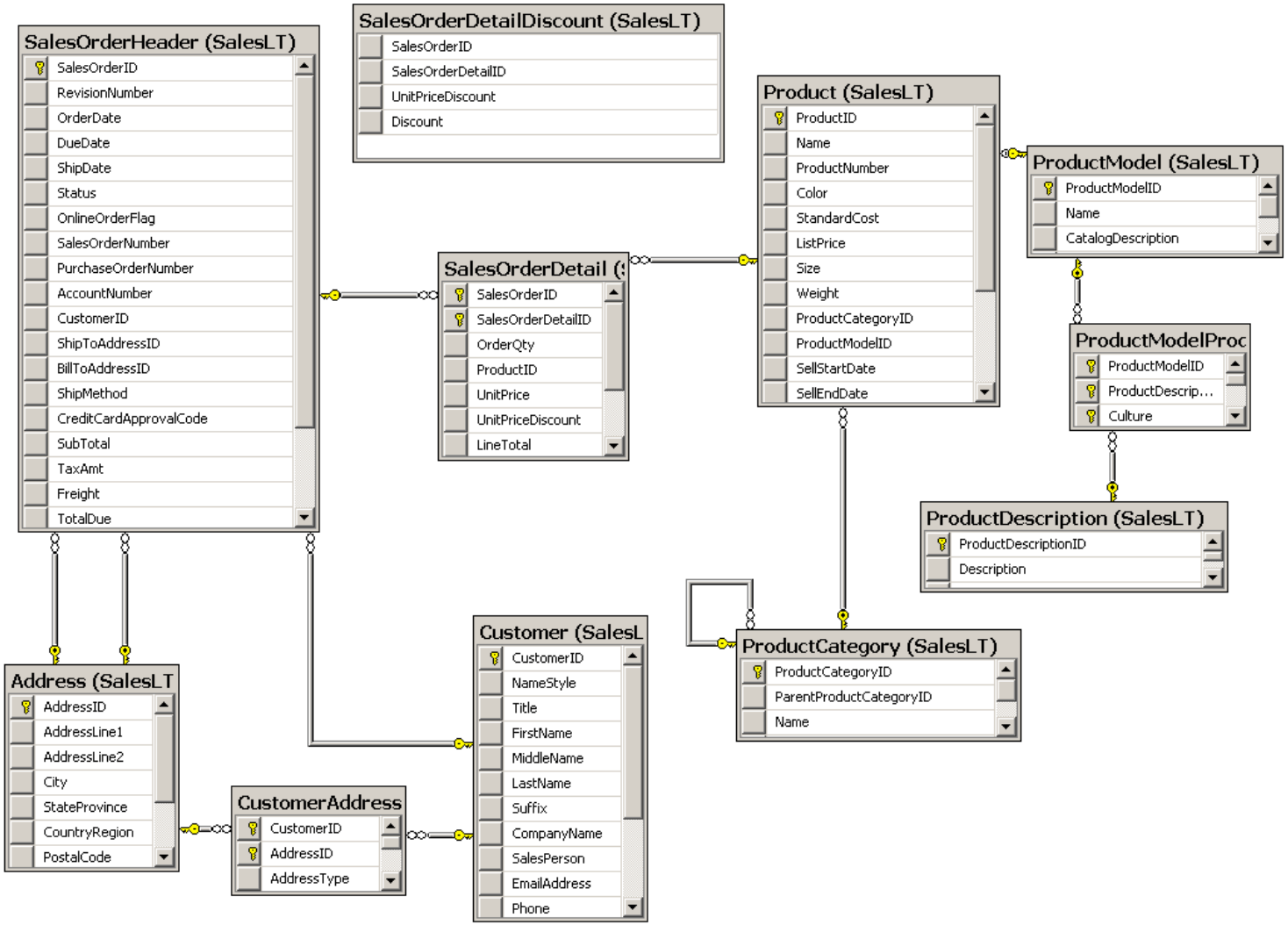
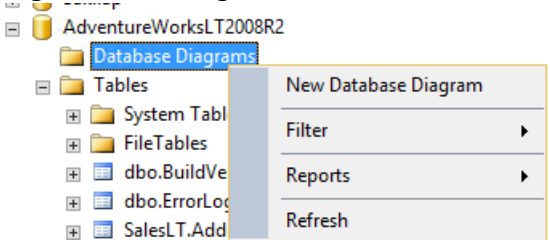
APPUNTI LEZIONE 22 OTTOBRE 2015

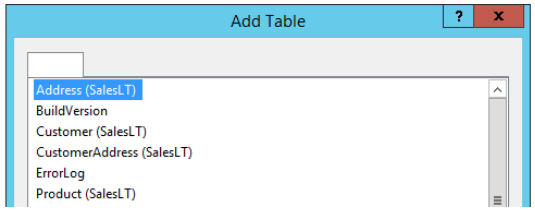
Diagramma relazionale del DB https://dl.dropboxusercontent.com/u/15491020/SitoWeb/AdventureWorksLT_SIA.bak



Per generare il diagramma relazionale :



quindi aggiungere le tabelle ad eccezione di BuildVersion ed errorLog



Si considerano le tre relazioni (alcuni attributi sono omessi, i nomi sono abbreviati)

ProductCategory **PC**(PC_ID, PPC_ID:PC, Name)

Product **P**(P_ID, Name, PC_ID:PC)

SalesOrderDetail **SOD**(SO_ID,SOD_ID,P_ID:P, Qty)

(siccome non si considera al momento la tabella SalesOrderHeader, non viene indicata la foreign key SO_ID:SOH)

Data Profiling

1. Name è AK in PC
   PC(PC_ID, PPC_ID:PC, Name)
   AK: Name
2. Name è AK in P
   P(P_ID, Name, PC_ID:PC)
   AK: Name
3. SO_ID, P_ID è AK in SOD, ovvero un prodotto è presente nel dettaglio di un ordine al Massimo una volta!
   SOD(SO_ID,SOD_ID,P_ID:P, Qty)
   AK: SO_ID, P_ID

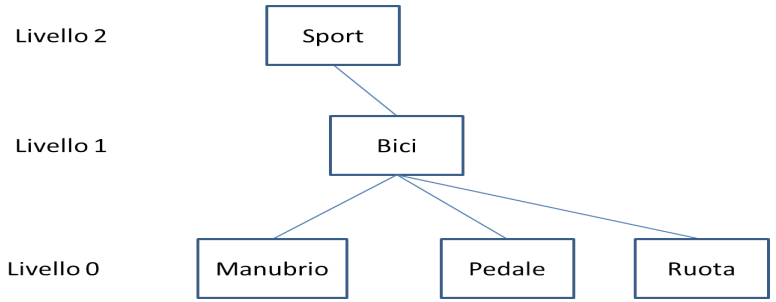
Si vuole analizzare nel dettaglio la relazione

ProductCategory **PC**(PC_ID, PPC_ID:PC, Name)

Che presenta una foreign key riferita a se stessa, usata per *classificare* le categorie:
la PPC (ParentProductCategory) di una categoria è la categoria più generale, ad esempio

PC_ID	PPC_ID	Name
1	4	Pedale
2	4	Manubrio
3	4	Ruota
4	5	Bici
5	NULL	Sport

Corrisponde alla seguente *gerarchia di classificazione* delle categorie



Consideriamo anche le istanze delle altre due tabelle (nella tabella PC sono state inserite altre categorie)

PC

PC_ID	PPC_ID	Name
1	4	Pedale
2	4	Manubrio
3	4	Ruota
4	5	Bici
5	NULL	Sport
6	8	Maglia
7	8	Scarpa
8	NULL	Abbigliam.

