Domenico Beneventano
Sonia Bergamaschi
Francesco Guerra
Maurizio Vincini

Progetto di Basi di Dati Relazionali

lezioni ed esercizi

Indice

Prefazione

L'obiettivo del volume è fornire al lettore le nozioni fondamentali di progettazione e di realizzazione di applicazioni di basi di dati relazionali.
Relativamente alla progettazione, vengono trattate le fasi di progettazione concettuale e logica e vengono presentati i modelli dei dati Entity-Relationship e Relazionale che costituiscono gli strumenti di base, rispettivamente, per la progettazione concettuale e la progettazione logica.
Viene inoltre introdotto lo studente alla teoria della normalizzazione di basi di dati relazionali.
Relativamente alla realizzazione, vengono presentati elementi ed esempi del linguaggio standard per RDBMS (Relational Database Management Systems) SQL. Ampio spazio è dedicato ad esercizi svolti sui temi trattati.

Il volume nasce dalla pluriennale esperienza didattica condotta dagli autori nei corsi di Basi di Dati e di Sistemi Informativi per studenti dei corsi di laurea e laurea specialistica della Facoltà di Ingegneria di Modena, della Facoltà di Ingegneria di Reggio Emilia e della Facoltà di Economia "Marco Biagi" dell'Università degli Studi di Modena e Reggio Emilia.
Il volume attuale estende notevolmente le edizioni precedenti arricchendo la sezione di progettazione logica e di SQL. La sezione di esercizi è completamente nuova, inoltre, ulteriori esercizi sono reperibili sul sito web `www.dbgroup.unimo.it/librobdati` in formato pdf (password: bdati2007). Come le edizioni precedenti, costituisce piú una *collezione di appunti* che un vero *libro* nel senso che tratta in modo *rigoroso* ma *essenziale* i concetti forniti. Inoltre, non esaurisce tutte le tematiche di un corso di Basi di Dati, la cui altra componente fondamentale è costituita dalla *tecnologia delle basi di dati*. Questa componente è, a parere degli autori, trattata in maniera eccellente da un altro testo di Basi di Dati (Lezioni di Basi di Dati [14]), scritto dai nostri colleghi e amici Paolo Ciaccia e Dario Maio dell'Università di Bologna.
Il volume, pure nella sua essenzialità, è ricco di esercizi svolti e quindi può costituire un ottimo strumento per gruppi di lavoro che, nell'ambito di software house, si occupino di progettazione di applicazioni di basi di dati relazionali.
Il libro si articola in 6 capitoli:

- *Progettazione Concettuale.* Vengono presentati i modelli concettuali dei dati, i loro meccanismi di astrazione fondamentali ed il modello Entity-Relationship. Vengono inoltre illustrate le metodologie per il progetto concettuale e l'attività di progettazione concettuale da requisiti in linguaggio naturale e da file esistenti. Questo capitolo riassume concetti fondamentali tratti da un testo in lingua inglese ormai storico Conceptual Database Design: an Entity-Relationship approach di Batini, Ceri, Navathe [10]. In questa nuova edizione sono stati approfonditi alcuni

concetti, quali i vincoli di integrità e la reificazione ed è stata trattata la documentazione di schemi E/R.

- *Elementi di Teoria Relazionale.* Viene presentato il modello Relazionale dei dati e l'Algebra Relazionale.

- *Progettazione Logica.* Vengono presentate le due fasi fondamentali di progettazione logica: di alto livello e relazionale. Questo capitolo estende i concetti fondamentali tratti da [10] e le edizioni precedenti.

- *Teoria della normalizzazione.* Vengono presentati gli elementi fondamentali della teoria della normalizzazione, il cui obiettivo è quello di produrre uno schema di basi di dati con proprietà di qualità. Per una trattazione più approfondita si rimanda alla sezione 7 del testo [37].

- *Elementi ed Esempi del Linguaggio SQL.* Vengono presentati gli elementi fondamentali del linguaggio standard per basi di dati relazionali *SQL*: definizione dei dati, interrogazioni, manipolazione dei dati, Viste, Privilegi e Cataloghi. Vengono inoltre introdotti in questa nuova edizione Embedded SQL, stored procedure e Trigger.

- *Esercizi.* Vengono presentati numerosi esercizi svolti di progettazione concettuale e logica, di algebra relazionale e SQL, di normalizzazione di schemi relazionali. Questa edizione è stata completamente riscritta rispetto alle edizioni precedenti.

In Appendice vengono riportati alcuni lucidi del seminario "An Introduction to Relational Data Base" tenuto nel 1976 da C. J. Date ai Laboratori IBM di San Jose, gentilmente concessi dal Prof. Paolo Tiberio, Preside della Facoltà di Ingegneria di Modena dellUniversità di Modena e Reggio Emilia.

Un ringraziamento a R.Carlos Nana Mbinkeu, Mirko Orsini, Laura Po e Antonio Sala, studenti della scuola di dottorato in Information and Communication Technologies dell'Università degli Studi di Modena e Reggio Emilia che hanno contribuito alla realizzazione tecnica della terza edizione di questo volume.
Un ringraziamento speciale va rivolto ai nostri studenti che hanno contribuito al miglioramento della didattica sulle Basi di Dati e quindi al nostro libro.

Capitolo 1

Progettazione Concettuale

L'obiettivo della *progettazione concettuale* è quello di rappresentare la realtà di interesse ad un alto livello di astrazione. In questo capitolo, dopo una breve introduzione dei principali meccanismi di astrazione dei modelli concettuali dei dati, verrà presentato il modello Entity-Relantionship (E/R), concepito da Chen nel 1976 e successivamente esteso con altre primitive di rappresentazione, che costituisce lo standard *de facto* per la progettazione concettuale delle applicazioni per basi di dati [9].

Nella prima parte del capitolo verranno illustrati i concetti fondamentali del modello E/R con la relativa rappresentazione grafica. Si mostrerà attraverso esempi come esprimere vincoli di integrità, ovvero condizioni che i dati devono soddisfare per riflettere una certa realtà.

La seconda parte del capitolo introduce una metodologia per il progetto concettuale, che definisce la costruzione di uno schema E/R come un processo incrementale: la nostra percezione della realtà è progressivamente raffinata e arricchita e lo schema concettuale è formalmente sviluppato.

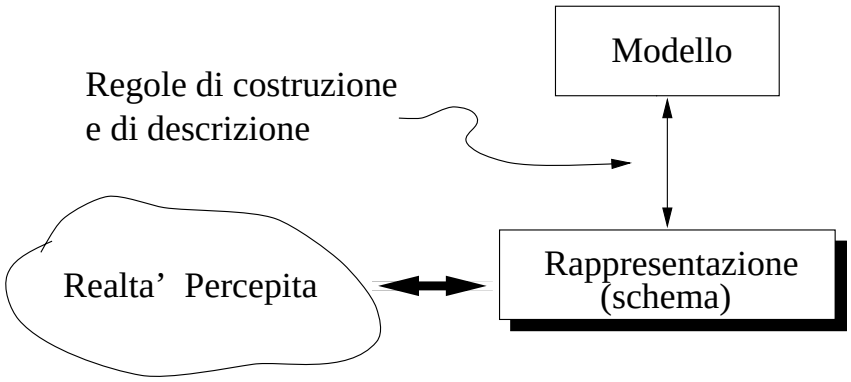
Nella terza parte del capitolo vengono trattati due casi particolarmente rilevanti di progettazione concettuale: progettazione da requisiti in linguaggio naturale e progettazione da file esistenti.

Nell'ultima parte del capitolo viene discusso come corredare uno schema E/R con una documentazione di supporto allo scopo di facilitare l'interpretazione dello schema stesso.

1.1 I modelli concettuali dei dati

♢ **modello**: l'insieme di regole e strutture che permettono la rappresentazione della realtà di interesse

♢ **schema**: la rappresentazione di una specifica realtà secondo un determinato modello



♢ il modello fornisce le regole per la costruzione della rappresentazione

♢ la rappresentazione è data da un insieme di simboli posti in corrispondenza con la realtà di interesse

♢ la realtà di interesse è una porzione di mondo reale così come è percepita da chi costruisce la rappresentazione

Astrazione

♢ è un procedimento mentale che si adotta quando si evidenziano alcune proprietà e caratteristiche di un insieme di oggetti

♢ altre proprietà, giudicate non rilevanti, vengono trascurate

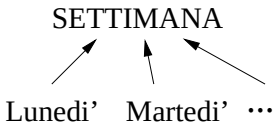
♢ primitive di astrazione comuni a tutti i modelli:

- **classificazione**
- **aggregazione**
- **generalizzazione**

Primitive di astrazione

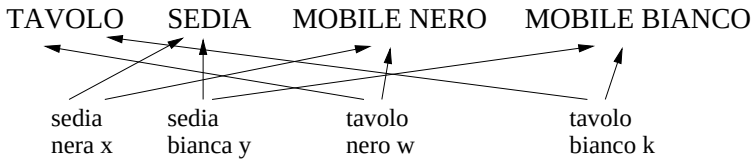
Classificazione: conduce alla definizione di una *classe* a partire da un insieme di oggetti caratterizzati da proprietà comuni.

♢ ESEMPIO: i membri della Classe SETTIMANA sono Lunedì, ..., Giovedì, ... Domenica.



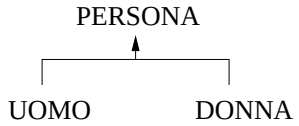
♢ relazione MEMBER_OF

- esiste tra un elemento della classe e la classe stessa
  Giovedì è MEMBER_OF SETTIMANA
- gli stessi oggetti possono essere classificati in modi diversi



Generalizzazione: definisce una relazione di sovrainsieme tra una classe padre e altre classi figlie (sottoclassi).

♢ ESEMPIO: la classe PERSONA è una generalizzazione delle classi UOMO e DONNA



♢ la generalizzazione stabilisce una corrispondenza tra gli elementi di una sottoclasse e gli elementi della superclasse

GENERALIZZAZIONE

♢ Stabilisce un mapping tra gli elementi di una classe e gli elementi delle classi generalizzate. Per tale mapping si definiscono le proprietà di copertura:

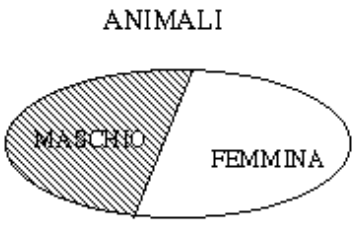
♢ **Copertura totale o parziale**

- **totale (t):** ogni elemento della classe generica è in relazione con almeno un elemento delle classi generalizzate;
- **parziale (p):** esistono alcuni elementi della classe generica che non sono in relazione con alcun elemento delle classi generalizzate;

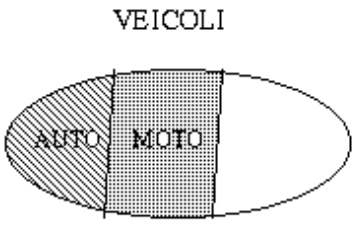
♢ **Copertura esclusiva o sovrapposta**

- **esclusiva (e):** ogni elemento della classe generica è in relazione con al massimo un elemento delle classi generalizzate;
- **sovrapposta (o):** esistono alcuni elementi della classe generica che sono in relazione con elementi di due o più classi generalizzate.

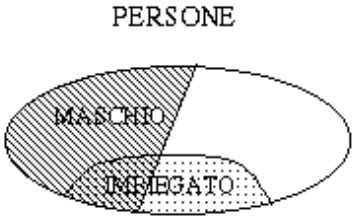
♢ **Esempi**



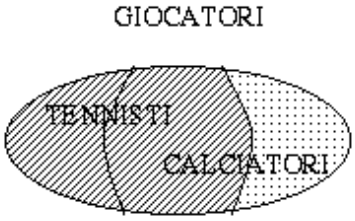
totale, esclusiva



parziale, esclusiva



parziale, sovrapposta



totale, sovrapposta

AGGREGAZIONE

Aggregazione: permette di giungere alla definizione di un concetto a partire da altri concetti che ne rappresentano le parti componenti.

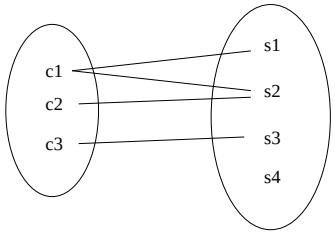
♢ ESEMPIO: il concetto BICICLETTA è la classe i cui componenti sono RUOTA, PEDALE e MANUBRIO

♢ relazione **PART_OF** : esiste tra un concetto componente ed il concetto composto; ad esempio, PEDALE è PART-OF BICICLETTA

♢ l'aggregazione stabilisce una corrispondenza (mapping) tra gli elementi di una classe componente e gli elementi della classe composta

Aggregazione binaria: : è una corrispondenza stabilita tra due classi.

♢ ESEMPIO: ESAME è una aggregazione binaria delle classi CORSO e STUDENTE, e stabilisce una corrispondenza tra le due classi
Rappresentazione grafica



CARDINALITÀ della partecipazione alla corrispondenza

♢ sia A aggregazione di C_1 e C_2

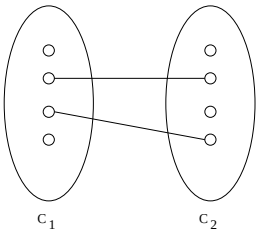
- *cardinalità minima di C_1 in A* : min-card(C_1,A)
  è il minimo numero di corrispondenze nell'aggregazione A alle quali ogni membro di C_1 deve partecipare
- *cardinalità massima di C_1 in A* : max-card(C_1,A)
  è il massimo numero di corrispondenze nell'aggregazione A alle quali ogni membro di C_1 può partecipare

- *partecipazione opzionale* : min-card(C_1,A) = 0
  alcuni elementi di C_1 possono non essere aggregati tramite A a elementi di C_2

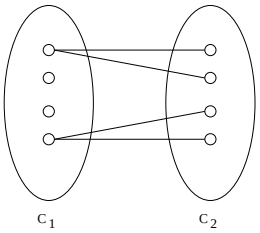
- *partecipazione obbligatoria* : min-card(C_1,A) > 0
  ad ogni elemento di C_1 deve essere aggregato, tramite A, almeno un elemento di C_2

Cardinalità della corrispondenza

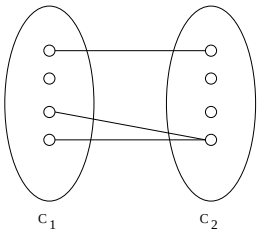
♢ *Uno a uno* (One-to-one)
max-card(C_1,A) = 1
max-card(C_2,A) = 1



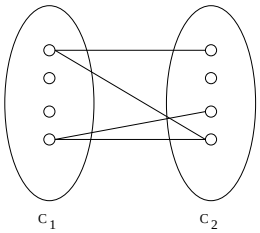
♢ *Uno a molti* (One-to-many)
max-card(C_1,A) = n
max-card(C_2,A) = 1



♢ *Molti a uno (simmetrica)* (Many-to-one)
max-card(C_1,A) = 1
max-card(C_2,A) = n

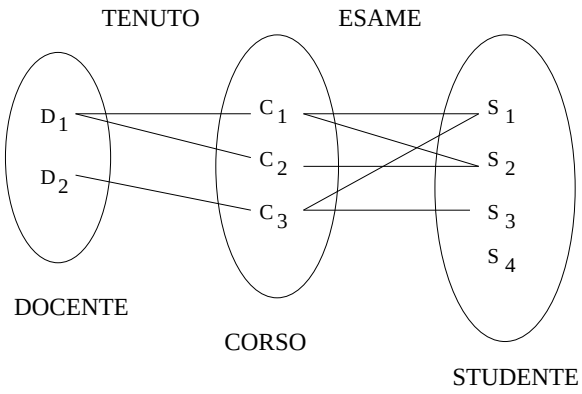


♢ *Molti a molti* (Many-to-many)
max-card(C_1,A) = n
max-card(C_2,A) = n



NB: max-card(C,A) = n significa che il massimo numero di partecipazioni di un elemento di C_1 in A è limitato soltanto dal numero di elementi in C_2.

Esempi di aggregazione binaria



- per ogni corso c'è stato almeno un esame:
  min-card(CORSO,ESAME)=1
- per ogni corso c'è un numero generico di esame:
  max-card(CORSO,ESAME)=n
- alcuni studenti non hanno sostenuto esami:
  min-card(STUDENTE,ESAME)=0
- uno studente può sostenere più esami:
  max-card(STUDENTE,ESAME)=n

- un corso è tenuto da uno ed un solo docente:
  min-card(CORSO,TENUTO)=1
  max-card(CORSO,TENUTO)=1
- un docente tiene almeno un corso:
  min-card(DOCENTE,TENUTO)=1
- un docente può tenere un generico numero di corsi:
  max-card(DOCENTE,TENUTO)=n

  - card(CORSO,ESAME) = (1,n)
  - card(STUDENTE,ESAME) = (0,n)
  - card(CORSO,TENUTO) = (1,1)
  - card(DOCENTE,TENUTO) = (1,n)

♢ Si possono avere due o più aggregazioni tra la stessa coppia di concetti

Ad esempio, tra i concetti CORSO e STUDENTE si può stabilire l'ulteriore aggregazione SEGUE

Aggregazione n-aria

♢ È stabilita tra tre o più classi C_1,C_2, ... ,C_n

- *cardinalità minima di C_i in A* : min-card(C_i,A)
  è il minimo numero di corrispondenze nell'aggregazione A alle quali ogni membro di C_i deve partecipare
- *cardinalità massima di C_i in A* : max-card(C_i,A)
  è il massimo numero di corrispondenze nell'aggregazione A alle quali ogni membro di C_i può partecipare

♢ **Esempio:**

ORARIO_SETTIMANALE è una aggregazione ternaria delle classi CORSO, GIORNO e AULA

- ogni corso può tenersi da 1 a 5 volte la settimana:
  min-card(CORSO,ORARIO_SETTIMANALE) = 1
  max-card(CORSO,ORARIO_SETTIMANALE) = 5

- in ogni giorno della settimana si può tenere un qualsiasi numero di lezioni:
  min-card(GIORNO,ORARIO_SETTIMANALE) = 0
  max-card(GIORNO,ORARIO_SETTIMANALE) = n

- ogni aula ospita al massimo 40 lezioni per settimana:
  min-card(AULA,ORARIO_SETTIMANALE) = 0
  max-card(AULA,ORARIO_SETTIMANALE) = 40

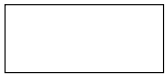
  - card (CORSO,ORARIO_SETTIMANALE) = (1,5)
  - card (GIORNO,ORARIO_SETTIMANALE) = (0,n)
  - card (AULA,ORARIO_SETTIMANALE) = (0,40)

1.2 Il Modello Entity Relationship (E/R)

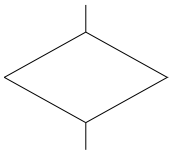
È stato sviluppato da Chen nel 1976 (P.P. Chen, "The Entity-Relantionship: Toward a Unified View of Data", ACM Transactions on Database Systems 1, no. 1, 9-37, March 1976).
È stato poi esteso nell'ambito della metodologia di progettazione di basi di dati DATAID sviluppata nel progetto italiano omonimo (S. Ceri, "Methodology and Tools for Data Base Design", North-Holland, 1983)

Elementi di base

ENTITÀ: Una entità rappresenta un insieme di oggetti della realtà di cui si individuano proprietà comuni[1].
Rappresentazione grafica:

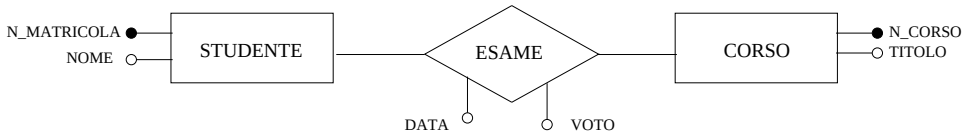


ASSOCIAZIONE: Una associazione rappresenta un legame logico tra due o più entità.
Rappresentazione grafica:



ATTRIBUTO: L'attributo rappresenta proprietà elementari di entità o associazioni.
Rappresentazione grafica:

ESEMPIO:



♢ STUDENTE e CORSO sono entità.

♢ ESAME è una associazione tra STUDENTE e CORSO.

♢ N_MATRICOLA e NOME sono attributi di STUDENTE.

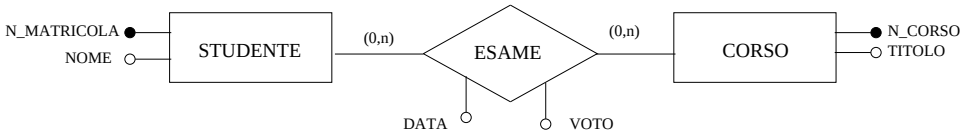
♢ DATA e VOTO sono attributi di ESAME.

♢ N_CORSO e TITOLO sono attributi di CORSO.

[1] entità, associazione corrispondono in realtà ad insiemi di entità, insiemi di associazioni che hanno proprietà comuni; per semplicità vengono indicate come entità e associazioni.

ASSOCIAZIONI

♢ Le associazioni sono caratterizzate in termini di cardinalità.



♢ uno studente può aver sostenuto zero, uno o più esami; con cardinalità minima 0; si afferma che uno studente esiste anche se non ha sostenuto esami:

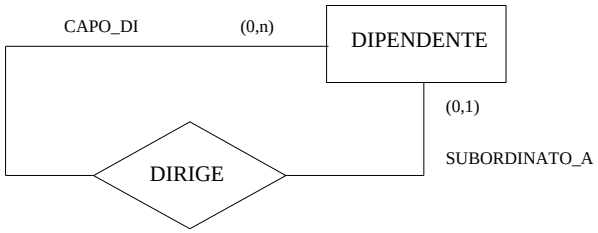
$$\text{min-card(STUDENTE,ESAME)} = 0$$
$$\text{max-card(STUDENTE,ESAME)} = n$$
$$\Downarrow$$
$$\text{card(STUDENTY,ESAME)} = (0,n)$$

♢ un corso può esistere indipendentemente dal fatto che vi siano o meno studenti che abbiano già sostenuto esami relativi a quel corso:

$$\text{min-card(CORSO,ESAME)} = 0$$
$$\text{max-card(CORSO,ESAME)} = n$$
$$\Downarrow$$
$$\text{card(CORSO,ESAME)} = (0,n)$$

ANELLI: Un anello è una associazione binaria tra un'entità e se stessa. Il *ruolo* di un'entità è indicato tramite una *label*.

ESEMPIO:
DIRIGE connette i capi ai subordinati, entrambi istanze dell'entità DIPENDENTE (ruoli CAPO_DI e SUBORDINATO_A).



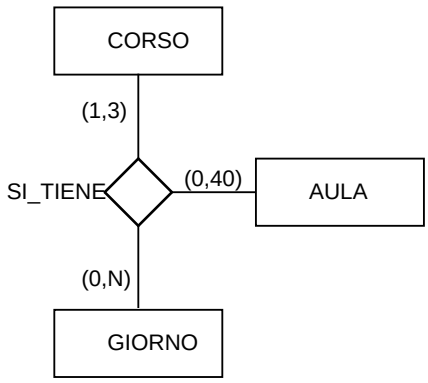
♢ Ogni capo può dirigere (CAPO-DI) più dipendenti.

♢ Ogni dipendente è direttamente SUBORDINATO_A un solo capo.

ASSOCIAZIONI n-arie

Una associazione n-aria connette più di due entità.

Esempio:
SI_TIENE è una associazione ternaria che connette le entità CORSO, GIORNO e AULA, con le seguenti cardinalità:



card(CORSO,SI_TIENE)= (1,3)
card(GIORNO,SI_TIENE)= (0,n)
card(AULA,SI_TIENE)=(0,40)

ATTRIBUTI

Dominio dell'attributo: insieme di valori legali per l'attributo.

Un attributo è detto semplice se è definito su un solo dominio.

Sia E un'entità o un'associazione e A un attributo di E:

Cardinalità minima: min-card(A,E)
è il minimo numero di valori dell'attributo associati a ogni istanza dell'entità o associazione E.

Cardinalità massima: max-card(A,E)
è il massimo numero di valori dell'attributo associato a ogni istanza dell'entità o associazione E.

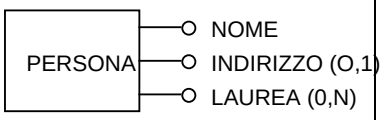
opzionale: min-card(A,E) = 0
può essere non specificato il valore dell'attributo

obbligatorio: min-card(A,E) = 1
almeno un valore dell'attributo deve essere specificato

valore-singolo: max-card(A,E) = 1

valore-multiplo: max-card(A,E) > 1

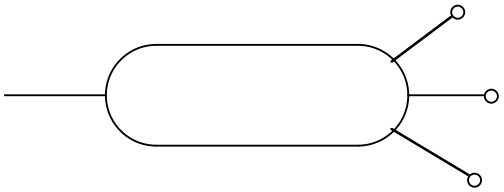
- una persona ha esattamente un nome
- una persona può avere al massimo un indirizzo
- una persona può avere più lauree



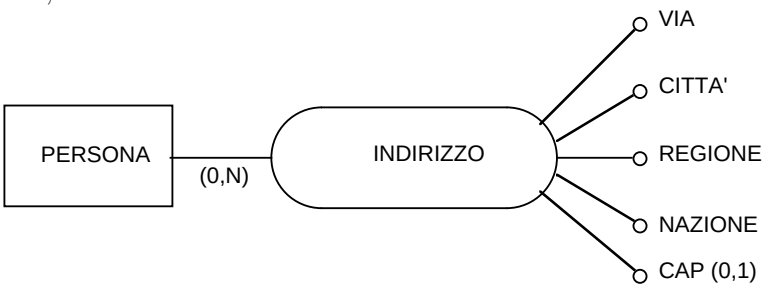
ATTRIBUTI COMPOSTI

♢ Un attributo composto è costituito da un gruppo di attributi che hanno affinità nel significato o nell'uso.

Rappresentazione grafica:



Esempio: INDIRIZZO denota il gruppo di attributi VIA, CITTA, REGIONE, NAZIONE e CAP.



Si noti la maggiore capacità espressiva:

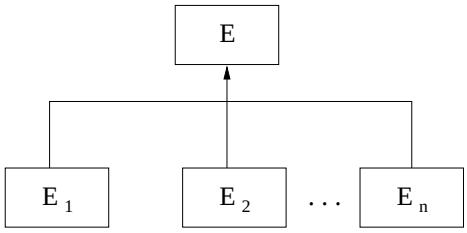
♢ Con la cardinalità sull'attributo composto INDIRIZZO si dice che una persona può avere più indirizzi, ciascuno dei quali composto da via, città, regione, nazione e c.a.p (opzionale).

♢ Se invece si fossero usati cinque attributi semplici si poteva solo stabilire la cardinalità di ciascuno di essi indipendentemente da quella degli altri.

GERARCHIE DI GENERALIZZAZIONE

Un'entità E è una **generalizzazione** di un gruppo di entità E_1, E_2, ..., E_n se ogni oggetto delle classi E_1, E_2, ..., E_n è anche un oggetto della classe E.

Rappresentazione grafica:



Proprietà di copertura della generalizzazione:

♢ totale ed esclusiva: **(t,e)**

♢ parziale ed esclusiva: **(p,e)**

♢ parziale e sovrapposta: **(p,o)**

♢ totale e sovrapposta: **(t,o)**

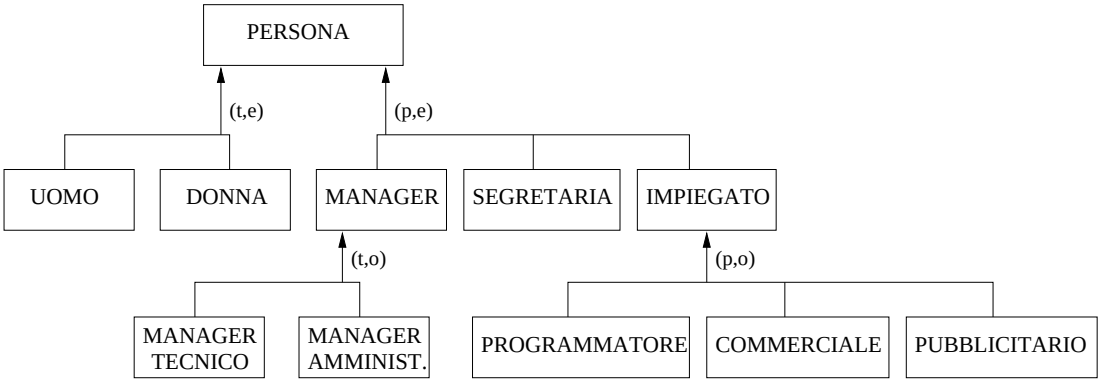
Esempio:

♢ la generalizzazione delle persone basata sul sesso è **(t,e)**;

♢ vi sono persone che non sono nè impiegati, nè segretari e nè manager: la generalizzazione basata sull'impiego è **(p,e)**;

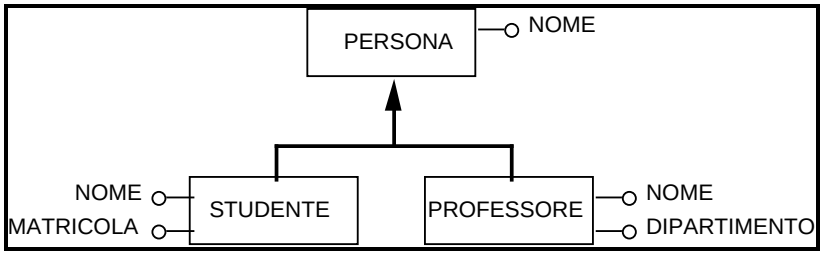
♢ gli impiegati possono avere più di un lavoro, anche diverso da quelli rappresentati in figura: tale generalizzazione è **(p,o)**;

♢ tutti i manager ricoprono il ruolo tecnico e/o amministrativo: la generalizzazione basata sul ruolo manageriale è **(t,o)**;

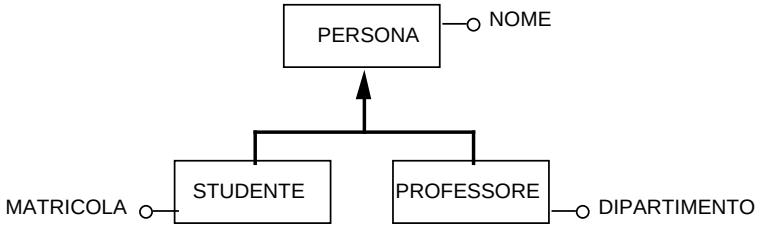


Ereditarietà delle proprietà

♢ Nell'astrazione di generalizzazione tutte le proprietà dell'entità generica sono **ereditate** dalle entità generalizzate. Nel modello E/R ogni attributo, associazione e generalizzazione definita per l'entità generica E è **ereditata automaticamente** da tutte le entità generalizzate E_1, E_2, ..., E_n.



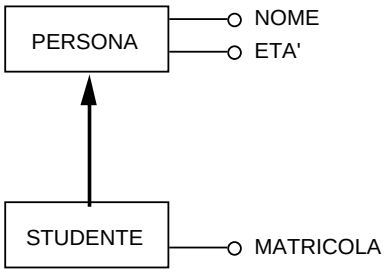
L'attributo NOME della classe PERSONA è anche attributo delle classi STUDENTE e PROFESSORE, pertanto essi possono essere eliminati da tali classi ottenendo il seguente schema semplificato:



SUBSET: Un subset è una gerarchia di generalizzazione con una sola entità generalizzata.
La copertura di un subset è **parziale**.

Esempio:

STUDENTE ha, oltre agli attributi ereditati da PERSONA, l'attributo addizionale MATRICOLA



IDENTIFICATORI

Un identificatore di un'entità E è una collezione di attributi o di entità in associazione con E che individua in modo univoco tutte le istanze di E.
Definizione formale di identificatore Sia E un'entità e siano

- A_1, ..., A_n: attributi a **valore singolo** ed **obbligatori** per E
- E_1, ..., E_m: entità diverse da E e connesse ad E tramite associazioni binarie R_1, ..., R_m **obbligatorie** (min-card(E,R_i)=1) e **one-to-one** o **many-to-one** (max-card(E,R_i)=1)

Possibili identificatori $I = \{A_1, ..., A_n, E_1, ..., E_m\},\ n \geq 0, m \geq 0, n+m \geq 1.$

Valore dell'identificatore di un'istanza di E: l'insieme di tutti i valori degli attributi A_i, i=1, ..., n e di tutte le istanze E_j,j=1, ..., m.

I è un identificatore di E se:

1. non ci sono due istanze di E con lo stesso valore dell'identificatore;
2. eliminando un attributo A_i oppure un'entità E_j da I, la proprietà 1. non è più valida.

♢ Per le ipotesi fatte sulla cardinalità degli attributi e delle entità nelle associazioni che costituiscono un identificatore, il valore di un identificatore è sempre ben definito. Cioè per ogni istanza E, esiste **uno ed un solo** valore dell'attributo A_i oppure **una ed una sola** istanza dell'entità E_j.

Classificazione degli identificatori:

♢ **semplice** se n+m=1; **composto** se n+m>1.

♢ **interno** se m=0; **esterno** se n=0.

♢ **mixed** se n>0 e m>0.

♢ Ogni entità deve avere almeno un identificatore.

Gli identificatori interni sono preferibili rispetto a quelli esterni.

Gli identificatori semplici sono preferibili rispetto a quelli composti.

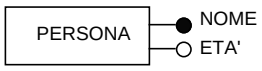
♢ È importante evitare circolarità nella definizione degli identificatori: se l'entità E_j è usata nell'identificazione dell' entità E_i, allora E_i non può essere usata nell'identificazione di E_j.

♢ **entità forte**: ha almeno un identificatore interno.

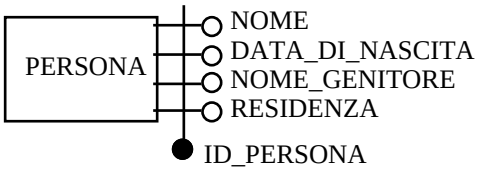
♢ **entità debole**: non ha alcun identificatore interno.

Esempi di identificatori

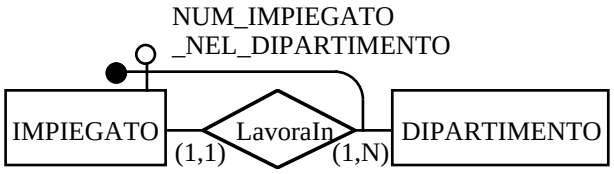
a) identificatore semplice ed interno I={NOME}



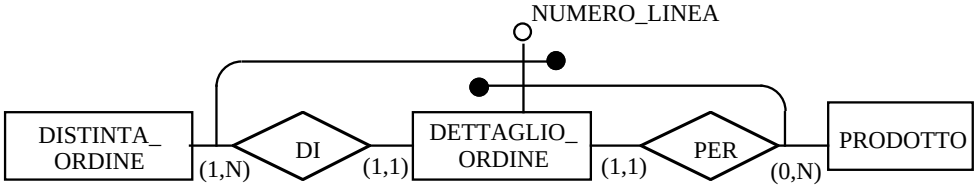
b) identificatore composto ed interno
I = {NOME, DATA_DI_NASCITA, NOME_GENITORE, RESIDENZA}
(il nome dell'identificatore, ID_PERSONA è opzionale)



c) identificatore composto e mixed
I = { NUM_IMPIEGATO_NEL_DIPARTIMENTO, DIPARTIMENTO }



♢ **identificatori per l'entità** DETTAGLIO_ORDINE



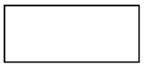
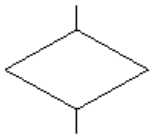
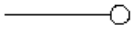
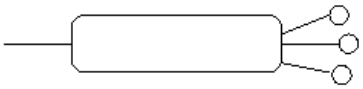
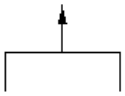
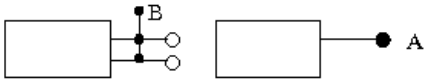
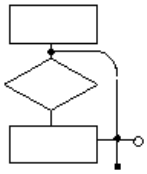
Si assume che due istanze dell'entità DETTAGLIO_ORDINE non possano riferirsi alla stessa istanza dell'entità PRODOTTO e alla stessa istanza dell'entità DISTINTA_ORDINE. In altre parole, si assume che in una distinta d'ordine, non può comparire due volte lo stesso prodotto.

$$\Downarrow$$

card(DETTAGLIO_ORDINE,DI) = (1,1)
card(DETTAGLIO_ORDINE,PER) = (1,1)

A) composto e esterno : I = { DISTINTA_ORDINE, PRODOTTO }
B) composto e mixed : I = { DISTINTA_ORDINE, NUM_LINEA }

SIMBOLI GRAFICI DEL MODELLO E/R

Concetto	**Rappresentazione Grafica**
Entita'	
Associazione	
Attributo	
Attributo composto	
Gerarchia di generalizzazione	
Subset	
Identificatore	
Identificatore mixed	

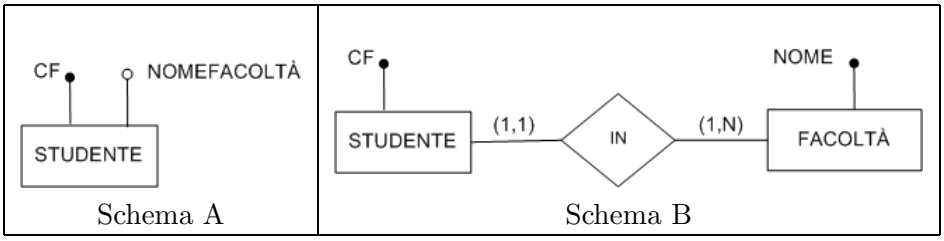
Attributo vs Associazione (1)

♢ Nel modello E/R ci sono due costrutti per descrivere le proprietà di un'entità:

- **attributo:** per descrivere proprietà *elementari*
- **associazione:** per descrivere relazioni con altre entitá.

♢ Nel seguito è discusso, in maniera intuitiva tramite degli esempi, quando conviene usare un attributo e quando conviene usare un'associazione.

♢ Entrambi gli schemi seguenti rappresentano che "Uno studente è descritto da un codice fiscale (CF) univoco e da una facoltà (identificata dal nome)" :



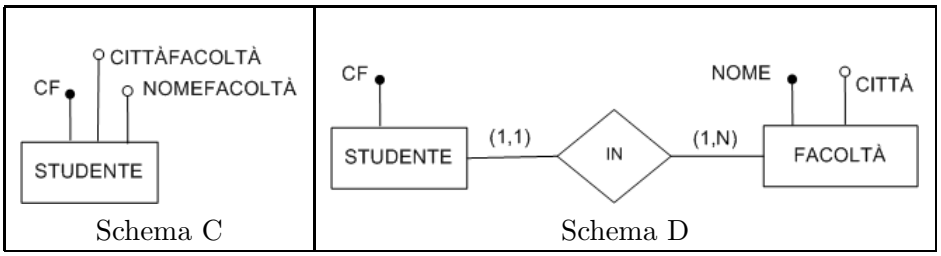
Schema A Schema B

♢ I due schemi sono *equivalenti*, ovvero rappresentano la stessa informazione; è da preferire lo schema A in quanto più semplice.

♢ Si noti che nello schema A, `NOMEFACOLTÀ` non può essere identificatore di `STUDENTE`, altrimenti in una facoltà ci sarebbe un solo studente. Cioè, mettere `NOMEFACOLTÀ` come identificatore di `STUDENTE` nello schema A equivale a mettere max-card`(FACOLTÀ,IN)=1` nello schema B.

Attributo vs Associazione (2)

♢ Nell'esempio precedente, lo schema A può considerarsi come una semplificazione dello schema B : tale semplificazione è possibile in quanto nell'entità FACOLTÀ tutti gli attributi fanno parte dell'identificatore.
Infatti supponiamo che della facoltà debba essere descritto, oltre al nome (identificativo), anche la città e quindi consideriamo i seguenti due schemi:



Schema C Schema D

♢ Lo schema C con una sola entità comporta una *ridondanza*:
per tutti gli studenti della stessa facoltà devo ripetere la città di tale facoltà.
Il concetto di ridondanza nei dati verrà definita formalmente e studiata nel modello relazionale tramite il concetto di *Forme Normali*; tale concetto verrà quindi esteso anche agli schemi E/R (vedi pagina 239).

♢ Possiamo concludere questo discorso introducendo una semplice regola per decidere se usare un attributo oppure un'entità per descrivere un oggetto:
se un oggetto (es. FACOLTÀ) ha un identificatore ed altre proprietà allora deve essere descritto da un'entità.

1.3 Vincoli di Integrità

♢ Per riflettere una certa realtà è necessario che i dati rispettino determinate condizioni, note come *vincoli di integrità*. In altri termini, un vincolo di integrità è una condizione che deve essere soddisfatta dalle istanze di una base di dati.

♢ I vincoli di integrità esprimibili variano in base al modello o linguaggio utilizzato per descrivere i dati.