P

P_ID	Name	PC_ID
1	PedaleXWD	1
2	PedaleBik	1
3	ManubrTres	2
4	RuotaLentEW	3
5	RuotaRagEZ	3

SOD

SO_ID	SOD_ID	P_ID	Qty
OrdineA	1	1	20
OrdineA	2	3	10
OrdineA	3	5	20
OrdineB	1	2	30
OrdineB	2	5	60

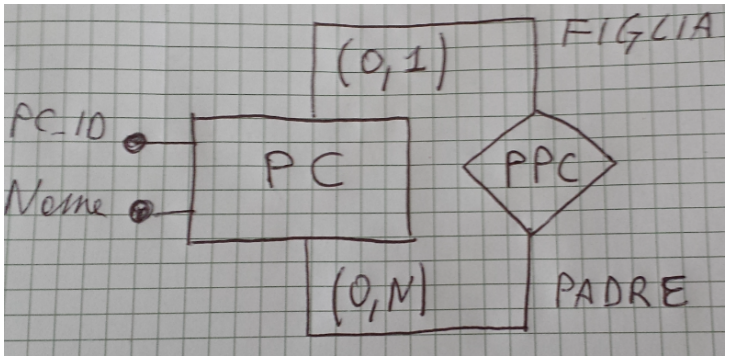
Reverse Engineering in ER

In ER, una foreign key riferita a se stessa

PC(PC_ID, PPC_ID:PC, Name)

AK: Name

Corrisponde ad un *anello* o *associazione ricorsiva*, ovvero associazione tra un'entità e se stessa:

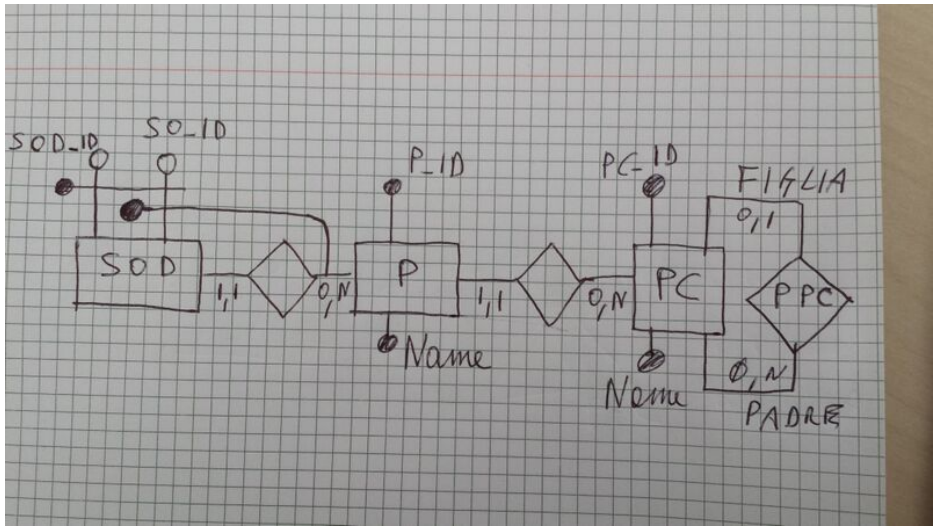


Il (nome del) *ruolo* (FIGLIA, PADRE nell'esempio) è importante per distinguere il differente *ruolo* dell'entità nell'associazione.

Dato lo schema relazionale del nostro esempio ridotto

PC(PC_ID, PPC_ID:PC, Name)
AK: Name
P(P_ID, Name, PC_ID:PC)
AK: Name
SOD(SO_ID,SOD_ID,P_ID:P, Qty)
AK: SO_ID, P_ID

L'ER completo è il seguente:



PROGETTAZIONE DEL Data Mart

Requisiti : Dimensioni **D = {SO_ID, PRODOTTO}**

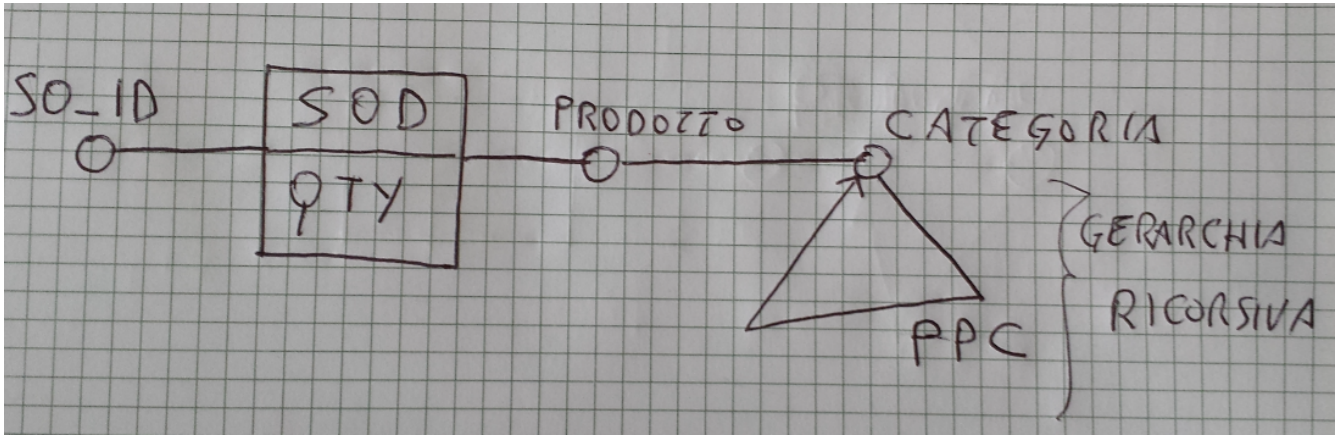
(la misura è la quantità QTY)

Note:

1) **SO_ID** è degenere, non avendo nello schema ridotto altre informazioni sull'ordine a parte il suo identificatore
2) Del prodotto non interessa il suo P_ID (potato) e si considera il Name (che è chiave): per chiarezza si rinomina in PRODOTTO
3) Della categoria non interessa il suo PC_ID (potato) e si considera il Name (che è chiave): per chiarezza si rinomina in CATEGORIA

Aspetto principale:

l'anello/associazione ricorsiva PPC di CATEGORIA (entità PC) corrisponde ad una **gerarchia ricorsiva**:



Non è stato discusso a lezione (verrà fatto la prossima volta)

ma è facile verificare che questo schema è **TRANSAZIONALE**!!