♢ Nel modello E/R i vincoli di integrità esprimibili possono essere classificati nelle seguenti categorie:

- *vincoli di cardinalità* (per le associazioni e gli attributi) che abbiamo già discusso nelle sezioni precedenti;
- *vincoli di copertura* (per le generalizzazioni): la definizione delle proprietà di copertura per una generalizzazione su una entità E implica delle condizioni sulle istanze di E e sulle istanze delle entità generalizzate, come già discusso nelle sezioni precedenti;
- *vincoli di identificazione* (per le entità): la definizione di un identificatore per un'entità implica una condizione di unicità dei valori dell'identificatore.

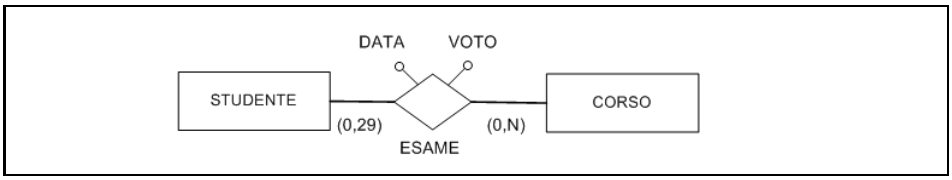
♢ Il concetto di vincolo di identificazione mette in evidenza il ruolo duale di un identificatore: da una parte esso serve per identificare in maniera univoca le istanze di una entità, dall'altra impone condizioni sulle istanze di un'entità. Ad esempio, consideriamo un'entità `VIAGGIO` con attributi `DATA`, `PERSONA` e `LUOGO`; dire che l'insieme di attributi `(DATA,PERSONA)` è un identificatore di `VIAGGIO` comporta che:

1. un viaggio possa essere identificato in base alla coppia di valori degli attributi `(DATA,PERSONA)`;
2. le istanze dell'entità `VIAGGIO` soddisfino la condizione di unicità sulla coppia di valori degli attributi `(DATA,PERSONA)`;
   quindi l'identificatore impone il seguente vincolo di identificazione: "una persona, in una certa data, può fare un solo viaggio".

D'altra parte, rappresentando `VIAGGIO` non come un'entità ma come un'associazione tra le entità `PERSONA` e `LUOGO` e con l'attributo `DATA`, il vincolo di integrità "una persona, in una certa data, può fare un solo viaggio" non può essere direttamente espresso come vincolo di identificazione in quanto `VIAGGIO` è un'associazione. In questi casi è utile *reificare* l'associazione, come discuteremo a pagina 24.

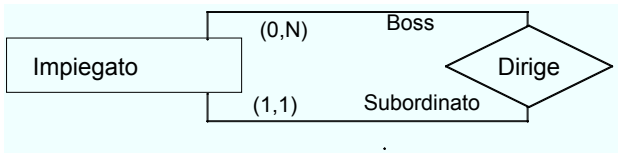
Osservazioni ed esempi di vincoli di integrità

♢ Data la seguente associazione binaria:



- Consideriamo il vincolo di integrità "Uno studente non può sostenere due o più volte un esame per lo stesso corso" e mostriamo come esso sia già espresso dall'associazione binaria `ESAME`. A tale scopo, consideriamo un'istanza:
  - *Istanza* di `Studente` $= \{s_1, s_2, s_3\}$
  - *Istanza* di `Corso` $= \{c_1, c_2\}$
  - *Istanza* di `Esame` $= \{e_1 = (s_3, c_1), e_2 = (s_3, c_2)\}$
- L'istanza di `Esame` è un insieme e pertanto non può essere inserita $e_3 = (s_3, c_2)$, ovvero non posso asserire che lo studente s_3 sostiene due volte l'esame per il corso c_2: lo schema E/R include il vincolo richiesto.

♢ Data la seguente associazione binaria ricorsiva o *Anello*:



- Consideriamo il vincolo di integrità: "un impiegato non può essere boss di se stesso" e ci chiediamo se esso è già espresso o è esprimibile nello schema E/R.
  A tale scopo, consideriamo un'istanza dello schema
  - *Istanza* di `Impiegato` $= \{i_1, i_2, i_3\}$
  - *Istanza* di `Dirige` $= \{d_1 = (i_1, i_2), d_2 = (i_1, i_3)\}$

♢ Dopo aver osservato che in un'ennupla è rilevante la posizione ((i_1, i_2) è diversa da (i_2, i_1)) assumiamo che in (i_1, i_2) il primo elemento i_1 sia nel ruolo di `BOSS`, ovvero (i_1, i_2) stabilisce che i_1 è il boss di i_2.

♢ Nell'istanza di `Dirige` posso inserire l'ennupla (i_1, i_1) per asserire che i_1 è il boss di i_1. Questo mostra che nello schema *non* è espresso il vincolo di integrità "un impiegato non può essere boss di se stesso".

♢ Questa tipologia di vincoli di integrità non può essere espressa con il modello E/R.

1.3.1 Reificazione

♢ Per reificazione si intende la rappresentazione di un'associazione in una forma equivalente tramite un'entità:

Sia A un'associazione tra $E_1, E_2, , E_n$. La *reificazione* di A è un entità, che continueremo ad indicare con A e chiameremo A reificata, collegata con ciascuna delle entità E_i che partecipano all'associazione A, tramite un'associazione binaria uno-a-molti A_E_i con

- $card(A, A\_E_i) = (1, 1)$ e
- $card(E_i, A\_E_i) = card(E_i, A)$.

♢ Come definire l'identificatore dell'associazione reificata?

- Nel modello E/R per le associazioni non è definito il concetto di identificatore: le associazioni sono identificate dalle entità che vi partecipano.

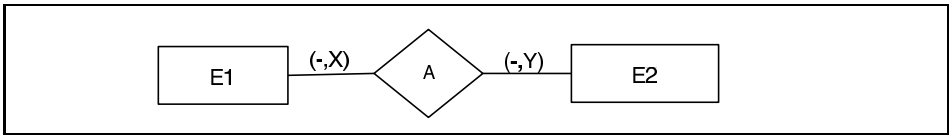
- Sulla base del fatto che l'insieme di istanze di un'associazione è una relazione matematica e che quindi non può contenere ennuple ripetute, possiamo definire l'identificatore di un'associazione reificata come segue:

Sia A un'associazione tra $E_1, E_2, , E_n$. Ogni E_i tale che *max-card*$(E_i, A) = 1$ è un identificatore di A reificata; altrimenti (cioè per ogni E_i si ha che *max-card*$(E_i, A) > 1$) A ha un unico identificatore costituito dalle entità $E_1, E_2, , E_n$.

♢ La reificazione è sempre possibile; tuttavia, lo schema con associazione è più semplice e sintetico di quello equivalente con reificazione e quindi è da preferire a quest'ultimo, effettuando la reificazione solo quando necessario.

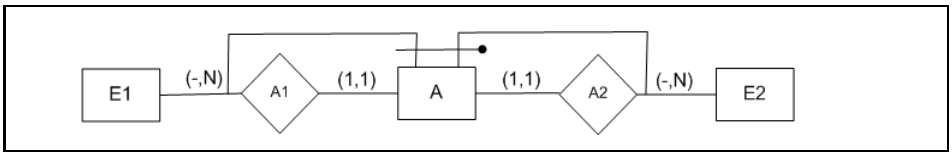
Esempi di reificazione

♢ Mostriamo la reificazione di una associazione binaria al variare della cardinalità massima di partecipazione delle due entità; la cardinalità minima non viene considerata in quanto non influisce sulla reificazione dell'associazione.

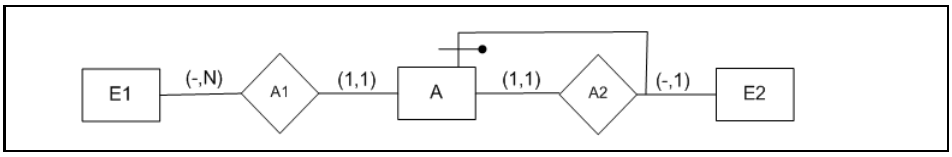


Si hanno i seguenti tre casi:

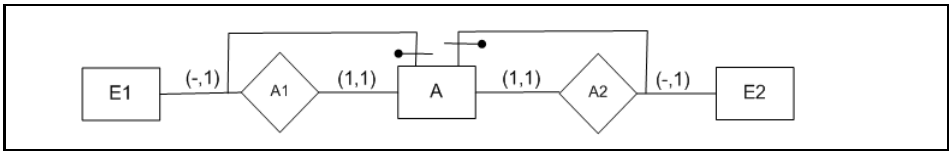
- Associazione binaria molti-a-molti ($X = N$ e $Y = N$):



- Associazione binaria molti-a-uno ($X = N$ e $Y = 1$):



- Associazione binaria uno-a-uno ($X = 1$ e $Y = 1$):



1.3.2 Esempi di schema E/R con vincoli di integrità

♢ **Esempio 1**

In questo primo esempio, si vogliono rappresentare gli esami che gli studenti sostengono per i vari corsi, riportandone la data e il voto. Uno studente può sostenere fino ad un massimo di 29 esami. Nessuna limitazione per il numero di esami registrabili per un corso. Verranno imposti di volta in volta i seguenti vincoli di integrità differenti e si discuterà quindi come essi siano riportati in E/R.

1. Per un dato studente e un dato corso può essere registrato un unico esame, con il relativo voto e la relativa data.
2. Per un dato studente e un dato corso possono essere registrati più esami, ciascuno con il relativo voto e la relativa data.
3. Per un dato studente e un dato corso possono essere registrati più esami, ma in date differenti. Il vincolo è equivalente al seguente: per un dato studente, un dato corso e una certa data, può essere registrato un unico esame, con il relativo voto.
4. Uno studente non può registrare due o più esami nella stessa data, cioè per una certa data e per un certo studente si può registrare un unico esame (relativo ad un preciso corso).
5. Modellare il vincolo espresso dal punto 4 e quello dal punto 1.

♢ **Esempio 2**
Si vuole rappresentare l'orario settimanale delle lezioni dei corsi, riportando l'indicazione dell'aula e dell'orario. L'orario è rappresentato a livello di singola *ora* e quindi trattandosi di un orario settimanale, con il concetto di orario si intenderà un'ora di un giorno della settimana, ad esempio `Lunedì, 10`.
L'orario settimanale verrà rappresentato con un'associazione tra i corsi, le aule e gli orari; i vincoli di integrità per le cardinalità di tale associazione sono:

- ogni corso si tiene da tre a cinque volte la settimana;
- in un aula si tiene, durante la settimana, almeno un corso (ovvero si vogliono memorizzare solo le aule in cui si tiene un corso);
- in un certo orario si tiene almeno un corso (ovvero si vogliono memorizzare solo gli orari in in cui c'è una lezione).

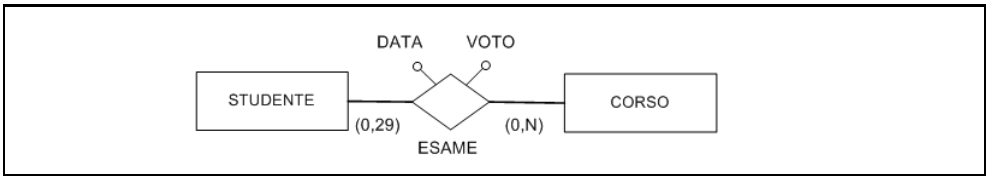
Inoltre si considerano i seguenti due vincoli di integrità:

- **non sovrapposizione :** Un'aula, in una certa ora di un giorno, deve ospitare un unico corso.
- **non sdoppiamento :** Un corso, in una certa ora di un giorno, deve essere in un'unica aula.

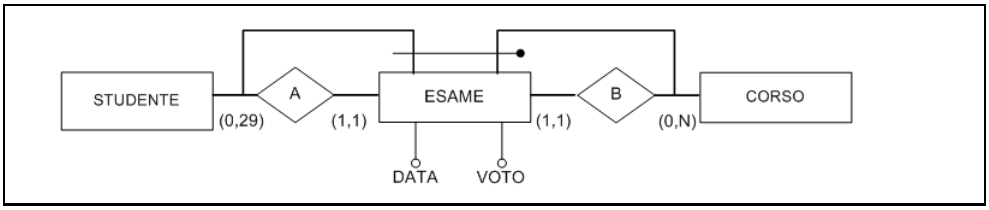
Soluzione Esempio 1

1. Per un dato studente e un dato corso può essere registrato un unico esame, con il relativo voto e la relativa data.

Come discusso a pagina 23 l'associazione binaria tra studente ed esame rispetta già questo vincolo:



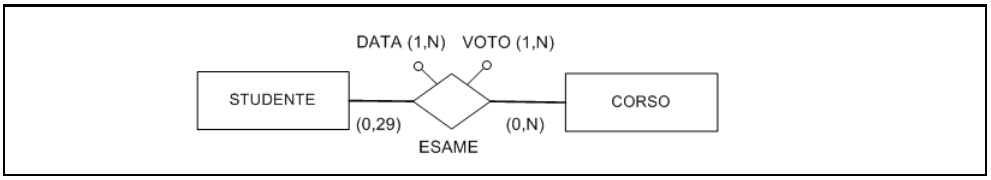
Pertanto in questo caso, ovvero in presenza del solo vincolo di integrità **1.** è inutile reificare l'associazione ESAME:



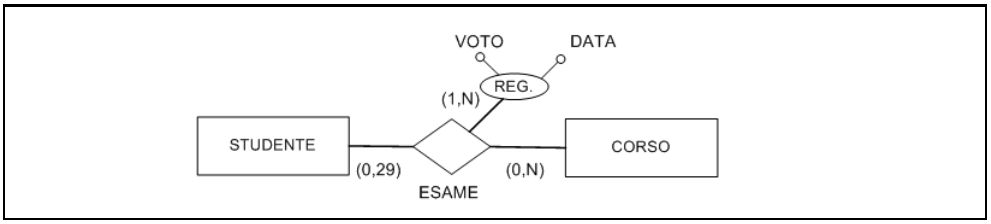
In quanto si ottiene uno schema equivalente ma più complesso di quello con la semplice associazione binaria ESAME.
La reificazione dell'associazione binaria ESAME è necessaria quando il vincolo già espresso da tale associazione si vuole *togliere* (come nel caso `4.`) oppure ad esso si vuole *aggiungere* un nuovo vincolo di integrità (come nel caso `5.`). Queste situazioni verranno discusse nel seguito.

2. Per un dato studente e un dato corso possono essere registrati più esami, ciascuno con il relativo voto e la relativa data.

La specifica di registrare più esami viene interpretata nel seguente modo: per un dato studente e un dato corso ci possono essere più registrazioni più voti, ciascuna con il relativo voto e la relativa data Si potrebbe ipotizzare di utilizzare attributi a valore-multiplo, come nello schema che segue:

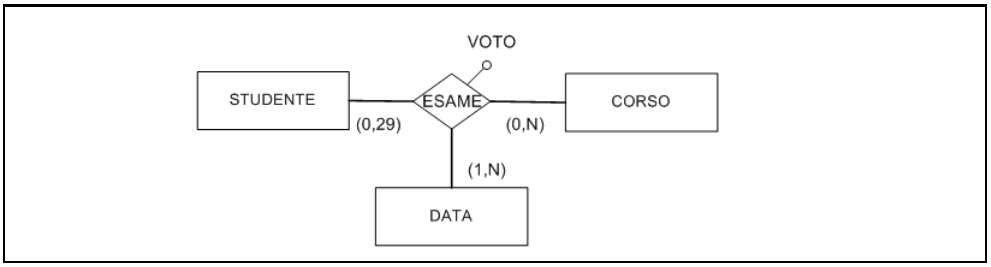


In questo modo però le due informazioni sul voto e sulla data di un certo esame possono essere scorrelate: il modello non impone alcuna corrispondenza tra i valori dei due attributi. È quindi corretto utilizzare un attributo composto multiplo:



3. Per un dato studente e un dato corso possono essere registrati più esami, ma in date differenti. Il vincolo è equivalente al seguente: per un dato studente, un dato corso e una certa data, può essere registrato un unico esame, con il relativo voto.

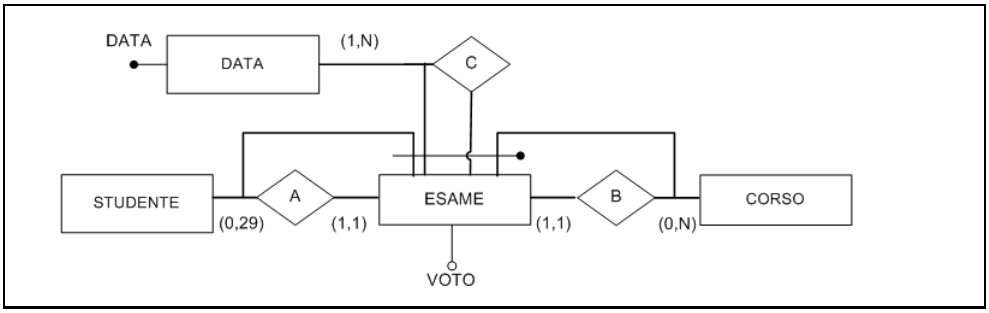
Si tratta di un vincolo maggiormente restrittivo rispetto a quello espresso al punto precedente, e che può essere modellato attraverso una associazione ternaria.



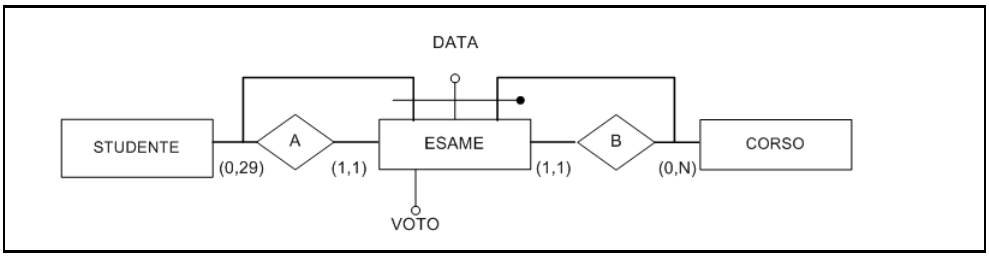
Si noti che con la partecipazione obbligatoria dell'entità DATA nell'associazione ESAME, cioè tramite card-min(DATA,ESAME)=1, una data può esistere solo se nella data è stata effettuata una registrazione d'esame. Invece, con la partecipazione opzionale dell'entità DATA nell'associazione ESAME, card-min(DATA,ESAME)=1, una data esiste indipendentemente dal fatto sia stata effettuata una registrazione d'esame.

L'associazione ternaria ESAME può essere reificata ottenendo una rappresentazione equivalente del precedente schema (si consideri il caso di card-

min(DATA,ESAME)=1, ma lo stesso vale anche nel caso di card-min(card-min(DATA,ESAME)=0):



Lo schema con l'associazione reificata è equivalente, ma graficamente più complesso, dello schema equivalente con l'associazione ternaria, pertanto quest'ultimo è sicuramente da preferire per rappresentare la situazione in questione. D'altra parte lo schema con l'associazione reificata si presta alla seguente semplificazione. L'entità DATA ha come unico attributo DATA e la card-min(DATA,ESAME)=1 implica che una data esiste solo in quanto data di registrazione di un esame. E' possibile quindi semplificare riportando l'attributo DATA direttamente sull'entità ESAME:

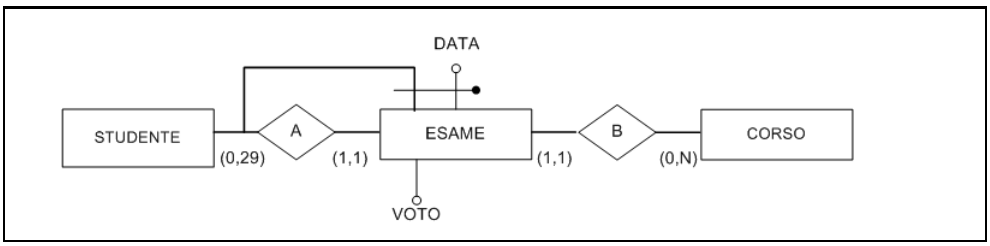


Tale semplificazione non è invece corretta, cioè lo schema semplificato non è equivalente a quello originale, nel caso di card-min(DATA,ESAME)=0: infatti nello schema semplificato una data non può esistere indipendentemente dall'esame. Nel seguito consideriamo il caso di card-min(DATA,ESAME)=1 e quindi useremo lo schema reificato semplificato.

4. Uno studente non può registrare due o più esami nella stessa data, cioè per una certa data e per un certo studente si può registrare un unico esame (relativo ad un preciso corso).

È un vincolo più restrittivo rispetto al vincolo espresso al punto 3. Partendo dallo schema con la associazione reificata, questo vincolo viene ottenuto

togliendo la componente dell'identificazione relativa al corso all'identificatore dell'entità ESAME:



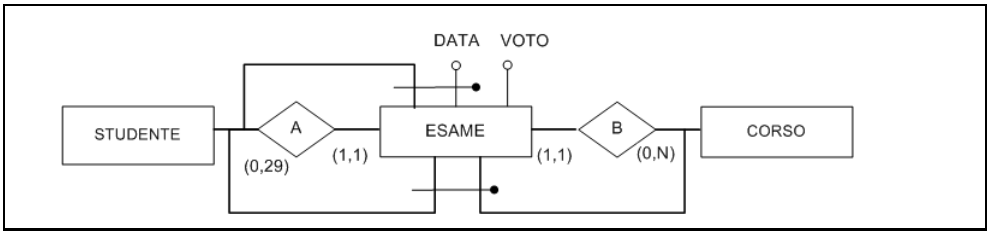
Si noti che questo schema si può considerare come ottenuto attraverso i seguenti passi:

1. la reificazione dell'associazione binaria ESAME;
2. l'eliminazione del vincolo 1, ovvero l'eliminazione dell'identificatore di ESAME costituito da STUDENTE e CORSO. In altre parole, la reificazione è servita in questo caso per poter eliminare il vincolo imposto dall'associazione binaria.
3. l'aggiunta del vincolo di identificazione 4.

Quindi in questo caso la reificazione associazione binaria ESAME è indispensabile per esprimere qualcosa di più complesso e non esprimibile rispetto alla semplice associazione binaria ESAME.

5. Modellare il vincolo espresso dal punto 4 e quello dal punto 1.

Partendo dallo schema precedente nel quale il vincolo del punto 4 è già rispettato, per rispettare il vincolo del punto 1 si introduce un nuovo identificatore: questo identificatore deve ricostruire l'associazione binaria tra studente ed esame:



Si noti che questo schema si può considerare come ottenuto attraverso i seguenti passi:

1. la reificazione dell'associazione binaria ESAME;

2. l'aggiunta del vincolo di identificazione 4.

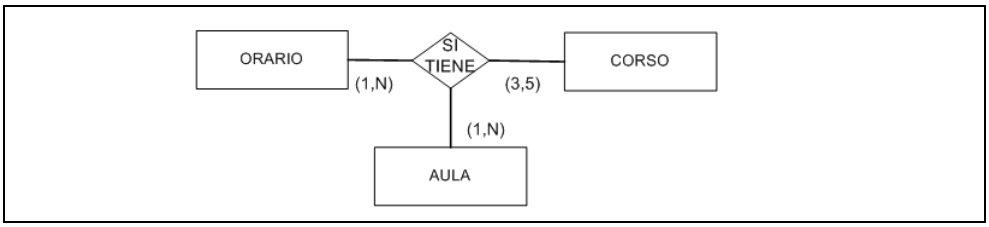
Anche in questo caso la reificazione si è resa necessaria per esprimere qualcosa di più complesso rispetto alla semplice associazione binaria ESAME.

Soluzione Esempio 2

L'orario delle lezioni è modellabile attraverso l'associazione ternaria SI_TIENE tra le entità ORAIO, CORSO e AULA:

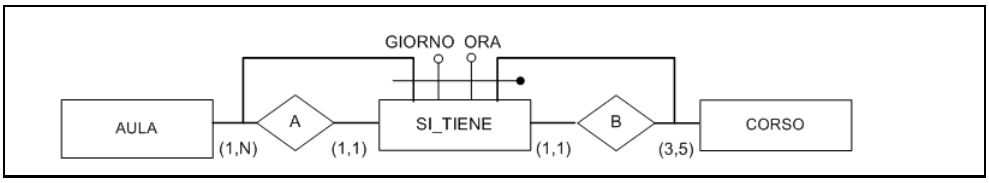
- ogni corso si tiene da tre a cinque volte la settimana: card(CORSO,SI_TIENE)=(3,5);
- in un aula si tiene, durante la settimana, almeno un corso:

  card(AULA,SI_TIENE)=(1,N);
- in un certo orario settimanale si tiene almeno un corso: card(ORARIO,SI_TIENE)=(1,N).



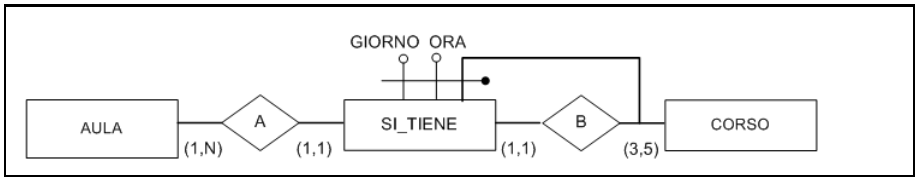
Come detto nelle specifiche l'entità ORARIO modella il giorno e l'ora della settimana, quindi ha dua attributi GIORNO e ORA che assieme costituiscono il suo identificatore. Per le altre due entità gli attributi e gli identificatori non vengono riportati in quanto non influenzano la discussione sui vincoli di integrità.

Dopo aver verificato che l'associazione ternaria non impone i vincoli richiesti (*non sovrapposizione* e *non sdoppiamento*) si procede come nel caso dell'associazione ESAME: si reifica l'associazione ternaria e si semplifica eliminando l'entità ORARIO (la semplificazione produce uno schema equivalente in quanto la partecipazione di ORARIO è obbligatoria):

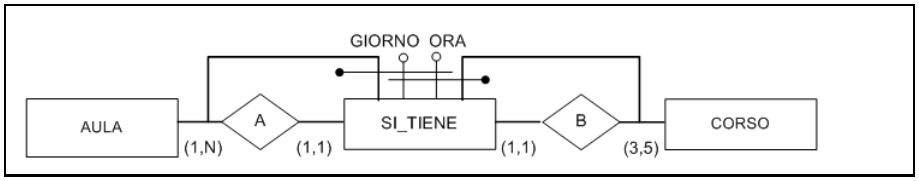


e si modifica lo schema per ottenere i vincoli di integrità richiesti:

- il seguente identificatore esprime il vincolo di *non sdoppiamento* (infatti dato un `CORSO`, un `GIORNO` ed un'`ORA` ho un'unica istanza di `SI TIENE` alla quale è associata un'unica `AULA`):



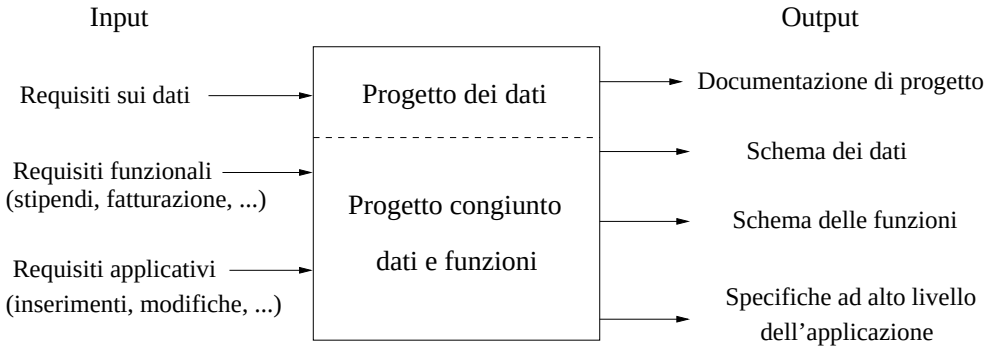
- mentre il seguente ulteriore identificatore esprime il vincolo di *non sovrapposizione* (infatti dato un'`AULA`, un `GIORNO` ed un'`ORA` ho un'unica istanza di `SI TIENE` alla quale è associata un unico `CORSO`):



1.4 Metodologia per il progetto concettuale

Questa sezione è tratta dal libro C. Batini, S. Ceri, S. Navathe, "Conceptual Database Design - An Entity-Relationship approach", Benjamin, Cummings 1992.

La metodologia di progetto generale è presentata in figura.



I requisiti sui dati possono essere forniti secondo diverse modalità:

1. requisiti in linguaggio naturale
2. moduli
3. tracciati record o schermate
4. schemi di dati

Nel seguito verranno trattati i requisiti sui dati relativamente ai soli requisiti in linguaggio naturale e tracciati record.
I requisiti funzionali e applicativi verranno espressi nella forma di schemi navigazionali E/R.

Metodologia per il progetto concettuale

La costruzione di uno schema E/R è un processo incrementale: la nostra percezione della realtà è progressivamente raffinata e arricchita e lo schema concettuale è formalmente sviluppato.

Primitive di raffinamento: trasformazioni che applicate allo schema iniziale producono lo schema finale.

Strategie di progetto
- **top-down**
- **bottom-up**
- **mixed**

Primitive di raffinamento + **Strategie di progetto** = **Metodologia di progetto**

Una metodologia di progetto dovrebbe idealmente essere un compromesso tra due aspetti contrastanti:

1. **rigore:** metodologia come approccio formale nel quale ogni processo di decisione dovrebbe idealmente corrispondere ad un algoritmo;
2. **flessibilità:** metodologia flessibile in modo che ogni progettista possa adattarla ad un problema specifico e seguire il proprio stile di progettazione.

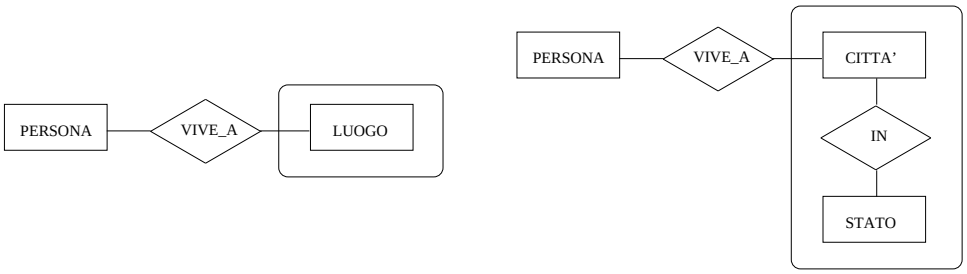
1.4.1 Primitive per il progetto concettuale

Il progetto di uno schema concettuale è un processo iterativo durante il quale, partendo da uno schema iniziale, si effettuano delle **trasformazioni di schema** per produrre lo schema finale.

Caratteristiche delle trasformazioni dello schema:

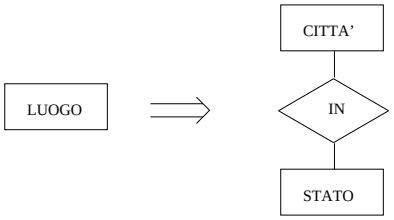
1. Ogni trasformazione si applica ad uno **schema iniziale** e produce uno **schema finale**.
   Nell'esempio successivo, nello schema iniziale è presente un'entità LUOGO e nello schema finale LUOGO è rappresentato da una coppia di entità CITTÀ, STATO connesse da un'associazione IN.
2. Ogni trasformazione mappa **nomi** di concetti dello schema iniziale in nomi di concetti dello schema finale.
   Nell'esempio al nome iniziale LUOGO corrisponde l'insieme di nomi CITTÀ, IN, STATO.

3. I concetti dello schema finale devono ereditare tutte le **connessioni logiche** definite per i concetti dello schema iniziale.
   Nelle'esempio l'associazione VIVE_A tra PERSONA e LUOGO è ereditata dall'entità CITTÀ.



Schema iniziale

Schema finale



Trasformazione

Trasformazioni primitive

Le trasformazioni primitive sono trasformazioni che non possono essere decomposte in altre più semplici.

PRIMITIVE TOP-DOWN

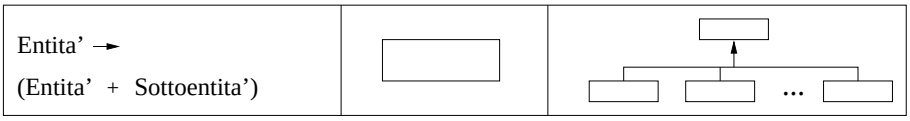
- Hanno una struttura semplice: lo schema iniziale è un concetto semplice e lo schema finale consiste di un piccolo insieme di concetti;
- Tutti i nomi sono raffinati in nuovi nomi che descrivono il concetto originale ad un livello di astrazione più basso;
- Le connessioni logiche dello schema iniziale devono essere ereditate dai concetti dello schema finale.

PRIMITIVE BOTTOM-UP

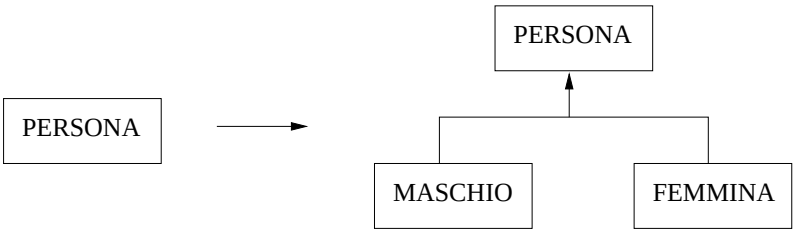
- Introducono nuovi concetti e proprietà che non comparivano nelle precedenti versioni dello schema oppure modificano alcuni concetti esistenti;
- Sono usate quando alcune caratteristiche del dominio di applicazione non sono rappresentate nelle versioni precedenti dello schema;
- Sono applicate per integrare schemi differenti in uno schema globale più comprensivo (**integrazione di schemi**).

Esempi di primitive top-down

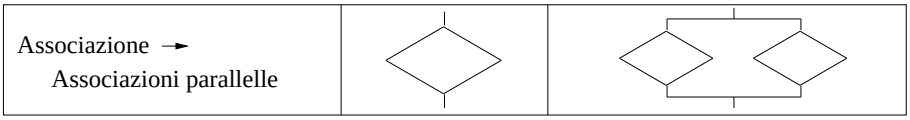
♢ Un'entità è trasformata in una gerarchia di generalizzazione o in un subset.



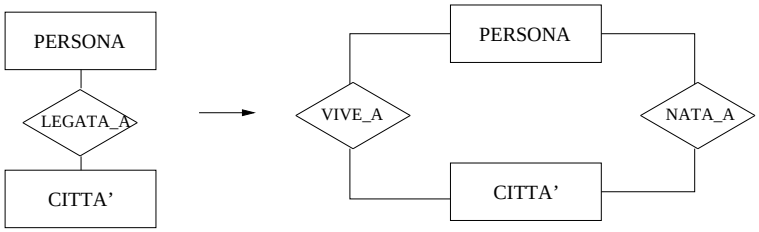
Esempio
L'entità PERSONA è trasformata in una generalizzazione che include MASCHIO e FEMMINA.



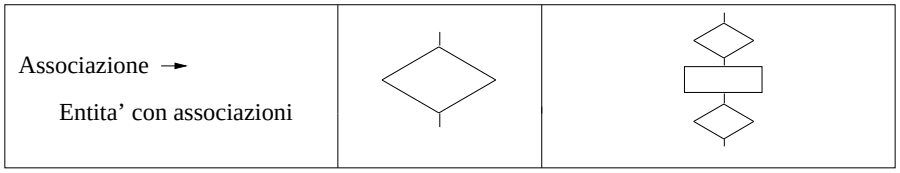
♢ Un'associazione è trasformata in una o più associazioni tra le stesse entità.



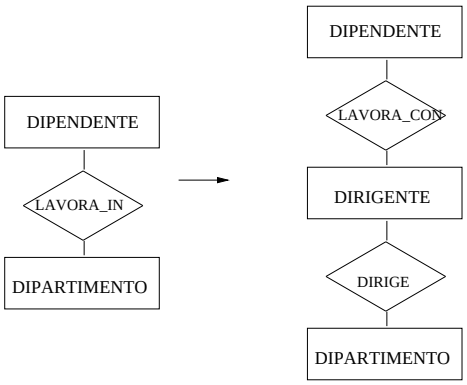
Esempio
L'associazione LEGATA_A tra PERSONA e CITTÀ è raffinata in due associazioni, VIVE_A e NATA_A, tra le stesse entità.



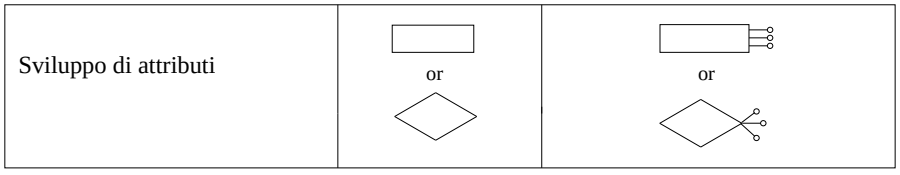
♢ Un'associazione è trasformata in un *cammino* di entità e associazioni. Questo corrisponde a riconoscere che un'associazione tra due entità deve essere espressa tramite una terza entità e due associazioni. Questa trasformazione è detta anche *reificazione*.



Esempio
L'associazione LAVORA_IN tra DIPENDENTE e DIPARTIMENTO è raffinata in una aggregazione più complessa che include l'entità DIRIGENTE e due nuove associazioni.



♢ Un'entità o un'associazione è trasformata introducendo i suoi attributi.

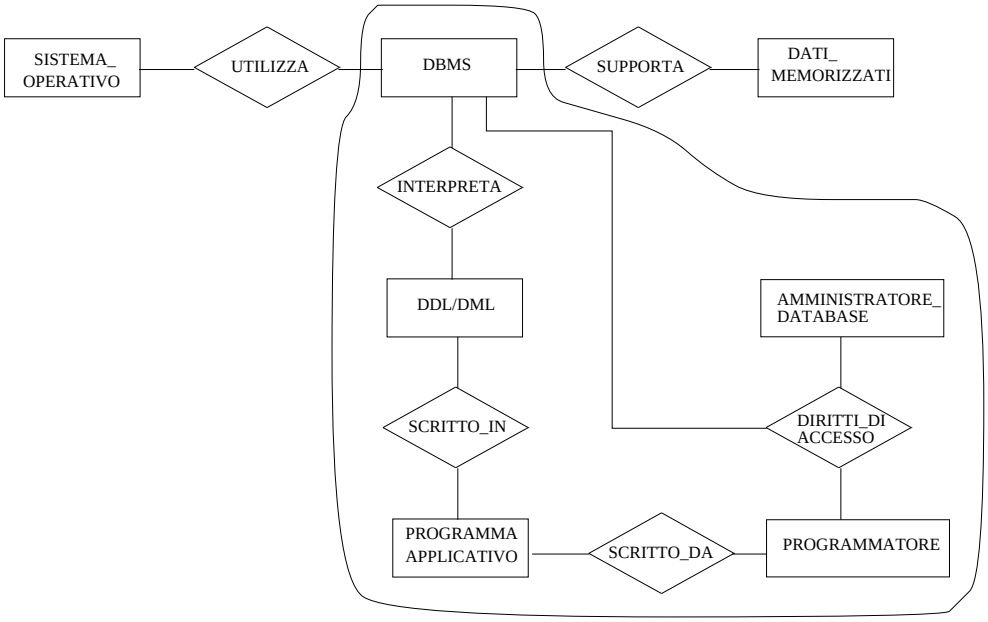


Esempio
Per l'entità PERSONA sono introdotti gli attributi NOME, SESSO ed ETÀ.

Esempio di applicazione delle primitive top-down

♢ Le primitive presentate sono alcune tra le più significative di tipo top-down;

♢ Altre e più complesse trasformazioni possono essere classificate come top-down, come mostrato nel seguente esempio, dove un'entità è trasformata in un insieme di entità e associazioni.

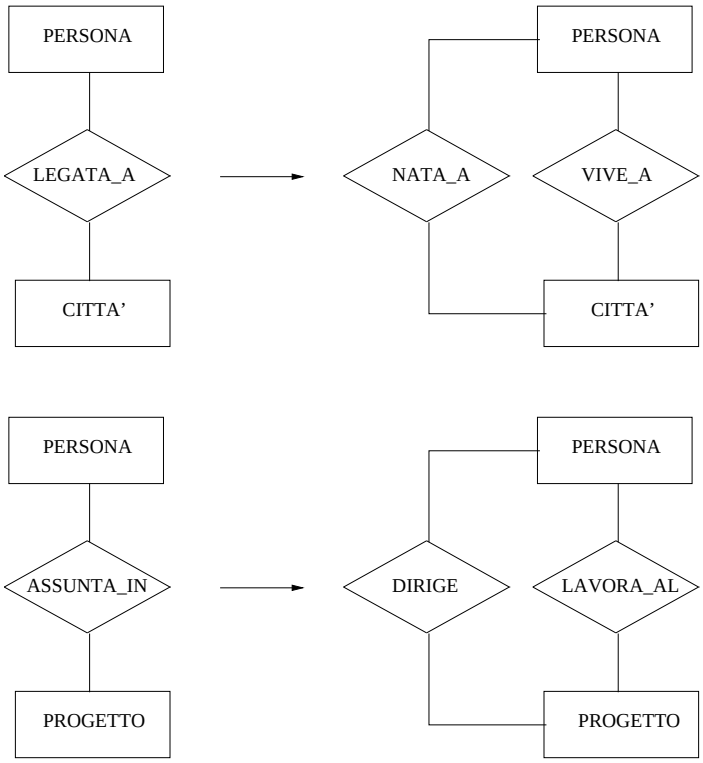


♢ Nell'applicazione delle primitive di raffinamento si devono rispettare alcuni vincoli.