Come ottenere il pattern CATEGORIA, ovvero un report con indicate le categorie con le relative quantità vendute

CATEGORIA	**QTY**
BICI	140
PEDALE	50
…	…

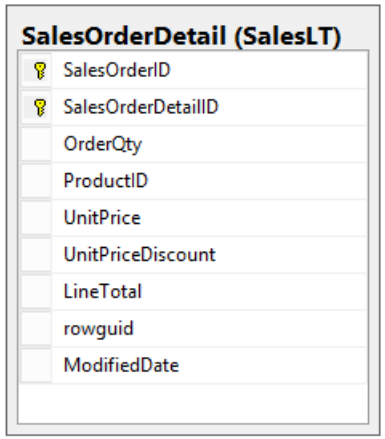
Possibili soluzioni:

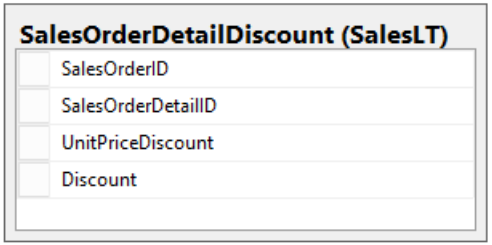
1) Strumenti ad-hoc del sistema OLAP
2) In SQL

Queste soluzioni verranno valutate nelle prossime lezioni.

Seconda Parte

Consideriamo la tabella SalesOrderDetailDiscount (SODD), per la quale non è stata dichiarata nello schema la chiave e nessuna foreign key:





In fase di Data Profiling è fondamentale determinare le chiavi e le foreign key (associazioni con altre tabelle)

Abbreviamo i nomi: SODD (SO_ID, SOD_ID, UPD, Discount)

Intuitivamente SODD è un sottoinsieme di SOD .... In relazionale un sottoinsieme si specifica con una foreign key che è anche chiave, ricordate l'esempio del CHECK_IN o quello classico LAUREANDO come sottoinsieme di STUDENTE:

STUDENTE(MATR,ETA) LAUREANDO(MATR:STUDENTE, TitoloTesi)

Verifico la chiave SO_ID, SOD_ID

```
SELECT * FROM SODD
where SO_ID is null OR SOD_ID is null

SELECT SO_ID, SOD_ID
  FROM SSOD
group by SO_ID, SOD_ID
having count(*) > 1
```

Entrambe vuote quindi è chiave:

SODD (SO_ID, SOD_ID, UPD, Discount)

Per verificare la foreign key in SODD riferita a SOD,

cioè **FK: SO_ID, SOD_ID REFERENCES SOD**

abbreviate come

SODD (SO_ID, SOD_ID : SOD, UPD, Discount)

Devo verificare che in SODD **non** ci siano coppie di SO_ID, SOD_ID non associate a coppie di SO_ID, SOD_ID in SOD

Ovvero che il seguente left join restituisca l'insieme vuoto

```
SELECT        *
FROM            SODD LEFT JOIN
                   SOD ON SOD.SO_ID = SODD.SO_ID AND SOD.SOD_ID = SODD.SOD_ID
where SOD.SOD_ID is null
```

Verifichiamo anche che il viceversa non è vero

```
SELECT        *
FROM            SOD LEFT JOIN
                        SODD ON SOD.SO_ID = SODD.SO_ID AND SOD.SOD_ID = SODD.SOD_ID
where SODD.SOD_ID is null
```

restituisce 449 righe: sono quelle senza DISCOUNT

In definitiva

SODD (SO_ID, SOD_ID : SOD, UPD, Discount)

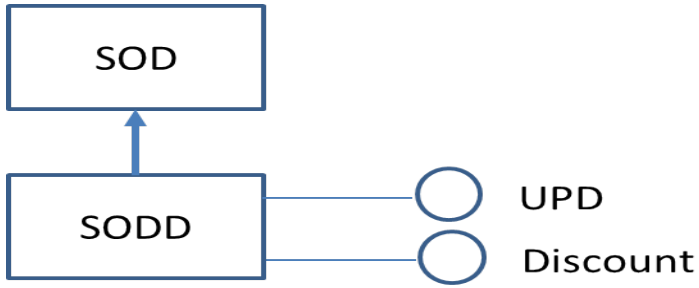
Nel database reale dove devo aggiungere la Key e la FK individuate?

Si ***dovrebbero*** nel DBO riconciliato, che pu⊠ essere un nuovo database (architetture a tre livelli) oppure quello dato modificato.

Nella tesina la Key e la FK vanno riportate **SEMPLICEMENTE** nella **DOCUMENTAZIONE,** cioè nel report da consegnare scriveremo

SODD (SO_ID, SOD_ID : SOD, UPD, Discount)

e quindi tramite reverse engineering in ER come SUBSET



La progettazione concettuale verrà fatta nella prossima lezione.

TERZA PARTE (svolta il 29 Ottobre)

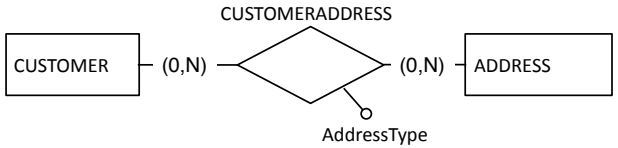
Consideriamo le tre relazioni

CUSTOMER(<u>CustomerID</u>, …

CUSTOMERADDRESS(<u>CustomerID:CUSTOMER, AddressID:ADDRESS</u>, AddressType)

ADDRESS(<u>AddressID</u>, …)

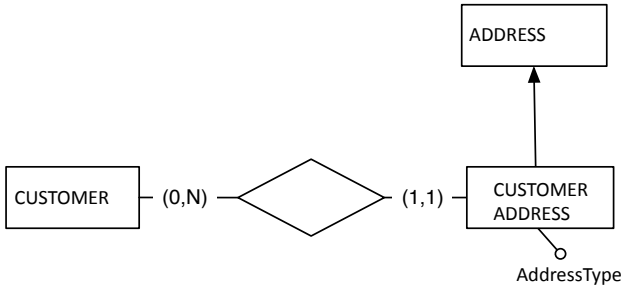
Lo schema ER corrispondente è il seguente



Dal Data Profiling risulta che AddressID è chiave di CUSTOMERADDRESS (quindi CustomerID, AddressID superchiave), cioè

CUSTOMERADDRESS(CustomerID:CUSTOMER, <u>AddressID:ADDRESS</u>, AddressType)

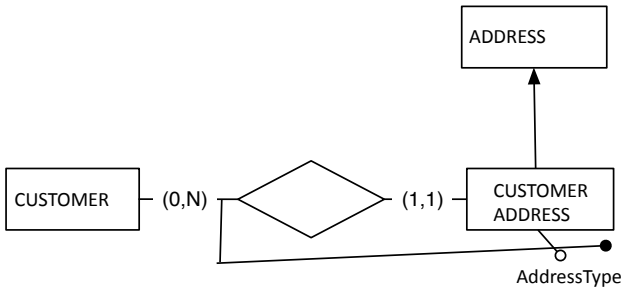
Lo schema ER corrispondente cambia quindi nel seguente schema