Consideriamo l'applicazione della trasformazione di un'associazione in *cammino* di entità e associazioni: esiste un legame tra le cardinalità, minima e massima, dell'associazione nello schema di partenza e quelle delle due nuove associazioni introdotte. Ad esempio, se nello schema di partenza si ha un'associazione one-to-one, lo schema risultante non può includere un'associazione many-to-many.

♢ I vincoli dipendono anche dal significato dei concetti.

Consideriamo due diverse applicazioni della suddetta primitiva:

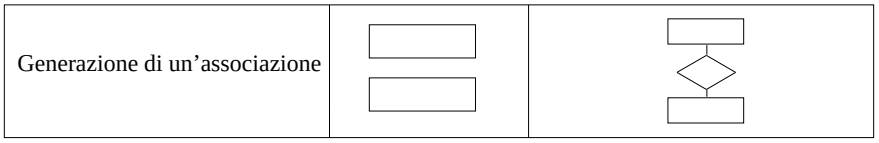


Nel primo caso le associazioni prodotte dalla trasformazione sono indipendenti da quella originale.
Nel secondo caso invece le istanze dell'associazione ASSUNTA_IN sono partizionate nelle istanze delle associazioni DIRIGE e LAVORA_AL.

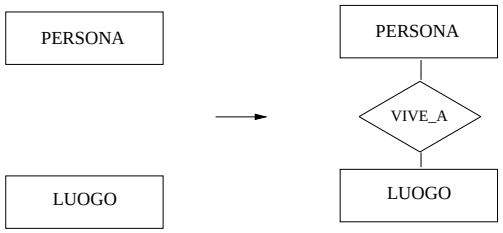
Esempi di primitive bottom-up

♢ Generazione di una nuova entità.
È usata per introdurre un nuovo concetto nello schema.

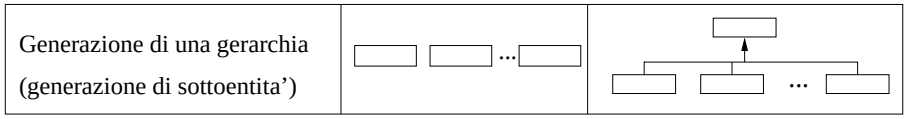
♢ Generazione di una nuova associazione tra entità precedentemente definite.



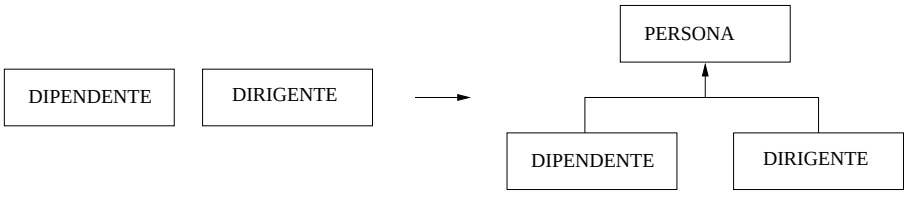
Esempio
La nuova associazione VIVE_A è stabilita tra le entità PERSONA e LUOGO.



♢ Generazione di una nuova entità come generalizzazione di entità esistenti.

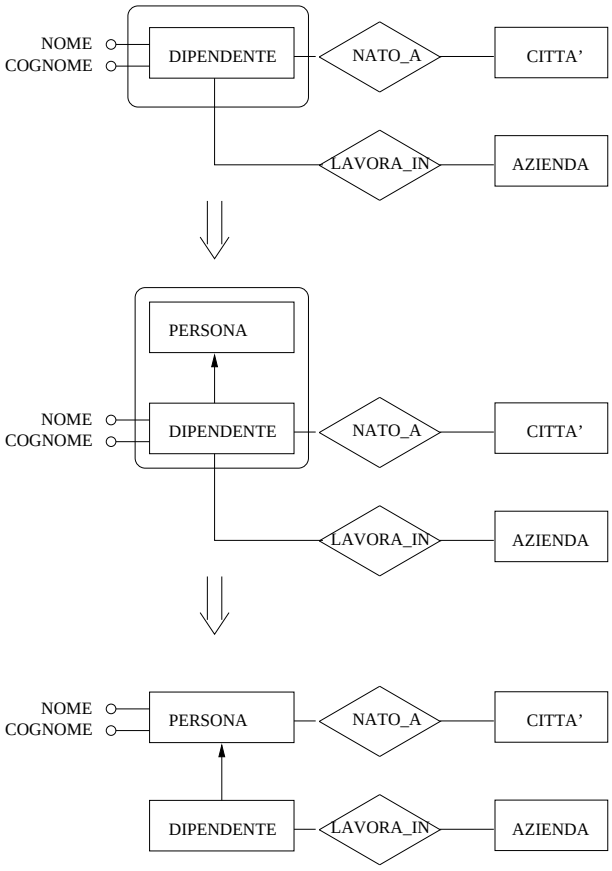


Esempio
Si forma una gerachia di generalizzazione, introducendo la classe PERSONA.



L'applicazione della primitiva di generazione di una gerarchia rende necessario controllare se le proprietà possono migrare tra le entità per effetto della nuova generalizzazione.

Nell'esempio seguente, usando la primitiva di generazione di una gerarchia, si introduce una nuova entità PERSONA e una relazione di generalizzazione tra PERSONA e DIPENDENTE. Di conseguenza gli attributi NOME e COGNOME e l'associazione NATO_A devono essere spostate e assegnate all'entità generalizzazione. Si noti invece come l'associazione LAVORA_IN rimane assegnata all'entità DIPENDENTE.



1.4.2 Strategie per il progetto concettuale

1. **top-down**
   Lo schema è ottenuto applicando solo le primitive top-down. Il processo finisce quando tutti i requisiti sono stati rappresentati.
   In una strategia top-down *pura* tutti i concetti da rappresentare nello schema finale devono essere presenti in ogni schema di raffinamento intermedio (naturalmente ad un diverso livello di astrazione).
   Il processo termina quando tutti i requisiti sono stati rappresentati.

2. **bottom-up**
   Lo schema è ottenuto applicando solo le primitive bottom-up, iniziando da concetti elementari e costruendo concetti più complessi.
   I concetti vengono progressivamente integrati nello schema finale.

3. **mixed**
   Utilizzo sia delle strategie top-down che di quelle bottom-up.
   I requisiti, se il dominio è molto complesso, sono partizionati in sottinsiemi e considerati separatamente; nello stesso tempo si produce uno **schema scheletro** che serve come collegamento tra le varie partizioni.

Applicazione delle strategie ad un esempio

Base di dati demografica che rappresenta persone e luoghi.

Requisiti:

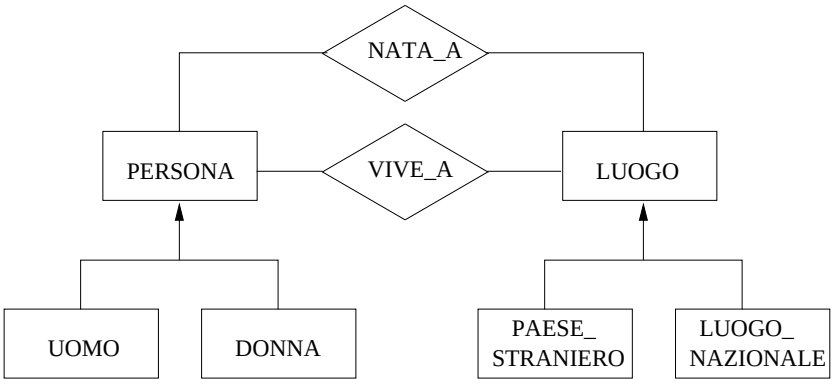
- Per le persone sono considerate le seguenti proprietà: nome, cognome, sesso, età, luogo di nascita e di residenza, durata della residenza, posizione militare per gli uomini e il nome da nubile per le donne.

- Un luogo può essere uno stato estero o una città nazionale; ognuno di essi ha un nome e una popolazione. Per uno stato estero deve essere indicato il continente e per una città nazionale il nome della regione.

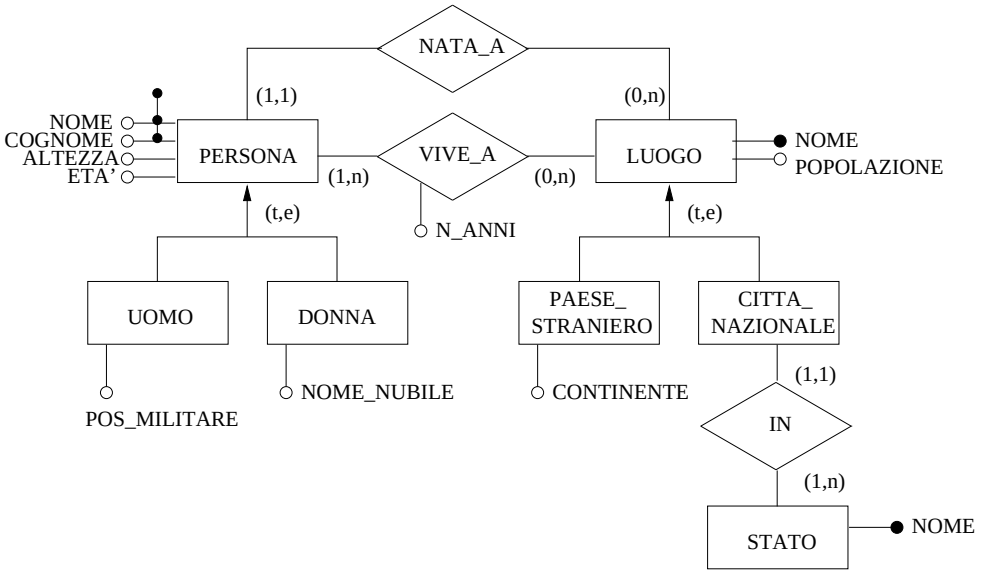
Applicazione della strategia top-down

L'entità è raffinata in due entità e un'associazione tra di esse.

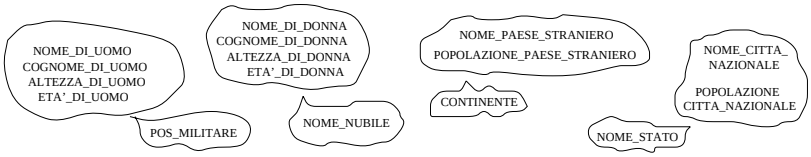
L'associazione LEGATI_A è raffinata in due associazioni differenti e le entità PERSONA e LUOGO sono trasformate in due gerarchie di generalizzazione.



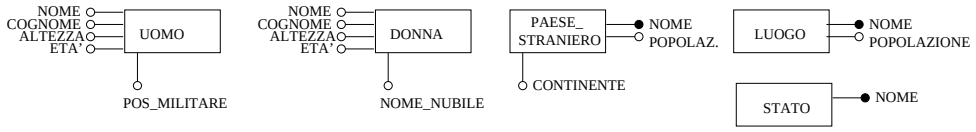
L'entità LUOGO_NAZIONALE è raffinata in due entità e un'associazione tra di esse.
Per ottenere lo schema finale sono specificati gli attributi, gli identificatori, la cardinalità di partecipazione alle associazioni.



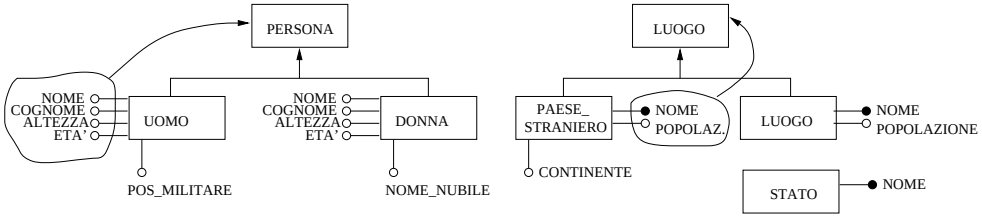
Applicazione della strategia bottom-up



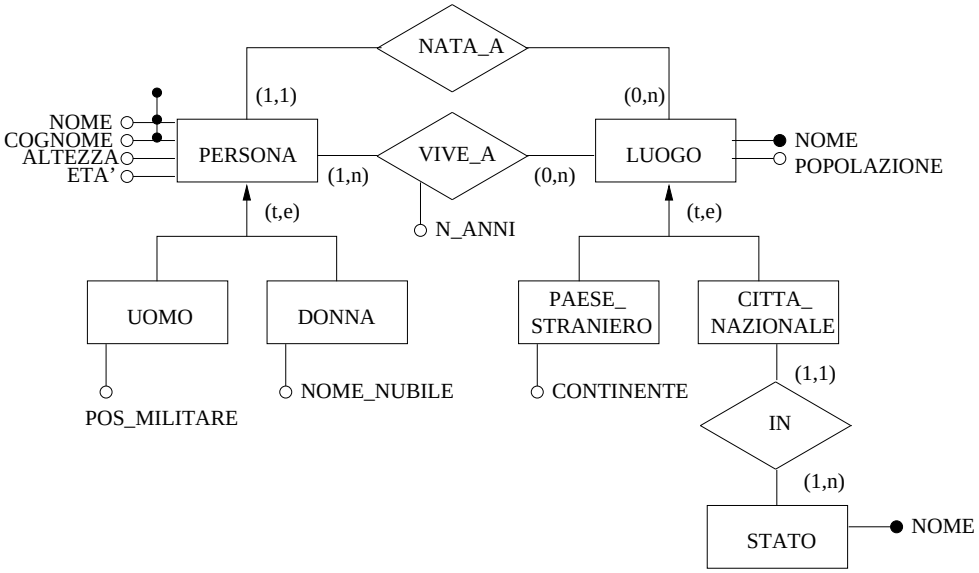
Si introducono le entità che rappresentano l'aggregazione degli attributi dati.



Si formano due gerarchie di generalizzazione sulle entità; le proprietà dovranno migrare per effetto delle generalizzazioni introdotte.



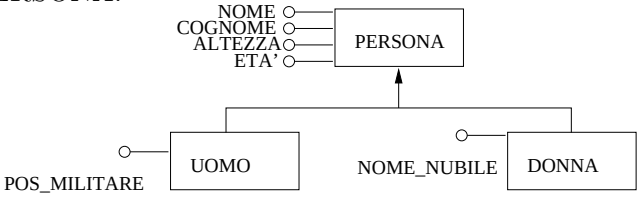
Vengono definite associazioni tra le entità.
Si introducono cardinalità e identificatori.



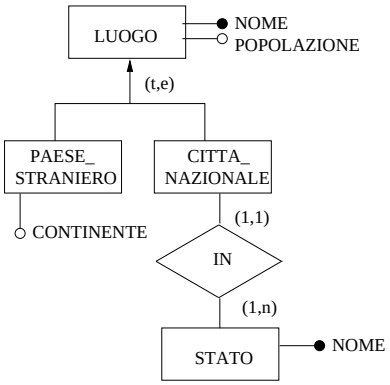
Applicazione della strategia mixed

Schema scheletro:
sono individuate le entità PERSONA e LUOGO e tra di esse viene stabilita un'associazione. Gli schemi di PERSONA e LUOGO sono ottenuti separatamente.

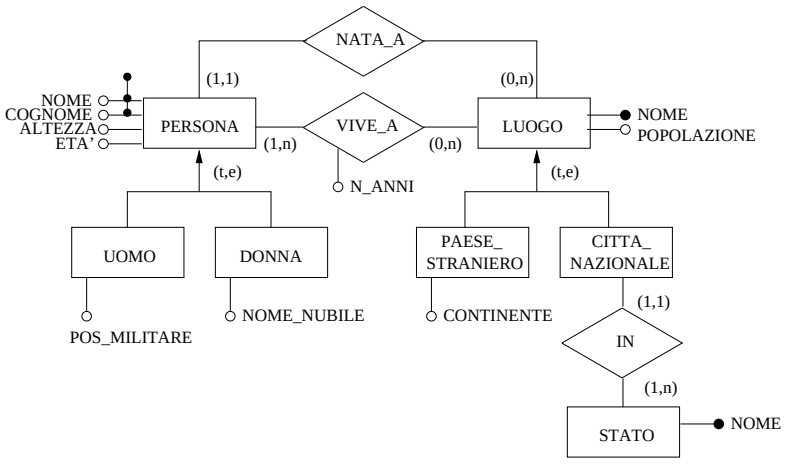
Schema di PERSONA:



Schema di LUOGO:

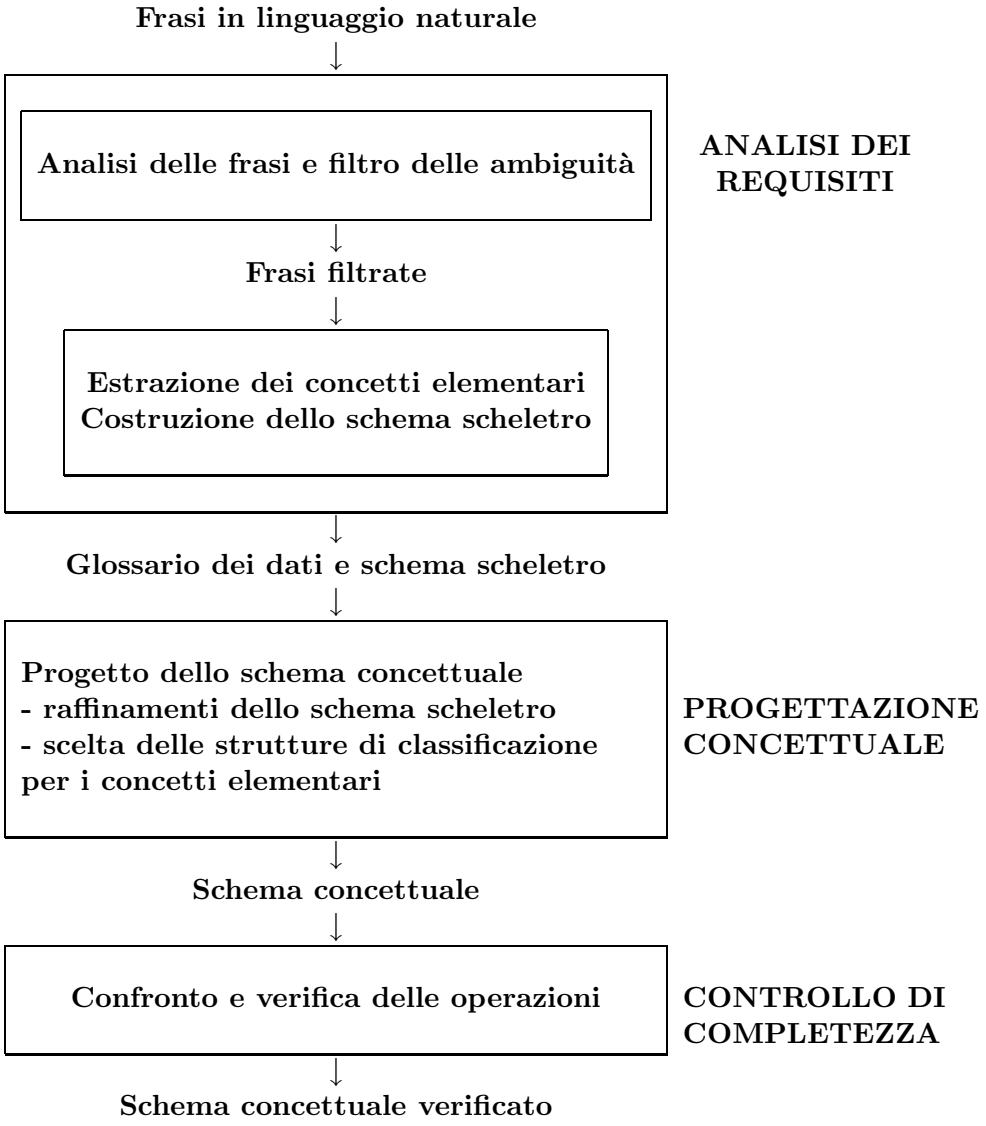


Schema integrato:
gli schemi di PERSONA e di LUOGO sono connessi allo schema scheletro e l'associazione LEGATI_A è raffinata in due associazioni:



1.5 Progettazione da requisiti in linguaggio naturale

Fasi delle attività di progettazione concettuale a partire da requisiti espressi in linguaggio naturale



Requisiti per un Database di Società Sportiva:

1.	Si vuole costruire un database per gestire le informazioni relative agli
2.	atleti e alle squadre di una società sportiva. Per gli atleti della società
3.	è necessario memorizzare il numero della tessera di iscrizione, il codice
4.	fiscale, il nome, il cognome, il sesso, l'indirizzo, la data ed il luogo di
5.	nascita e la squadra di appartenenza. Gli atleti possono frequentare
6.	corsi annuali organizzati dalla società. Per gli atleti frequentatori
7.	interessano: la data dell'ultima visita medica, i corsi annuali che
8.	hanno seguito prima, con l'esito ottenuto e i corsi che stanno
9.	seguendo attualmente. Per gli atleti professionisti si indica la
10.	disciplina sportiva e il preparatore atletico. In generale, per i corsi
11.	si rappresenta il codice corso e la sua descrizione; per i corsi correnti
12.	si rappresentano, oltre al loro costo e al numero dei partecipanti,
13.	i giorni, le relative ore di inizio e di fine e il luogo in cui si svolgono,
14.	con relativa descrizione, indirizzo e telefono. Un corso si può svolgere
15.	una o più volte nello stesso giorno, in più impianti o nello stesso
16.	impianto. Ogni corso è tenuto da un allenatore per il quale si
17.	rappresenta il codice fiscale, il nome, il cognome, il sesso, l'indirizzo,
18.	la squadra di appartenenza e la specializzazione sportiva.

Termini ambigui nei requisiti e correzioni possibili:

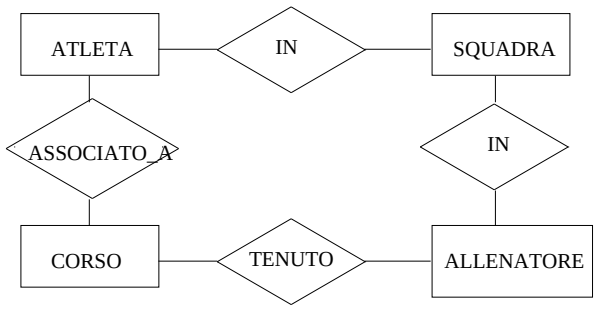
Linea	**Termine**	**Nuovo termine**	**Motivo correzione**
8	Prima	In anni precedenti	Parola generica
9	Attualmente	Nell'anno corrente	Parola ambigua
10	Preparatore atletico	Allenatore	Termine più specifico
15	Giorno	Giorno della sett.	Termine più specifico
18	Specializzazione	Disciplina	Termine più specifico

Requisiti dopo il filtraggio delle ambiguità:

1.	Si vuole costruire un database per gestire le informazioni relative agli
2.	atleti e alle squadre di una società sportiva. Per gli atleti della società
3.	è necessario memorizzare il numero della tessera di iscrizione, il codice
4.	fiscale, il nome, il cognome, il sesso, l'indirizzo, la data ed il luogo di
5.	nascita e la squadra di appartenenza. Gli atleti possono frequentare
6.	corsi annuali organizzati dalla società. Per gli atleti frequentatori
7.	interessano: la data dell'ultima visita medica, i corsi annuali che
8.	hanno seguito in anni precedenti , con l'esito ottenuto e i corsi che
9.	stanno seguendo nell'anno corrente. Per gli atleti professionisti si
10.	indica la disciplina sportiva e l'allenatore. In generale, per i corsi
11.	si rappresenta il codice corso e la sua descrizione; per i corsi correnti
12.	si rappresentano, oltre al loro costo e al numero dei partecipanti,
13.	i giorni, le relative ore di inizio e di fine e il luogo in cui si svolgono,
14.	con relativa descrizione, indirizzo e telefono. Un corso si può svolgere
15.	una o più volte nello stesso giorno della settimana, in più impianti
16.	o nello stesso impianto. Ogni corso è tenuto da un allenatore per il
17.	quale si rappresenta il codice fiscale, il nome, il cognome, il sesso,
18.	l'indirizzo, la squadra di appartenenza e la disciplina sportiva.

Progetto Iniziale:

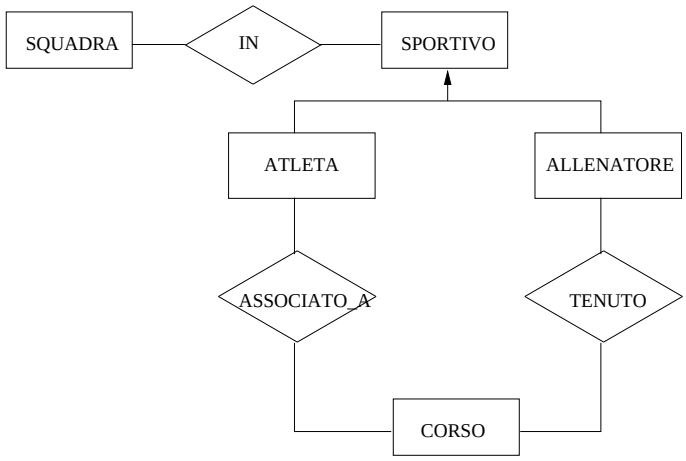
Si deve costruire uno schema scheletro con i concetti più evidenti espressi nei requisiti. I concetti referenziati con maggiore frequenza nei requisiti sono buoni candidati per tale scelta:
ATLETA, CORSO, ALLENATORE, SQUADRA.
A tali entità si impongono le associazioni più evidenti; l'associazione ASSOCIATO_A è deliberatamente vaga e sarà specificata dopo.



Schema scheletro iniziale:

Prima di continuare il progetto, si controlla se è possibile ristrutturare lo schema scheletro precedente.
Osservando l'associazione IN, si scoprono similarità tra le entità ATLETA e ALLENATORE: si introduce SPORTIVO come generalizzazione di tali entità e si riferisce ad essa l'associazione SPORTIVO.

Raffinamento dello schema scheletro:



Progetto dello schema

Raffinamenti top-down :

1. l'entità ATLETA è ridefinita in termini di due sottoentità: ATLETA_FREQUENTATORE e ATLETA_PROFESSIONISTA
2. l'associazione ASSOCIATO_A pe ridefinita con due associazioni: HANNO_SEGUITO e SEGUONO

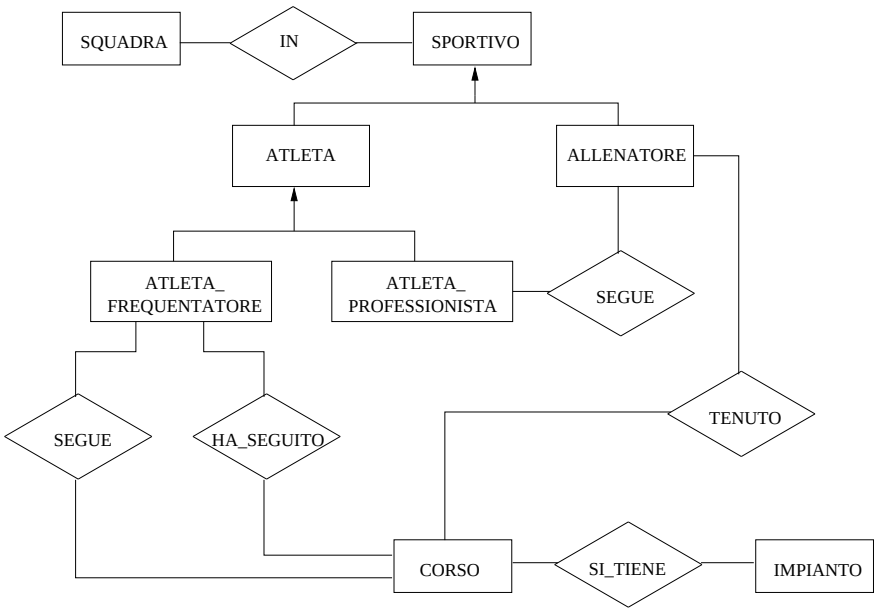
Raffinamenti bottom-up :

introduzione dell'associazione SEGUE tra ALLENATORE e ATLETA_PROFESSIONISTA

Raffinamenti inside-out :

poichè una proprietà di CORSO è l'impianto in cui viene svolto che ha a sua volta diverse proprietà (indirizzo,telefono) si rappresenta l'impianto come entità e si esprime il legame logico tra CORSO e IMPIANTO con l'associazione SI_TIENE.

Ne risulta il seguente schema:



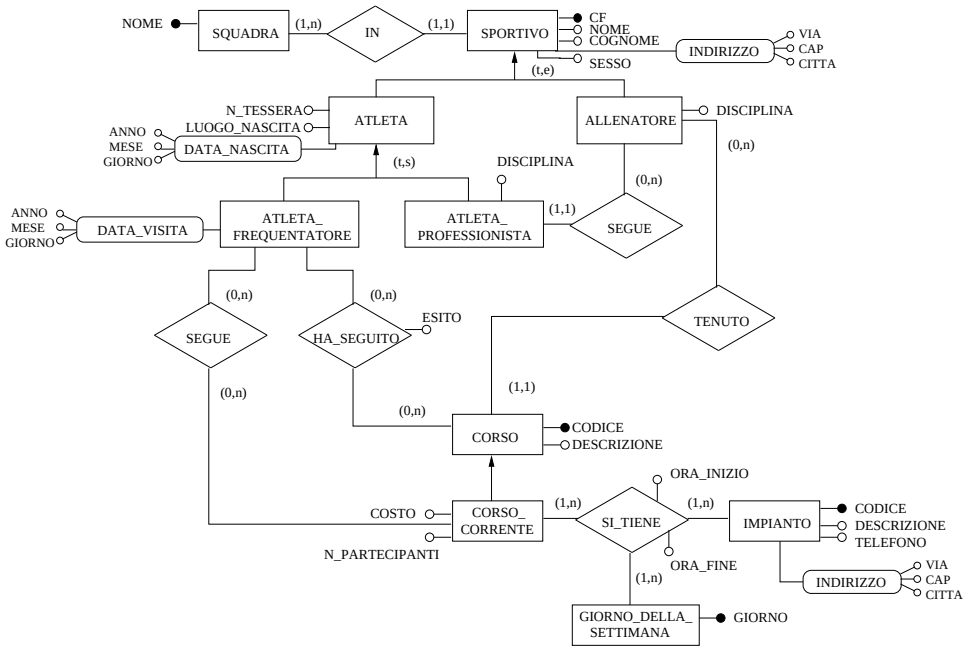
Ulteriori raffinamenti :

1. l'associazione SI_TIENE viene raffinata per rappresentare l'orario dei vari corsi. Poichè un CORSO si può tenere una o più volte nello stesso giorno, si introduce l'entità GIORNO_DELLA_SETTIMANA e si trasforma l'associazione SI_TIENE in ternaria

2. si specializza l'entità CORSO con un subset CORSO_CORRENTE: solo per i corsi correnti viene specificato l'orario tramite l'associazione SI_TIENE

3. attributi, identificatori, cardinalità delle associazioni e coperture delle gerarchie

Si ottiene il seguente schema finale:



Operazioni per il Database Società Sportiva

Operazione 1 ISCRIZIONE DI UN NUOVO ATLETA :

Si prende in input il numero tessera e si controlla che non esista già lo sportivo; si inseriscono quindi i dati, considerando il fatto che l'atleta iscritto sia professionista e/o frequentatore. Mediamente ci sono 100 iscrizioni nuove al mese.

Operazione 2 CANCELLAZIONE DI UN CORSO CORRENTE E REGISTRAZIONE DELLE FREQUENZE:

Si immette il semestre e l'anno di interesse e si cancellano i corsi correnti con data di inizio nel semestre individuato. Per ogni corso da cancellare si individuano gli atleti iscritti e la loro frequenza viene archiviata assieme allesito e alla data di fine corso. Per ogni corso da cancellare si elimina infine il suo impegno degli impianti. Questa operazione è effettuata al termine di ogni semestre.

Nel seguito vengono presentati gli schemi di navigazione delle operazioni indicate assieme al volume dei dati ed alle valutazioni del costo di accesso ai dati per eseguire le operazioni.

Volume dei dati

Il volume dei dati, con le principali ipotesi fatte, è il seguente:

Tavola dei volumi

CONCETTO	TIPO	VOLUME
SPORTIVO	E	2100
ALLENATORE	E	100
ATLETA	E	2000
ATLETA_FREQUENTATORE	E	2000
ATLETA_PROFESSIONISTA	E	600 **(A)**
CORSO	E	200
CORSO_CORRENTE	E	100
SEGUE	R	25*100=2500 **(B)**
HA_SEGUITO	R	200*25*2 **(C)**
IMPIANTI	E	10
SI_TIENE	R	300 **(D)**
GIORNO_DELLA_SETT	E	6

(A) La ripartizione tra atleti frequentatori e professionista è stata fatta considerando che il 30% sono gli atleti professionisti che normalmente sono anche frequentatori.

(B) Ogni corso corrente è seguito in media da 25 atleti.

(C) Per gli atleti frequentatori si memorizzano i corsi frequentati nei 4 anni precedenti. Supponendo che un corso venga tenuto ad anni alterni, questo comporta che un corso sia stato già tenuto 2 volte nei 4 anni precedenti, per un totale di 50 presenze.

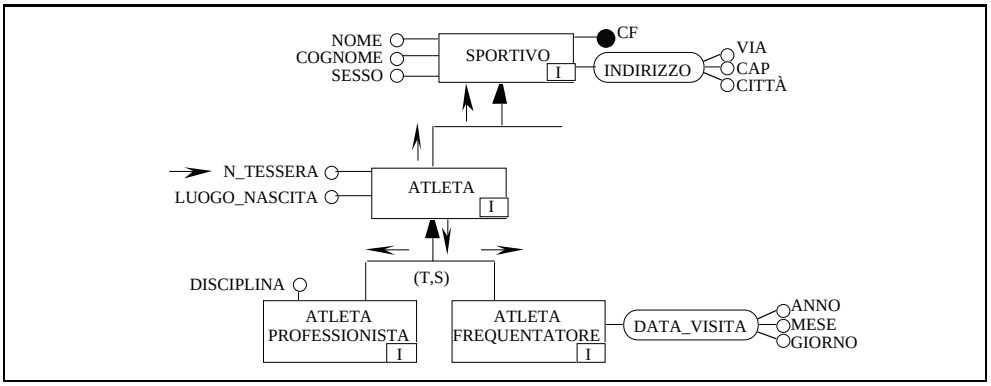
(D) Ogni corso corrente è tenuto in media 3 volte la settimana.

Operazione 1

ISCRIZIONE DI UN NUOVO ATLETA :

Si prende in input il numero tessera e si controlla che non esista già lo sportivo; si inseriscono quindi i dati, considerando il fatto che l'atleta iscritto sia professionista e/o frequentatore. Mediamente ci sono 100 iscrizioni nuove al mese.

Lo schema di navigazione è il seguente:



Nello schema di navigazione sono stati indicati gli attributi interessati all'operazione; in particolare, la freccia sull'attributo N_TESSERA indica che l'accesso all'entità ATLETA avviene tramite il numero tessera.
Per ogni entità il tipo di accesso è **I**: il controllo che non esista già uno sportivo con il numero tessera specificato viene preso in considerazione nella tabella degli accessi, tramite un accesso il lettura sull'entità SPORTIVO:

CONCETTO	**ACCESSI**	**TIPO**
SPORTIVO	1	L
SPORTIVO	1	S
ATLETA	1	S
ATLETA_PROFESSIONISTA	.3	S
ATLETA_FREQUENTATORE	1	S

Considerando pari a 100/mese la frequenza di questa operazione, il suo costo mensile è il seguente:

(1 + 2 + 2 + .6 + 2)*100 = 760

Operazione 2

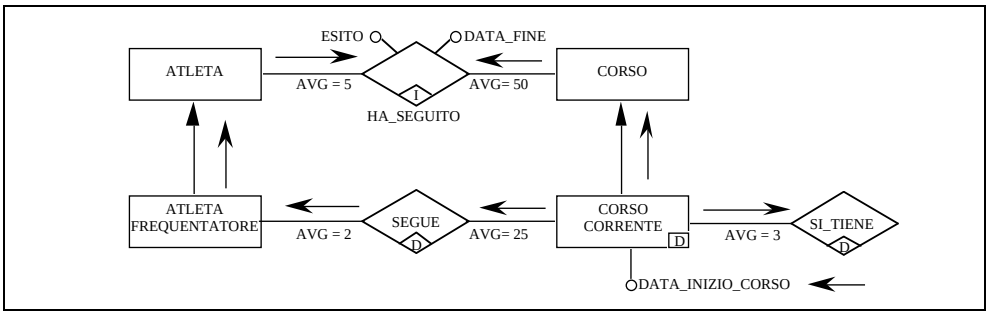
CANCELLAZIONE DI UN CORSO CORRENTE E REGISTRAZIONE DELLE FREQUENZE

Supponiamo che questa operazione venga eseguita nel seguente modo:

1. Un confronto tra la data dell'operazione e la DATA_INIZIO_CORSO consente di individuare i corsi di CORSO_CORRENTE da eliminare. Supponendo la data di inizio di un corso uniformemente distribuita in un anno, dei 100 CORSO_CORRENTE ne devono essere cancellati 50.
2. Per ogni corso da cancellare si individuano, tramite l'associazione SEGUE, gli ATLETI_FREQUENTATORI (i loro identificatori) iscritti al corso: ogni coppia di identificatori `<atleta,corso>` viene eliminata da SEGUE ed inserita in HA_SEGUITO, assieme all'esito e alla data di fine corso.
3. Per ogni corso da cancellare si elimina il suo impegno degli impianti cancellandolo dallassociazione SI_TIENE.

Per effettuare questa operazione occorre completare lo schema E/R, presentato a pagina 39 , aggiungendo l'attributo DATA_INIZIO_CORSO sull'entità CORSO_CORRENTE e l'attributo DATA_FINE sullassociazione HA_SEGUITO.

Lo schema di navigazione e la tabella degli accessi sono i seguenti:



CONCETTO	ACCESSI	TIPO
CORSO_CORRENTE	1	L
CORSO_CORRENTE	.5	S
SEGUE	.5*25=12.5	L
SEGUE	.5*25=12.5	S
HA_SEGUITO	.5*25=12.5	S
SI_TIENE	.5*3=1.5	L
SI_TIENE	.5*3=1.5	S

Questa operazione viene effettuata una volta a semestre, per ciascuno dei 100 CORSO_CORRENTE, per una frequenza totale di 100/semestre; il suo costo semestrale è il seguente:
(1 + 1 + 12.5 + 25 + 25 + 1.5 + 3)*100 = 6900

1.6 Progettazione da file esistenti

Le applicazioni commerciali usano file, cioè insiemi di record. Spesso le applicazioni di basi di dati devono tenere conto di applicazioni pre-esistenti basate su file e file-system[2]. Per sviluppare un progetto a partire dalla progettazione concettuale è quindi necessario compiere un reverse-engineering che ha come input un insieme di file e che deve produrre come output una vista concettuale E/R da integrare. I linguaggi più usati sono COBOL, PL/1, FORTRAN e C.

Esempio

Consideriamo la dichiarazione in linguaggio COBOL di un file FATTURA. Alcuni campi sono elementari (QUANTITA, CODICE_ARTICOLO, CLIENTE) mentre altri campi sono composti (AMMONTARE, DATA_DI_EMISSIONE, ...).

```
01   FATTURA.
     02 NUMERO_FATTURA                  PIC X(5).
     02 DATA_DI_EMISSIONE.
        03 ANNO_DI_EMISSIONE            PIC 9(4).
        03 MESE_DI_EMISSIONE            PIC 9(2).
        03 GIORNO_DI_EMISSIONE          PIC 9(2).
     02 DATA_DI_PAGAMENTO.
        03 ANNO_DI_PAGAMENTO            PIC 9(4).
        03 MESE_DI_PAGAMENTO            PIC 9(2).
        03 GIORNO_DI_PAGAMENTO          PIC 9(2).
     02 AMMONTARE.
        03 PREZZO_NETTO                 PIC 9(9)V99.
        03 IVA                          PIC 9(2).
        03 COSTO_IVA                    PIC 9(9) COMPUTATIONAL.
        03 PREZZO_TOTALE                PIC 9(9) COMPUTATIONAL.
     02 LINEA_FATTURA OCCURS 15 TIMES.
        03 NUM_LINEA                    PIC 9(2).
        03 CODICE_ARTICOLO              PIC X(5).
        03 QUANTITA                     PIC 9(3).
        03 PREZZO_UNITARIO              PIC 9(9)V99.
     02 CLIENTE                         PIC X(9).
     02 AZIENDA                         PIC X(9).
```

In COBOL varie clausole nella definizione del file specificano il ruolo di un campo, la sua allocazione fisica, il tipo di accesso previsto per il file e altre caratteristiche. Queste informazioni sono molto importanti per determinare il significato dei campi, le loro associazioni logiche interne e le astrazioni definite tra di essi in modo da poter rappresentare il file in termini di uno schema E/R.

Il progetto degli schemi E/R a partire da file esistenti inizia con l'introduzione di una singola entità per rappresentare il file (infatti il file è una collezione di

[2]Questa situazione è nota nel mondo database, in lingua inglese, come legacy system.

dati con la stessa struttura) con lo stesso nome del file.

Quindi si considerano le clausole definite sul file allo scopo di individuare ulteriori proprietà strutturali del file.

In questo modo la rappresentazione iniziale del file è progressivamente arricchita tramite l'introduzione di nuove entità, associazioni, generalizzazioni, attributi e altri concetti.

Nel seguito si analizzano alcune clausole che possono comparire nella definizione di un file e vengono dati metodi generali per la loro traduzione in concetti del modello E/R.

Campi semplici e composti

♢ Un campo è semplice quando ha un singolo valore in ogni record istanza;

♢ I campi semplici sono trasformati in attributi semplici.

♢ I campi composti sono trasformati in attributi composti.

Esempio

Queste linee sono tradotte in attributi semplici dell'entità FATTURA

```
02 CLIENTE              PIC X(9).
02 AZIENDA              PIC X(9).
```

Queste linee sono tradotte in attributi composti dell'entità FATTURA

```
02 DATA_DI_EMISSIONE.
   03 ANNO_DI_EMISS.    PIC 9(4).
   03 MESE_DI_EMISS.    PIC 9(2).
   03 GIORNO_DI_EMISS.  PIC 9(2).
```

Campi ripetitivi

♢ Un campo ripetitivo ha valori multipli. Esso è definito tramite la clausola OCCURS che specifica il numero di volte che il campo compare in un record istanza;

♢ Un campo ripetitivo con una **singola** clausola OCCURS viene trasformato in un attributo che ha entrambe min-card e max-card uguali al valore specificato in OCCURS;

♢ Un campo ripetitivo con **più** clausole OCCURS (tabella o array) viene trasformato introducendo una nuova entità.

Esempio La tabella mostra l'andamento delle vendite di un prodotto, classificata per mese e anno.

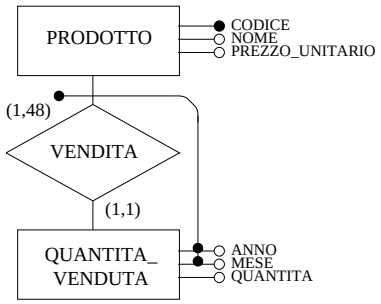
QUANTITÀ VENDUTA DI PRODOTTO

Mese	1994	1996	1997	1998
Gen	81	34	87	48
Feb	12	60	26	27
Mar	67	93	57	36
Apr	54	73	54	09
Mag	57	38	34	05
Giu	55	18	56	09
Lug	23	29	63	18
Ago	65	58	25	39
Set	67	36	95	55
Ott	48	49	48	49
Nov	38	39	75	40
Dic	48	75	39	80

La tabella è descritta dal record VENDITE_PRODOTTO in linguaggio COBOL, tramite l'uso dei campi ripetitivi.

```
01  VENDITE_PRODOTTO.
    02 NOME              PIC X(10).
    02 CODICE            PIC X(4).
    02 PREZZO_UNITARIO   PIC 9(9)V99.
    02 QUANTITA_DISPONIBILE.
       03 QUANTITA_VENDUTA_PER_ANNO OCCURS 4 TIMES.
          04 QUANTITA_VENDUTA_PER_MESE PIC X(3) OCCURS 12 TIMES.
```

Per trasformare il record VENDITE_PRODOTTO si introduce una nuova entità, QUANTITA_VENDUTA, che è connessa all'entità VENDITE_PRODOTTO tramite l'associazione VENDITA.
Si noti che
card(VENDITE_PRODOTTO, VENDITA) = (1,48)



Campi ridefiniti

Si può definire la stessa porzione di un record usando differenti clausole.

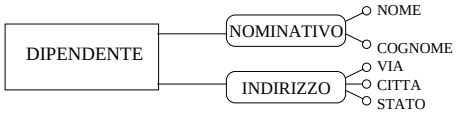
La ridefinizione di campo può essere usata per due motivi:

1. vedere gli stessi dati da differenti punti di vista;

   Esempio
   Consideriamo il record DIPENDENTE per la memorizzazione dei dipendenti di una azienda, dove memorizziamo il nome e il cognome e l'indirizzo di ciascuna persona. Il nominativo e la città sono aggregati in due modi diversi: il campo NOMINATIVO è utile per inviare comunicazioni al dipendente; il campo RICERCA può essere utile per operazioni di ricerca.

```
01  DIPENDENTE.
    02 DATI-PERSONALI.
       03 NOMINATIVO.
          04 COGNOME          PIC X(20).
          04 NOME             PIC X(20).
       03 INDIRIZZO.
          04 VIA              PIC X(20).
          04 CITTA            PIC X(10).
          04 STATO            X(10).
    02 DATI-PERSONALI-BIS  REDEFINES DATI-PERSONALI.
       03 RICERCA.
          04 NOMINATIVO-R     PIC X(40).
          04 CITTA-R          PIC X(10).
```

   ♢ La rappresentazione concettuale può essere una delle due o una loro combinazione.
   È preferibile avere tutti gli attributi; si ottiene quindi lo schema concettuale riportato in figura:



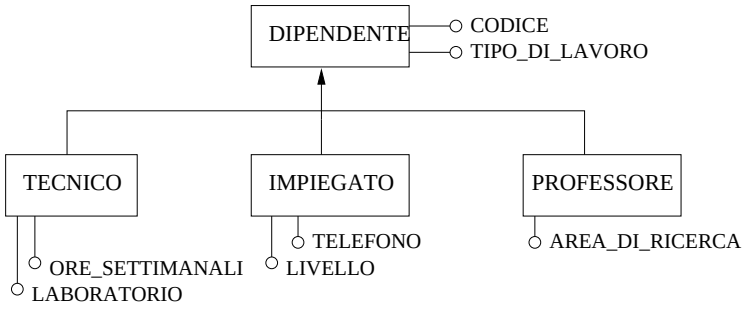
2. ottimizzare lo spazio di memorizzazione fisica.

Questo usualmente è indice della presenza di una gerarchia di generalizzazione tra i concetti descritti nel file.

Esempio
Per il record DIPENDENTE il campo INFO_TECNICO è ridefinito due volte nei campi INFO_IMPIEGATO e INFO_PROFESSORE.

```
01  DIPENDENTE.
    02 CODICE              PIC X(7).
    02 TIPO_DI_LAVORO      PIC X.
    02 INFO_TECNICO.
       03 ORE_SETTIMANALI  PIC 99.
       03 LABORATORIO      PIC X(6).
    02 INFO_IMPIEGATO      REDEFINES INFO_TECNICO.
       03 LIVELLO          PIC 9(2).
       03 TELEFONO         PIC 9(7).
    02 INFO_PROFESSORE     REDEFINES INFO_IMPIEGATO.
       03 AREA_DI_RICERCA  PIC X(20)
```

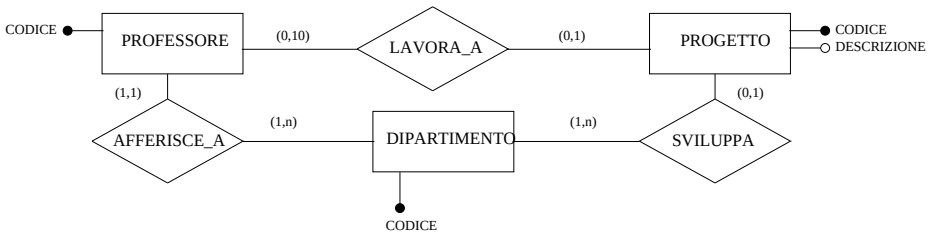
Il record viene tradotto in uno schema con l'entità DIPENDENTE e con una gerarchia di generalizzazione con sottoclassi TECNICO, IMPIEGATO e PROFESSORE.



Puntatore simbolico: è un campo di un record che denota l'identificatore di un altro record. I puntatori sono tradotti in associazioni:
queste associazioni connettono le entità con il campo puntatore all'entità corrispondente al file che contiene il record. Le cardinalità dell'associazione dipendono dal significato specifico dei campi.
Esempio
Consideriamo tre record relativi a PROFESSORE, DIPARTIMENTO e PROGETTO. Le associazioni tra questi concetti sono espressi tramite tre differenti campi usati come puntatori:
CODICE-DIP lega i professori ai loro dipartimenti di appartenenza
CODICE-PROGETTO connette i professori ai progetti ai quali lavorano
CODICE-DIP lega i progetti ai dipartimenti che li controllano.

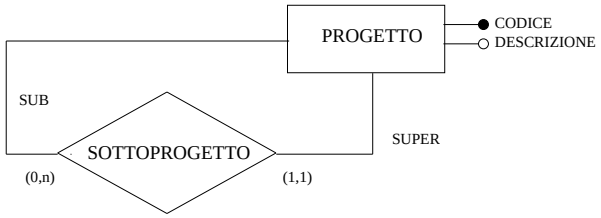
```
02  PROFESSORE.
    03 CODICE           PIC X(0).
    03 CODICE-DIP       PIC X(5).
    03 CODICE-PROGETTO  PIC X(7) OCCURS 10 TIMES.
02  DIPARTIMENTO.
    03 CODICE           PIC X(5).
02  PROGETTO.
    03 CODICE           PIC X(7).
    03 CODICE-DIP       PIC X(5).
    03 DESCRIZIONE      PIC X(30).
```

Lo schema E/R risultante è quindi il seguente:



Un puntatore che si riferisce al record stesso nel quale è definito, è tradotto in un'associazione binaria "ad anello" sull'entità stessa:

```
01  PROGETTO.
    02 CODICE
    02 DESCRIZIONE
    02 CODICE-SOTTOPROGETTO
    02 BUDGET
```



Flag

Flag: flag è riferito ad un campo (o ad un gruppo di campi) e indica se il campo (o il gruppo di campi) del record istanza ha un valore o è lasciato vuoto.
I flag possono essere tradotti in due modi differenti:

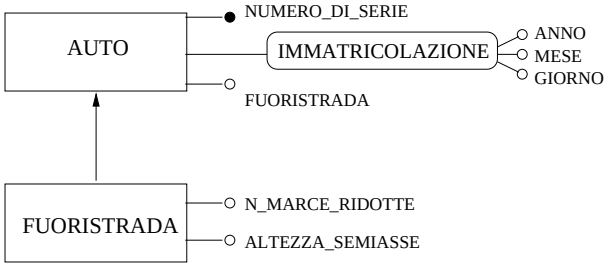
1. al file corrisponde un' unica entità con il campo indicato dal flag come attributo opzionale (min-card = 0);

2. al file corrispondono due entità legate da una relazione di generalizzazione.

Esempio
Consideriamo il file relativo alle auto di un concessionario. Poichè solo alcune auto sono considerate fuoristrada, si introduce il flag FLAG-FUORISTRADA: il valore è 0 per le auto fuoristrada e 1 per le altre. Se il valore è 0 allora si devono specificare i campi N_MARCE_RIDOTTE e ALTEZZA_SEMIASSE.

```
01  AUTO.
    02 NUMERO_DI_SERIE    PIC X(10).
    02 DATA-DI-IMMATRICOLAZIONE.
       03 ANNO     PIC 9(2).
       03 MESE    PIC 9(2).
       03 GIORNO     PIC 9(2).
    02 FLAG-FUORISTRADA    PIC 9.
       88 SI-FUORISTRADA     VALUE 0.
       88 NO-FUORISTRADA    VALUE 1.
    02 N_MARCE_RIDOTTE    PIC 9(2).
    02 ALTEZZA_SEMIASSE           PIC 9(5).
```

Poichè l'esempio ricade nel caso 2, lo schema E/R risultante introduce due entità legate da una relazione di generalizzazione:



Regole per i valori di un campo

Regole: l'uso specifico dei campi può essere espresso in termini di regole che definiscono il valore dei record istanza.

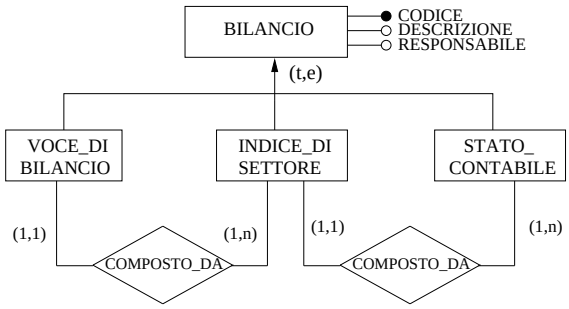
Esempio
Consideriamo un file relativo alla strutturazione del bilancio di una compagnia. Sono previsti tre livelli diversi: STATO_CONTABILE, INDICE_DI_SETTORE e VOCE_DI_BILANCIO. Ciascuno STATO_CONTABILE raggruppa un insieme di INDICI_DI_SETTORE e un INDICE_DI_SETTORE raggruppa un insieme di VOCE_DI_BILANCIO. Pertanto, nel file di BILANCIO, il campo INDICE_DI_SETTORE (codice di un indice di settore) è significativo solo per le voci di bilancio e il campo STATO_CONTABILE (codice di uno stato contabile) è significativo solo per gli indici di settore.

```
01  BILANCIO.
    02 CODICE             PIC 9(4).
    02 DESCRIZIONE        PIC 9(4).
    02 INDICE_DI_SETTORE  PIC 9(4).
    02 STATO_CONTABILE    PIC X(5).
    02 RESPONSABILE       PIC X(30).
```

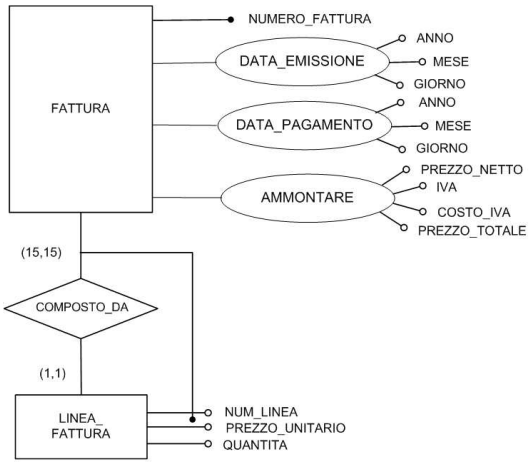
Le seguenti regole stabiliscono l'appartenenza di un record istanza ad uno dei tre livelli:

SE IL CODICE È TRA	ALLORA IL CODICE INDIVIDUA
1 e 999	uno stato contabile
1000 e 2999	un indice di settore
3000 e 9999	una voce di bilancio

Si può concludere che il file è la fusione di tre tipi di record logicamente differenti, connessi gerarchicamente. Un possibile schema E/R è il seguente, dove la struttura gerarchica è espressa da due associazioni:



Il file FATTURA precedentemente descritto è tradotto in uno schema con due entità, FATTURA e LINEA_FATTURA.



1.7 Documentazione di schemi E/R

♢ Uno schema E/R può essere uno strumento incompleto per rappresentare tutti gli aspetti e vincoli di un dominio applicativo, per varie ragioni:

1. i nomi delle entità e degli attributi possono non esser sufficiente per comprendere il significato di quello che rappresentano;
2. vari tipi di vincoli di integrità (proprietà dei dati rappresentati) non possono essere espressi direttamente dai costrutti del modello.

♢ **Documentazione di schemi E/R**: uno schema E/R è corredato con una documentazione di supporto allo scopo di facilitare l'interpretazione dello schema stesso e di descrivere vincoli di integrità non esprimibili in E/R

♢ **Regole aziendali (business rules)**: la descrizione di una proprietà che non è possibile rappresentare direttamente nel modello concettuale può essere espressa mediante delle regole aziendali (o business rules). Una regola aziendale può essere:

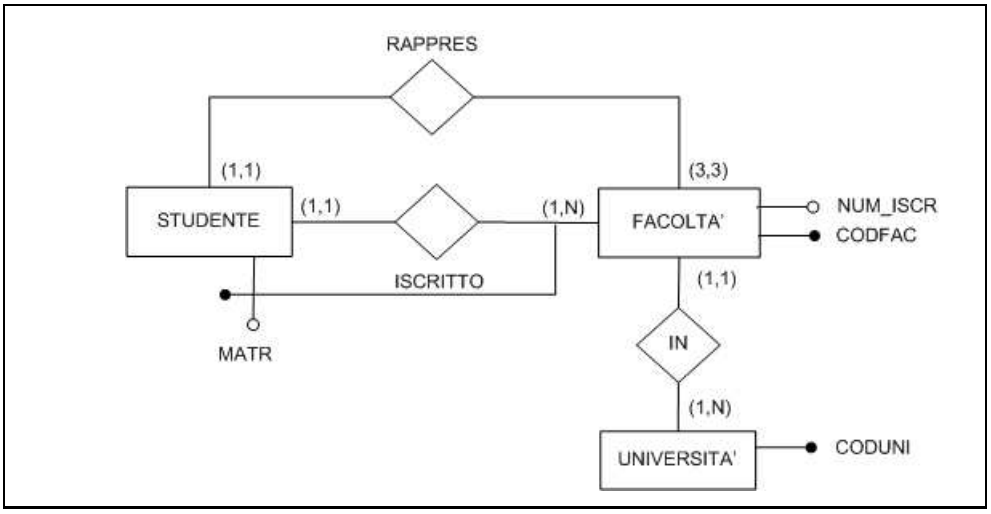
1. Una **descrizione di un concetto** (entità, associazione e attributo) dello schema E/R (l'insieme delle regole di questo tipo viene indicato come il **dizionario dei dati**);
2. Un **vincolo di integrità**, sia esso la documentazione di un vincolo espresso nello schema E/R o la descrizione di un vincolo non esprimibile in E/R;
3. Una **derivazione** ovvero un concetto che può essere ottenuto attraverso un'inferenza o un calcolo da altri concetti dello schema (**dato derivato**).

Non esiste un formalismo standard per esprimere le regole aziendali: normalmente si usano definizioni in linguaggio naturale opportunamente strutturate. Una metodologia condivisa di documentazione degli schemi E/R definisce il dizionario dei dati attraverso due tabelle che rappresentano rispettivamente le entità (indicando il nome, la descrizione, gli attributi e l'identificatore) e le associazioni (indicando il nome, la descrizione, le entità coinvolte e gli attributi). Per quanto riguarda le regole aziendali, si utilizza una ulteriore tabella nella quale si indicano le regole specificando di volta in volta la tipologia di appartenenza.

♢ Quando lo schema concettuale viene tradotto in uno schema logico e quindi in SQL, le regole aziendali devono essere opportunamente implementate. Esistono diverse modalità per effettuare tali implementazioni:

- definizione di vincoli in linguaggio SQL all'atto della definizione dello schema logico;
- definizione di trigger;
- definizione di opportune procedure in un linguaggio di programmazione.

Un esempio semplificato



Un esempio semplificato

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatore
Studente	Studente universitario	Matr,CodFac	Matr, CodFac
Facoltà	Facoltà di una Università	CodFac, Num_iscr	CodFac
Università	Ateneo universitario	Coduni	Coduni

Relazione	Descrizione	Entità Coinvolte	Attributi
Rappres	Associa a una Facoltà i propri rappresentanti	Studente (0,1), Facoltà (3,3)	Data
Iscritto	Associa a una Facoltà i propri iscritti	Studente(1,1), Facoltà (1,N)	-
In	Associa a una Università le Facoltà che la compongono	Facoltà (1,1), Università (1,N)	-

Regole Aziendali

Regole di Vincolo	
RV1	I rappresentanti di una Facoltà devono essere studenti della stessa Facoltà
RV2	Uno studente non può essere iscritto a duè Facoltà contemporaneamente sia della stessa Università sia di una diversa Università
RV3	Una matricola è univoca solo all'interno della Facoltà di appartenenza (espressa)
RV4	Una Facoltà ha esattamente tre rappresentanti (espressa)
RV5	Una Facoltà ha almeno uno studente iscritto (espressa)
...	...
Regole di Derivazione	
RD1	Il valore dell'attributo `NUM_ISCR` è pari al numero di studenti iscritti alla Facoltà (associati tramite l'associaztione `ISCRITTO`

♢ Non tutte le regole aziendali potranno trovare una implementazione nella base di dati. Nella sezione 3.2.2 si discuterà la definizione delle regole aziendali espresse in termini di progettazione logica relazionale.

Capitolo 2

Elementi di Teoria Relazionale

In questo capitolo vengono presentati i concetti fondamentali del Modello Relazionale dei dati, introdotto da Codd nel 1970, e dell'Algebra Relazionale [5, 36, 17].

Nella prima parte del capitolo, dopo aver introdotto il concetto matematico di relazione sul quale è basato il modello relazionale, vengono presentati i vincoli di integrità più importanti che possono essere espressi sui valori di una relazione.

La seconda parte introduce gli operatori principali dell'algebra relazionale ed alcuni esempi di interrogazioni di una base di dati relazionale tramite tali operatori.

2.1 Modello Relazionale

Il modello relazionale dei dati è stato introdotto da Codd nel 1970 (E.F. Codd, "A relational model of data for large shared data banks", *Comm. of the ACM*, 1970) ed è basato sul concetto matematico di relazione.

Dominio : insieme di valori $D = \{v_1, v_2, \ldots, v_k\}$

Tupla :

Dati n domini $D_1, D_2, \ldots, D_n$, non necessariamente distinti una *ennupla* (*tupla*) ordinata è definita come

$$t = (v_1, v_2, \ldots, v_n), v_i \in D_i, \forall 1 \leq i \leq n$$

Prodotto Cartesiano :

Il *prodotto cartesiano* di n domini $D_1, D_2, \ldots, D_n$, non necessariamente distinti, indicato con $D_1 \times D_2 \times \ldots \times D_n$, è l'insieme di tutte le tuple t su $D_1, D_2, \ldots, D_n$.

Relazione :

Una *relazione* R su n domini $D_1, D_2, \ldots, D_n$, non necessariamente distinti, è un sottoinsieme del prodotto cartesiano $D_1 \times D_2 \times \ldots \times D_n$:

$$R \subseteq D_1 \times D_2 \times \ldots \times D_n$$

- Il valore di n viene chiamato **Grado** della relazione
- Il numero di tuple viene chiamata **Cardinalità** della relazione

Esempi :

$D_1 = \{v_{11}, v_{12}\}$ $\quad$ $D_2 = \{v_{21}, v_{22}, v_{23}\}$

$D_1 \times D_2 = \{(v_{11}, v_{21}), (v_{11}, v_{22}), (v_{11}, v_{23}), (v_{12}, v_{21}), (v_{12}, v_{22}), (v_{12}, v_{23})\}$

$R_1 = \{(v_{11}, v_{21}), (v_{11}, v_{22})\}$ $\quad$ $R_2 = \{\}$

$R_3 = \{(v_{11}, v_{21}), (v_{11}, v_{22}), (v_{12}, v_{21}), (v_{12}, v_{22})\}$

Rappresentazione di relazioni :

Una relazione viene rappresentata generalmente tramite una tabella con un numero di righe pari alla cardinalità e un numero di colonne pari al grado

R_3

v_{11}	v_{21}
v_{11}	v_{22}
v_{12}	v_{21}
v_{12}	v_{22}

Schema ed Istanza

Attributo : Nome dato ad un dominio in una relazione

- si ottiene l'**indipendenza** dall'ordinamento dei domini
- si attribuisce **significato** ai valori del dominio

Schema di relazione :

Uno *schema di relazione* è una coppia costituita dal nome della relazione R e da un insieme di nomi degli attributi $X = (A_1, A_2, \ldots, A_n)$, indicato con $R(X)$.

Dato uno schema $R(X)$ si dice anche che lo schema della relazione R è X.

♢ I nomi degli attributi A_i devono essere tutti diversi.

Istanza di relazione :

Una *istanza di relazione* o *relazione* su uno schema $R(A_1, A_2, \ldots, A_n)$ è un insieme r di tuple su $(A_1, A_2, \ldots, A_n)$

Schema di base di dati :

È un insieme di schemi di relazioni $\mathbf{R} = \{R_1(X_1), R_2(X_2), \ldots, R_n(X_n)\}$

Tutti i nomi di relazione R_i devono essere differenti.

Istanza di base di dati :

Dato uno schema di base di dati $\mathrm{R} = \{R_1(X_1), R_2(X_2), \ldots, R_n(X_n)\}$, una *istanza* su $\mathbf{R}$ è un insieme di relazioni

$$\mathbf{r} = \{r_1, r_2, \ldots, r_n\}, \text{ dove } r_i \text{ è una relazione su } R_i, \ \forall 1 \leq i \leq n$$

♢ Differenza tra *tabella* (usata nei DMBS) e *relazione*
- Righe duplicate vs tuple distinte

Notazione :

♢ **Insieme di attributi** : Un insieme di attributi Y dello schema $R(X)$, cioè $Y \subseteq X$, può essere denotato anche con $R.Y$
A denota $\{A\}$, XY denota $\{X\} \cup \{Y\}$,

♢ **Tuple** : Data una tupla t su $R(X)$, un attributo $A \in X$ e un sottoinsieme $Y \subseteq X$
$t[A]$, oppure $t.A$, denota il valore di t su A
$t[Y]$, denota la sottotupla di t ottenuta considerando i valori degli attributi in Y

Esempio

Schema di base di dati :

DB_UNIVERSITÀ = { STUDENTE(MATR, NOME, CITTÀ, ACORSO)
CORSO(CODCOR, NOME, CODDOC)
DOCENTE(CODDOC, CF, CITTÀ)
FREQUENZA(MATR, CODCOR) }

Istanza di base di dati :

STUDENTE

MATR	NOME	CITTÀ	ACORSO
M1	Marco Quaranta	SA	1
M2	Giacomo Tedesco	PA	2
M3	Maria Mei	BO	1
M4	Ugo Rossi	MO	2
M5	Sara Neri	MO	2
M6	Agata Verdi	MI	1

CORSO

CODCOR	NOME	CODDOC
C1	Fisica 1	D1
C2	Analisi Matematica 1	D2
C3	Fisica 2	D1
C4	Analisi Matematica 2	D2
C5	Meccanica Razionale	D4

DOCENTE

CODDOC	CF	CITTÀ
D1	CF1	MO
D2	CF2	BO
D3	CF3	MO
D4	CF4	FI

FREQUENZA

MATR	CODCOR
M1	C1
M1	C3
M2	C1
M2	C2
M3	C1
M3	C2
M3	C3
M3	C4

Chiavi

♢ Informalmente, per *chiave* di una relazione si intende un sottoinsieme dei suoi attributi che identifica univocamente ogni tupla della relazione stessa.

♢ Dato uno schema di relazione $R(X)$, un sottoinsieme di attributi $K \subseteq X$ è detto *chiave* dello schema di relazione $R(X)$ se e solo se per ogni relazione r su $R(X)$ valgono le seguenti proprietà:

1. **Univocità :** $\forall t_1, t_2 \in r, t_1[K] = t_2[K] \Longrightarrow t_1 \equiv t_2$

   cioè, non esistono due tuple distinte di r con lo stesso valore della chiave

2. **Minimalità :** $\forall A_i \in K, K - A_i$ non verifica la proprietà 1.

   cioè, non esiste un sottoinsieme proprio di K con la proprietà di univocità.

♢ Un insieme di attributi $Y \subseteq X$ che contiene in modo stretto una chiave K, $Y \supset K$, è detto **superchiave** dello schema di relazione $R(X)$.

In altri termini, una superchiave soddisfa **solo** la proprietà 1. di Univocità.

♢ Per ogni schema di relazione $R(X)$, l'insieme X è una superchiave in quanto una relazione, essendo un insieme, contiene tuple distinte.

Quindi, ogni schema di relazione $R(X)$ ha almeno una chiave.

♢ Uno schema $R(X)$ può avere più chiavi dette **chiavi candidate**

Tra le chiavi candidate ne viene scelta una detta **chiave primaria**

Le rimanenti chiavi vengono dette **chiavi alternative**

Notazione :

♢ La **chiave primaria** di uno schema $R(X)$ si indica sottolineando gli attributi che la compongono: $R(\underline{K_1}, \underline{K_2}, \ldots, \underline{K_m}, A_2, \ldots, A_n)$

♢ Una **chiave alternativa** di uno schema $R(X)$ è riportata di seguito allo schema, contraddistinta dalla parola chiave **AK**:
$R(X)$
AK: $K_1, \ldots K_m$

Esempio :

DOCENTE (<u>CODDOC</u>, CF, CITTÀ)
AK: CF

Valori nulli e Vincolo di Entity Integrity

Valori nulli :

Ogni dominio di relazione viene esteso con un particolare valore, detto *valore nullo* e denotato con `null`, che rappresenta assenza di informazione. In questo modo è possibile introdurre nelle relazioni anche tuple in cui il valore di uno o più attributi non è disponibile.

♢ Ad esempio, con riferimento allo schema di relazione

`DOCENTE (CODICE, CF, NOME, CITTA_DI_NASCITA)`

si possono elencare vari casi in cui si deve inserire nella relazione `DOCENTE` una tupla il cui valore di un attributo non è disponibile:

- `CITTA_DI_NASCITA` del docente è sconosciuto
- il `CF` non è previsto per docenti di determinati paesi
- ...

♢ Il gruppo di standardizzazione ANSI (*American National Standard Institute*) ha specificato 14 diverse interpretazione per il valore `null`.

♢ Nei sistemi relazionali è possibile specificare se un attributo può o meno assumere il valore `null`.

Vincoli di Integrità :

I *vincoli* (o *regole*) di *integrità* stabiliscono condizioni di correttezza delle informazioni nella base di dati.

♢ La stessa dichiarazione di chiave K di uno schema di relazione R è una dichiarazione di vincolo di integrità in quanto stabilisce l'univocità dei valori assunti dagli attributi di K.

♢ In presenza di valori nulli, non sarebbe quindi possibile controllare l'univocità dei valori assunti dagli attributi di una chiave. Per questo motivo viene imposto il seguente vincolo.

Vincolo di Entity Integrity :

Tale vincolo stabilisce che gli attributi che costituiscono la chiave primaria o alternativa di una relazione non possono assumere valore nullo.

♢ Formalmente, un'istanza r di uno schema di relazione R con chiave primaria $K_1, K_2, \ldots, K_m$, $R(\underline{K_1}, \underline{K_2}, \ldots, \underline{K_m}, A_2, \ldots, A_n)$, soddisfa il vincolo di Entity integrity se e solo se

$$\forall t \in r, t[K_i] \neq \texttt{null},\ \forall 1 \leq i \leq m$$

Vincolo di Integrità Referenziale

♢ Nel Modello Relazionale, i riferimenti tra le tuple delle relazioni vengono stabiliti tramite i valori assunti dagli attributi nelle tuple.

Ad esempio, nella relazione `CORSO`, il docente di "Fisica1" viene stabilito assegnando all'attributo `CODDOC` il valore `D1`, che corrisponde al valore della chiave `CODICE`, nella relazione `DOCENTE`, della tupla `(D1,CF1,MO)`.

Il vincolo di integrità referenziale assicura che quando in una tupla si utilizza il valore di un attributo per riferirsi ad un'altra tupla, quest'ultima sia una tupla esistente.

♢ In uno schema di base di dati $\mathbf{R}$ un *vincolo di integrità referenziale* viene dichiarato specificando:

Foreign Key o Chiave Esterna : insieme di attributi $FK = \{FK_1, FK_2, \ldots, FK_n\}$ di uno schema di relazione $R_1 \in \mathbf{R}$

Chiave della Relazione riferita : schema di relazione $R_2 \in \mathbf{R}$, non necessariamente distinto da R_1, con una chiave $K = \{K_1, K_2, \ldots, K_m\}$, con $m = n$.

♢ Informalmente, un'istanza r $= \{r_1, r_2, \ldots\}$ su $\mathbf{R}$ soddisfa il vincolo di integrità referenziale se i valori sulla foreign key FK di ciascuna tupla di r_1 sono valori della chiave K di r_2, o sono valori nulli.

♢ Formalmente, un'istanza r $= \{r_1, r_2, \ldots\}$ su $\mathbf{R}$, soddisfa il vincolo di integrità referenziale se e solo se

$$\forall t_1 \in r_1, (t_1[FK_i] = \texttt{null} \vee \exists t_2 \in r_2 : t_1[FK_i] = t_2[K_i]),\ \forall 1 \leq i \leq n$$

Notazione : $R_1(X)$
FK: $FK_1, \ldots FK_n$ **REFERENCES** $R_2(K_1, \ldots K_n)$
• Se $K_1, \ldots K_n$ è la chiave primaria di R_2 può essere omessa:
FK: $FK_1, \ldots FK_n$ **REFERENCES** R_2

Esempio : `CORSO (CODICE, NOME, CODDOC)`
FK: `CODDOC` **REFERENCES** `DOCENTE`

Istanza Legale di Base di Dati :

♢ Dato uno schema di basi di dati $\mathbf{R} = \{R_1(X_1), R_2(X_2), \ldots R_n(X_n)\}$, un'istanza $\mathbf{r} = \{r_1, r_2, \ldots r_n\}$ su $\mathbf{R}$ tale che

• ciascuna relazione r_i soddisfa il vincolo di entity integrity
• $\mathbf{r}$ soddisfa tutti i vincoli di integrià referenziale imposti su $\mathbf{R}$

verrà detta istanza **legale** della base di dati $\mathbf{R}$.

2.2 Algebra Relazionale

♢ L'*Algebra Relazionale* è un insieme di operatori che si applicano alle relazioni e restituiscono relazioni.

♢ Tramite le *espressioni* dell'Algebra Relazionale è possibile formulare *interrogazioni* anche complesse sulla base di dati.

♢ L'Algebra Relazionale è composta da cinque operatori di base a partire dai quali si possono definire altri operatori, detti *derivati*.

♢ **Operatori Base :**

- **Unari :**
  - Selezione
  - Proiezione
- **Binari :**
  - Unione
  - Differenza
  - Prodotto Cartesiano

♢ **Principali Operatori Derivati :**

- Intersezione
- Join
- Divisione

Operatori Unari

Selezione : Operatore σ

$\diamondsuit$ Data una relazione r con schema X e un *predicato di selezione* F

- F è una espressione booleana di predicati semplici p
- un predicato semplice p è il confronto tra due espressioni che utilizzano nomi di attributi, costanti e operazioni aritmetiche

l'operazione di selezione $\sigma_F(r)$ ha come risultato una relazione con schema X definita dal sottoinsieme di tuple di r che soddisfano il predicato F

$$\sigma_F(r) = \{t \mid t \in r, F(t) = true\}$$

Es. "Studenti del secondo anno di corso"

$\sigma_{\texttt{ACORSO=2}}\ (\texttt{STUDENTE})$

MATR	NOME	CITTÀ	ACORSO
M2	Giacomo Tedesco	PA	2
M4	Ugo Rossi	MO	2
M5	Sara Neri	MO	2

Proiezione : Operatore π

$\diamondsuit$ Data una relazione r con schema X e un insieme di attributi $Y \subseteq X$, il risultato dell'operazione di proiezione $\pi_Y(r)$ è una relazione con schema Y definita dall'insieme di sottotuple di r ottenute considerando solo i valori su Y:

$$\pi_Y(r) = \{t[Y] \mid t \in r\}$$

Es. "Città e anno di corso degli studenti"

$\pi_{\texttt{CITTÀ,ACORSO}}\ (\texttt{STUDENTE})$

CITTÀ	ACORSO
SA	1
PA	2
BO	1
MO	2

Es. "Città degli studenti del secondo anno di corso"

$\pi_{\texttt{CITTÀ}}\ (\sigma_{\texttt{ACORSO=2}}\ (\texttt{STUDENTE}))$

CITTÀ
PA
MO

Operatori Binari

Unione : Operatore $\cup$

$\diamondsuit$ Date due relazioni r e s con lo stesso schema X, il risultato dell'operazione di unione $r \cup s$ è una relazione che ha come schema X ed è definita dall'unione delle tuple di r con le tuple di s

$$r \cup s = \{t \mid t \in r \vee t \in s\}$$

Es. "Città di studenti o docenti"

$$\pi_{\text{CITTÀ}}(\texttt{STUDENTE}) \cup \pi_{\text{CITTÀ}}(\texttt{DOCENTE})$$

Differenza : Operatore $-$

$\diamondsuit$ Date due relazioni r e s con lo stesso schema X, l'operazione di differenza $r - s$ è una relazione che ha come schema X ed è definita dalla differenza tra le tuple di r e quelle di s

$$r - s = \{t \mid t \in r \wedge t \notin s\}$$

Es. "Città di studenti ma non di docenti"

$$\pi_{\text{CITTÀ}}(\texttt{STUDENTE}) - \pi_{\text{CITTÀ}}(\texttt{DOCENTE})$$

Intersezione : Operatore $\cap$

$\diamondsuit$ L'operatore di intersezione $\cap$ è un **operatore derivato** definito come

$$r \cap s = r - (r - s) = \{t \mid t \in r \wedge t \notin (r - s)\} = \{t \mid t \in r \wedge t \in s\}$$

Es. "Città di studenti e di docenti"

$$\pi_{\text{CITTÀ}}(\texttt{STUDENTE}) \cap \pi_{\text{CITTÀ}}(\texttt{DOCENTE})$$

Prodotto Cartesiano e Join

Prodotto Cartesiano : Operatore $\times$

♢ Date due relazioni r e s con schema $R(X)$ e $S(Y)$, il risultato dell'operazione di prodotto cartesiano $r \times s$ è una relazione che ha come schema $R.X \cup S.Y$ ed è definita dall'insieme di tuple

$$r \times s = \{t \mid t = t_R t_S : t_R \in r \wedge t_S \in s\}$$

dove $t_R t_S$ indica la concatenazione della tupla t_R e della tupla t_S.

- Nello schema del prodotto cartesiano $R.X \cup S.Y$, un attributo A che è solo in $R.X$ $(S.Y)$ può essere indicato semplicemente con A.

Theta-Join : Operatore $\bowtie_F$

♢ L'operazione di *join* serve per combinare tuple prese da due o più relazioni sulla base di condizioni espresse sugli attributi delle tuple.

♢ Il Theta-join tra r e s definisce un sottoinsieme del prodotto cartesiano $r \times s$ ottenuto tramite una operazione di selezione.

♢ Date due relazioni r e s con schema $R(X)$ e $S(Y)$, e un *predicato di join* F costituito da una espressione booleana di predicati di confronto $A\Theta B$, dove $A \in X$, $B \in Y$ e Θ è un operatore di confronto, il Theta-join tra r e s è definito come

$$r \bowtie_F s = \sigma_F(r \times s)$$

♢ **Equijoin**
I predicati di confronto sono esclusivamente predicati di uguaglianza

Naturaljoin : Operatore $\bowtie$

♢ Date due relazioni r e s con schemi $R(X)$ e $S(Y)$, il *naturaljoin* (o *join naturale*) tra r e s, indicato con $r \bowtie s$, è il risultato dell'equijoin tra r e s su tutti gli attributi comuni a X e Y proiettato sull'unione degli attributi dei due schemi (XY)

$$r \bowtie s = \pi_{X \cup Y}(r \bowtie_{\bigwedge^i_{A_i \in X \cap Y}(R.A_i = S.A_i)} s)$$

SemiJoin e OuterJoin

Semijoin : Operatore $\ltimes$

♢ Date due relazioni r e s con schemi $R(X)$ e $S(Y)$ il semijoin da s a r, indicato con $r \ltimes s$ è la proiezione su X del join naturale di r e s

$$r \ltimes s = \pi_X(r \bowtie s)$$

♢ Si può verificare che $r \ltimes s = r \bowtie (\pi_{X \cap Y} s)$

OuterJoin :

♢ Il risultato di un join $r \bowtie s$ comprende solo le tuple t_R di r (t_S di s) che possono essere messe in corrispondenza, in base al predicato di join, con una tupla t_S di s (t_R di r).

Una tupla t_R di r (t_S di s) che non partecipa a tale corrispondenza, e che quindi non contribuisce al join, è detta *dangling*.

Più precisamente, una tupla t_R di r è detta *dangling* se non esiste $t \in r \bowtie s$ tale che $t[X] = t_R$ (stessa cosa per t_S di s).