Dal Data Profiling risulta anche che CustomerID, AddressType è chiave di CUSTOMERADDRESS, cioè

CUSTOMERADDRESS(CustomerID:CUSTOMER, <u>AddressID:ADDRESS</u>, AddressType)
AK: CustomerID, AddressType

Questa chiave alternativa nello schema ER corrisponde ad aggiungere un ulteriore identificatore a CUSTOMERADDRESS

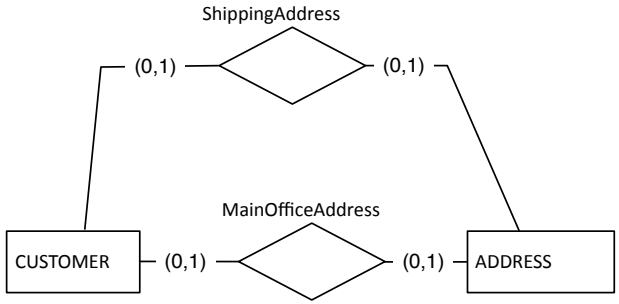


Questa nuova chiave alternativa, questo nuovo identificatore, assieme al fatto che ci sono solo due valori distinti di AddressType (Main Office e Shipping) implica che un CUSTOMER non possa avere più di due istanze in CUSTOMERADDRESS , ovvero la cardinalità N può essere cambiata in 2.

Siccome ci sono solo al massimo due indirizzi per un CUSTOMER, possiamo individuarli e riportarli come *colonne* di una vista/tabella, cioè possiamo definire

VA (CustomerID, ShippingAddress, MainOfficeAddress)

Si noti che in tale vista/tabella CustomerID non è mai nullo e non si ripete, quindi è chiave, mentre ShippingAddress, MainOfficeAddress sono ADDRESS e quindi riferiti tramite foreign key ad ADDRESS; siccome l'entità con chiave CustomerID è CUSTOMER, possiamo aggiungere queste due associazioni all'entità CUSTOMER, ottenendo



Consideriamo ora

ProductModelProductDescription (ProductModelID: ProductModel, ProductDescriptionID: ProductDescription, Culture)

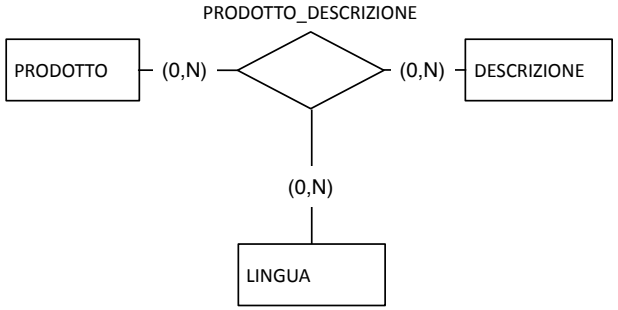
Usando nomi italiani e usando per la chiave e l'entità lo stesso nome avremo

PRODOTTO(PRODOTTO, ..

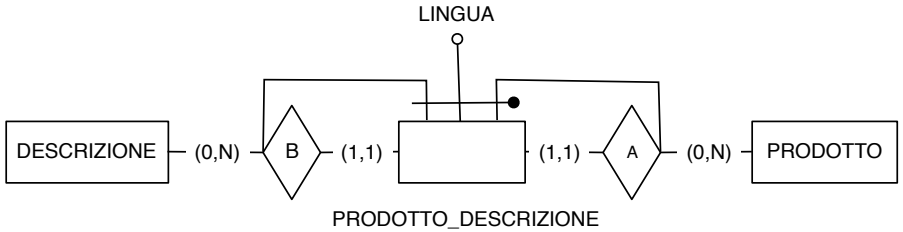
DESCRIZIONE(DESCRIZIONE, …

PRODOTTO_DESCRIZIONE (PRODOTTO:PRODOTTO, DESCRIZIONE:DESCRIZIONE, LINGUA)

PRODOTTO_DESCRIZIONE ha una chiave composta da tre elementi quindi è un'associazione ternaria



Essendo LINGUA solo un attributo e non una foreign key riferita ad una istanza, meglio rappresentarlo come



Il DataProfiling indica che in

PRODOTTO_DESCRIZIONE (PRODOTTO:PRODOTTO, DESCRIZIONE:DESCRIZIONE, LINGUA)

Ci sono due chiavi e non usa sola superchiave (simile al discorso precedente su CUSTOMER ADDRESS)

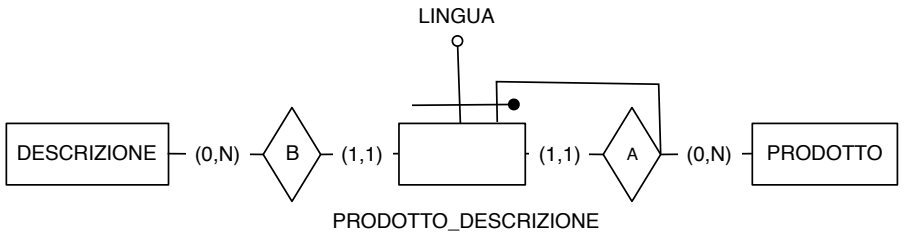
1) PRODOTTO, LINGUA
2) DESCRIZIONE

PRODOTTO_DESCRIZIONE (PRODOTTO:PRODOTTO, LINGUA , DESCRIZIONE:DESCRIZIONE)

Per la prima chiave

1) PRODOTTO, LINGUA

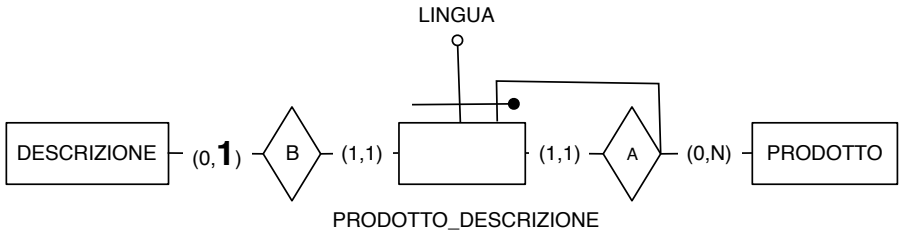
Lo schema ER diventa



Per la seconda chiave

2) DESCRIZIONE

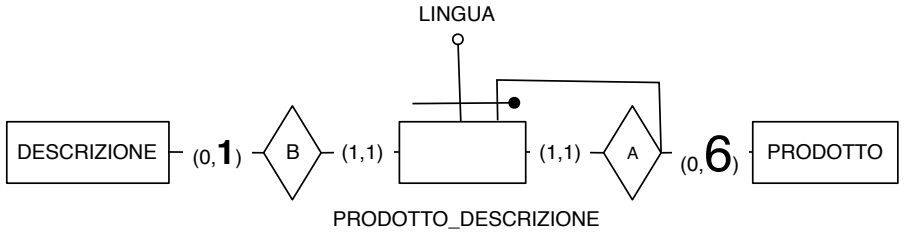
Lo schema ER diventa



Nel database ci sono sei valori distinti di LINGUA

```
SELECT distinct Culture   FROM SalesLT.ProductModelProductDescription
```

Quindi per PRODOTTO la cardinalità N si può specificare in 6



Questo è un livello di dettaglio non richiesto nella tesina.

Come nel caso degli address, essendo la cardinalità massima con un valore specificato (6 in questo caso) si *potrebbero* riportare 6 associazioni tra PRODOTTO e DESCRIZIONE, una per ogni lingua.

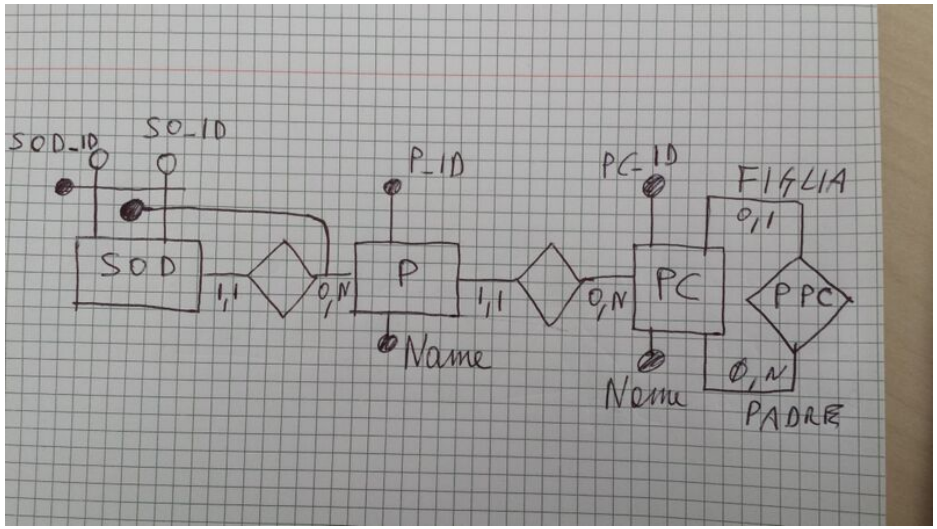
Essendo DESCRIZIONE non utile per le analisi, questa parte non viene ulteriormente sviluppata.

Ricostruiamo lo *schema ER complessivo*

Ricordiamo che per schema relazionale del nostro esempio ridotto

PC(PC_ID, PPC_ID:PC, Name)
AK: Name
P(P_ID, Name, PC_ID:PC)
AK: Name
SOD(SO_ID,SOD_ID,P_ID:P, Qty)
AK: SO_ID, P_ID

L'ER completo è il seguente:

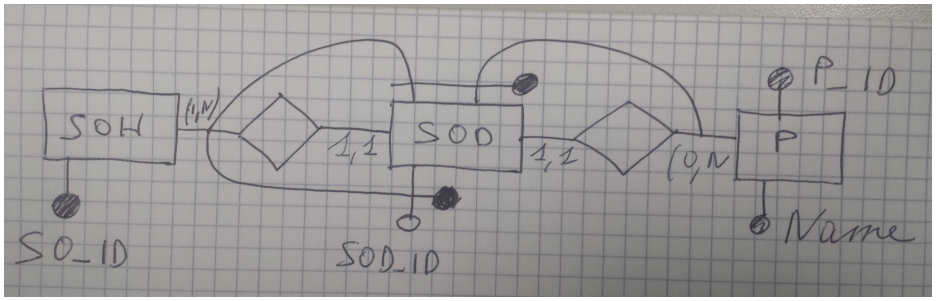


Nel seguito non consideriamo l'*anello /associazione ricorsiva* su PC: eventuali gerarchie ricorsive vengono esaminate nella seconda consegna.

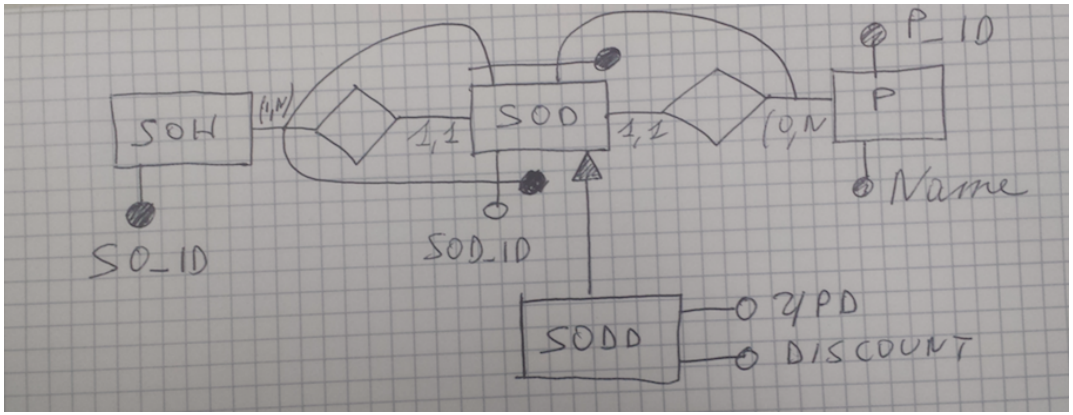
Se consideriamo anche la tabella SalesOrderHeader, deve essere indicata la foreign key SO_ID:SOH