♢ Gli operatori di OuterJoin servono per includere nel risultato del join anche le tuple dangling: tale tuple sono concatenate con tuple composte da valori nulli

left-outerjoin : $r =\!\bowtie s$

comprende le tuple dangling t_R di r

right-outerjoin : $r \bowtie\!= s$

comprende le tuple dangling t_S di s

full-outerjoin : $r =\!\bowtie\!= s$

comprende sia le tuple dangling t_R di r che le tuple dangling t_S di s

Esempi di Join

Es. "Studenti e docenti della stessa città "

$\texttt{STUDENTE} \bowtie_{\texttt{STUDENTE.CITTÀ=DOCENTE.CITTÀ}} \texttt{DOCENTE}$

MATR	NOME	STUDENTE. CITTÀ	ACORSO	CODICE	CF	DOCENTE. CITTÀ
M3	Maria Mei	BO	1	D2	CF2	BO
M4	Ugo Rossi	MO	2	D1	CF1	MO
M4	Ugo Rossi	MO	2	D3	CF3	MO
M5	Sara Neri	MO	2	D1	CF1	MO
M5	Sara Neri	MO	2	D3	CF3	MO

Oppure, usando il join naturale

$\texttt{STUDENTE} \bowtie \texttt{DOCENTE}$

MATR	NOME	CITTÀ	ACORSO	CODICE	CF
M3	Maria Mei	BO	1	D2	CF2
M4	Ugo Rossi	MO	2	D1	CF1
M4	Ugo Rossi	MO	2	D3	CF3
M5	Sara Neri	MO	2	D1	CF1
M5	Sara Neri	MO	2	D3	CF3

Es. "Studenti che frequentano i corsi (almeno uno) del docente D2"

$\texttt{FREQUENZA} \bowtie \sigma_{\texttt{CODDOC=D2}} \left(\texttt{CORSO}\right)$

MATR	CODCOR	NOME	CODDOC
M2	C2	Analisi Matematica 1	D2
M3	C2	Analisi Matematica 1	D2
M3	C4	Analisi Matematica 2	D2

La relazione risultante contiene solo la matricola degli studenti; per ottenere tutto lo schema della relazione STUDENTE:

$\texttt{STUDENTE} \bowtie \pi_{\texttt{MATR}} \left(\texttt{FREQUENZA} \bowtie \left(\sigma_{\texttt{CODDOC=D2}} \left(\texttt{CORSO}\right)\right)\right)$
oppure
$\texttt{STUDENTE} \ltimes \left(\texttt{FREQUENZA} \bowtie \sigma_{\texttt{CODICE='D1'}} \left(\texttt{CORSO}\right)\right)$

MATR	NOME	CITTÀ	ACORSO
M2	Giacomo Tedesco	PA	2
M3	Maria Mei	BO	1

Divisione

♢ Informalmente, date due relazioni r e s, l'operazione di *divisione* tra r (*dividendo*) e s (*divisore*), indicata con $r \div s$, serve per individuare le tuple di r associate a *tutte* le tuple di s

♢ Date due relazioni r e s con schemi $R(X)$ e $S(Y)$ tali che $Y \subset X$, l'operazione di *divisione* tra r e s , $r \div s$, ha come risultato una relazione che ha schema $(X - Y)$ ed è definita da

$$r \div s = \{t_D \mid \forall t_S \in s, t_D t_S \in r\}$$

Es. "Studenti che frequentano *tutti* i corsi del docente D1"

FREQUENZA $\div$ $\pi_{\texttt{CODCOR}}$ $\left(\sigma_{\texttt{CODDOC}='\texttt{D1}'}\ (\texttt{CORSO})\right)$

MATR	CODCOR
M1	C1
M1	C3
M2	C1
M2	C2
M3	C1
M3	C2
M3	C3
M3	C4

CODCOR
C1
C3

♢ L'operatore divisione $\div$ può essere derivato dagli operatori di base:

$$r \div s = \pi_{X-Y}(r) - \pi_{X-Y}((\pi_{X-Y}(r) \times s) - r)$$

Verifichiamo tale equivalenza sull'esempio precedente:

$\pi_{X-Y}(r) \times s$

MATR	CODCOR
M1	C1
M1	C3
M2	C1
M2	C3
M3	C1
M3	C3

$(\pi_{X-Y}(r) \times s) - r$

MATR	CODCOR
M2	C3

$\pi_{X-Y}((\pi_{X-Y}(r) \times s) - r)$

MATR
M2

$\pi_{X-Y}(r) -$
$\pi_{X-Y}((\pi_{X-Y}(r) \times s) - r)$

MATR
M1
M3

Esercizio

Consideriamo il seguente schema relazionale

AUTO(<u>CODAUTO</u>,COSTRUTTORE)
ACCESSORIO(<u>CODACC</u>,DESCRIZIONE)
INSTALLABILE(<u>CODAUTO</u>,<u>ANNOPROD</u>,<u>CODACC</u>)
FK: CODAUTO **REFERENCES** AUTO
FK: CODACC **REFERENCES** ACCESSORIO

L'accessorio CODACC è installabile sull'auto CODAUTO prodotta nell'anno ANNOPROD.

e una sua istanza:

AUTO

CODAUTO	COSTRUTTORE
Tipo	Fiat
Uno	Fiat
Fiesta	Ford
Escort	Ford

ACCESSORIO

CODAUTO	DESCRIZIONE
PSci9	PortaSci
FNeb11	FariAntiNebbia
VSport32	VolanteSportivo
RLega16	RuoteInLega

INSTALLABILE

CODAUTO	ANNOPROD	CODACC
Tipo	1989	PSci9
Tipo	1990	FNeb11
Fiesta	1990	PSci9
Uno	1989	VSport32
Uno	1989	PSci9
Uno	1990	PSci9
Uno	1991	FNeb11
Escort	1991	PSci9
Escort	1992	RLega16

a) selezionare i dati relativi agli accessori installabili su *almeno* un'auto 'Fiat';

Per determinare il codice degli accessori installabili su *almeno* un'auto 'Fiat' occorre fare il join naturale tra la relazione INSTALLABILE e la relazione AUTO ristretta con la condizione COSTRUTTORE='Fiat'.

$\left(\texttt{INSTALLABILE} \bowtie \sigma_{\texttt{COSTRUTTORE='Fiat'}} \left(\texttt{AUTO}\right)\right)$

CODAUTO	ANNOPROD	CODACC	COSTRUTTORE
Tipo	1989	PSci9	Fiat
Tipo	1990	FNeb11	Fiat
Uno	1989	VSport32	Fiat
Uno	1989	PSci9	Fiat
Uno	1990	PSci9	Fiat
Uno	1991	FNeb11	Fiat

In tale join c'è solo il codice degli accessori: per ottenere i dati (cioè tutti gli attributi) relativi a tali accessori, possiamo scrivere

$$\texttt{ACCESSORIO} \bowtie \left(\pi_{\texttt{CODACC}} \left(\texttt{INSTALLABILE} \bowtie \sigma_{\texttt{COSTRUTTORE}=\texttt{'Fiat'}} \left(\texttt{AUTO}\right)\right)\right)$$

oppure, in modo più sintetico, utilizzando l'operatore di semijoin

$$\texttt{ACCESSORIO} \ltimes \left(\texttt{INSTALLABILE} \bowtie \sigma_{\texttt{COSTRUTTORE}=\texttt{'Fiat'}} (\texttt{AUTO})\right)$$

b) selezionare i dati relativi agli accessori che non sono installabili su nessuna auto 'Fiat';

Gli accessori che non sono installabili su nessuna auto 'Fiat' si ottengono sottraendo dall'insieme di tutti gli accessori quelli installabili su almeno un'auto 'Fiat', ricavati in precedenza:

$$\texttt{ACCESSORIO} - \texttt{ACCESSORIO} \ltimes \left(\texttt{INSTALLABILE} \bowtie \sigma_{\texttt{COSTRUTTORE}=\texttt{'Fiat'}} (\texttt{AUTO})\right)$$

La differenza tra i due insiemi di tuple della relazione ACCESSORI può essere fatta considerando solo la chiave di tale relazione, CODACC:

$\pi_{\texttt{CODACC}} \left(\texttt{ACCESSORIO}\right)$

CODACC
PSci9
FNeb11
VSport32
RLega16

$\pi_{\texttt{CODACC}} \left(\left(\texttt{INSTALLABILE} \bowtie \sigma_{\texttt{COSTRUTTORE}=\texttt{'Fiat'}} \left(\texttt{AUTO}\right)\right)\right)$

CODACC
PSci9
FNeb11
VSport32

però a questo punto occorre fare ancora una operazione di join per ricavare tutti gli attributi della relazione ACCESSORI:

$$\texttt{ACCESSORIO} \bowtie \left(\pi_{\texttt{CODACC}} (\texttt{ACCESSORIO}) - \pi_{\texttt{CODACC}} \left(\texttt{INSTALLABILE} \bowtie \sigma_{\texttt{COSTRUTTORE}=\texttt{'Fiat'}} (\texttt{AUTO})\right)\right)$$

♢ Si noti che l'interrogazione **b)** non può essere risolta con un semplice join.

In particolare, con il seguente join:

$$\left(\texttt{INSTALLABILE} \bowtie \sigma_{\texttt{COSTRUTTORE}<>\texttt{'Fiat'}} \left(\texttt{AUTO}\right)\right)$$

CODAUTO	ANNOPROD	CODACC	COSTRUTTORE
Fiesta	1990	PSci9	Ford
Escort	1991	PSci9	Ford
Escort	1992	RLega16	Ford

si ottengono i codici degli accessori installabili su *almeno* un auto **non** 'Fiat';

c) selezionare i dati relativi agli accessori installabili su *tutte* le auto 'Fiat';

Questa interrogazione si risolve con un'operazione di divisione, in cui il divisore è costituito dal codice delle auto 'Fiat':

$$\pi_{\texttt{CODAUTO}}\big(\sigma_{\texttt{COSTRUTTORE}=\texttt{'Fiat'}}(\texttt{AUTO})\big)$$

Per quanto riguarda il dividendo, occorre far riferimento all'installabilità di un accessorio su un auto, indipendentemente dall'anno di produzione

$$\pi_{\texttt{CODACC,CODAUTO}}(\texttt{INSTALLABILE})$$

Quindi, in definitiva

$$\pi_{\texttt{CODACC,CODAUTO}}\ (\texttt{INSTALLABILE}\ )\ \ \div\ \pi_{\texttt{CODAUTO}}\ \big(\sigma_{\texttt{COSTRUTTORE}=\texttt{'Fiat'}}\ (\texttt{AUTO}\ )\ \big)$$

CODACC	CODAUTO
PSci9	Tipo
FNeb11	Tipo
PSci9	Fiesta
VSport32	Uno
PSci9	Uno
FNeb11	Uno
PSci9	Escort
RLega16	Escort

CODAUTO
Tipo
Uno

Lo schema risultante di tale divisione è CODACC: per ottenere i dati dello schema di ACCESSORI occorre fare il join con tale relazione:

$$\texttt{ACCESSORIO}\bowtie\big(\pi_{\texttt{CODACC,CODAUTO}}\ (\texttt{INSTALLABILE})\ \div\ \pi_{\texttt{CODAUTO}}\ \big(\sigma_{\texttt{COSTRUTTORE}=\texttt{'Fiat'}}\ (\texttt{AUTO})\big)\big)$$

♢ Si noti che se si considerasse come dividendo tutta la relazione INSTALLABILE

$$\texttt{INSTALLABILE}\ \ \div\ \pi_{\texttt{CODAUTO}}\ \big(\sigma_{\texttt{COSTRUTTORE}=\texttt{'Fiat'}}\ (\texttt{AUTO}\ )\ \big)$$

CODAUTO	ANNOPROD	CODACC
Tipo	1989	PSci9
Tipo	1990	FNeb11
Fiesta	1990	PSci9
Uno	1989	VSport32
Uno	1989	PSci9
Uno	1990	PSci9
Uno	1991	FNeb11
Escort	1991	PSci9
Escort	1992	RLega16

CODAUTO
Tipo
Uno

si otterrebbero i dati relativi agli accessori che nello *stesso anno di produzione* sono installabili su tutte le auto 'Fiat'.

Capitolo 3

Progettazione Logica

Con il termine *progettazione logica* si intende la traduzione di uno schema disegnato tramite un modello concettuale in uno schema disegnato tramite un modello logico. Il presente capitolo contiene le fasi principali della progettazione logica a partire da schemi concettuali realizzati tramite il modello E/R. Questa fase è indispensabile in quanto non esistono DBMS in grado di operare direttamente sugli oggetti di uno schema E/R.

La prima fase consiste in una semplificazione dello schema E/R (eliminazione di gerarchie e identificazioni esterne, normalizzazione di attributi composti o multipli, scelta di chiavi primarie) basata su criteri di ottimizzazione dello schema. Il risultato è ancora uno schema E/R, quindi, questa fase risulta indipendente dal modello logico scelto per l'implementazione della base di dati.

La seconda fase è riferita ad un particolare modello logico, il modello relazionale, e consiste nella vera e propria trasformazione dello schema E/R semplificato in uno schema relazionale. Verranno illustrate in dettaglio le regole di traduzione di entità e associazioni in schemi di relazioni e le ulteriori ottimizzazioni basate sulle caratteristiche del modello relazionale.

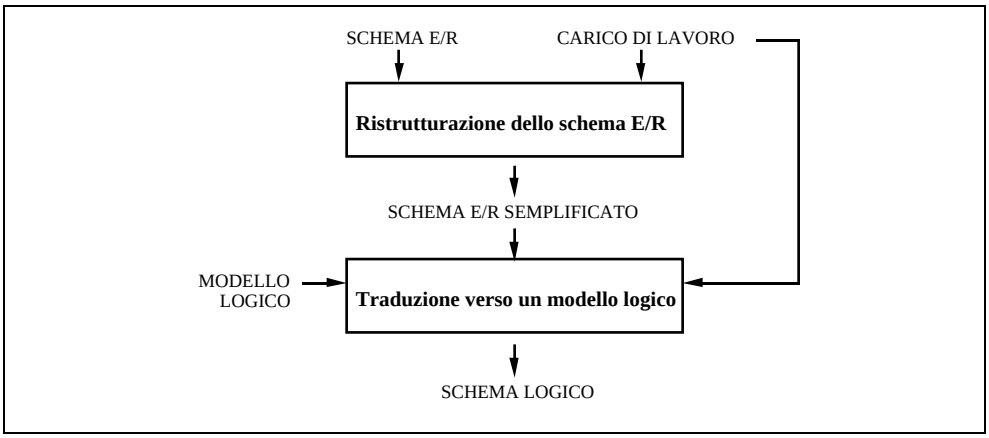
3.1 Progetto logico di alto livello con il modello E/R

♢ Non esistono DBMS in grado di operare direttamente sugli oggetti di uno schema E/R: è quindi necessario tradurli in modelli di dati supportati da DBMS, quali il gerarchico, reticolare, relazionale.

Fasi della progettazione logica

1) ristrutturazione dello schema E/R
- è indipendente dal modello logico
- si basa su criteri di ottimizzazione dello schema
- consiste nella eliminazione di gerarchie e identificazioni esterne, normalizzazione di attributi composti o multipli, scelta di chiavi primarie.

2) traduzione verso il modello logico
- è riferita ad un particolare modello logico: modello relazionale
- ulteriori ottimizzazioni basate sulle caratteristiche del modello logico vengono effettuate
- consiste nella traduzione di entità e associazioni in schemi di relazioni



♢ Dopo la prima fase, lo schema E/R è costituito soltanto da entità associazioni e attributi semplici.

♢ Ogni trasformazione impoverisce semanticamente lo schema; la semantica persa resterà sotto forma di vincoli di integrità che governeranno l'uso delle relazioni.

♢ In entrambe le fasi si deve considerare il carico di lavoro previsto, in termini di dimensione dei dati e caratteristiche delle operazioni.

♢ Si possono individuare alcune regole intuitive da seguire in questa fase:
- le proprietà logiche sono comunque primarie rispetto ai motivi di efficienza
- sullo stesso concetto vanno mantenute le informazioni che verranno di frequente consultate insieme
- informazioni che verranno consultate separatamente vanno suddivise su concetti distinti
- l'incidenza di valori nulli per attributi opzionali va limitata

Carico di Lavoro

Definizione: Il carico di lavoro sul DB è rappresentato sia dalla dimensione dei dati che dalle operazioni più significative che si stima saranno eseguite sul DB.
Regola 20-80: il 20% delle operazioni produce l'80% del carico.

♢ Si fa uso di informazioni non direttamente legate alla struttura fisica dei dati, in quanto questa non è ancora data.

Volume dei dati:
numero medio di istanze di ogni entità e associazione
cardinalità e dimensioni di ciascun attributo
percentuali di copertura di gerarchie

Tabella dei volumi:

Concetto	**Tipo**	**Volume dei dati**

Dove nel campo Concetto compare il nome, nel campo tipo il tipo che può essere E (entità), R (associazione) o A (attributo).

Descrizione delle operazioni:
tipo dell'operazione: Interattiva o Batch
frequenza: numero medio di esecuzioni in un certo periodo di tempo
schema di operazione: frammento dello schema E-R interessato dall'operazione sul quale viene disegnato il "cammino logico" da percorrere per accedere alle informazioni di interesse.

Tabella delle operazioni:

Operazione	**Tipo (I o B)**	**Frequenza**

♢ Con queste informazioni è possibile fare una stima del costo di un'operazione contando il numero di accessi alle istanze di entità e associazioni necessario per eseguire l'operazione.

Tabella degli accessi:

Concetto	**Accessi**	**Tipo**

Dove nel campo accessi compare il numero degli accessi.

♢ Le operazioni di scrittura (S) sono generalmente più onerose di quelle in lettura (L): il peso degli accessi in scrittura è doppio di quello delle letture.

Dati derivati

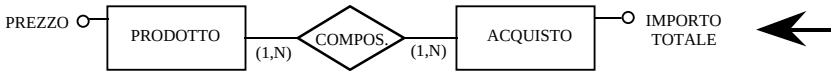
Definizione: Un dato derivato è un dato che può essere ottenuto attraverso una serie di operazioni da altri dati.
Sulla base delle operazioni e delle loro frequenze è possibile valutare se è conveniente o meno mantenere nello schema attributi derivati

Vantaggio: a tempo di accesso non è richiesta alcuna operazione per ricavare il valore dell'attributo

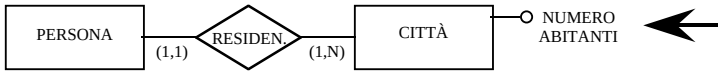
Svantaggi: occorre eseguire operazioni di aggiornamento per mantenere la consistenza dei dati,
si spreca memoria; tuttavia, allo stato attuale dei costi di CPU e memorizzazione si può ritenere in generale trascurabile l'onere dovuto ai maggiori costi di memorizzazione.

♢ Attributi derivabili da altri attributi della stessa entità o associazione

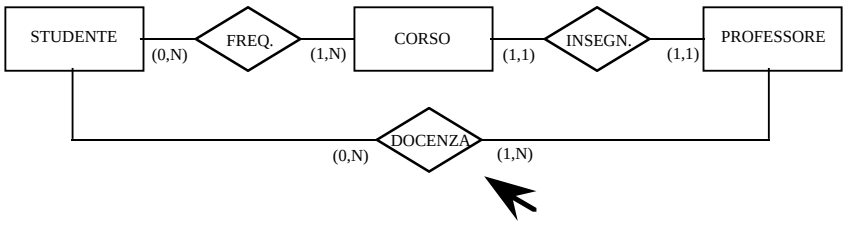
♢ Attributi derivabili da attributi di altre entità o associazioni



♢ Attributi derivabili da operazioni di conteggio di istanze



♢ Associazioni derivabili dalla composizione di altre associazioni



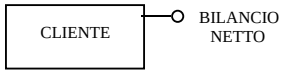
Esempio di Dato derivato

Esempio:

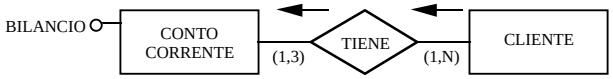
dato derivato: **Bilancio Netto**

operazione 1: lettura del bilancio netto di un cliente

Con il dato derivato



Senza il dato derivato



operazione 2: deposito su un conto corrente

Con il dato derivato

Senza il dato derivato

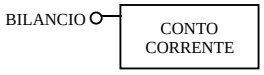


Tabella dei volumi

CONCETTO	TIPO	VOL.
Cliente	E	15000
ContoCorrente	E	20000
Tiene	R	30000

Tabella delle operazioni

OPER.	TIPO	FREQ.
Oper . 1	I	3000/Giorno
Oper . 2	I	1000/Giorno

Esempio di Dato derivato (2)

Con il dato derivato:

Occupazione di memoria: se ogni valore di "Bilancio Netto" richiede 6 bytes di memoria, la memoria richiesta è di 90 Kbytes.

Operazione 1	**CONCETTO**	**ACC.**	**TIPO**
1 accesso in lettura	Cliente	1	L
1*3000 = **3000**/giorno			
Operazione 2	ContoCorrente	1	L
4 accessi in lettura	ContoCorrente	1	S
2.5 accessi in scrittura	Tiene	1.5	L
	Cliente	1.5	L
9*1000 = **9000**/giorno	Cliente	1.5	S

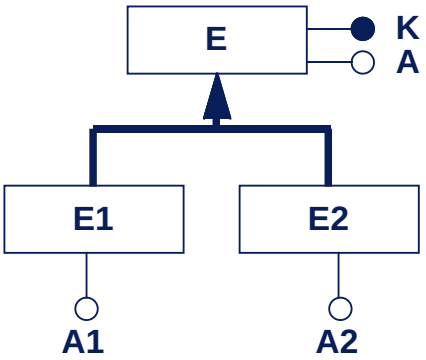
Senza il dato derivato:

Operazione 1	**CONCETTO**	**ACC.**	**TIPO**
5 accesso in lettura	Cliente	1	L
5*3000 = **15000**/giorno	Tiene	2	L
	ContoCorrente	2	L
Operazione 2	ContoCorrente	1	L
1 accesso in lettura	ContoCorrente	1	S
1 accesso in scrittura			
3*1000 = **3000**/giorno			

Conclusione: Conviene tenere il dato derivato.

Eliminazione delle gerarchie

♢ In questa attività si ottiene uno schema E/R semplificato in cui ogni gerarchia è sostituita da appropriate entità ed associazioni.



♢ Le alternative che si presentano sono tre:

1) mantenimento delle entità con associazioni,
2) collasso verso l'alto
3) collasso verso il basso

♢ Le trasformazioni delle gerarchie possono modificare, in generale, le cardinalità di attributi ed associazioni.

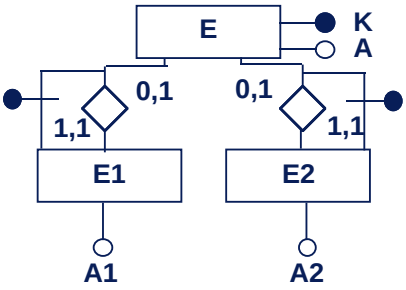
♢ l'applicabilità e la convenienza delle soluzioni dipendono dalle proprietà di copertura della gerarchia e dalle operazioni previste.

♢ le operazioni influiscono sulla convenienza delle varie soluzioni in base al modo in cui accedono a istanze e attributi della porzione di schema considerata:

- una operazione del tipo "fornire i valori di A e A1 per tutti gli E1" usa "insieme" gli attributi di E ed E1

- una operazione del tipo "fornire i valori di A per tutti gli E" usa "insieme" le istanze della gerarchia, con i soli attributi della classe generalizzazione

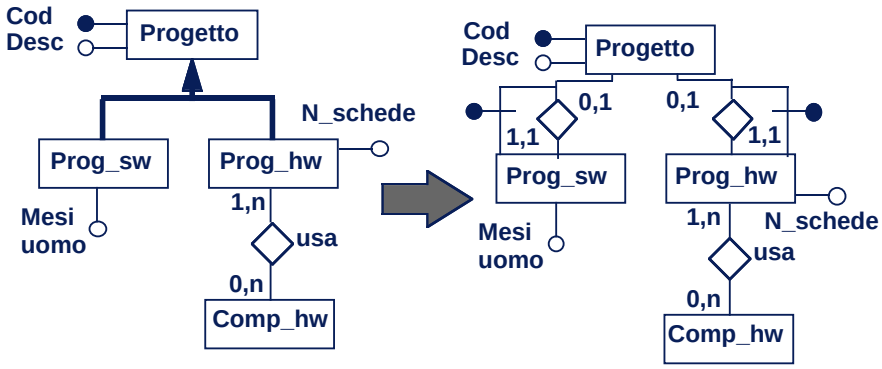
Mantenimento delle entità con associazioni



♢ Tutte le entità vengono mantenute: Le entità *figlie* sono in associazione binaria con l'entità *padre* e sono identificate esternamente.

♢ Questa soluzione è sempre possibile, indipendentemente dalla copertura.

Esempio:



Collasso verso l'alto

♢ Il collasso verso l'alto riunisce tutte le entità figlie nell'entità padre

♢ Sel è un attributo selettore che specifica se una singola istanza di E appartiene a una delle N sottoentità. Sel dipende dalla copertura:

- copertura totale esclusiva (t,e): Sel ha N valori, quante sono le sottoentità
- copertura parziale esclusiva (p,e): Sel ha N+1 valori; il valore in più serve per le istanze che non appartengono a nessuna sottoentità
- copertura sovrapposta (o): occorrono tanti selettori Sel_i quante sono le sottoentità , ciascuno a valore booleano, che è vero per ogni istanza di E che appartiene a E_i; se la copertura è parziale i selettori possono essere tutti falsi, oppure si può aggiungere un selettore

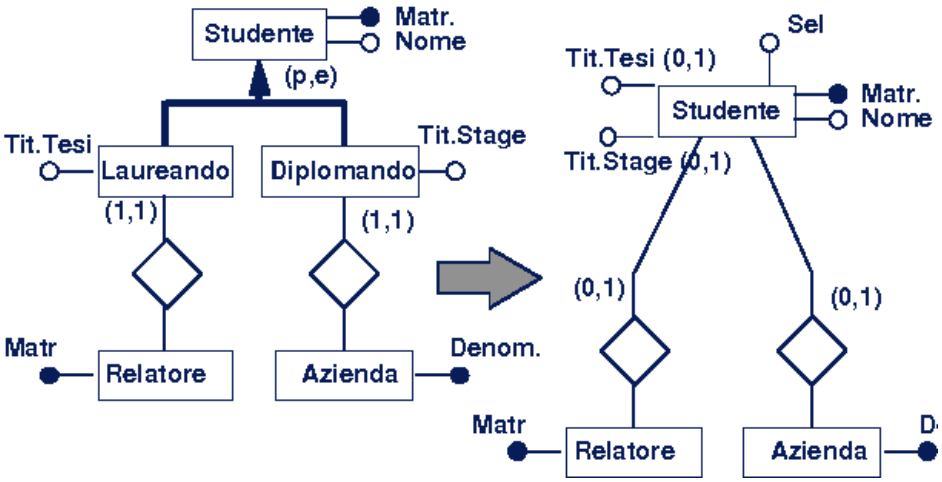
♢ Gli attributi obbligatori per le entità figlie divengono opzionali per l'entità padre oppure possono parzialmente sovrapporsi si avrà una certa percentuale di valori nulli

♢ le eventuali associazioni connesse alle sottoentità si trasportano su E, le eventuali cardinalità minime diventano 0

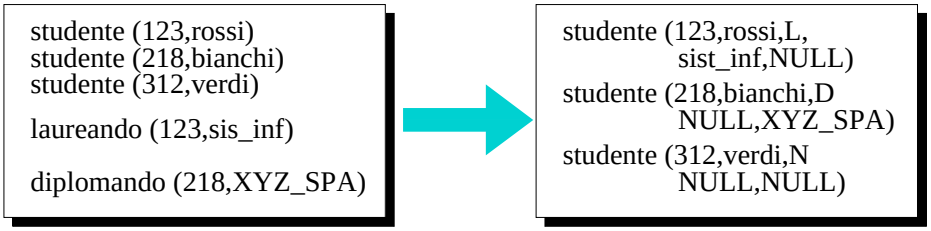
♢ Questa soluzione favorisce operazioni che consultano insieme gli attributi dell'entità padre e quelli di una entità figlia:

- in questo caso si accede a una sola entità, anzichè a due attraverso una associazione

Collasso verso l'alto esempio

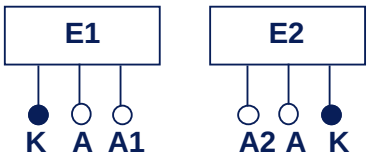


♢ esiste una precisa relazione tra il valore del selettore e i campi che possono avere valore diverso da NULL



♢ per una stima delle percentuali di NULL occorre conoscere le percentuali di laureandi e diplomandi

Collasso verso il basso



♢ Il collasso verso il basso elimina l'entità padre trasferendone attributi e associazioni su tutte le entità figlie. Una associazione dell'entità padre si moltiplica, tante volte quante sono le entità figlie

♢ Se la copertura non è completa *non si può fare*

- non si saprebbe dove mettere le istanze di E che non sono istanze né di E1, né di E2

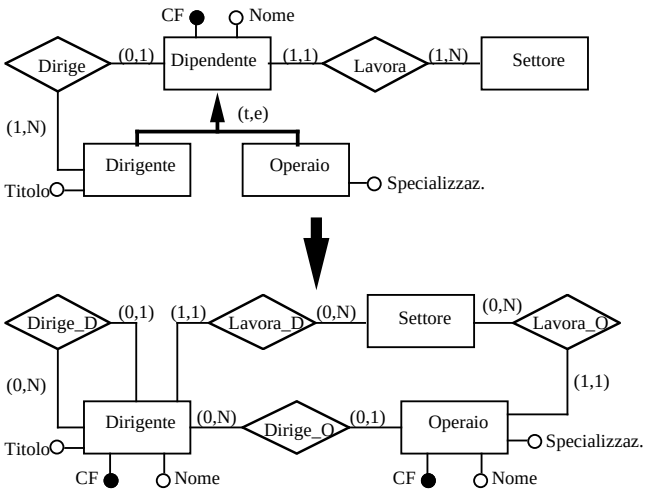
♢ Se la copertura non è esclusiva *introduce ridondanza*

- una certa istanza può essere sia in E1 che in E2, rappresentando due volte gli attributi che provengono da E

♢ Soluzione interessante in presenza di molti attributi di specializzazione. Favorisce le operazioni in cui si accede separatamente a ciascuna delle entità figlie

- esempio: fornire i valori di A per tutti gli E1

Esempio:



Ulteriori Trasformazioni (1)

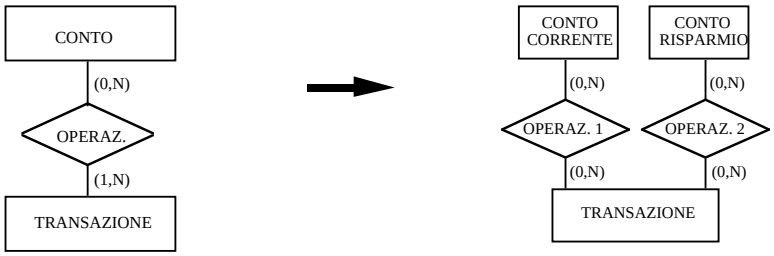
♢ **Partizionamento di entità**

Un'entità può essere partizionata orizzontalmente (istanze) o verticalmente (attributi) al fine di meglio rispondere alle operazioni previste.

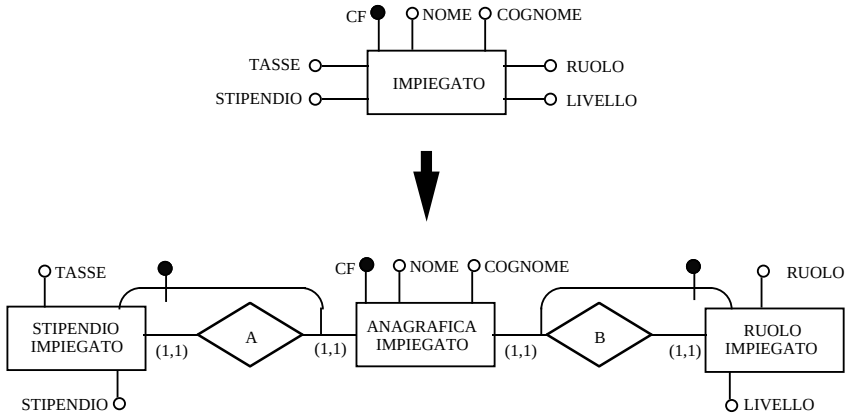
Il partizionamento (orizzontale e/o verticale) può anche essere indotto dalla presenza di diversi privilegi di accesso.

Il partizionamento (orizzontale e/o verticale) è utile soprattutto nel progetto di DBMS distribuiti.

Esempio di partizionamento orizzontale:



Esempio di partizionamento verticale:



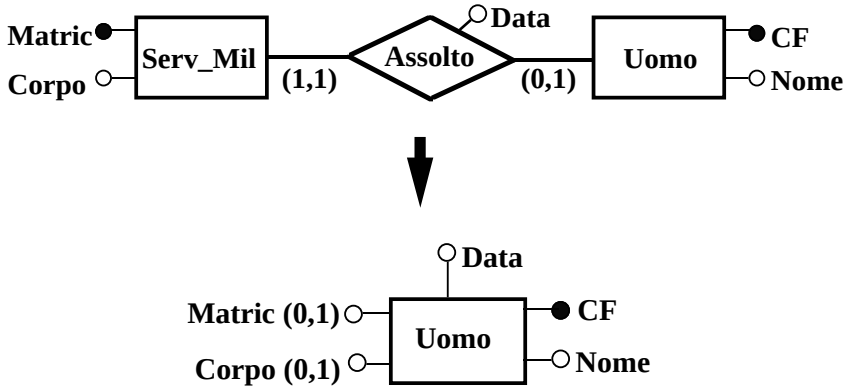
Ulteriori Trasformazioni (2)

♢ **Accorpamento di entità:**

Due entità partecipanti ad un'associazione uno a uno possono essere accorpate in un'unica entità contenente gli attributi di entrambi.

Per le associazioni uno a molti e molti a molti gli accorpamento di entità generano ridondanze.

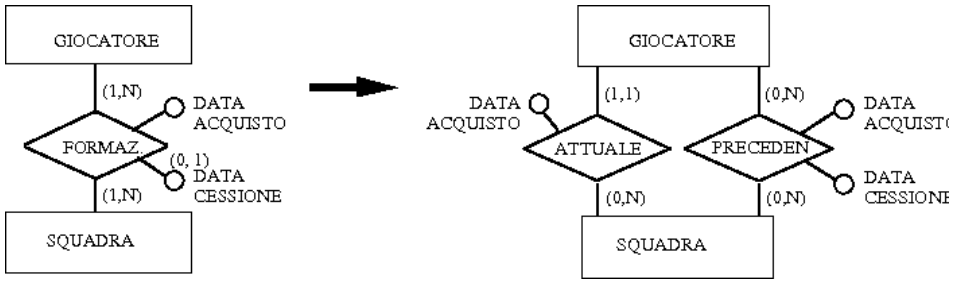
Esempio:



♢ **Partizionamento e accorpamento di associazioni:**

In questo caso il partizionamento è tipicamente verticale.

Esempio:



3.2 Progettazione logica relazionale

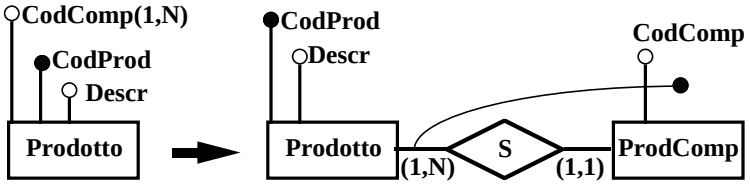
3.2.1 Trasformazione di attributi e identificatori

Attributi multivalore per le entità

♢ La **1NF** impone che, se una entità E ha un attributo multiplo A, si crei una nuova entità EA che ha A come attributo singolo ed è collegata ad E. Il tipo di collegamento dipende dai vari casi che verranno analizzati nel seguito, mostrando la semplificazione dell'attributo multivalore e quindi la traduzione in relazionale:

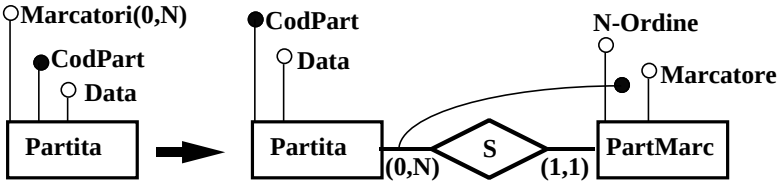
Caso a) un valore può comparire una volta sola nella ripetizione:

l'entità EA ha l'identificatore composto dall'entità E e l'attributo A

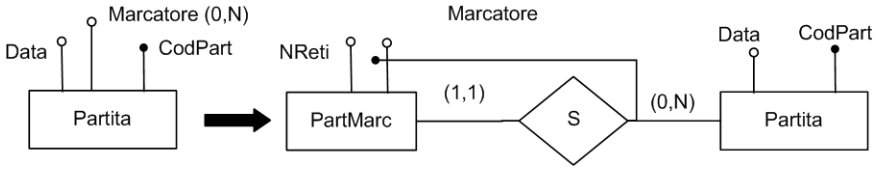


Caso b) un valore può comparire più volte nella ripetizione:

l'entità EA ha l'identificatore composto dall'entità E più un attributo identificante sintetico (ad esempio, un numero d'ordine)



*Soluzione Particolare per il **caso b)***: l'entità EA ha l'identificatore composto dall'entità E più l'attributo A; inoltre, l'entità EA ha un attributo che indica il numero di ripetizioni.



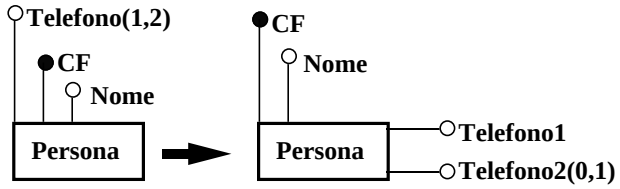
♢ Quando non specificato esplicitamente, si suppone di essere nel caso Caso **a)**: un valore può comparire una volta sola nella ripetizione.

Caso c) cardinalità massima nota: max-card(A, E)=K

l'entità EA ha come identificatore esterno l'entità E e K attributi , il cui valore sarà non nullo per i primi H, dove H=min-card(A,E)

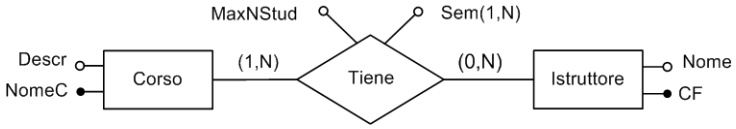
in questo caso si può evitare l'introduzione di EA, sostituendo direttamente l'attributo A con i K attributi

per evitare eccessivi valori nulli, tale soluzione conviene per K piccoli

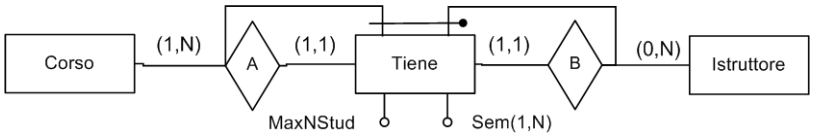


Attributi multivalore per le associazioni

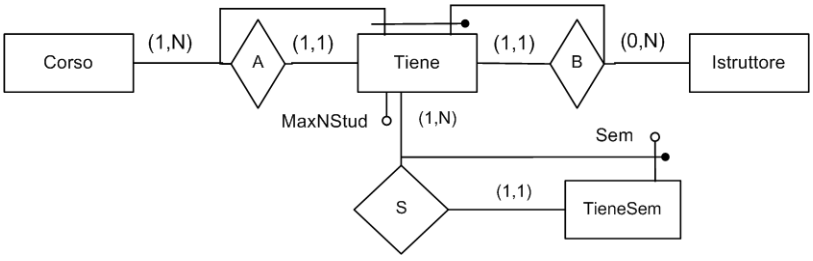
♢ Invece di introdurre regole per gli attributi multipli sulle associazioni, conviene **reificare** l'associazione: in questo modo si ha un attributo multiplo su un'entità e si applicano le regole già viste



Reifichiamo l'associazione `Tiene`:



Semplifichiamo l'attributo multiplo `Sem`:



Risoluzione degli Identificatori

♢ Ogni identificatore interno dell'entità E corrisponde ad una *chiave candidata* della relazione E.

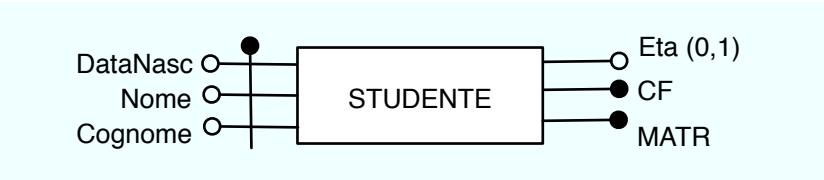
Scelta della chiave primaria : È necessario che tra le diverse chiave candidate di una entità sia designata la chiave primaria.

♢ Criteri euristici:

- scegliere la chiave che è usata più frequentemente per accedere direttamente alle istanze dell'entità
- preferire chiavi semplici a chiavi composte
- eventualmente introdurre una *chiave surrogata*, ovvero un attributo che non ha significato e di cui è garantita l'unicità all'interno dell'entità (es. numero progressivo, appositi codici interni di identificazione).