SOH(SO_ID …)
SOD(SO_ID : SOH,SOD_ID,P_ID:P, Qty)
AK: SO_ID, P_ID

quindi lo schema ER precedente diventa

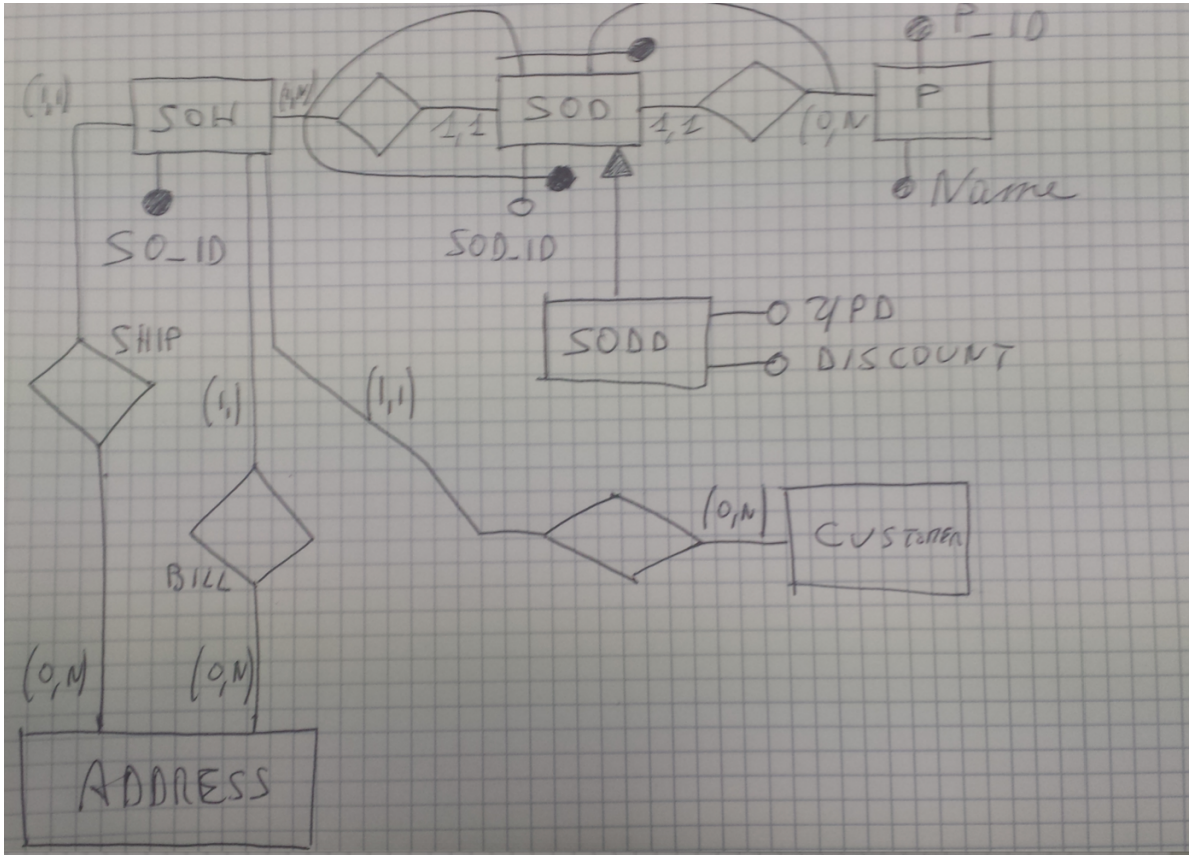


Si aggiunge anche l'entità SODD, ottenendo

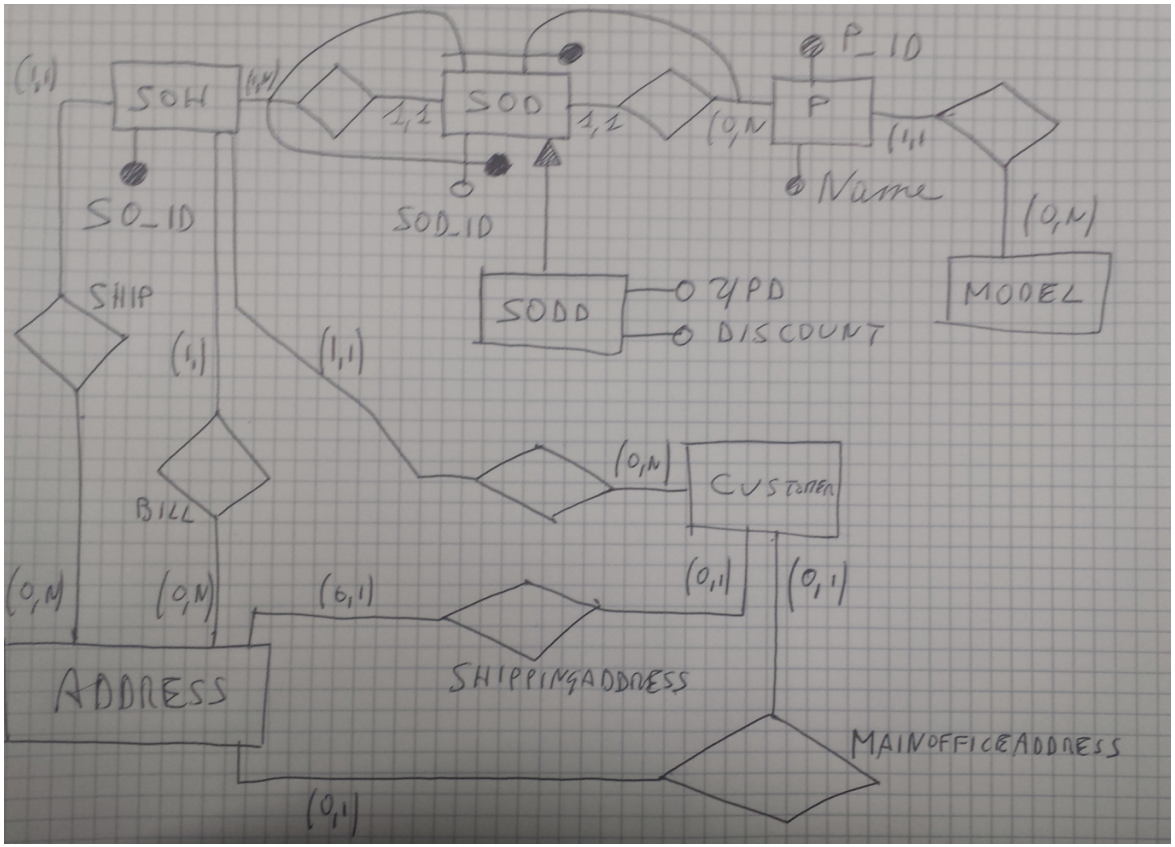


Quindi si aggiungono a SOH le due foreign key verso ADDRESS, ovvero ShipToAddressID e BillToAddressID,

E la foreign key verso CUSTOMER



Infine si aggiungono ShippingAddress e MainOfficeAddress a CUSTOMER e l'entità MODEL associata a PRODUCT (P)


SOH
SOD
P
P_ID
Name
SO_ID
SOD_ID
SODD
TYPD
DISCOUNT
MODEL
SHIP
BILL
CUSTOMER
ADDRESS
SHIPPINGADDRESS
MAINOFFICEADDRESS

Note

1) Sono indicate tutte le associazioni tra entità, ad eccezione di quelle che si è deciso di non considerare ora
2) Molti attributi non sono stati indicati

Considerazioni

1) SHIP e BILL coincidono sempre ➔ si considera una sola associazione SHIP_BILL

PROGETTAZIONE CONCETTUALE DELLO SCHEMA DI FATTO

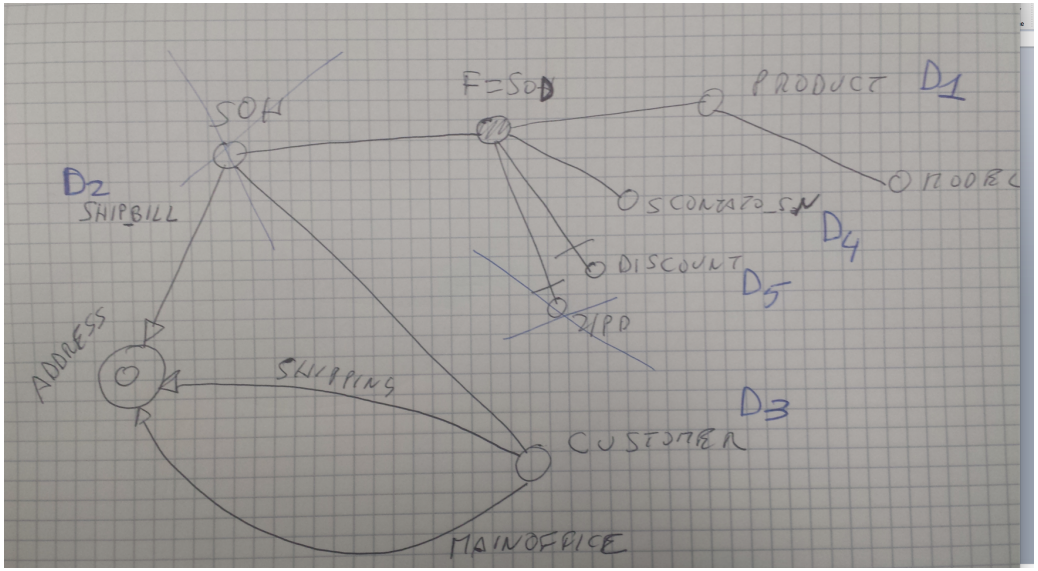
FATTO : SOD , che ha due chiavi : K1 = {P,SOH} e K2={SOD_ID,SOH}