♢ Tutte le chiavi di E diverse dalla chiave primaria sono denominate *chiavi alternative*.

♢ Esempio: Traduzione di una entità in relazionale

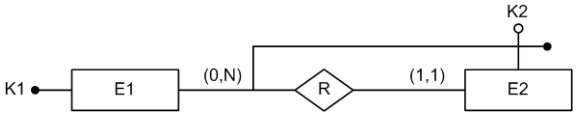



```
Studente(Matr,CF,Nome,Cognome,DataNasc,Eta)
   AK: CF
   AK: Nome,Cognome,DataNasc
```

Eliminazione degli identificatori esterni

♢ L'eliminazione delle identificazioni esterne è un'operazione di semplificazione già riferita al modello logico (nel nostro caso al modello relazionale).

♢ Una componente di identificazione esterna di una entità E2 da una entità E1 tramite una associazione R comporta il trasporto dell'identificatore di E1 su E2.

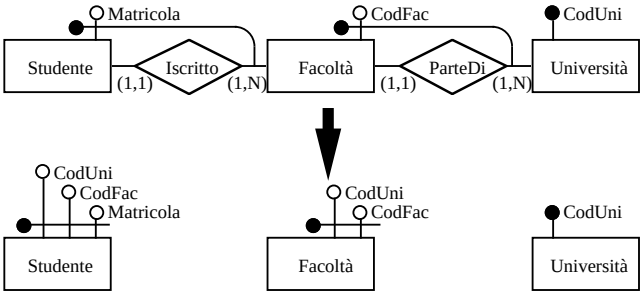


♢ Nella trasformazione in relazionale, l'identificatore di E1 trasportato nella relazione E2 diventa
- una parte della chiave di E2
- una foreign key riferita alla relazione E1.

♢ Grazie alla presenza in E2 della foreign key riferita alla relazione E1, l'associazione R tra E1 e E2 è automaticamente tradotta e può essere eliminata.

♢ In presenza di più identificazioni in cascata, è necessario iniziare la propagazione dall'entità che non ha identificazioni esterne

Nell'esempio quindi si parte dall'entità `Università`, si prosegue su `Facoltà` e poi su `Studente`.



Lo schema relazionale corrispondente sarà:

`Università(`<u>`CodUni`</u>`)`
`Facoltà(`<u>`CodFac`</u>`,`<u>`CodUni`</u>`)`
FK: `CodUni` **REFERENCES** `Università`
`Studente(`<u>`Matricola`</u>`,`<u>`CodFac`</u>`,`<u>`CodUni`</u>`)`
FK: `CodFac,CodUni` **REFERENCES** `Facoltà`

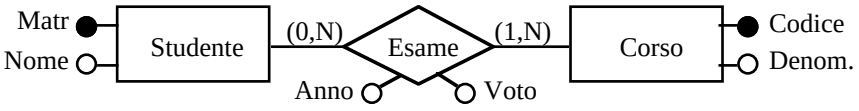
♢ Si noti che il passaggio intermedio dello schema semplificato mostrato in figura è ovvio e quindi conviene scrivere direttamente lo schema relazionale.

3.2.2 Traduzione di entità e associazioni

Traduzione standard

♢ Ogni entità è tradotta con una relazione con gli stessi attributi
- a chiave primaria della relazione è quella dell'entità stessa
- ogni altro identificatore dell'entità è chiave alternativa della relazione

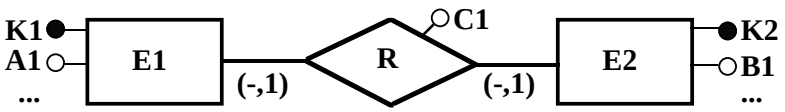
♢ Ogni associazione R tra le entità E1, E2, ..., En, è tradotta con una relazione con gli stessi attributi, cui si aggiungono le chiavi primarie di tutte le entità che essa collega
- ogni chiave primaria di un'entità Ei tale che max-card(Ei, R)=1, è una chiave (candidata) della relazione R; altrimenti la chiave della relazione è composta dall'insieme di tutte le chiavi primarie delle entità collegate (o da un sottoinsieme, nel caso che tale insieme denoti una superchiave);
- le chiavi primarie delle entità collegate sono chiavi straniere NOT NULL (FK) riferite alle corrispondenti entità;
- se la chiave straniera fa parte di una delle chiavi non necessario indicare NOT NULL.




```
Studente(Matr,Nome)
Corso(Codice,Denom)
Esame(Matr,Codice,Anno,Voto)
   FK: Matr REFERENCES Studente
   FK: Codice REFERENCES Corso
```

Altre traduzioni
- La traduzione standard è sempre possibile
- La traduzione standard è praticamente l'unica possibilità per le associazioni in cui tutte le entità partecipano con molteplicità maggiore di 1 (max-card(Ei, R) > 1, per ogni i)
- Altre forme di traduzione dell'associazione sono possibili per casi particolari di cardinalità
- Le altre forme di traduzione tendono a fondere in una stessa relazione entità e associazioni dando origine a un minore numero di relazioni e generano quindi uno schema più semplice
- richiedono un minore numero di operazioni di join per la navigazione attraverso una associazione, ovvero per conoscere le istanze di E1 connesse a E2 tramite R, ma penalizzano le operazioni che consultano soltanto gli attributi di una entità che è stata fusa

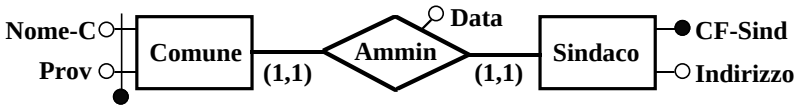
Associazione binaria uno a uno tradotta con una relazione



E12(K1,A1,A2,...,K2,B1,B2,...,C1,C2,...)

- se l'associazione è obbligatoria per entrambe le entità la chiave primaria può essere indifferentemente K1 o K2 (l'altra sarà chiave alternativa);

  ogni altro identificatore delle entità è chiave (alternativa) della relazione.




```
Ammin(Nome-C,Prov,CF-Sind,Indirizzo,Data)
   AK: CF-Sind
```

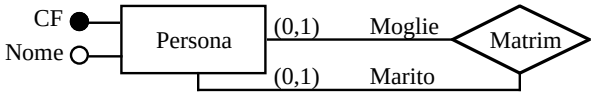
- se l'associazione è parziale per una entità (E1) e obbligatoria per l'altra (E2), la chiave deve essere K1;

  saranno possibili valori nulli per gli attributi di E2 e di R.


```
Uomo(CF,Nome,Matric,Corpo,Data)
   Matric UNIQUE
```

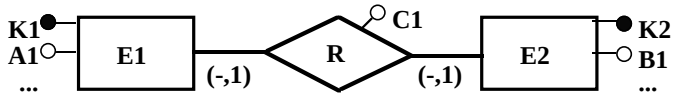
- se l'associazione è parziale per entrambe, occorre che entrambe le entità abbiano lo stesso identificatore; la chiave può essere indifferentemente K1 o K2 (l'altra sarà foreign key);

  saranno possibili valori nulli per gli attributi di E1, di E2 e di R.




```
Persona(CF,Nome,CFConsorte)
   FK: CFConsorte REFERENCES Persona
   CFConsorte UNIQUE
```

Associazione binaria uno a uno tradotta con due relazioni



- L'associazione si può compattatare in una delle entità, diciamo E1, includendo in E1 gli attributi di R e la chiave primaria di E2 come foreign key.

```
E2(K2,B1,B2,...)
E1(K1,A1,A2,...,K2,C1,C2,...))
   FK: K2 REFERENCES E2
```

- Per evitare i valori nulli, è preferibile compattare l'associazione in un'entità che partecipa obbligatoriamente.

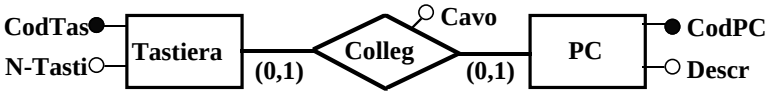

```
Uomo(CF,Nome)
Serv-Mil(Matric,Corpo,CF,Data)
   AK: CF
   FK: CF REFERENCES Uomo
```

- Sono possibili anche altre alternative:

```
Uomo(CF,Nome,Matric)
   FK: Matric REFERENCES Serv-Mil
   Matric UNIQUE
Serv-Mil(Matric,Corpo,CF,Data)
   AK: CF
   FK: CF REFERENCES Uomo
```

♢ **Associazione binaria uno a uno tradotta con tre relazioni**

- È preferibile se l'associazione è parziale per entrambe le entità . Praticamente forzata se, oltre alla parzialità , le due chiavi primarie delle entità hanno domini distinti




```
Tastiera(CodTas,N-Tasti)
PC(CodPC,Descr)
Colleg(CodPC,CodTas,Cavo)
   AK: CodTas
   FK: CodPC REFERENCES PC
   FK: CodTas REFERENCES Tastiera
```

Associazione binaria uno a molti

♢ **Traduzione con due relazioni**

L'associazione può essere compattata nell'entità che partecipa con molteplicità unitaria, diciamo E1, includendo in E1 gli attributi di R e la chiave primaria di E2 come foreign key:

`E2(K2,B1,B2,...)`

`E1(K1,A1,A2,...,K2,C1,C2,...)` **FK: K2 REFERENCES E2**


```
Provincia(Nome-P,Regione)
Persona(CF,Nome,Nome-P,Dal)
   FK: Nome-P REFERENCES Provincia NOT NULL
```

♢ **Traduzione con tre relazioni**

Se la partecipazione di E1 è parziale, per evitare i valori nulli, si può optare per la traduzione standard con tre relazioni:


```
Venditore(Nome,Telefono)
Ordine(Numero,Data)
Scrive(Numero,Nome,Sconto)
   FK: Numero REFERENCES Ordine
   FK: Nome REFERENCES Venditore NOT NULL
```

Associazioni unarie

Una associazione unaria può dar luogo ad una o due relazioni, dipendentemente dalle molteplicità in gioco.

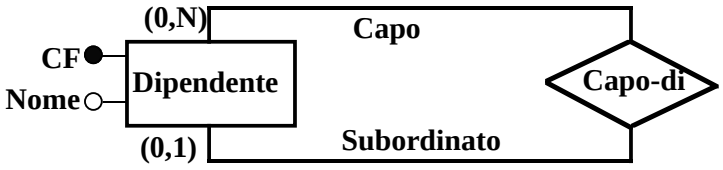
♢ **Anello molti a molti**

È tradotto con due relazioni, una per l'entità e una per l'associazione; la chiave della relazione che modella l'associazione è composta da due attributi, i cui nomi riflettono il diverso ruolo dell'entità ; ognuno di questi due attributi è anche foreign key.


```
Stato(Sigla,Nome)
Confina(Sigla,Sigla-Confinante,Lung-Conf)
   FK: Sigla REFERENCES Stato
   FK: Sigla-Confinante REFERENCES Stato
```

♢ **Anello uno a molti**

Oltre che con due relazioni, è traducibile con una sola relazione che contiene due volte l'attributo identificatore: una volta come chiave primaria e una volta come chiave esterna con un nome che riflette il ruolo dell'entità.



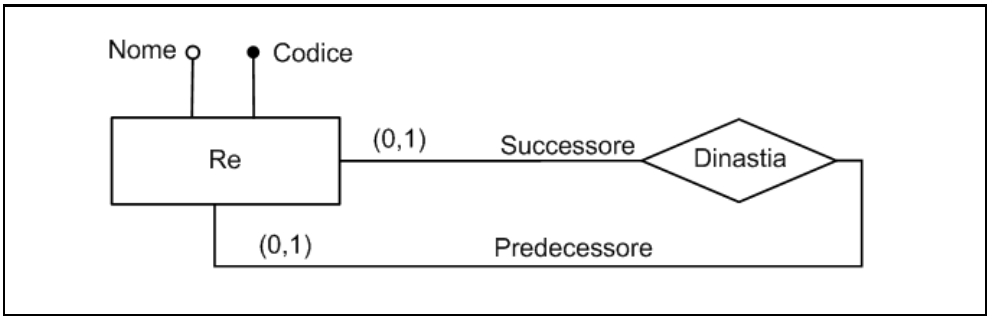

```
Dipendente(CF,Nome,CF-Capo)
   FK: CF-Capo REFERENCES Dipendente
```

La traduzione standard (che comporta due relazioni) del precedente schema è la seguente:

```
Dipendente(CF,Nome,CF-Capo)
Capo-Di(CF,CF-Capo)
   FK: CF REFERENCES Dipendente
   FK: CF-Capo REFERENCES Dipendente NOT NULL
```

♢ **Anello uno a uno**

Considerando il seguente schema:



Abbiamo due traduzioni possibili, entrambe corrette

(1)

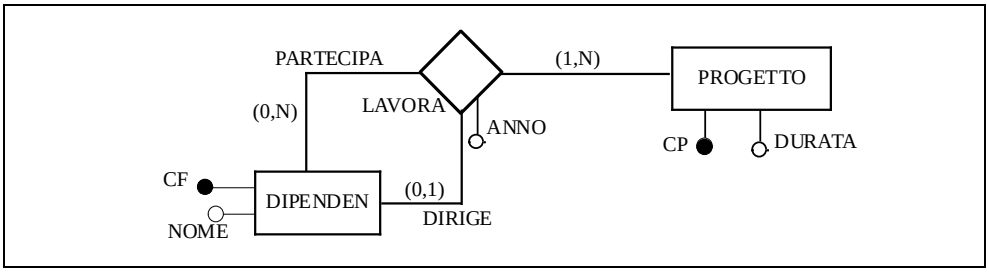
```
Re(Codice,Nome,Codice-Predecessore)
   FK: Codice-Predecessore REFERENCES Re
   Codice-Predecessore UNIQUE
```

(2)

```
Re(Codice,Nome,Codice-Successore)
   FK: Codice-Successore REFERENCES Re
   Codice-Successore UNIQUE
```

Supponiamo di modificare lo schema considerando la cardinalità `CARD-MIN(Re, Codice-Successore)=1`: ogni `Re` partecipa almeno una volta a `Dinastia` come Successore, ovvero ogni `Re` deve essere successore *di un altro* Re.
In questo modo non riesco a rappresentare il *primo re della dinastia*, in quanto tale Re non e' successore di nessuno.
D'altra parte si potrebbe dire che il *primo re della dinastia* è successore di se stesso (ricordiamo che in E/R non è esprimibile il vincolo che due istanze che partecipano ad una associazione siano diverse) riportandolo sia nel ruolo di successore che di predecessore; ma in questo modo non può essere più riportato come predecessore ...; in definitiva lo schema con `CARD-MIN(Re, Codice-Successore)=1` risulta essere inutilizzabile.

Esempio di traduzione di un anello



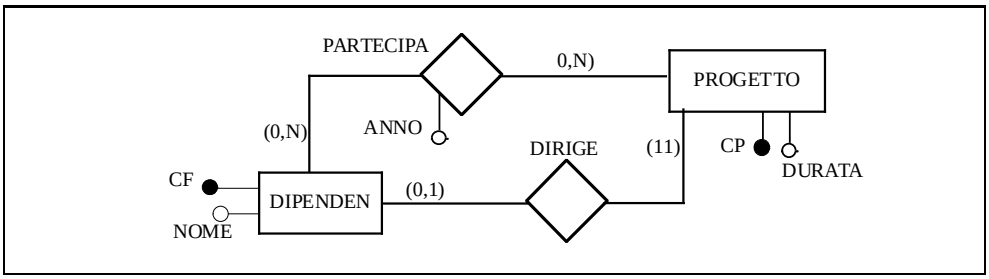
- Traduzione standard (l'associazione è tradotta con una relazione):

```
Dipendente(CF,Nome)
Progetto(CP,Durata)
Lavora(CF-Dirige,CF-Partecipa,CP,Anno)
   FK: CF-Dirige REFERENCES Dipendente
   FK: CF-Partecipa REFERENCES Dipendente NOT NULL
   FK: CP REFERENCES Progetto NOT NULL
```

- L'associazione può essere compattata nell'entità che partecipa con cardinalità massima uguale a 1, ovvero nell'entità `DIPENDENTE` con ruolo `DIRIGE`:

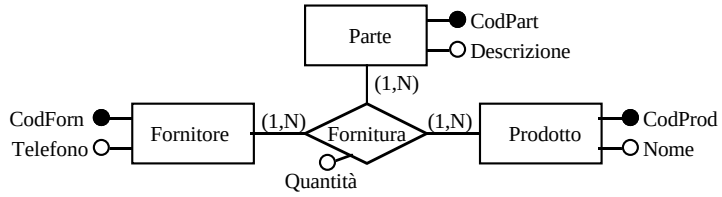
```
Progetto(CP,Durata)
Dipendente(CF,Nome,CF-Partecipa,CP,Anno)
   FK: CF-Partecipa REFERENCES Dipendente
   FK: CP REFERENCES Progetto
```

Entrambe le traduzioni di `LAVORA` sono corrette: in tale schema E/R una istanza dell'associazione `LAVORA` è una terna costituita dal progetto p, da un dipendente dd nel ruolo di dirigente e un dipendente dp nel ruolo di partecipante. Pertanto se si voleva rappresentare l'informazione che il progetto $p1$ è diretto da $d1$ e vi partecipano $d2$ e $d3$ tale schema non corretto.
Questo è un tipico esempio nel quale i problemi di uno schema E/R vengono individuati durante la progettazione logica (e la verifica dei requisiti). Rilevato il problema, esso può essere corretto nello schema E/R iniziale. Nell'esempio in questione, per rappresentare l'informazione che il progetto $p1$ è diretto da $d1$ e vi partecipano $d2$ $d2$, si deve modificare lo schema rendendo indipendenti l'associazione che rappresenta la direzione del progetto da quella che rappresenta la partecipazione
Una possibile soluzione è la seguente:



Associazione n-aria

♢ E' normalmente tradotta seguendo la traduzione standard




```
Prodotto(CodProd,Nome)
Fornitore(CodForn,Telefono)
Parte(CodPart,Descrizione)
Fornitura(CodPart,CodProd,CodForn,Quantità)
   FK: CodPart REFERENCES Parte
   FK: CodProd REFERENCES Prodotto
   FK: CodForn REFERENCES Fornitore
```

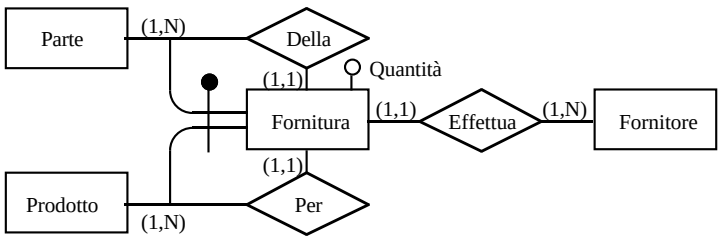
♢ Come per le associazioni binarie, in presenza di entità Ei che partecipano all'associazione n-aria R con $max - card(Ei, R) = 1$, sono possibili altre traduzioni applicabili in casi particolari, come discusso a pagina 119.

Associazione n-aria con vincoli

♢ Supponiamo di aggiungere al precedente schema E/R il seguente vincolo: ogni parte di un dato prodotto è fornita da un *unico* fornitore.

Tale vincolo introduce la seguente dipendenza funzionale: il fornitore dipende dalla parte e dal prodotto

1 il vincolo *viene espresso* nello schema E/R, *reificando* l'associazione tramite una entità Fornitura identificata da Prodotto e Parte



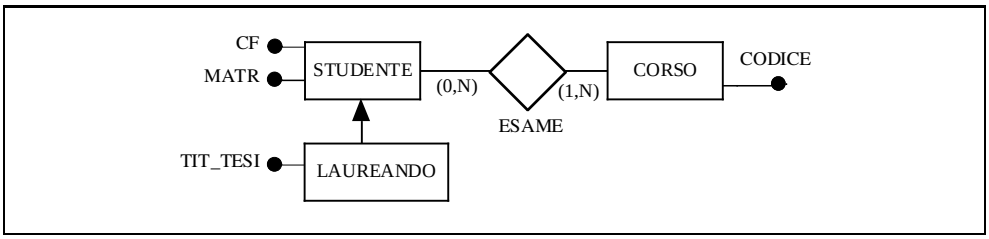

```
Fornitura(CodPart,CodProd,CodFornitore,Quantità)
```

2 il vincolo *non viene espresso* nello schema E/R che resta invariato (con associazione ternaria), ma viene aggiunto nello schema relazionale, cioè si usa ancora l'associazione ternaria Fornitura ma come chiave della relazione Fornitura si prende solo CodPart, CodProd:

```
Fornitura(CodPart,CodProd,CodFornitore,Quantità)
```

Uso di chiavi alternative nelle FK

♢ Nelle traduzioni delle associazioni abbiamo utilizzato sempre la chiave primaria; si può comunque utilizzare una qualsiasi *chiave alternativa*
Ad esempio, dato il seguente schema:

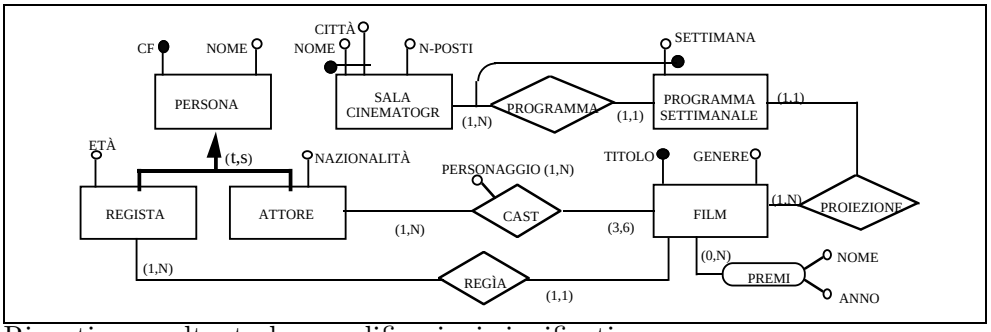


Il subset viene semplificato considerando il collasso verso l'alto e quindi riportando le proprietà di `LAUREANDO` in `STUDENTE`: l'identificatore di `LAUREANDO` sarà in `STUDENTE` un attributo `UNIQUE` che ammette dei valori nulli (non tutti gli studenti hanno un `TIT_TESI`).

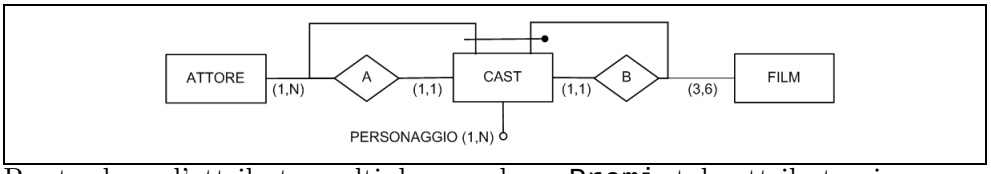
- Nella traduzione di `ESAME` si può usare sia la sua chiave primaria sia una sua chiave alternativa:

```
Studente(CF,Matr,Tit_Tesi)
   AK: Matr
   Tit_Tesi UNIQUE
Corso(Codice)
Esame(Matr,Codice)
   FK: Matr REFERENCES Studente(Matr)
   FK: Codice REFERENCES Corso
```

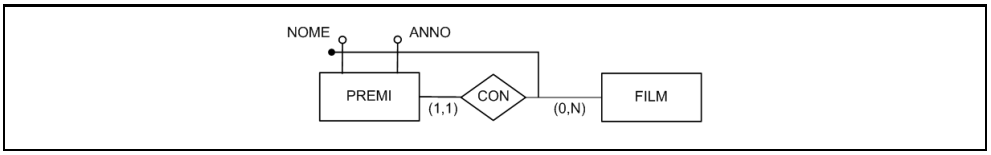
Esempio completo di traduzione da schema concettuale a schema logico relazionale



Riportiamo soltanto le semplificazioni significative.
Per tradurre l'attributo multiplo sull'associazione `Cast`, si reifica tale associazione per poi procedere con la normale traduzione di un attributo multiplo su un'entità:



Per tradurre l'attributo multiplo complesso `Premi`, tale attributo viene rappresentato tramite un'entità:

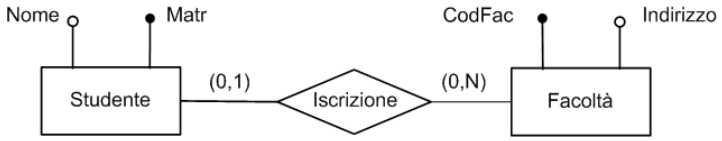


Lo schema relazionale corrispondente è il seguente:

```
Persona(CF,Nome)
Regista(CF,Età)
   FK: CF REFERENCES Persona
Attore(CF,Nazionalità)
   FK: CF REFERENCES Persona
Sala(Nome,Città,N-Posti)
Film(Titolo,Genere,Regista)
   FK: Regista REFERENCES Regista NOT NULL
Programma(Nome,Città,Settimana,Titolo)
   FK: Titolo REFERENCES Film NOT NULL
   FK: Nome,Città REFERENCES Sala
Premi(Titolo,Nome,Anno)
   FK: Titolo REFERENCES Film
Cast(Attore,Titolo)
   FK: Attore REFERENCES Attore
   FK: Titolo REFERENCES Film
Personaggio(Attore,Titolo,Personaggio)
   FK: Attore,Titolo REFERENCES Cast
```

Traduzione standard: esempio e precisazioni

Dato il seguente schema E/R



La traduzione standard è la seguente:

```
Studente(Matr,Nome)
Facoltà(CodFac,Indirizzo)
Iscrizione(Matr,CodFac)
   FK: Matr REFERENCES Studente
   FK: CodFac REFERENCES Facoltà NOT NULL
```

Per la discussione è utile considerare un'istanza dello schema relazionale:

`Studente`

Matr	Nome
123	Ugo
124	Ada
234	Leo

`Iscrizione`

Matr	CodFac
123	Geologia
124	Fisica

`Facoltà`

CodFac	Indirizzo
Geologia	via Alpi
Fisica	via E = MC2
Chimica	via H2O

Riportiamo e commentiamo la regola per la traduzione standard di associazioni

- Ogni associazione R tra E1, E2, ..., En, è tradotta con una relazione R con gli stessi attributi, cui si aggiungono le chiavi primarie di tutte le entità che essa collega

  – ogni chiave primaria di Ei tale che max-card(Ei, R)=1, è una **chiave candidata** di R; *altrimenti* la chiave della relazione è l'insieme di tutte le chiavi primarie delle entità collegate

  – le chiavi primarie delle entità collegate sono **foreign key non nulle** riferite alle corrispondenti entità. Infatti per realizzare un'istanza dell'associazione R tra E1, E2, ..., En devo avere necessariamente un elemento per ciascuna delle Ei: ne deriva che le foreign key che si mettono in R devono essere *non nulle*!
  Questo è sempre vero, indipendentemente dalla min-card(Ei, R).

Nell'esempio, per avere un'istanza di `Iscrizione` devo avere necessariamente un'istanza di `Studente` ed una di `Facoltà`. Il fatto che `min-card(Studente, Iscrizione) = 0` significa che ci possono essere studenti non iscritti: la matricola degli studenti non iscritti non comparirà in `Iscrizione`. Ad esempio,

lo studente con matricola `234` non è iscritto, cioè non è associato ad una facoltà, quindi non riporterò tale matricola in `Iscrizione`!
Si potrebbe pensare di mettere `234` ed associarlo ad un valore `null`, cioè mettere in `Iscrizione` la tupla `(234,null)`, ma normalmente (si consideri anche l'osservazione riportata di seguito) non si procede in questo modo in quanto in contrasto con il significato di `Iscrizione` come associazione binaria: per *fare una iscrizione* devo avere necessariamente *due* elementi, uno studente ed una facoltà.
Per questo si impone il vincolo `NON NULL` su tutte le foreign key.

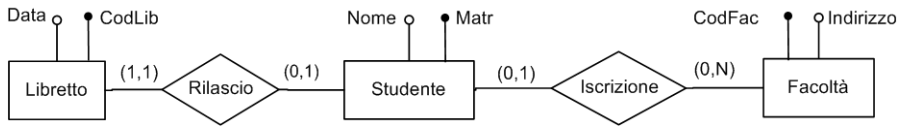
Osservazione: in un database relazionale si riportano solo *asserzioni positive*: la tupla `(123,Geologia)` in `Iscrizione` asserisce che `123` è iscritto a `Geologia`. **Non** si riportano *asserzioni negative*: non si riporta nessuna tupla per asserire che `234` non è iscritto.

Questo *modo di costruire* un database influenzerà il modo con il quale esso sarà interrogato (come vedremo nel contesto dei linguaggi di interrogazione):

- per sapere gli *studenti iscritti* basta vedere il contenuto della relazione `Iscrizione`
- per conoscere gli *studenti non iscritti* devo considerare tutti gli studenti (e quindi il contenuto della relazione `Studente`) e *sottrarre* quelli iscritti (e quindi il contenuto della relazione `Iscrizione`)

Traduzione non standard: esempio

♢ Nella traduzione di uno schema E/R generale con più associazioni, si applicano le regole di traduzione date singolarmente per ogni associazione e si possono compattare/inglobare nella stessa relazione più associazioni e più entità. Ad esempio, dato lo schema E/R:



Tradurre tale schema in modo da minimizzare il numero di relazioni, cioè traducendo l'associazione uno-a-molti `Iscrizione` con due relazioni e l'associazione uno-a-uno `Rilascio` con una relazione.
Consideriamo prima la traduzione dell'associazione `Iscrizione`:

```
Facoltà(CodFac,Indirizzo)
Studente(Matr,Nome,CodFac)
   FK: CodFac REFERENCES Facoltà
```

Nella relazione `Studente` non si deve imporre che `CodFac` sia NOT NULL in quanto `min-card(Studente, Iscrizione) = 0`: uno studente non iscritto avrà `CodFac` con valore `null`! Vediamo questo aspetto su un'istanza di tale schema relazionale:

`Studente`

Matr	Nome	CodFac
123	Ugo	Geologia
124	Ada	Fisica
234	Leo	null

`Facoltà`

CodFac	Indirizzo
Geologia	via Alpi
Fisica	via E = MC2
Chimica	via H2O

Consideriamo ora la traduzione dell'associazione `Rilascio` con una sola relazione. La regola afferma che la chiave primaria di tale relazione deve essere quella della entità che partecipa in modo opzionale, ovvero deve essere la chiave di `Studente`. Siccome nello schema abbiamo già una relazione `Studente`, aggiungo ad essa anche tale traduzione, cioè riporto in `Studente` anche tutti gli attributi derivanti dalla traduzione dell'associazione `Rilascio` e quindi dell'entità `Libretto`:

```
Facoltà(CodFac,Indirizzo)
Studente(Matr,Nome,CodFac,CodLib,Data)
   CodLib UNIQUE
   FK: CodFac REFERENCES Facoltà
```

Consideriamo un'istanza della relazione `Studente`:

Studente

Matr	Nome	CodFac	CodLib	Data
123	Ugo	Geologia	542376	15/10/00
124	Ada	Fisica	null	null
234	Leo	null	542376	15/10/00

Osservazione : Nello schema E/R iniziale `Iscrizione` e `Rilascio` sono due *associazioni indipendenti*, quindi posso avere uno studente iscritto (che partecipa ad `Iscrizione`) senza libretto (che non partecipa a `Rilascio`) e viceversa. Questo comportamento si riscontra anche nello schema relazionale corrispondente: posso avere uno studente iscritto (`124`) senza libretto e viceversa (`234`).

Come si può evitare questo *problema*? A livello di schema E/R si può usare una ternaria per associare `Studente`, `Libretto` e `Facoltà` (continuiamo a chiamare tale associazione `ISCRIZIONE`): in questo modo per *fare una iscrizione* devo avere necessariamente *tre* elementi, uno studente, una facoltà ed un libretto!

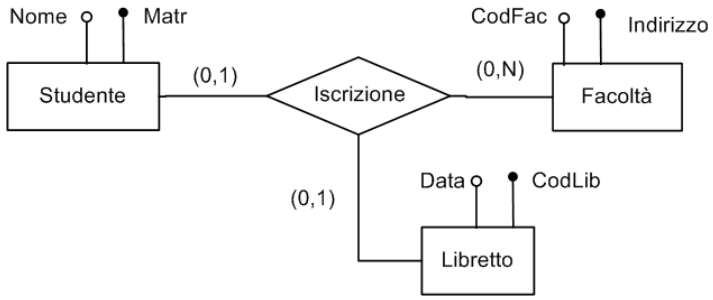
Nel seguito viene discusso la traduzione in relazionale di tale associazione ternaria.

Traduzione non standard di un'associazione n-aria

♢ Tale regola di traduzione si ottiene estendendo la regola "Traduzione con due relazioni" data per le associazioni binarie *uno-a-uno* e *uno-a-molti* :

> Un'associazione n-aria R tra $E1, E2, \ldots, En$ può essere compattata in una delle entità Ei che partecipa con molteplicità unitaria (cioè *max-card*$(Ei, R) = 1$) includendo in Ei gli attributi di R e le chiavi primarie di tutte le altre entità come foreign key; tali foreign key saranno vincolate ad essere `NOT NULL` se e solo se *min-card*$(Ei, R) = 1$.
> Ogni chiave primaria di $Ej \neq Ei$ tale che Ej partecipa con molteplicità unitaria (cioè *max-card*$(Ej, R) = 1$) sarà chiave alternativa.

♢ Per illustrare l'applicazione di questa regola, consideriamo un esempio di associazione ternaria:



- Consideriamo prima la sua traduzione standard:

```
Libretto(CodLib,Data)
Studente(Matr,Nome)
Facoltà(CodFac,Indirizzo)
Iscrizione(Matr,CodFac,CodLib)
   AK: CodLib
   FK: Matr REFERENCES Studente
   FK: CodFac REFERENCES Facoltà NOT NULL
   FK: CodLib REFERENCES Libretto NOT NULL
```

- È possibile verificare che nello schema relazionale non è più presente il problema discusso in precedenza: per inserire l'iscrizione di uno studente, va inserita una tupla nella relazione `Iscrizione` dove tutti gli attributi (`Matr`, `CodFac` e `CodLib`) devono essere specificati.

♢ Una possibile traduzione non-standard è la seguente, nella quale l'associazione è inclusa nella relazione `Studente`:

```
Libretto(CodLib,Data)
Facoltà(CodFac,Indirizzo)
Studente(Matr,Nome,CodFac,CodLib)
   CodLib UNIQUE
   FK: CodFac REFERENCES Facoltà
   FK: CodLib REFERENCES Libretto
```

Osservazione : È possibile verificare che questo schema relazionale presenta lo stesso problema già discusso, ovvero posso avere uno studente iscritto (`CodFac` ha un valore non nullo) senza libretto (`CodLib` ha un valore nullo) e viceversa.

Il problema non si può risolvere mettendo `NOT NULL` su `CodFac` e `CodLib` altrimenti non potrei inserire nel DB relazionale gli studenti che non sono iscritti e/o che non hanno libretto.

Il problema si può risolvere mettendo un vincolo che stabilice che `CodFac` e `CodLib` possono essere entrambi nulli oppure entrambi non nulli. Tale vincolo si può esprimere in SQL (nell'istruzione `create table`):

```
CHECK ( (CodFac IS NULL AND CodLib IS NULL) OR
(CodFac IS NOT NULL AND CodLib IS NOT NULL) )
```

Osservazione : Se `min-card(Libretto, Iscrizione)=1` e `min-card(Studente, Iscrizione)=1` nello schema relazionale deve essere riportato **CodFac NOT NULL** e **codLib NOT NULL** e la traduzione risulta essere corretta.

♢ Una soluzione più articolata pu essere definita utilizzando il concetto di trigger SQL.

Esempio di implementazione di regole aziendali

Nella sezione 1.7 si è definito un semplice schema E/R e un insieme di regole aziendali per tale schema. Lo schema logico corrispondente a tale rappresentazione E/R è il seguente:

```
UNIVERSITA'(CODUNI)
FACOLTA'(CODFAC,NUM_ISCR,CODUNI)
   FK: CODUNI REFERENCES UNIVERSITA'
STUDENTE(MATR,CODFAC,PAGINA)
   FK: CODFAC REFERENCES FACOLTA'
RAPPRES(MATR,CODFAC,DATA)
   FK: MATR,CODFAC REFERENCES STUDENTE
   FK: CODFAC1 REFERENCES FACOLTA' NOT NULL
```

- La regola aziendale **RV3** è correttamente implementata nella relazione `STUDENTE`

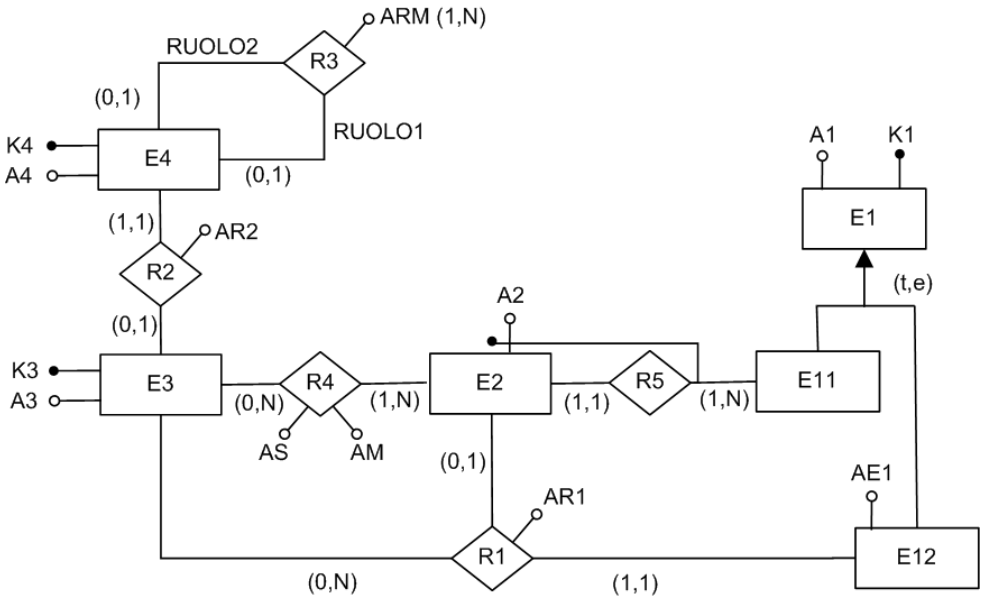
- La regola aziendale **RV1** è implementabile in relazionale imponendo che gli attributi `CODFAC` e `CODFAC1` rappresentino lo stesso valore. Conseguentemente la definizione della relazione `RAPPRES` diventa

  ```
  RAPPRES(MATR,CODFAC)
     FK: CODFAC REFERENCES FACOLTA'
  ```

- Le regole aziendali **RV2**, **RV4**, **RV5**, **RD1** non sono esprimibili in relazionale e possono essere implementate attraverso opportuni TRIGGER.

3.2.3 Esempio di progettazione logica

Scopo di questa sezione è quello di presentare e commentare le regole di progettazione logica relazionale introdotte nelle sezioni precedenti. A tale scopo si considera il seguente schema E/R:



La traduzione in relazionale si può effettuare applicando per le associazioni le regole di traduzione date:

1. traduzione standard, sempre applicabile
2. altre traduzioni non standard, applicabili quando in un'associazione R vi sono entità Ei che partecipano con molteplicità unitaria:
   $max\text{-}card(Ei, R) = 1$.

Dato uno schema E/R, nella sua traduzione si possono tradurre alcune associazioni con la traduzione standard ed altre con traduzioni non standard.
Nota : Nella traduzione standard devo introdurre una relazione per ciascuna associazione, *ad eccezione* delle associazioni relative a componenti di identificatori esterni: infatti, come discusso a pagina 104 tale associazione è automaticamente semplificata e tradotta grazie alla introduzione della foreign key.
A tale proposito si noti inoltre che normalmente nelle associazioni relative a componenti di identificatori esterni non vengono riportati attributi: infatti un attributo su tale associazione può essere rappresentato (e viene normalmente rappresentato) in forma equivalente come attributo sull'entità identificata esternamente.