Dimensioni

1) PRODUCT
2) SHIP_BILL : indirizzo SHIP_BILL dell'ordine
3) CUSTOMER

Sia per PRODUCT che per CUSTOMER non specifico se considerare il suo ID oppure il suo Name: è ovvio che se Name è chiave (alternativa) nelle analisi conviene usarlo come attributo dimensionale, mentre l'ID può destare come descrittivo.

ALBERO DEGLI ATTRIBUTI



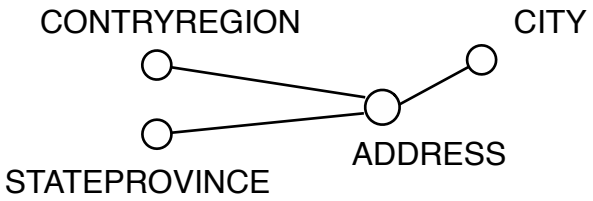
Durante la progettazione si decide di aggiungere anche le dimensioni SCONTATO_S_N e DISCOUNT.

Consideriamo ADDRESS

Inizialmente ADDRESS (inteso come identificatore della tabella ADDRESS)
ha CITY, COUNTRYREGION e STATEPROVINCE come figli (POSTALCODE viene considerato dopo), in quanto

ADDRESS(<u>ADDRESS</u>, CITY, COUNTRYREGION, STATEPROVINCE, …

Quindi

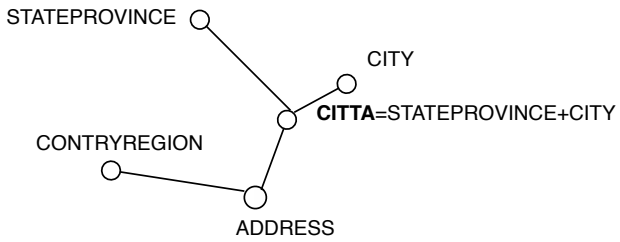


Con il data profiling è stata individuata la dipendenza funzionale

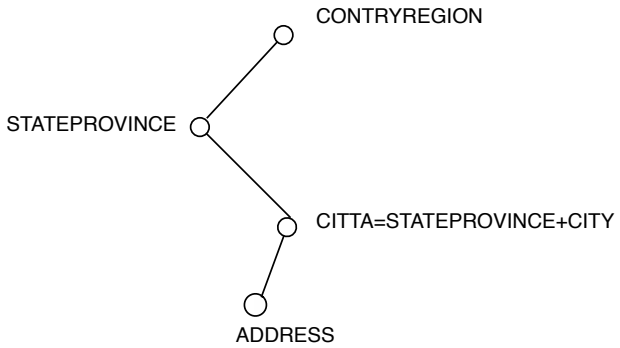
CITY, STATEPROVINCE → COUNTRYREGION

ed è stato verificato che *il solo valore* di CITY non determina né STATEPROVINCE né COUNTRYREGION, cioè **non vale che**
CITY → COUNTRYREGION
CITY → STATEPROVINCE
Quindi per individuare una *città* (per costruire un'anagrafica di città) dobbiamo considerare la coppia CITY, STATEPROVINCE.
Questo nuovo concetto, la *città*, intesa come CITY, STATEPROVINCE diventa un attributo dimensionale dell'albero degli attributi.
Può essere denotato sia come CITY, STATEPROVINCE sia come CITY + STATEPROVINCE (in quanto il nome delle città viene costruito concatenando il valore di CITY con quello di STATEPROVINCE e + è l'operatore di concatenazione tra stringhe in SQL)

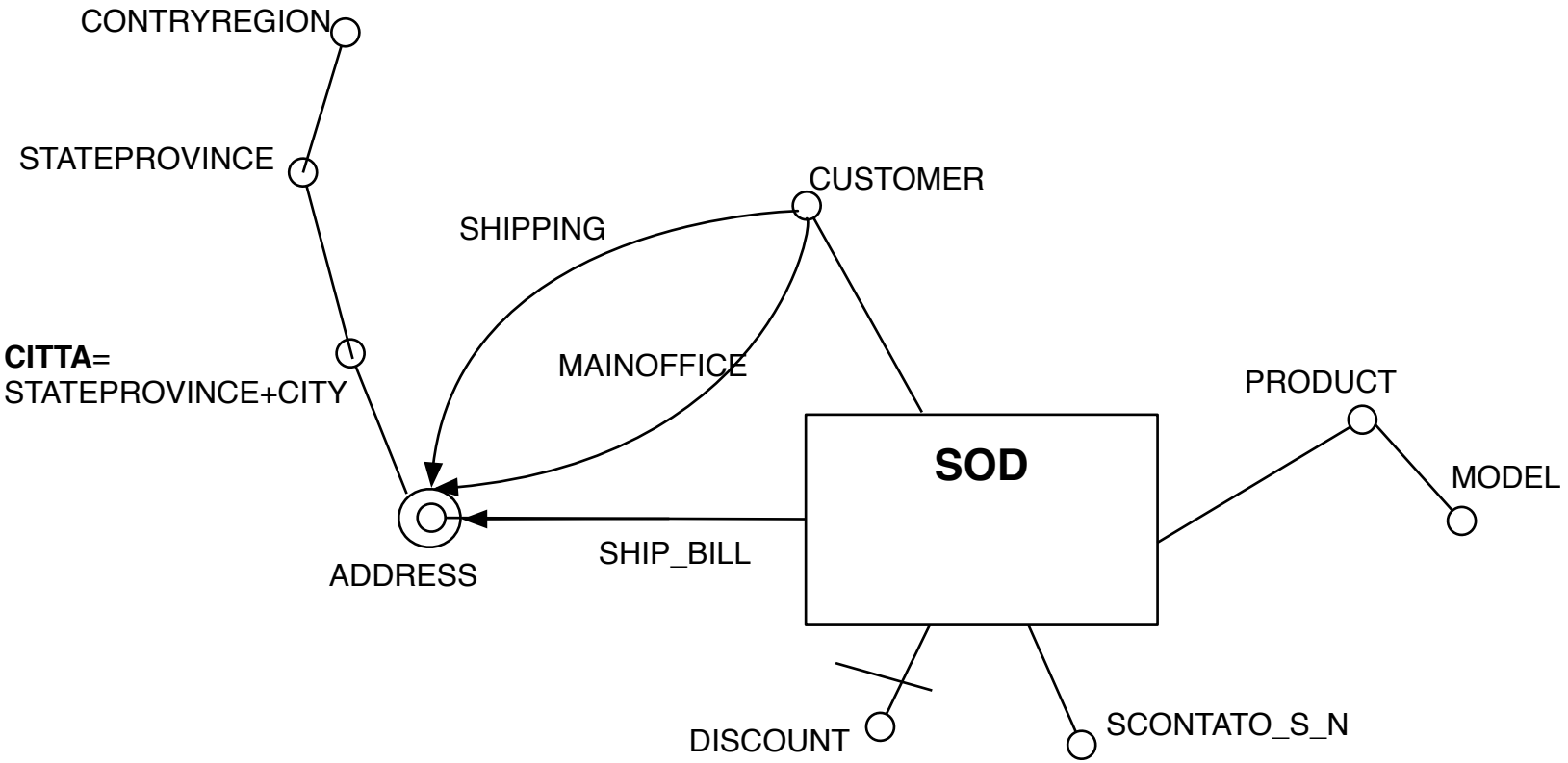
Si ottiene quindi



L’attributo CITY si può togliere come attributo dimensionale in quanto da solo non rappresenta niente di significativo.
Infine si aggiunge anche la dipendenza funzionale STATEPROVINCE → COUNTRYREGIONE



SCHEMA DI FATTO



Schema di Fatto TEMPORALE: nessuna delle due chiavi è inclusa tra le dimensioni, in quanto ho tolto SOH.

Dipendenze funzionali tra le dimensioni:

DISCOUNT → SCONTATO_S_N

CONDIVISIONE/CONVERGENZA :

SHIP_BILL che relazione ha con lo SHIPPING e MAINOFFICE address del customer?

Verrà trattato nella seconda consegna.

Quello che segue non è stato ancora svolto è verrà trattato mercoledì prossimo.

Dimensione SCONTATO_S_N

SCONTATO_S_N è un attributo dimensionale che non ho nello schema originale **ma devo *calcolare*.**

SCONTATO_S_N = 'SI' quando SOD è scontato, ovvero **quando un SOD è anche in SODD,** mentre è 'NO' negli altri casi

Per calcolarlo, ricordiamo che

SODD (SO_ID, SOD_ID : SOD, UPD, Discount)

Quindi devo fare un **SOD left join SODD**: quando un SOD è anche in SODD, e quindi SODD.SO_ID è non nullo, allora 'SI', altrimenti NO

Il calcolo viene effettuato tramite una query che *salvo* in una vista che posso chiamare appunto SCONTATO_S_N.
Tale vista avrà come *chiave* la chiave di SOD in quanto il valore di SCONTATO_S_N dipende dall'SOD

```
CREATE VIEW SCONTATO_S_N
AS
    SELECT SO_ID, SOD_ID,
        SCONTATO_S_N=CASE WHEN SODD.SO_ID IS NULL THEN 'NO' ELSE 'SI' END
    FROM SOD LEFT JOIN SODD ON (SOD.SO_ID= SODD.SO_ID  AND SOD.SOD_ID= SODD.SOD_ID)
```

Questo completa il calcolo di SCONTATO_S_N.

Siccome occorre considerare anche DISCOUNT come dimensione e occorre verificare che DISCOUNT → SCONTATO_S_N, tenuto conto che anche DISCOUNT dipende da SOD, riporto anche DISCOUNT nella vista SCONTATO_S_N.

Ho due possibilità:
modificare (ALTER) quella già creata, aggiungendo DISCOUNT

```
ALTER VIEW SCONTATO_S_N
AS
        SELECT SO_ID, SOD_ID,
                SCONTATO_S_N=CASE WHEN SODD.SO_ID IS NULL THEN 'NO' ELSE 'SI' END,
                DISCOUNT
        FROM SOD LEFT JOIN SODD ON (SOD.SO_ID= SODD.SO_ID  AND SOD.SOD_ID= SODD.SOD_ID)
```

Crearne una nuova

```
CREATE VIEW SCONTATO_S_N_E_DISCOUNT
AS
        SELECT SO_ID, SOD_ID,
                SCONTATO_S_N=CASE WHEN SODD.SO_ID IS NULL THEN 'NO' ELSE 'SI' END,
                DISCOUNT
        FROM SOD LEFT JOIN SODD ON (SOD.SO_ID= SODD.SO_ID  AND SOD.SOD_ID= SODD.SOD_ID)
```

Finalmente possiamo verificare DISCOUNT → SCONTATO_S_N (la vista è a tutti gli effetti una tabella …)

```
SELECT DISCOUNT
FROM  SCONTATO_S_N  --- OPPURE SCONTATO_S_N_E_DISCOUNT
GROUP BY DISCOUNT
HAVING COUNT(DISTINCT SCONTATO_S_N)>1
-- OK, è vuota
```

Vengono riportate nel seguito le query per il data profiling relativo a POSTALCODE (**importante**: vedere come effettuare il count distinct di una coppia di elementi)

Il risultato è il seguente: **nessuna** delle seguenti dipendenze funzionali è vera (per CITTA si intende CITY+STATEPROVINCE)

CITY → POSTALCODE
POSTALCODE → CITY
CITTA → POSTALCODE
POSTALCODE → CITTA

Quindi POSTALCODE resta come figlio di ADDRESS: ADDRESS → POSTALCODE.

```
SELECT     City,StateProvince
FROM SalesLT.Address
GROUP BY CITY,StateProvince
HAVING  count(distinct PostalCode) > 1
-- non vuota, quindi non vale StateProvince,CITY --> PostalCode
-- coe' non vale CITTA --> Postalcode

SELECT     City
FROM SalesLT.Address
GROUP BY CITY
HAVING  count(distinct PostalCode) > 1
-- non vuota, quindi non vale CITY --> PostalCode

SELECT     PostalCode
FROM SalesLT.Address
GROUP BY  PostalCode
HAVING  count(distinct CITY) > 1
-- non vuota, quindi non vale PostalCode --> CITY


-- resta da vedere se Postalcode --> CITTA
-- cioe' se   Postalcode --> CITY,StateProvince
 -- per fare il count(distinct CITY,StateProvince)
```

```
 -- devo rendere CITY,StateProvince un solo valore
 -- tramite concatenazione tra stringhe, cioe'
--- CITY + StateProvince



SELECT     PostalCode
FROM SalesLT.Address
GROUP BY  PostalCode
HAVING  count(distinct CITY+StateProvince) > 1
-- non vuota, quindi non vale PostalCode --> CITTA



SELECT     *
FROM AdventureWorksLT2008R2.SalesLT.Address
WHERE PostalCode IN (SELECT     PostalCode
FROM AdventureWorksLT2008R2.SalesLT.Address
GROUP BY  PostalCode
HAVING  count(distinct CITY) > 1)
```