♢ **Traduzione standard :**

Lo schema relazionale risultante e' il seguente

E1(K1-E1,A1)

E11(K1-E11)
FK: K1-E11 **REFERENCES** E1

E12(K1-E12,AE1)
FK: K1-E12 **REFERENCES** E1

E2(K1-E11,A2)
FK: K1-E11 **REFERENCES** E11

E3(K3,A3)

E4(K4,A4)

R1(K1-E11,A2,K3,K1-E12,AR1)
AK: K1-E12
FK: K1-E11,A2 **REFERENCES** E2
FK: K3 **REFERENCES** E3 **NOT NULL**
FK: K1-E12 **REFERENCES** E12

R2(K4,K3,AR2)
AK: K3
FK: K3 **REFERENCES** E3
FK: K4 **REFERENCES** E4

R3(K4-RUOLO1,K4-RUOLO2)
AK: K4-RUOLO2
FK: K4-RUOLO1 **REFERENCES** E4
FK: K4-RUOLO2 **REFERENCES** E4

R4(K1-E11,A2,K3,AS,AM)
FK: K1-E11,A2 **REFERENCES** E2
FK: K3 **REFERENCES** E3

ARM(K4-RUOLO1,ARM)
FK: K4-RUOLO1 **REFERENCES** R3

♢ **Altre traduzioni non standard**

1) Inglobare le associazioni R3 ed R2 in E4: in questo modo non si devono più introdurre le relazioni R3 ed R2.
E3(K3,A3) (invariata rispetto alla traduzione standard)
E4(K4,A4, K3, AR2, K4-RUOLO2)
AK: K3
FK: K3 **REFERENCES** E3
FK: K4-RUOLO2 **REFERENCES** E4
Si noti che nella traduzione di R3 si poteva usare anche, indifferentemente, K4-RUOLO1:
L'unico punto particolare è la traduzione dell'attributo multiplo ARM: come già fatto per la traduzione standard, prima di tradurre l'attributo multiplo devo reificare R3 e quindi introdurre una relazione ARM in cui metto come chiave l'insieme ARM + *ChiaveDiR3*. Però adesso non c'è una relazione R3 perchè è in E4: allora in ARM come *ChiaveDiR3* si mette la chiave di E4 in quanto R3 è adesso inglobata in E4. Avremo quindi:
E4(K4,A4, K3, AR2, K4-RUOLO1)
AK: K3
FK: K3 **REFERENCES** E3
FK: K4-RUOLO1 **REFERENCES** E4
ARM(ARM,K4)
FK: K4 **REFERENCES** E4

Per meglio evidenziare questa soluzione facciamo riferimento ad un caso reale, invece che simbolico, considerando le seguenti corrispondenze:
E3 = STUDENTE, K3 = MATR A3 = NOME
R2 = RILASCIO, AR2 = DATA
E4 = LIBRETTO, K4 = CODLIB, A4 = TIPO
R3 = SOSTITUZIONE, RUOLO1 = SOSTITUTO, RUOLO2 = SOSTITUITO,
ARM = MOTIVI

Si ottiene
STUDENTE(MATR,NOME)
LIBRETTO(CODLIB,TIPO, MATR, DATA, CODLIB-SOSTITUITO)
AK: MATR
AK: CODLIB-SOSTITUITO
FK: MATR **REFERENCES** STUDENTE **NOT NULL**
FK: CODLIB-SOSTITUITO **REFERENCES** LIBRETTO
MOTIVI(MOTIVI, CODLIB)
FK: CODLIB **REFERENCES** LIBRETTO

2) Inglobare la relazione ternaria `R1` in `E12`.
Per le associazioni ternarie (ed n-arie in generale) è possibile applicare una traduzione non standard così come discusso a pagina 119. Applichiamola al caso di `R1`, inglobandola in `E12`.
`E2(K1-E11,A2)` (invariata rispetto alla traduzione standard)
`FK: K1-E11 REFERENCES E11`
`E3(K3,A3)` (invariata rispetto alla traduzione standard)
`E12(K1-E12, AE1, K3, K1-E11,A2, AR1)`
`AK: K1-E11,A2`
`FK: K1-E12 REFERENCES E1`
`FK: K3 REFERENCES E3 NOT NULL`
`FK: K1-E11,A2 REFERENCES E2`
Si noti che essendo *min-card*$(E12, R1) = 1$ per le foreign key relative alle altre entità (`E3` e `E2`) devo mettere `NOT NULL`.

3) Inglobare l'associazione ternaria `R1` in `E2`. Rispetto a prima, adesso abbiamo *min-card*$(E2, R1) = 0$ quindi per le foreign key relative alle altre entità (`E3` e `E12`) non si deve mettere `NOT NULL`
`E12(K1-E12,AE1)` (invariata rispetto alla traduzione standard)
`FK: K1-E12 REFERENCES E1`
`E3(K3,A3)` (invariata rispetto alla traduzione standard)
`E2(K1-E11,A2, K3, K1-E12, AR1)`
`FK: K1-E11 REFERENCES E11`
`FK: K3 REFERENCES E3`
`FK: K1-E12 REFERENCES E12`

Capitolo 4

Elementi ed esempi del linguaggio SQL

Il linguaggio SQL (*Structured Query Language*) è il linguaggio standard per la definizione, manipolazione e interrogazione delle basi di dati relazionali [18, 39, 11]. In questo capitolo ne verranno presentati gli elementi fondamentali.

Nella prima parte illustreremo l'uso di SQL come *DDL* (*Data Definition Language*), cioè come linguaggio per la definizione di schemi relazionali, la dichiarazione di vincoli di integrità e l'introduzione di indici.

Nella parte centrale descriveremo l'uso di SQL come *DML* (*Data Manipulation Language*), cioè come linguaggio per l'interrogazione e l'aggiornamento (inserimento, modifica e cancellazione) delle istanze della base di dati.

Quindi mostreremo, brevemente, le caratteristiche di SQL come *DCL* (*Data Control Language*), cioè come linguaggio per la gestione degli utenti e l'assegnazione dei privilegi di accesso.

Nell'ultima parte mostreremo l'uso di SQL all'interno di linguaggi di programmazione (*embedded SQL*), introdurremo brevemente il concetto di *Stored Procedure* (procedure eseguite dal RDBMS su esplicita richiesta delle applicazioni o degli utenti) e quindi tratteremo la definizione di *Trigger* (procedure attivate automaticamente dal RDBMS al verificarsi di determinate condizioni).

Il linguaggio SQL

♢ Il linguaggio SQL (*Structured Query Language*) è il linguaggio standard per la definizione, manipolazione e interrogazione delle basi di dati relazionali.

♢ Il linguaggio SQL è stato originariamente sviluppato nei laboratori di ricerca IBM per il sistema relazionale System R.

♢ Il linguaggio SQL è stato successivamente sottoposto a varie fasi di standardizzazione che hanno portato allo standard attuale, noto come SQL92 (o SQL2) concluso nel 1992.

In seguito faremo riferimento allo standard SQL92.

Caratteristiche principali del linguaggio :

♢ è dichiarativo

♢ opera su *multiset* di tuple, cioè su insiemi di tuple che possono contenere duplicati

♢ permette di esprimere tutte le interrogazioni formulabili in Algebra Relazionale

♢ Notazione utilizzata per la sintassi degli elementi del linguaggio:

- elementi **TERMINALI**
- elementi `<non-terminali>`
- `[termine-opzionale]`
- $\texttt{termine}_1$ `|` $\texttt{termine}_2$ `| ...|` $\texttt{termine}_n$
  elenco di termini in alternativa di cui deve esserne scelto uno;
  per delimitare elenchi consecutivi si useranno le parentesi `<...>`

4.1 Definizione dei dati

♢ L'SQL mette a disposizione i seguenti principali **tipi di dati**

Caratteri :

CHARACTER [VARYING] [(<lunghezza>)]
[CHARACTER SET <set-di-caratteri>]

- Lunghezza fissa o variabile, con indicazione della lunghezza massima
- Possibilità di scegliere tra insiemi di caratteri
- Abbreviazioni: lungh. fissa: **CHAR(X)**, lungh. variabile: **VARCHAR(X)**

Bit :

BIT [VARYING] [(<lunghezza>)]

Numerico Esatto :

- Interi : **INTEGER** (**INT**) e **SMALLINT**
- Decimali : **DECIMAL [(<precisione> [,<scala>])]**

Numerico Approssimato :

- **FLOAT [(<precisione>)]**
- **REAL**
- **DOUBLE PRECISION**

Datetime :

- **DATE** : Anno-Mese-Giorno, 'AAAA-MM-GG'
- **TIME** : Ore:Minuti:Secondi, 'OO-MM-SS'
- **TIMESTAMP** : Anno-Mese-Giorno Ore:Minuti:Secondi

Intervallo :

INTERVAL <campo-iniziale> TO <campo-finale>

- utilizzati per rappresentare periodi continui di tempo
- Esempio: `INTERVAL YEAR(2) TO MONTH`
  Intervallo con valori compresi tra 0 anni - 0 mesi e 99 anni - 11 mesi

Valori nulli

♢ Per considerare i valori nulli l'SQL utilizza una logica a tre valori basata sulle seguenti tabelle di verità

AND	true	null	false
true	true	null	false
null	null	null	false
false	false	false	false

OR	true	null	false
true	true	true	true
null	true	null	null
false	true	null	false

NOT	
true	false
null	null
false	true

♢ Un qualsiasi confronto con il valore `null` restituisce ancora `null` (e quindi non risulta essere `true`)

`(X <op> null) = (null <op> X) = null`

dove `<op>` è un operatore algebrico o relazionale e `X` è una costante o variabile

- due valori `null` sono considerati diversi
- due valori `null` sono considerati uguali ai fini dell'indicizzazione e ai fini del raggruppamento
- Operatore di test per i valori nulli : `<attr> IS [NOT] NULL`

Valori nulli (2)

♢ Come è stato detto a pagina 75, introducendo il concetto di valore nullo nel modello relazionale, è possibile dare diverse interpretazioni per il valore `null`.

Siccome l'SQL non distingue tra queste diverse interpretazioni di `null`, occorre diversificare la condizione di ricerca a secondo del test da effettuare, come evidenziato nel seguente esempio:

Consideriamo il seguente schema:

`Studente(Matricola,Media)`

e la sua istanza:

Matricola	Media
M1	28
M2	null
M3	23
M4	26

♢ Supponiamo di voler selezionare gli studenti sulla base della condizione "media superiore a 24". Siccome l'attributo `Media` può assumere il valore `null` occorre fare attenzione sul significato di tale condizione quando si interroga la base di dati. Nella seguente tabella sono riportati alcuni possibili test da effettuare con la condizione "media superiore a 24", con la relativa condizione di ricerca in SQL ed il risultato ottenuto sulla precedente istanza:

Test effettuato	**Condizione di ricerca**	**Risultato**
è noto che (`Media > 24`)	(`Media > 24`)	`M1, M4`
è possibile che (`Media > 24`)	(`Media > 24`) `OR` (`Media IS NULL`)	`M1, M2, M4`
è noto che non è (`Media > 24`)	`NOT` (`Media > 24`)	`M3`
non è noto che (`Media > 24`)	`NOT` (`Media > 24`) `OR` (`Media IS NULL`)	`M2, M3`
`Media` è non nota	(`Media IS NULL`)	`M2`
`Media` è nota	(`Media IS NOT NULL`)	`M1, M3, M4`

Schema

♢ Uno schema è una collezione di oggetti del database (tabelle, domini, indici, viste, privilegi,. . .)

Tutti gli oggetti dello schema hanno lo stesso proprietario

CREATE SCHEMA [<nome-schema>]
[**AUTHORIZATION** <nome-proprietario>]
(<oggetti-dello-schema>)

- Se `AUTHORIZATION <nome-proprietario>` viene omesso, si assume come proprietario l'utente che ha eseguito il comando
- Se `<nome-schema>` viene omesso, si assume come nome dello schema il nome del proprietario
- La dichiarazione degli oggetti dello schema può avvenire al di fuori del comando `CREATE SCHEMA`.

Tabella :

Una tabella è costituita da una collezione ordinata (lista) di uno o più attributi (colonne) e da un insieme di zero o più vincoli (*constraint*)

CREATE TABLE <nome-tabella>
(<nome-colonna> <dominio> [<vincoli-di-colonna>],
...
<nome-colonna> <dominio> [<vincoli-di-colonna>],
[<vincoli-di-tabella>]
)

```
CREATE TABLE ESAME
   ( CC CHAR(5) REFERENCES CORSO,
   MATR CHAR(9) REFERENCES STUDENTE(MATR),
   VOTO INTEGER CHECK ((VOTO>=18) AND (VOTO<=30)) OR (VOTO=33),
   PRIMARY KEY(CC,MATR),
   ...);
```

Schema (2)

Domini : Un dominio è un insieme di valori consentiti

Un dominio può essere un tipo di dato base dell'SQL oppure può essere definito dall'utente tramite la seguente sintassi:

CREATE DOMAIN `<nome-dominio>` **AS** `<tipo-di-dati>`
`[<valore-default> ] [<vincoli-di-dominio>]`

```
CREATE DOMAIN VOTOESAME AS INTEGER
CHECK ((VOTO>=18) AND (VOTO<=30)) OR (VOTO=33));

CREATE TABLE ESAME
   ...
   VOTO VOTOESAME
```

Indici : I comandi di definizione degli indici non fanno parte dello standard del linguaggio.

Comunque, una sintassi accettata da vari sistemi è la seguente

CREATE [UNIQUE] INDEX `<nome-indice>`
ON `<tabella> (<lista-colonne>)`

- **UNIQUE** : non sono ammessi valori duplicati degli attributi specificati in `<lista-colonne>`
  equivale quindi a dichiarare `<lista-colonne>` come chiave di `<tabella>`.

♢ Altri oggetti di uno schema, quali viste e privilegi, verranno introdotti in sezione 4.4.

Rimozione di uno schema e dei suoi oggetti :

È possibile rimuovere interamente uno schema con tutti i suoi oggetti

DROP SCHEMA `<nome-schema>`

oppure singolarmente i vari oggetti, come ad esempio:

DROP TABLE `<nome-tabella>`

Vincoli di tabella

♢ Permettono di controllare i valori che possono essere registrati in una tabella.

Primary Key :

PRIMARY KEY (<lista-colonne>)

- esprime la chiave primaria di una tabella
- può essere definito solo una volta in una tabella

Foreign Key :

FOREIGN KEY (<lista-colonne>)
REFERENCES <tabella> [(<lista-colonne>)]

- <column list> deve essere definita in <tabella> come chiave
- in assenza di (<column list>) è riferita alla primary key di <tabella>

Check :

CHECK (<condizione>)

- esprime un generico vincolo sulla tabella tramite una espressione che deve essere vera per tutte le tuple della tabella

```
CREATE TABLE CORSO
(  CC CHAR(5)
   ...,
   PRIMARY KEY (CC));
CREATE TABLE STUDENTE
(  MATR CHAR(9)
   CF CHAR(9)
   ...,
   PRIMARY KEY(MATR),
   CHECK (CF UNIQUE) );
CREATE TABLE ESAME
(  CC CHAR(5),
   MATR CHAR(9),
   VOTO INTEGER,
   PRIMARY KEY(CC,MATR),
   FOREIGN KEY(CC) REFERENCES CORSO(CC),
   FOREIGN KEY(MATR) REFERENCES STUDENTE,
   CHECK (((VOTO>=18) AND (VOTO<=30)) OR (VOTO=33)));
```

♢ Ad un vincolo può essere dato un nome con **CONSTRAINT** <nome>:

```
CONSTRAINT FK-CC-TO-CORSO FOREIGN KEY(CC) REFERENCES CORSO(CC),
```

Vincoli di colonna

♢ Permettono di controllare i valori che possono essere registrati in una singola colonna.

Tutti i vincoli di colonna possono essere espressi come vincoli di tabella.

Not Null :

stabilisce che l'attributo non può assumere il valore `null`;

Unique :

esprime l'unicità dell'attributo;

- tramite `Not Null Unique` si dichiara che l'attributo è una chiave alternativa della tabella; (vedere discussione nella prossima pagina)

Primary Key :

stabilisce che l'attributo è la chiave primaria della tabella;

References :

esprime il vincolo della Foreign Key: la colonna nella tabella riferita può essere la primary key (ed in questo caso può essere omessa) oppure una colonna esplicitamente indicata;

Check :

esprime un generico vincolo sulla colonna tramite una espressione logico-relazionale;

```
CREATE TABLE CORSO
   ( CC CHAR(5) PRIMARY KEY
   ...);
CREATE TABLE STUDENTE
   ( MATR CHAR(9) PRIMARY KEY
   CF CHAR(9) NOT NULL UNIQUE
   ...);
CREATE TABLE ESAME
   ( CC CHAR(5) REFERENCES CORSO,
   MATR CHAR(9) REFERENCES STUDENTE(MATR),
   VOTO INTEGER CHECK ((VOTO>=18) AND (VOTO<=30)) OR (VOTO=33)
   PRIMARY KEY(CC,MATR),
   ...);
```

Dichiarazioni di Alternative Key in SQL

♢ In SQL92, mentre per dichiarare la chiave primaria di una relazione esiste un esplicito vincolo `PRIMARY KEY`, non esiste un analogo esplicito vincolo per dichiarare una chiave alternativa: una chiave alternativa si ottiene imponendo sia il vincolo `UNIQUE` che il vincolo `NOT NULL`.

♢ In SQL92, occorre imporre sia il vincolo `UNIQUE` che il vincolo `NOT NULL`, in quanto il vincolo `UNIQUE` non implica automaticamente il vincolo `NOT NULL`.

- È interessante osservare il differente comportamento del vincolo `UNIQUE` nelle varie implementazioni del linguaggio SQL nei DBMS commerciali.

♢ Consideriamo ad esempio:

```
CREATE TABLE STUDENTE
  (  MATR CHAR(9)  PRIMARY KEY,
     CF   CHAR(9)  UNIQUE
  )
```

- In Oracle, Access, MySQL posso avere varie tuple tutte con un valore `NULL` nell'attributo `CF`.

- In DB2 la precedente istruzione di `CREATE TABLE` non è consentita in quanto il vincolo `UNIQUE` può essere dichiarato solo per attributi `NOT NULL`.

- In SQL-SERVER posso avere al massimo una tupla con valore `NULL` nell'attributo `CF`.

♢ D'altra parte, le varie implementazioni del linguaggio SQL nei DBMS commerciali, si comportano nello stesso modo quando viene utilizzato `NOT NULL UNIQUE`:

```
CREATE TABLE STUDENTE
  (  MATR CHAR(9)  PRIMARY KEY,
     CF   CHAR(9)  NOT NULL UNIQUE
  )
```

Violazione di vincoli di integrità

♢ Generalmente, quando il sistema rileva una violazione di un vincolo di integrità, il comando di aggiornamento viene rifiutato e viene segnalato l'errore all'utente.

♢ Per i vincoli di integrità referenziale SQL92 permette di scegliere quale reazione adottare quando viene rilevata una violazione.

FOREIGN KEY (<lista-colonne>)
REFERENCES <tabella> [(<lista-colonne>)]
ON <**DELETE**|**UPDATE**>
<**CASCADE**|**SET NULL**|**SET DEFAULT**|**NO ACTION**>

♢ Una operazione sulla tabella che effettua il riferimento, detta *dipendente*, che viola il vincolo (inserimento o modifica degli attributi referenti) viene rifiutata.

♢ Ad una operazione sulla tabella *riferita* che viola il vincolo (modifica di attributi riferiti o cancellazione) si può rispondere con:

CASCADE

In caso di modifica, il nuovo valore dell'attributo della tabella riferita viene riportato su tutte le corrispondenti righe della tabella dipendente. In caso di cancellazione, tutte le righe della tabella dipendente corrispondenti alla riga cancellata vengono cancellate.

SET NULL

all'attributo referente viene assegnato il valore null al posto del valore modificato/cancellato nella tabella riferita

SET DEFAULT

all'attributo referente viene assegnato il valore di default al posto del valore modificato/cancellato nella tabella riferita

NO ACTION

non viene eseguita alcuna reazione

Modifica della struttura di una tabella

♢ Il comando **ALTER TABLE** `<nome-tabella>` `<tipo-modifica>` permette di modificare la struttura di una tabella. In `<tipo-modifica>` si possono specificare le seguenti azioni

aggiunta di una colonna :

ADD [**COLUMN**] `<nome-colonna>` `<dominio>` [`<vincoli-di-colonna>`]

rimozione di una colonna :

DROP [**COLUMN**] `<nome-colonna>` **RESTRICT|CASCADE**

Con `RESTRICT` la colonna viene eliminata solo se non ci sono dipendenze da tale colonna in altre definizioni, mentre con `CASCADE` si forza l'eliminazione della colonna e di tutte le dipendenze nelle altre definizioni.

Ad esempio, il comando `ALTER TABLE CORSO DROP CC` fallisce con `RESTRICT` in quanto nella tabella `ESAME` c'è un vincolo di foreign key che usa `CC`, mentre con `CASCADE` viene cancellata sia la colonna `CC` che il vincolo di foreign key in `ESAME`.

aggiunta di un vincolo di tabella :

ADD `<vincolo-di-tabella>`

rimozione di un vincolo di tabella :

DROP `<nome-vincolo>` **RESTRICT|CASCADE**

Con `RESTRICT` un vincolo di unicità non viene eliminato se è usato in un vincolo di foreign key, mentre con `CASCADE` si forza l'eliminazione anche del vincolo di foreign key.

imposta il valore di default per una colonna :

ALTER [**COLUMN**] `<nome-colonna>` **SET DEFAULT** `<valore>`

annulla il valore di default per una colonna :

ALTER [**COLUMN**] `<nome-colonna>` **DROP DEFAULT**

♢ Il comando **ALTER DOMAIN** `<nome-dominio>` `<tipo-modifica>` permette di modificare alcune proprietà di un dominio di valori. Le proprietà modificabili sono solo quelle che non influenzano i dati reali, infatti in `<tipo-modifica>` si possono specificare le seguenti azioni

imposta/annulla il valore di default per il dominio

aggiunta/rimozione di un vincolo di dominio

4.2 Interrogazioni

♢ L'istruzione base dell'SQL per costruire interrogazioni di complessità arbitraria è lo statement `SELECT`

♢ Sintassi di base:

SELECT [DISTINCT|ALL] <lista-select>
FROM <lista-from>
[WHERE <condizione>]
[ORDER BY <lista-order>]

ALL (Default): non c'è l'eliminazione dei duplicati (semantica del multiset)
DISTINCT: eliminazione dei duplicati

♢ `<lista-select>`:

- Uno o più attributi delle relazioni della `lista-from`
- Funzioni aggregate: `COUNT`,`AVG`,`MIN`,`MAX`,`SUM`
- Costanti
- Una generica espressione matematica che coinvolge uno o più degli oggetti precedenti
- Il simbolo $*$: tutti gli attributi

♢ `<lista-from>`:

- Una o più tabelle o viste
- Operazioni di join tra una o più tabelle o viste

♢ `<condizione>`: espressione booleana di:

- Predicati semplici
- Predicati di join
- Predicati con subquery

♢ `<lista-order>`:

- Uno o più attributi della `<lista-select>`
- Ordine ascendente (default) o discendente (**DESC**)

Esempio di riferimento per le interrogazioni

S(Matr,SNome,Città,ACorso)
C(CC,CNome,CD)
FK: CD REFERENCES D
D(CD,CNome,Città)
E(Matr,CC,Data,Voto)
FK: Matr REFERENCES S
FK: CC REFERENCES C

S

Matr	SNome	Città	ACorso
M1	Lucia Quaranta	SA	1
M2	Giacomo Tedesco	PA	2
M3	Carla Longo	MO	1
M4	Ugo Rossi	MO	1
M5	Valeria Neri	MO	2
M6	Giuseppe Verdi	BO	1
M7	Maria Rossi	null	1

C

CC	CNome	CD
C1	Fisica 1	D1
C2	Analisi Matematica 1	D2
C3	Fisica 2	D1
C4	Analisi Matematica 2	D3

D

CD	CNome	Città
D1	Paolo Rossi	MO
D2	Maria Pastore	BO
D3	Paola Caboni	FI

E

Matr	CC	Data	Voto
M1	C1	06-29-1995	24
M1	C2	08-09-1996	33
M1	C3	03-12-1996	30
M2	C1	06-29-1995	28
M2	C2	07-07-1996	24
M3	C2	07-07-1996	27
M3	C3	11-11-1996	25
M4	C3	11-11-1996	33
M6	C2	01-02-1996	28
M7	C1	06-29-1995	24
M7	C2	04-11-1996	26
M7	C3	06-23-1996	27

Predicati semplici

- **Operatori relazionali** : `<attr> <op-rel> <cost>`

  dove `<op-rel>` $\in \{=, <>, >, >=, <, <=\}$

  Es. "Studenti del secondo anno di corso"

  ```
  SELECT *
  FROM   S
  WHERE  ACorso=2
  ```

  Es. "Esami con voto compreso tra 24 e 28"

  ```
  SELECT *
  FROM   E
  WHERE  Voto >= 24
  AND    Voto <= 28
  ```

- **Operatore di range** : `<attr>` **BETWEEN** `<cost1>` **AND** `<cost2>`

  Es. "Esami con voto compreso tra 24 e 28"

  ```
  SELECT *
  FROM   E
  WHERE  Voto BETWEEN 24 AND 28
  ```

- **Operatore di set** : `<attr>` **IN** `(<cost1>, ..., <costN>)`

  Es. "Esami con voto pari a 29, 30 oppure con lode (voto pari a 33)"

  ```
  SELECT *
  FROM   E
  WHERE  Voto IN (29,30,33)
  ```

- **Operatore di confronto stringhe** : `<attr>` **LIKE** `<stringa>`

  dove `<stringa>` può contenere i caratteri speciali `_` (carattere arbitrario) e `%` (stringa arbitraria)

  Es. "Studenti il cui nome inizia con A e termina con O "

  ```
  SELECT *
  FROM   S
  WHERE  SNome LIKE 'A%O'
  ```

Predicati semplici (2)

- **Operatori quantificati** :

  `<attr> <op-rel>[ANY|ALL] (<cost1>,...,<costN>)`

  Es. "Esami con voto pari a 29, 30 oppure con lode (voto pari a 33)"

  ```
  SELECT *
  FROM   E
  WHERE  Voto =ANY (29,30,33)
  ```

  Es. "Esami con voto diverso da 29, 30 e 33"

  ```
  SELECT *
  FROM   E
  WHERE  Voto <> ALL (29,30,33)
  ```

  ♢ Confronto con l'insieme vuoto `()`:

  - `<attr> <op-rel>ANY ()` ha valore `false`
  - `<attr> <op-rel>ALL ()` ha valore `true`

- **Operatore di confronto con valori NULL** : `<attr> IS [NOT] NULL`

  Es. "Studenti con l'attributo città non specificato"

  ```
  SELECT *
  FROM   S
  WHERE  Città IS NULL
  ```

♢ **Ordinamento del risultato:**

Es. "Studenti di Modena ordinati in senso ascendente rispetto all'anno di corso"

```
SELECT   Matr,ACorso
FROM     S
WHERE    Città='MO'
ORDER BY ACorso
```

- L'ordinamento deve essere fatto rispetto a uno o più elementi della `<lista-select>`: un tale elemento può essere indicato anche riportando la sua posizione nella `<lista-select>`.

Es. "Esami del corso C1 ordinati in senso discendente rispetto al voto espresso in sessantesimi, e a parità di voto rispetto alla matricola"

```
SELECT   Matr,CC,(60*Voto)/30
FROM     E
WHERE    CC='C1'
ORDER BY 3 DESC, Matr
```

Prodotto Cartesiano e Join

♢ Il **prodotto cartesiano** di due o più relazioni si ottiene riportando le relazioni nella `<lista-from>` della clausola `FROM`, senza clausola `WHERE`.

♢ Il **join** viene espresso generalmente riportando nella clausola `FROM` le relazioni interessate e nella clausola `WHERE` le *condizioni di join*.

Es. "Combinazioni di studenti e di docenti residenti nella stessa città"

```
SELECT S.Matr,S.Città,D.CD
FROM   S,D
WHERE  S.Città=D.Città
```

S.Matr	S.Città	D.CD
M6	BO	D2
M3	MO	D1
M4	MO	D1
M5	MO	D1

♢ Con l'SQL92 è possibile esprimere le operazioni di join nella clausola `FROM`:

```
<tabella1> [INNER|LEFT|RIGHT|FULL] JOIN <tabella2>
ON <condizione>
```

Il default è il join interno è quindi la parola `INNER` può essere omessa.

Es. "Combinazioni di studenti e di docenti residenti nella stessa città"

```
SELECT S.Matr,S.Città,D.CD
FROM   S JOIN D ON (S.Città=D.Città)
```

♢ Join con predicati locali

Es. "Studenti residenti nella stessa città di residenza del docente D1"

```
SELECT S.*
FROM   S,D
WHERE  S.Città = D.Città
AND    D.CD='D1'
```

oppure

```
SELECT S.*
FROM   S JOIN D ON (S.Città = D.Città)
WHERE  D.CD='D1'
```

Matr	SNome	Città	ACorso
M3	Carla Longo	MO	1
M4	Ugo Rossi	MO	1
M5	Valeria Neri	MO	2

Esempi di interrogazioni di Join (1)

♢ Negli esempi che seguono si vogliono selezionare gli esami (relazione `E`) con in aggiunta altre proprietà (il nome dello studente, del corso, ...) memorizzate in altre relazioni. Quindi occorre fare il join tra la relazione `E` con le relazioni in cui tali proprietà sono presenti, join effettuato sugli attributi chiave esterna/chiave.

1. Selezionare, per ogni esame, Nome studente, CodiceCorso e voto

```
SELECT S.SNome, E.CC, E.Voto
FROM E, S
WHERE E.Matr=S.Matr
```

2. Selezionare, per ogni esame, Nome studente, Nome del Corso e voto

```
SELECT S.SNome, C.CNome, E.Voto
FROM E, S, C
WHERE E.Matr=S.Matr
AND E.CC=C.CC
```

3. Selezionare, per ogni esame, Nome studente, Nome del Corso, Nome del Docente del Corso e voto

```
SELECT S.SNome, C.CNome, D.CNome, E.Voto
FROM E, S, C, D
WHERE E.Matr=S.Matr
AND E.CC=C.CC
AND C.CD=D.CD
```

4. Selezionare, per ogni esame di un corso di 'Fisica' sostenuto da uno studente di 'MO', voto, Nome studente, Nome del corso e Nome del docente del corso

```
SELECT S.SNome, C.CNome, D.CNome, E.Voto
FROM E, S, C, D
WHERE E.Matr=S.Matr
AND E.CC=C.CC
AND C.CD=D.CD
AND C.CNome LIKE 'Fisica%'
AND S.Citta='MO'
```

Esempi di interrogazioni di Join (2)

♢ Un'altra tipologia di interrogazione risolvibile con il join sono interrogazioni del tipo "Selezionare il nome degli studenti che hanno sostenuto **almeno un** esame": Infatti se uno studente ha sostenuto almeno un esame, allora sarà presente nel join tra la relazione `S` e la relazione `E`.

1. Selezionare gli studenti (tutti gli attributi di studente) che hanno sostenuto almeno un esame

```
SELECT S.*
FROM  S, E
WHERE E.Matr=S.Matr
```

2. Selezionare gli studenti (tutti gli attributi di studente) che hanno sostenuto almeno un esame con voto maggiore di 24

```
SELECT S.*
FROM  S, E
WHERE E.Matr=S.Matr
AND E.Voto > 24
```

3. Selezionare gli studenti (tutti gli attributi di studente) che hanno sostenuto almeno un esame con voto maggiore di 24 di un corso di 'Fisica'

```
SELECT S.*
FROM E, S, C
WHERE E.Matr=S.Matr
AND E.CC=C.CC
AND C.CNome LIKE 'Fisica%'
AND E.Voto > 24
```

4. Selezionare gli studenti (tutti gli attributi di studente) che hanno sostenuto almeno un esame con voto maggiore di 24 di un corso tenuto da un docente di 'BO'

```
SELECT S.*
FROM E, S, C, D
WHERE E.Matr=S.Matr
AND E.CC=C.CC
AND C.CD=D.CD
AND D.Citta = 'BO'
AND E.Voto > 24
```

Outer-Join

♢ Oltre al join interno, con lo standard SQL92 sono stati introdotti gli operatori di outer join :

`<tabella1>` **LEFT JOIN** `<tabella2>` **ON** `<condizione>`:
mantiene le tuple di `<tabella1>` per cui non esiste corrispondenza in `<tabella2>`

`<tabella1>` **RIGHT JOIN** `<tabella2>` **ON** `<condizione>`:
mantiene le tuple di `<tabella2>` per cui non esiste corrispondenza in `<tabella1>`

`<tabella1>` **FULL JOIN** `<tabella2>` **ON** `<condizione>`:
mantiene le tuple di `<tabella1>` e di `<tabella2>` per cui non esiste corrispondenza in `<tabella2>` e `<tabella1>` rispettivamente

♢ Le tuple senza corrispondenza vengono concatenate con tuple di valori `null`, di lunghezza opportuna.

Es. "Tutti gli studenti con i relativi esami sostenuti inclusi gli studenti che non hanno sostenuto alcun esame"

```
SELECT S.Matr,E.CC
FROM   S LEFT JOIN E ON (S.Matr=E.Matr)
```

Es. "Combinazioni di studenti e di docenti residenti nella stessa città inclusi gli studenti (docenti) che risiedono in una città che non ha corrispondenza nella relazione dei docenti (studenti)"

```
SELECT S.Matr,S.Città,D.CD, D.Città
FROM   S FULL JOIN D ON (S.Città=D.Città)
```

S.Matr	S.Città	D.CD	D.Città
M6	BO	D2	BO
M3	MO	D1	MO
M4	MO	D1	MO
M5	MO	D1	MO
M1	SA	null	null
M2	PA	null	null
M7	null	null	null
null	null	D3	FI

Outer-Join (2)

♢ Nella clausola `FROM` è possibile esprimere più di un'operazione di join.

Es. "Per ogni esame con voto superiore a 24 riportare il nome dello studente e il codice del docente del corso"

```
SELECT  S.SNome,C.CD
FROM   (S JOIN E ON (S.Matr=E.Matr))
        JOIN C ON (E.CC=C.CC)
WHERE  Voto > 24
```

In modo equivalente

```
SELECT S.SNome,C.CD
FROM   S,E,C
WHERE  S.Matr=E.Matr
AND    E.CC=C.CC
AND    Voto > 24
```

Es. "Matricole degli studenti con i codici dei corsi dei relativi esami sostenuti, inclusi gli studenti che non hanno sostenuto alcun esame e i corsi per i quali non ci sono esami sostenuti"

```
SELECT DISTINCT S.Matr,C.CC
FROM   (S LEFT JOIN E ON (S.Matr=E.Matr))
        FULL JOIN C ON (E.CC=C.CC)
```

S.Matr	C.CC
M1	C1
M1	C2
M1	C3
M2	C1
M2	C2
M3	C2
M3	C3
M4	C3
M5	null
M6	C2
M7	C1
M7	C2
M7	C3
null	C4

Riepilogo degli operatori di Join

♢ Per riepilogare le operazioni di join introdotte, consideriamo le seguenti relazioni:

Studente (S)

Nome	Anno
Pio	1
Ada	2

Lavoratore (L)

Nome	Stipendio
Ada	10000
Ugo	15000

- **JOIN INTERNO (INNER JOIN)**
  "Nome,Anno e Stipendio degli *studenti lavoratori*".

  Le seguenti tre espressioni SQL sono equivalenti

```
SELECT S.Nome AS N, Anno, Stipendio
FROM S, L
WHERE S.Nome=L.Nome

SELECT S.Nome AS N, Anno, Stipendio
FROM S join L on S.Nome=L.Nome

SELECT S.Nome AS N, Anno, Stipendio
FROM S inner join L on S.Nome=L.Nome
```

  e forniscono come risultato la tabella

N	Anno	Stipendio
Ada	2	10000

- **JOIN ESTERNO (OUTER JOIN)**
  "Nome,Anno e Stipendio degli studenti; se lavoratori anche lo stipendio, altrimenti null".

  Le seguenti due espressioni SQL sono equivalenti

```
SELECT S.Nome as N, Anno, Stipendio
FROM S left join L on S.Nome=L.Nome

SELECT S.Nome as N, Anno, Stipendio
FROM S left outer join L on S.Nome=L.Nome
```

  e forniscono come risultato la tabella

N	Anno	Stipendio
Ada	2	10000
Pio	1	NULL

- *Left* and *Right* join sono duali:
  "Nome e Stipendio dei lavoratori; se studenti anche l'anno, altrimenti null".

  Le seguenti due espressioni SQL sono equivalenti

```
SELECT L.Nome as N, Stipendio, Anno
FROM S right join L on S.Nome=L.Nome

SELECT L.Nome as N, Stipendio, Anno
FROM L left join S on S.Nome=L.Nome
```

  e forniscono come risultato la tabella

N	Anno	Stipendio
Ada	2	10000
Ugo	NULL	15000

- **FULL OUTER JOIN**
  "Nome e Stipendio dei lavoratori; Se studenti anche l'anno, altrimenti null; Se lavoratori anche lo stipendio, altrimenti null."

```
SELECT S.Nome, Stipendio, Anno, L.Nome
FROM S full join L on S.Nome=L.Nome
```

S.Nome	Stipendio	Anno	L.Nome
NULL	15000	NULL	Ugo
Ada	10000	2	Ada
Pio	NULL	1	NULL

  ♢ Come fare il *merge* tra `S.Nome` e `L.Nome`, in modo da ottenere un'unica colonna con `N`?

N	Anno	Stipendio
Ugo	NULL	15000
Ada	2	10000
Pio	1	NULL

  - Si può utilizzare la funzione `ISNULL(EXP1,EXP2)`
    se il valore di `EXP1` è `NULL` riporta il valore specificato in `EXP2`,
    altrimenti riporta il normale valore di `EXP1`

```
SELECT ISNULL(S.Nome,L.Nome) AS N, Anno, Stipendio
FROM S full join L on S.Nome=L.Nome
```

Self Join

♢ Nel join tra una tabella e se stessa occorre necessariamente utilizzare dei sinonimi (*alias*) per distinguere le diverse occorrenze della tabella.

Es. "Coppie di studenti residenti nella stessa città"

```
SELECT S1.Matr,S2.Matr
FROM   S S1, S S2
WHERE  S1.Città = S2.Città
AND    S1.Matr < S2.Matr
```

oppure

```
SELECT S1.Matr,S2.Matr
FROM   S S1 Join S S2
       ON (S1.Città = S2.Città)
WHERE  S1.Matr < S2.Matr
```

S1.Matr	S2.Matr
M3	M5
M3	M4
M4	M5

Es. "Matricole degli studenti che hanno sostenuto almeno uno degli esami sostenuti dallo studente di nome 'Giuseppe Verdi' "

```
SELECT E1.Matr
FROM   S, E E1, E E2
WHERE  E2.Matr = S.Matr
AND    E1.CC = E2.CC
AND    S.SNome='Giuseppe Verdi'
```

Interrogazioni innestate

♢ Una interrogazione viene detta *innestata* o *nidificata* se la sua condizione è formulata usando il risultato di un'altra interrogazione, chiamata *subquery*.

In generale, un'interrogazione innestata viene formulata con:

Operatori quantificati : `<attr> <op-rel>[ANY|ALL] <subquery>`

Operatore di set : `<attr> [NOT] IN <subquery>`

Quantificatore esistenziale : `[NOT] EXISTS <subquery>`

♢ **Interrogazione innestata con operatori quantificati**

Il confronto tra un attributo e il risultato di una interrogazione,

`<attr> <op-rel> <subquery>`

non è in generale corretto in quanto si confronta il singolo valore assunto da `<attr>` con l'insieme di valori restituiti da `<subquery>`. Tuttavia, il confronto è possibile quando `<subquery>` produce (run-time) un valore atomico.
Nel confronto tra un attributo e il risultato di una interrogazione occorre specificare, dopo l'operatore relazionale `<op-rel>`, `ANY` oppure `ALL`.

Es. "Nome degli studenti che hanno sostenuto l'esame del corso C1"

```
SELECT SNome
FROM   S
WHERE  Matr =ANY  (  SELECT Matr
                     FROM   E
                     WHERE  CC='C1')
```

Es. "Studenti con anno di corso più basso"

```
SELECT *
FROM   S
WHERE  ACorso <= ALL  (  SELECT ACorso
                         FROM   S)
```

Interrogazioni innestate con l'operatore di set

Es. "Nome degli studenti che hanno sostenuto l'esame del corso C1"

```
SELECT SNome
FROM   E,S
WHERE  E.Matr=S.Matr
AND    E.CC = 'C1'
```

oppure

```
SELECT SNome
FROM   S
WHERE  Matr IN  (  SELECT Matr
                   FROM   E
                   WHERE  CC='C1')
```

La subquery restituisce l'insieme `(M1,M2,M7)` e pertanto la condizione dell'interrogazione innestata equivale a `Matr IN (M1,M2,M7)`.

Es. "Nome degli studenti che hanno sostenuto l'esame di un corso del docente D1"

```
SELECT SNome
FROM   E,S,C
WHERE  E.Matr=S.Matr
AND    E.CC=C.CC
AND    CD='D1'
```

oppure

```
SELECT SNome
FROM   S
WHERE  Matr IN  (  SELECT Matr
                   FROM   E
                   WHERE  CC IN  (  SELECT CC
                                    FROM   C
                                    WHERE  CD='D1'))
```

La subquery più interna restituisce l'insieme `(C1,C3)` e pertanto la condizione della subquery intermedia equivale a `CC IN (C1,C3)`;

La subquery intermedia restituisce quindi l'insieme `(M1,M2,M3,M4,M7)` e pertanto la condizione dell'interrogazione innestata equivale a `Matr IN (M1,M2,M3,M4,M7)`.

♢ Si noti che la query espressa tramite join e quella espressa tramite `IN` restituiscono lo stesso risultato a meno dell'eliminazione dei duplicati.

Subquery correlate

♢ Una subquery viene detta correlata se la sua condizione è formulata usando relazioni e/o sinonimi definite nella query esterna.

Es. "Nome degli studenti che hanno sostenuto l'esame del corso C1"

```
SELECT SNome
FROM   S
WHERE  'C1' IN  (  SELECT CC
                   FROM   E
                   WHERE  E.Matr=S.Matr)
```

Per ogni tupla della relazione S della query esterna, detta *tupla corrente*, si valuta la subquery che ha come risultato il codice degli esami sostenuti dallo studente corrente: se C1 è tra questi esami, il nome dello studente corrente viene riportato in uscita.

♢ Per una maggiore leggibilità è conveniente far uso di sinonimi

```
SELECT S1.SNome
FROM   S S1
WHERE  'C1' IN  (  SELECT E1.CC
                   FROM   E E1
                   WHERE  E1.Matr=S1.Matr)
```

♢ I sinonimi sono indispensabili quando una stessa relazione compare sia nella query esterna che nella subquery (analogalmente al self join)

Es. "Per ogni città, il nome degli studenti con anno di corso più alto"

```
SELECT S1.Città,S1.SNome
FROM   S S1
WHERE  S1.ACorso >= ALL  (  SELECT S2.ACorso
                            FROM   S S2
                            WHERE  S1.Città=S2.Città)
```

♢ Per gli attributi non qualificati avviene la qualificazione automatica con il nome della relazione *più vicina*.

Ad esempio, la precedente interrogazione si può scrivere come:

```
SELECT Città,SNome
FROM   S S1
WHERE  ACorso >= ALL  (  SELECT ACorso
                         FROM   S
                         WHERE  S1.Città=Città)
```

Il quantificatore esistenziale

♢ Il predicato

```
EXISTS (<subquery>)
```

ha valore `true` se e solo se l'insieme di valori restituiti da `<subquery>` è non vuoto.

Es. "Nome degli studenti che hanno sostenuto l'esame del corso C1"

```
SELECT SNome
FROM   S
WHERE  EXISTS  (  SELECT *
                  FROM   E
                  WHERE  E.Matr=S.Matr
                  AND    E.CC='C1' )
```

♢ Il predicato

```
NOT EXISTS (<subquery>)
```

ha valore `true` se e solo se l'insieme di valori restituiti da `<subquery>` è vuoto.

Es. "Nome degli studenti che non hanno sostenuto l'esame del corso C1"

```
SELECT SNome
FROM   S
WHERE  NOT EXISTS  (  SELECT *
                      FROM   E
                      WHERE  E.Matr=S.Matr
                      AND    E.CC='C1' )
```

♢ Generalmente, al fine di formulare query *significative* con `EXISTS` è indispensabile utilizzare subquery correlate:

```
SELECT SNome
FROM   S
WHERE   EXISTS  (  SELECT *
                   FROM   E
                   WHERE  E.CC='C1' )
```

Restituisce tutti gli studenti (nessun studente) se c'è (se non c'è) un esame del corso C1.

Riduzioni di query innestate (1)

♢ Le query innestate formulate con i seguenti operatori si possono ridurre a query di join equivalenti (stessa risposta per ogni possibile istanza della base di dati):

- `IN`
- `ANY` (con qualsiasi operatore di confronto)
- `EXISTS` con subquery correlata

Es. "Nome degli studenti che hanno sostenuto l'esame del corso C1"

1.
```
SELECT SNome
FROM   S
WHERE  Matr IN  (  SELECT Matr
                   FROM   E
                   WHERE  CC='C1')
```
2.
```
SELECT SNome
FROM   S
WHERE  Matr =ANY  (  SELECT Matr
                     FROM   E
                     WHERE  CC='C1')
```
3.
```
SELECT SNome
FROM   S
WHERE  EXISTS  (  SELECT *
                  FROM   E
                  WHERE  E.Matr=S.Matr
                  AND    E.CC='C1' )
```

Queste tre query sono equivalenti (a meno dell'eliminazione dei duplicati) alla seguente query di join:

```
SELECT SNome
FROM   E,S
WHERE  E.Matr=S.Matr
AND    E.CC='C1'
```

Riduzioni di query innestate (2)

♢ Le query innestate formulate con i seguenti operatori non si possono ridurre:

- `NOT IN`
- `ALL` (con qualsiasi operatore di confronto)
- `NOT EXISTS` con subquery correlata

Es. "Nome degli studenti che non hanno sostenuto l'esame del corso C1"

1.
```
SELECT SNome
FROM   S
WHERE  Matr NOT IN  (  SELECT Matr
                       FROM   E
                       WHERE  CC='C1')
```
2.
```
SELECT SNome
FROM   S
WHERE  Matr <> ALL  (  SELECT Matr
                       FROM   E
                       WHERE  CC='C1')
```
3.
```
SELECT SNome
FROM   S
WHERE  NOT EXISTS  (  SELECT *
                      FROM   E
                      WHERE  E.Matr=S.Matr
                      AND    E.CC='C1' )
```

Queste tre query sono equivalenti tra di loro ma non si possono ridurre ad una query di join.
Si noti infatti che la query di join:

```
SELECT SNome
FROM   E,S
WHERE  E.Matr=S.Matr
AND    E.CC <> 'C1'
```

restituisce il nome degli studenti che hanno sostenuto almeno un esame di un corso diverso da C1.

Considerazioni sulle interrogazioni SQL

♢ Una stessa interrogazione può essere scritta in vari modi equivalenti in SQL. Negli esempi fatti finora è stato evidenziato che:

- query del tipo "studenti che **hanno sostenuto** almeno un esame" possono essere espresse utilizzando il join, oppure, in modo equivalente, gli operatori `IN`, `ANY` ed `EXISTS`.
- query del tipo "studenti che **non hanno sostenuto** nessun esame" possono essere espresse in modo equivalente utilizzando gli operatori `NOT IN`, `ALL` e `NOT EXISTS`.
  In effetti, a causa di alcune limitazioni sul linguaggio SQL nei DBMS commerciali (vedere pagina 169) alcune interrogazioni possono essere espresse tramite il `NOT EXISTS` ma non tramite il `NOT IN`; un esempio di tale situazione è riportato a pagina 185.

♢ Un'altra importante tipologia di interrogazioni sono quelle del tipo "studenti che **hanno sostenuto 5** esami". Questo tipo di interrogazioni non vengono ovviamente espresse con gli operatori introdotti, in quanto richiederebbero delle complicate espressioni: nel caso di studente con 5 esami potrei fare quattro self join per considerare 5 tabelle `E` ...)

- Per esprimere query di questo tipo si usano le *Funzioni aggregate*.

Funzioni aggregate (column functions)

♢ SQL mette a disposizione una serie di funzioni per elaborare i valori di un attributo (`MAX`, `MIN`, `AVG`, `SUM`) e per contare le tuple che soddisfano una condizione (`COUNT`).

Opzioni: **`* :`** conteggio del numero di righe (`COUNT`)
`ALL :` trascura i `null` (`COUNT`,`AVG`,`SUM`); default
`DISTINCT :` trascura i `null` ed elimina i duplicati (`COUNT`,`AVG`,`SUM`)

♢ Allo scopo di evidenziare le varie opzioni ed in particolare il fatto che il valore `NULL` viene sempre trascurato nel calcolo delle funzioni aggregate, consideriamo la seguente tabella :

A	B
4	2
1	null
4	4

e riportiamo il valore restituito da alcune funzioni aggregate:

- `COUNT(*) = 3`
- `COUNT(A) = COUNT(ALL A) = 3`
- `COUNT(B) = COUNT(ALL B) = 2`
- `COUNT(DISTINCT A) = 2`
- `COUNT(DISTINCT B) = 2`
- `AVG(A) = AVG(ALL A) = 3`
- `AVG(DISTINCT A) = 2,5`
- `AVG(B) = AVG(ALL B) = 3`
- `AVG(DISTINCT B) = 3`

♢ In alcuni DBMS commerciali, il `COUNT( DISTINCT ...)` non è accettato: occorre effettuarlo in *due passi* usando le viste.

Esempi di funzioni aggregate

Es. "Numero di studenti presenti"

```
SELECT COUNT(*)
FROM   S
```

Es. "Numero di studenti che hanno sostenuto almeno un esame"

```
SELECT COUNT(DISTINCT Matr)
FROM   E
```

Es. "Numero di studenti con anno di corso non nullo"

```
SELECT COUNT(ACorso)
FROM   S
```

Es. "Numero di anni di corso di studenti presenti"

```
SELECT COUNT(DISTINCT ACorso)
FROM   S
```

Es. "Numero di coppie distinte matricola-voto"

```
SELECT COUNT(DISTINCT Matr,Voto)
FROM   E
```

Es. "Voto medio degli esami sostenuti dalla matricola M1"

```
SELECT AVG(Voto)
FROM   E WHERE Matr='M1'
```

che è equivalente a

```
SELECT SUM(Voto)/COUNT(Voto)
FROM   E WHERE Matr='M1'
```

Es. "Studenti il cui anno di corso è minore di quello massimo presente"

```
SELECT *
FROM   S
WHERE  ACorso <  (  SELECT MAX(ACorso)
                    FROM   S)
```

♢ Non è lecita la presenza contemporanea nella `<lista-SELECT>` di nomi di campi e funzioni aggregate (ciò è consentito solo nel caso di raggruppamento, come vedremo nel seguito), pertano la seguente interrogazione non è corretta:

```
SELECT Matr,MAX(Voto)
FROM   E
```

Raggruppamento: la clausola GROUP BY

♢ In una istruzione SELECT è possibile formare dei gruppi di tuple che hanno lo stesso valore di specificati attributi, tramite la clausola GROUP BY:

```
SELECT [DISTINCT|ALL] <lista-SELECT>
FROM <lista-FROM>
[WHERE <condizione>]
[GROUP BY <lista-group> ]
```

♢ Il risultato della SELECT è un **unico record per ciascun gruppo**:
pertanto nella <lista-SELECT> possono comparire solo:

- Uno o più attributi di raggruppamento, cioè specificati in <lista-group>
- Funzioni aggregate: tali funzioni vengono valutate, e quindi forniscono un valore unico, per ciascun gruppo

Es. "Voto massimo ottenuto per ogni studente"

```
SELECT   Matr,MAX(Voto)
FROM     E
GROUP BY Matr
```

Es. "Voto massimo e minimo ottenuto per ogni studente, escludendo il corso C1"

```
SELECT   Matr,MAX(Voto),MIN(Voto)
FROM     E
WHERE    CC <> 'C1'
GROUP BY Matr
```

Es. "Codice e nome di un corso, e relativo numero di esami sostenuti"

```
SELECT   C.CC,C.CNome,COUNT(*)
FROM     E,C
WHERE    E.CC=C.CC
GROUP BY C.CC,C.CNome
```

♢ La seguente interrogazione non è corretta:

```
SELECT   C.CC,C.CNome,COUNT(*)
FROM     E,C
WHERE    E.CC=C.CC
GROUP BY C.CC
```

Opzione ALL nel GROUP BY

♢ In presenza di una clausola `WHERE`, il raggruppamento riporta solo i valori che soddisfano la condizione della `WHERE`

Es. "Numero esami e somma dei voti per ogni studente, escludendo il corso C2"

```
SELECT   Matr,COUNT(*),SUM(Voto)
FROM     E
WHERE    CC <> 'C2'
GROUP BY Matr
```

♢ Tale query non restituisce alcuna informazione sulla matricola `M6`, in quanto nessuna tupla di `E` corrispondente a `M6` soddisfa la condizione

♢ Con l'opzione `ALL` nella clausola `GROUP BY`, cioè `GROUP BY ALL`:

```
SELECT   Matr,COUNT(*),SUM(Voto)
FROM     E
WHERE    CC <> 'C2'
GROUP BY ALL Matr
```

si ottiene anche la tupla `M6` per la quale viene riportato `COUNT`(*) pari a `0` e `SUM`(`Voto`) pari a `NULL`; questi due valori sono giustificati dal fatto che alla tupla `M6` corrisponde un *gruppo vuoto* e quindi con un numero di tuple pari zero (`COUNT`(*) = `0`) e valore medio nullo (infatti la funzione `SUM` applicata ad un insieme vuoto restituisce `NULL`

- In questo modo è possibile distinguere la matricola `M6`, che ha fatto esami ma nessuno soddisfa la condizione richiesta, dalla matricola `M5`, che non ha fatto alcun esame

♢ Come ottenere `0` invece di `NULL` anche per la funzione `SUM`(`Voto`) ?
Si utilizza la funzione `ISNULL` introdotta a pagina 150:

```
SELECT   Matr,COUNT(*), ISNULL(SUM(Voto),0)
FROM     E
WHERE    CC <> 'C2'
GROUP BY ALL Matr
```

Raggruppamento: La clausola HAVING

♢ La clausola `HAVING` è l'equivalente della clausola `WHERE` applicata a gruppi di tuple: ogni gruppo costruito tramite `GROUP BY` fa parte del risultato dell'interrogazione solo se il predicato specificato nella clausola `HAVING` risulta soddisfatto.

[GROUP BY <lista-group>]

[HAVING <condizione>]

Il predicato espresso nella clausola `HAVING` è formulato utilizzando

- Uno o più attributi specificati in `<lista-group>`
- Funzioni aggregate

Es. "Numero di esami per ogni ogni voto compreso tra 22 e 26"

```
SELECT   Voto,COUNT(*)
FROM     E
GROUP BY Voto
HAVING   Voto BETWEEN 22 AND 26
```

che è equivalente a

```
SELECT   Voto,COUNT(*)
FROM     E
WHERE    Voto BETWEEN 22 AND 26
GROUP BY Voto
```

Es. "Media dei voti per ogni esame sostenuto da più di due studenti"

```
SELECT   CC,AVG(Voto)
FROM     E
GROUP BY CC
HAVING   COUNT(*) > 2
```

Es. "Matricola degli studenti che hanno sostenuto almeno due esami con lo stesso voto"

```
SELECT   Matr
FROM     E
GROUP BY Matr,Voto
HAVING   COUNT(CC) >= 2
```

Considerazioni su raggruppamento e dipendenze funzionali

♢ Data la relazione $R(A, B)$, se A determina funzionalmente B, allora il raggruppamento su A restituisce *gli stessi gruppi* ottenuti raggruppando su A, B

♢ Tale considerazione può essere utile nella formulazione delle interrogazioni. Supponiamo ad esempio di voler ricavare gli "Studenti con cinque esami"

♢ Usando `HAVING` ottengo facilmente la `MATR` degli studenti

```
SELECT MATR
FROM E
GROUP BY MATR
HAVING COUNT(*) =   5
```

♢ Per avere il nome degli studenti si deve effettuare un join con la tabella `S`:

```
SELECT S.MATR,S.SNOME
FROM S,E
WHERE S.MATR=E.MATR
GROUP By S.MATR,S.SNOME
HAVING COUNT(*) = 5
```

♢ L'inserimento dell'attributo `S.NOME` nel `GROUP BY` non cambia la *logica di raggruppamento*, ovvero si ottengono gli stessi gruppi, in quanto `S.MATR` determina `S.NOME`. L'attributo `S.NOME` è stato inserito nel `GROUP BY` soltanto per poterlo poi riportare in uscita.

♢ La stessa interrogazione si può risolvere anche con:

```
SELECT S.MATR,S.SNOME
FROM S
WHERE  5  = (SELECT COUNT(*)
             FROM E
             WHERE E.MATR=S.MATR)


SELECT S.MATR,S.SNOME
FROM S
WHERE  MATR  IN (
              SELECT MATR
              FROM E
              GROUP BY MATR
              HAVING COUNT(*) =   5)
```

Unione, Differenza, Intersezione

♢ Sono definiti i seguenti operatori insiemistici relazionali:

Unione : `<subquery> UNION [ALL] <subquery>`

Differenza : `<subquery> EXCEPT [ALL] <subquery>`

Intersezione : `<subquery> INTERSECTION [ALL] <subquery>`

♢ Con `ALL` si considera la semantica del *multiset*

- $\{a, b\}$ `UNION ALL` $\{a\} = \{a, a, b\}$
- $\{a, a\}$ `EXCEPT ALL` $\{a\} = \{a\}$
- $\{a, a\}$ `INTERSECTION ALL` $\{a, b\} = \{a, a\}$

Es. "Città di studenti o docenti presenti"

```
SELECT Città FROM S
UNION
SELECT Città FROM D
```

Es. "Città di studenti ma non di docenti"

```
SELECT Città FROM S
EXCEPT
SELECT Città FROM D
```

Es. "Città di studenti e di docenti"

```
SELECT Città FROM S
INTERSECTION
SELECT Città FROM D
```

♢ Le interrogazioni formulate tramite gli operatori `EXCEPT` e `INTERSECTION` possono essere riscritte, in maniera equivalente, utilizzando operatori introdotti in precedenza, (ad esempio `NOT IN` e `IN` rispettivamente):

Es. "Città di studenti ma non di docenti"

```
SELECT Città
FROM   S
WHERE  Città NOT IN  (  SELECT Città
                        FROM   D)
```

Es. "Città in cui ci sono studenti e docenti"

```
SELECT Città
FROM   S
WHERE  Città IN  (  SELECT Città
                    FROM   D)
```

Questo non è valido per l'operatore `UNION`. Anche per questo motivo generalmente nelle implementazioni di SQL l'unico degli operatori presenti è `UNION`.

Divisione

♢ L'operazione di divisione non è definita in SQL. Pertanto, le interrogazioni che richiedono tale operatore, come ad esempio la seguente:

> "Studenti che hanno sostenuto **tutti** gli esami relativi a corsi del docente D1"

vengono in genere riformulate con una doppia negazione nel seguente modo"

> "Studenti per i quali **non** esiste alcun corso del docente D1 di cui **non** hanno sostenuto l'esame"

```
SELECT *
FROM   S
WHERE  NOT EXISTS
         ( SELECT *
           FROM   C
           WHERE  CD='D1'
           AND    NOT EXISTS
                    ( SELECT *
                      FROM   E
                      WHERE  E.Matr=S.Matr
                      AND    E.CC=C.CC))
```

Es. "Studenti che hanno sostenuto **tutti** gli esami sostenuti dallo studente M7", riformulata come

> "Studenti per i quali **non** esiste alcun esame sostenuto da M7 che essi **non** hanno sostenuto"

```
SELECT *
FROM   S
WHERE  Matr <> 'M7'
AND    NOT EXISTS
         ( SELECT *
           FROM   E E1
           WHERE  Matr='M7'
           AND    NOT EXISTS
                    ( SELECT *
                      FROM   E E2
                      WHERE  E2.Matr=S.Matr
                      AND    E2.CC=E1.CC))
```

Alcuni esempi di interrogazioni (1)

♢ Questo primo esempio riassume tutte le clausole di uno statement `SELECT`, mettendone in evidenza l'ordine con il quale esse vengono considerate per determinare il risultato dell'interrogazione.

Es. "Riportare, per ogni studente, il voto massimo ottenuto, escludendo gli esami con voto pari a 33 e considerando solo gli studenti con più di 10 esami; ordinare il risultato per voti massimi crescenti"

Per risolvere tale interrogazione occorre eseguire i seguenti passi:

1. considerare la relazione `E`:
   `FROM E`
2. selezionare gli esami con voto diverso da 33:
   `WHERE Voto <> 33`
3. raggruppare tali esami sulla base del valore di `Matr`:
   `GROUP BY Matr`
4. per ciascuno di tali gruppi contare il numero di esami (tuple) e controllare che sia maggiore di 10:
   `HAVING COUNT(*) > 10`
5. per ciascuno dei gruppi selezionati determinare il massimo voto e riportarlo in uscita assieme alla `Matr`:
   `SELECT Matr, MAX(Voto)`
6. ordinare il risultato sulla base del massimo voto calcolato:
   `ORDER BY 2`

L'interrogazione SQL risulta pertanto essere la seguente:

```
(5) SELECT Matr, MAX(Voto)
(1) FROM E
(2) WHERE Voto <> 33
(3) GROUP BY Matr
(4) HAVING COUNT(*) > 10
(6) ORDER BY 2
```

Alcuni esempi di interrogazioni (2)

Es. "Riportare, per ogni docente, il corso per il quale sono stati sostenuti il maggior numero di esami"

Per risolvere tale interrogazione occorre eseguire i seguenti passi:

1. effettuare un join tra `C` e `E`, in quanto il docente del corso (`CD`) è specificato in `C` e gli esami sono memorizzati in `E`:

   ```
   SELECT ...
   FROM   C C1,E E1
   WHERE  C1.CC=E1.CC
   ```

2. formare dei gruppi di tuple con lo stesso valore della coppia `CD`,`CC` e contare per ciascun gruppo, tramite `COUNT(*)` oppure `COUNT(Matr)`, il numero di tuple che lo compongono: il risultato del conteggio fornisce il numero di esami corrispondenti al docente `CD` per il suo corso `CC`

   ```
   SELECT   ...
   FROM     C C1,E E1
   WHERE    C1.CC=E1.CC
   GROUP BY C1.CD,C1.CC
   ```

3. una coppia `CD`,`CC` viene riportata in uscita dal `SELECT` se relativo conteggio delle tuple restituisce il valore più grande tra tutte le coppie corrispondenti allo stesso docente, cioè allo stesso valore di `CD`.

   Questa condizione deve essere espressa nella clausola `HAVING`:

   ```
   GROUP BY C1.CD,C1.CC
   HAVING COUNT(*) >= ALL (...)
   ```

   dove `(...)` sono i conteggi delle tuple relativi allo stesso docente, che si ottengono ripetendo il raggruppamento del punto 2, considerando però solo gli esami di c1.`CD`. Quindi in definitiva l'interrogazione SQL risulta essere la seguente:

   ```
   SELECT   C1.CD,C1.CC
   FROM     C C1,E E1
   WHERE    C1.CC=E1.CC
   GROUP BY C1.CD,C1.CC
   HAVING   COUNT(*) >= ALL  ( SELECT   COUNT(*)
                               FROM     C C2,E E2
                               WHERE    C2.CC=E2.CC
                               AND      C2.CD=C1.CD
                               GROUP BY C2.CD,C2.CC)
   ```

Si noti che nella subquery il raggruppamento può essere fatto solo su `C2.CC`, in quanto tutte le tuple di tale subquery hanno lo stesso valore di `C2.CD`, che è uguale a `C1.CD`.

Alcuni esempi di interrogazioni (3)

Es. "Docenti che hanno registrato esami per **tutti** i corsi da essi sostenuti"

può essere risolta riformulandola con una doppia negazione:

"Docenti per i quali **non** esiste un loro corso con **nessun** esame registrato"

```
SELECT *
FROM   D
WHERE  NOT EXISTS
         ( SELECT *
           FROM   C
           WHERE  C.CD=D.CD
           AND    NOT EXISTS
                    ( SELECT *
                      FROM   E
                      WHERE  E.CC=C.CC))
```

♢ Come ultima osservazione sulle interrogazioni formulabili in SQL, notiamo che in questo testo non abbiamo considerato la possibilità, permessa in SQL92, di esprimere condizioni su n-ple di attributi e non solo su un unico attributo. Ad esempio, in SQL92 si può scrivere la condizione `(ACorso,Città)=(2,'MO')`, equivalente a `(ACorso=2) AND (Città='MO')`. Un ulteriore esempio è il seguente:

Es. "Matricole degli studenti che hanno sostenuto un esame insieme (stesso corso, stessa data) alla matricola M2"

```
SELECT Matr
FROM   E
WHERE  (Data,CC) IN  (  SELECT (Data,CC)
                        FROM   E
                        WHERE  Matr = 'M2')
```

equivalente a:

```
SELECT E1.Matr
FROM   E E1
WHERE  EXISTS  (  SELECT *
                  FROM   E E2
                  WHERE  E2.Data = E1.Data
                  AND    E2.CC = E1.CC
                  AND    E2.Matr = 'M2' )
```

4.3 Manipolazione dei dati

- **Inserimento di record** :

  ♢ Inserimento esplicito di un singolo record:
  INSERT INTO `<tabella> [<lista-attr>]` **VALUES** `<lista-valori>`

  ♢ Inserimento del risultato di una interrogazione:
  INSERT INTO `<tabella> [<lista-attr>] <subquery>`

  - I valori riportati in `<lista-valori>` devono corrispondere in numero e tipo agli attributi specificati in `<lista-attr>`.

  - `<lista-attr>` deve essere contenuta nella lista di attributi di `<tabella>`: i valori corrispondenti agli attributi mancanti vengono posti uguali al valore di default oppure a `null`

  ♢ **Esempi** :

```
INSERT INTO S(Matr,SNome,Città,ACorso)
       VALUES ('M1','Lucia Quaranta','SA',1)

INSERT INTO S(Matr,SNome,ACorso)
       VALUES ('M1','Maria Rossi',1)
```

```
CREATE TABLE MEDIA
             (MATR CHAR(9) NOT NULL PRIMARY KEY,
             MEDIA INTEGER );
```

```
INSERT INTO MEDIA
        SELECT   Matr,AVG(Voto)
        FROM     E
        GROUP BY Matr
```

  - Nel caso in cui la `<lista-attr>` non sia specificata, essa viene assunta uguale alla lista di attributi nella creazione di `<tabella>`:

```
INSERT INTO S
       VALUES ('M2','Giacomo Tedesco','PA',2)
```

- **Modifica di record** :

  UPDATE <tabella> **SET** <attr> = <espressione>
  [**WHERE** <condizione>]

  Es. " Sostituire il codice corso C8 con C10 nella tabella degli esami"

```
UPDATE E
SET    CC='C10'
WHERE  CC='C8'
```

  Es. " Incrementare del 10% i voti inferiori a 24"

```
UPDATE E
SET    Voto=1.1*Voto
WHERE  Voto < 24
```

  Es. " Incrementare di 1 l'anno di corso degli studenti che hanno sostenuto più di 3 esami"

```
UPDATE S
SET    ACorso=ACorso + 1
WHERE   3 <  (  SELECT COUNT(*)
                FROM   E
                WHERE  E.Matr=S.Matr)
```

  Es. " Incrementare di 1 l'anno di corso di tutti gli studenti"

```
UPDATE S
SET    ACorso=ACorso+1
```

- **Cancellazione di record** :

  DELETE FROM <tabella> [**WHERE** <condizione>]

  Es. " Cancellare i corsi del docente D1"

```
DELETE FROM E
WHERE   CC in  (  SELECT CC
                  FROM   C
                  WHERE  CD='D1')
```

  Es. " Cancellare tutti i corsi "

```
DELETE FROM C
```

4.4 Viste, privilegi e cataloghi

Viste

♢ Una vista è una tabella "virtuale" definita, tramite un'interrogazione, da altre tabelle base o da altre viste (non sono ammesse però dipendenze ricorsive).

Es. "La vista `STUDENTIBIENNIO` riporta la matricola, il nome e la città degli studenti con anno di corso minore o uguale a 2"

```
CREATE VIEW STUDENTIBIENNIO(Matr,Città,ACorso)
  AS  SELECT Matr,Città,ACorso
      FROM   S
      WHERE  ACorso <= 2
```

♢ L'interrogazione non viene eseguita all'atto di creazione della vista ma quando la vista viene utilizzata.

♢ **Viste aggiornabili:**

sono possibili operazioni di modifica (`INSERT`, `UPDATE`, `DELETE`), tradotte in operazioni di modifica sulle tabelle base

♢ In SQL una vista è aggiornabile solo quando una sola riga di ciascuna tabella di base corrisponde ad una riga della vista.

In particolare, quindi **non** sono aggiornabili viste ottenute:

- tramite `GROUP BY` e funzioni aggregate
- tramite `DISTINCT` senza inclusione di una chiave
- tramite riferimenti a viste non aggiornabili

♢ In pratica, le viste aggiornabili variano da sistema a sistema, e tipicamente includono quelle definite come proiezioni e/o selezioni di una singola tabella di base; in alcuni sistemi viene anche richiesto che l'insieme di attributi della lista contenga una chiave. La restrizione più comune riguarda la non aggiornabilità di viste definite come join (2 o più variabili nella clausola `FROM`).

Viste - WITH CHECK OPTION

♢ Tale opzione, utilizzabile solo per viste aggiornabili, specifica che, dopo una operazione di inserimento o di modifica, le righe aggiornate devono continuare ad appartenere alla vista.

```
CREATE VIEW STUDENTIBIENNIO(Matr,Città,ACorso)
  AS  SELECT Matr,Città,ACorso
      FROM   S
      WHERE  ACorso <= 2
  WITH CHECK OPTION
```

Lo statement:

```
UPDATE STUDENTIBIENNIO
 SET ACORSO = 2
 WHERE MATR = M1
```

viene accettato e la tabella base S viene modificata, mentre lo statement:

```
UPDATE STUDENTIBIENNIO
 SET ACORSO = 3
 WHERE MATR = M1
```

non viene accettato (viene generato un errore e la tabella base S non viene modificata) in quanto la tupla modificata non soddisferebbe più il predicato che definisce la vista.

♢ **WITH [LOCAL | CASCADED] CHECK OPTION**

LOCAL : si controllano solo i predicati locali

CASCADED : si controllano tutti i predicati (default)

```
CREATE VIEW STUDENTIBIENNIOMODENA
  AS  SELECT *
      FROM   STUDENTIBIENNIO
      WHERE  Città = 'MO'
  WITH [LOCAL | CASCADED] CHECK OPTION
```

Lo statement:

```
INSERT INTO STUDENTIBIENNIO
VALUES ('M13','MO',3)
```

viene accettato con LOCAL ma non con CASCADED.

Uso delle viste per la formulazione di interrogazioni

♢ Le viste sono utili per:

1. Formulare interrogazioni non esprimibili tramite una singola `SELECT`.

   Ad esempio, per determinare "Il numero medio di esami sostenuti dagli studenti si dovrebbe:

   (a) raggruppare la relazione `E` su `Matr`: `GROUP BY Matr`

   (b) contare per ogni gruppo il numero esami: `COUNT(CC)`

   (c) calcolare la media dell'insieme di valori ottenuti in b: `AVG(COUNT(CC))`

   ♢ In SQL le funzioni aggregate **non si possono comporre**, quindi la seguente espressione non è corretta:

   ```
   SELECT   AVG(COUNT(CC))
   FROM     E
   GROUP BY Matr
   ```

   ♢ La composizione può essere effettuata usando le viste:

   ```
   CREATE VIEW STOT(Matr,NEsami)
     AS  SELECT   Matr,COUNT(CC)
         FROM     E
         GROUP BY Matr

   SELECT AVG(NEsami)
   FROM   STOT
   ```

2. Semplificare interrogazioni complesse.

   Es. "Matricola dello studente che ha sostenuto il maggior numero di esami"

   ```
   SELECT   Matr
   FROM     E
   GROUP BY Matr
   HAVING   COUNT(*) >= ALL  ( SELECT   COUNT(*)
                               FROM     E
                               GROUP BY Matr)
   ```

   Usando la vista `STOT`:

   ```
   SELECT Matr
   FROM   STOT
   WHERE  NESAMI =  (   SELECT MAX(NESAMI)
                        FROM   STOT)
   ```

Privilegi

♢ SQL include il concetto di *identificatore d'autorizzazione* (IA), o *user*, cioè il nome con cui un utente è conosciuto al sistema.

♢ L'utente che crea una risorsa ne è il proprietario ed è autorizzato a compiere su di essa qualsiasi operazione. L'utente può specificare quali sono le risorse a cui possono accedere gli altri utenti.

♢ Il sistema basa il controllo di accesso sul concetto di **privilegio**: per eseguire un'operazione, l'utente deve avere il privilegio necessario.

♢ Ogni privilegio è caratterizzato da:

1. **risorsa:** tabelle, attributi, viste, domini, indici
2. **utente che concede il privilegio** (grantor): UC
3. **utenti che ricevono il privilegio** (grantee): UR_i
4. **azione permessa sulla risorsa:**
   - **insert** (tabelle e attributi)
   - **update** (tabelle e attributi)
   - **delete** (tabelle)
   - **SELECT** (tabelle; per gli attributi si deve definire una vista)
   - **references** (tabelle e attributi): un utente URi può definire, in una sua tabella, una foreign key riferita a tabelle e/o attributi di UC: all'utente UC possono quindi essere vietate alcune operazioni su tali tabelle e/o attributi.
   - **usage** (domini)
5. **possibilità di trasmettere o meno il privilegio ad altri utenti**

♢ In fase di installazione del DBMS, uno user particolare, detto **System Administrator (SA)** riceve il privilegio di eseguire qualsiasi operazione legale nel sistema.

Grant

♢ L'utente UC concede su `<risorsa>` i privilegi specificati in `<azioni>` agli utenti elencati in `<utenti>` tramite il comando:

GRANT ALL|`<azioni>` **[ON** `<risorsa>`**]**

TO PUBLIC|`<utenti>` **[WITH GRANT OPTION];**

ALL : tutti i privilegi posseduti da UC sulla risorsa

PUBBLIC: tutti gli utenti, presenti e futuri, del sistema

WHIT GRANT OPTION: L'utente che ha ricevuto il privilegio può trasmetterlo ad altri

♢ **Esempi:**

```
SARA> CREATE TABLE STUDENTE
      (MATRICOLA CHAR(9) NOT NULL PRIMARY KEY
      CF CHAR(9) NOT NULL UNIQUE
      ...  );

SARA> GRANT UPDATE,SELECT,REFERENCES,REFERENCES(CF)
      ON STUDENTE TO ROCCO WITH GRANT OPTION;

ROCCO> GRANT UPDATE(CCORSO),REFERENCES(CF)
       ON STUDENTE TO MARIA, CARLO WITH GRANT OPTION;

MARIA> GRANT ALL ON STUDENTE TO PUBBLIC

MARIA> CREATE TABLE ESAME
       (MATRICOLA CHAR(9) NOT NULL REFERENCES STUDENTE
       ...  );

CARLO> CREATE TABLE ESAME
       (SCF CHAR(9) NOT NULL REFERENCES STUDENTE(CF)
       ...  );
```

♢ Si noti che i nomi degli oggetti vengono implicitamente qualificati con i nomi dei proprietari: `MARIA.ESAME` e `CARLO.ESAME`.

♢ Se `MARIA` e/o `CARLO` fanno degli inserimenti nelle proprie tabelle `ESAME`, il proprietario della tabella `STUDENTE`, `SARA`, non potrà cancellare e/o modificare i record di `STUDENTE` riferiti in `STUDENTE` da `MARIA` e/o `CARLO`.

Revoke

♢ L'utente UC può sottrarre alcuni o tutti i privilegi che aveva precedentemente accordato ad un altro utente tramite il comando:

REVOKE ALL|GRANT OPTION|<azioni> **[ON** <risorsa>**]**
FROM PUBLIC |<utenti> **[RESTRICT|CASCADE];**

GRANT OPTION: si revoca la possibilità di trasmettere ad altri il privilegio

RESTRICT (default): la revoca del privilegio non è possibile se vi sono oggetti o privilegi da esso dipendenti

CASCADE: si forza l'esecuzione del comando e la revoca viene propagata in cascata

♢ **Esempio:**

```
SARA> CREATE TABLE STUDENTE
      ...  );

SARA> GRANT SELECT ON STUDENTE TO ROCCO, CIRO
      WITH GRANT OPTION;

ROCCO> GRANT ALL ON STUDENTE TO MATTEO
       WITH GRANT OPTION;

CIRO> CREATE VIEW V1 AS SELECT ACORSO FROM STUDENTE

MATTEO> CREATE VIEW V2 AS SELECT SNOME FROM STUDENTE

SARA> REVOKE GRANT OPTION ON STUDENTE
      FROM ROCCO RESTRICT;

SARA> REVOKE SELECT ON STUDENTE
      FROM CIRO RESTRICT;
```

♢ I due comandi di revoca non vengono eseguiti. Invece con:

```
SARA> REVOKE GRANT OPTION ON STUDENTE
      FROM ROCCO CASCADE;
```

il comando viene eseguito e la revoca si propaga a "cascata": la vista V2 viene distrutta.

Cataloghi SQL

♢ La struttura di un database è memorizzata in un insieme di cataloghi. I cataloghi sono, a loro volta, tabelle e pertanto sono interrogabili in SQL.

Anche se la struttura dei cataloghi è definita solo in parte dallo standard SQL92, tutti le implementazioni gestiscono un proprio insieme di cataloghi; nel seguito, allo scopo di fare alcuni esempi, considereremo come riferimento il DBMS DB2.

♢ I principali cataloghi previsti in SQL92 sono i seguenti (tra parentesi è riportato il nome utilizzato da DB2):

Users (SYSCAT.DBAUTH): identificatori di autorizzazione di utenti e gruppi;

Tables (SYSCAT.TABLES): Tabelle, viste e alias;

Views (SYSCAT.VIEWS): In aggiunta alle informazioni di Tables c'è lo statement che definisce la vista, l'indicazione di check option, l'opzione readonly.

Indexes (SYSCAT.INDEXES): Contiene informazioni sul tipo di indice e il nome della tabella sul quale è definito;

Columns (SYSCAT.COLUMNS): Colonne di qualsiasi tabella, vista o alias;

Table_Constraint (SYSCAT.TABCONST): Vincoli di tabella di tipo CHECK, UNIQUE, PRIMARY KEY o FOREIGN KEY;

Key_Column_Usage (SYSCAT.KEYCOLUSE): Colonne che partecipano nella definizione di chiavi di tipo UNIQUE, PRIMARY KEY o FOREIGN KEY; (SYSCAT.COLCHECKS): Colonne referenziate da vincoli di tipo CHECK;

Referential_Constraint (SYSCAT.REFERENCES): Ogni riga rappresenta un vincolo di integrità referenziale (FOREIGN KEY);

Table_ Privileges (SYSCAT.TABAUTH): privilegi di tabella.

♢ Principali informazioni sulla struttura dei cataloghi (DB2):

```
SELECT TABNAME, TYPE
FROM   SYSCAT.TABLES
WHERE  TABSCHEMA ='SYSCAT'
AND    OWNER ='SYSIBM'

SELECT COLNAME
FROM   SYSCAT.COLUMNS
WHERE  TABNAME ='COLUMNS'
```

Cataloghi (DB2) : esempio

♢ Per mostrare un esempio completo di come la struttura di un database sia memorizzata nei cataloghi, definiamo uno schema relazionale in SQL e riportiamo alcune interrogazioni sui cataloghi per avere informazioni sulla struttura di tale schema. Lo schema relazionale è il seguente:

```
CREATE TABLE AUTO(
CODAUTO VARCHAR(10) NOT NULL PRIMARY KEY,
COSTR VARCHAR(10) NOT NULL,
SIGLA VARCHAR(10) NOT NULL,
CONSTRAINT AKTARGA UNIQUE(COSTR,SIGLA));

CREATE TABLE ACCESS(
CODACC VARCHAR(10) NOT NULL PRIMARY KEY,
DESCRIZIONE VARCHAR(20));

CREATE TABLE INSTALL(
COSTR VARCHAR(10) NOT NULL,
SIGLA VARCHAR(10) NOT NULL,
ANNOPROD INTEGER NOT NULL,
CODACC VARCHAR(10) NOT NULL REFERENCES ACCESS,
CONSTRAINT PKINSTALL PRIMARY KEY
(COSTR,SIGLA,CODACC,ANNOPROD),
CONSTRAINT FK1INSTALL FOREIGN KEY (COSTR,SIGLA)
REFERENCES AUTO(COSTR,SIGLA),
CONSTRAINT VINCOLO CHECK (SIGLA<>COSTR));
```

- Elencare le colonne della tabella `INSTALL`

```
SELECT T.TABNAME, C.COLNAME
FROM   SYSCAT.TABLES AS T, SYSCAT.COLUMNS AS C
WHERE  T.TABNAME = C.TABNAME
AND    T.TABNAME='INSTALL';
```

- Elencare i vincoli sulla tabella `INSTALL`

```
SELECT CONSTNAME, TYPE
FROM   SYSCAT.TABCONST
WHERE  WHERE TABNAME='INSTALL';
```

CONSTNAME	TYPE
PKINSTALL	P
SQL010509102242600	F
FK1INSTALL	F
VINCOLO	K

♢ I vincoli definiti su una tabella possono essere dei seguenti tipi:

SYMBOL	TYPE
F	foreign key
K	check
P	primary key
U	unique

- Per ottenere le colonne coinvolte nei vincoli definiti sulla tabella `INSTALL` è possibile utilizzare la seguente interrogazione:

```
SELECT TB.CONSTNAME, TYPE, COLSEQ, C.COLNAME
FROM   SYSCAT.TABCONST TB, SYSCAT.KEYCOLUSE C
WHERE  TB.CONSTNAME=C.CONSTNAME
AND    TB.TABNAME=C.TABNAME
AND    TB.TABNAME='INSTALL' ORDER BY 1
```

CONSTNAME	TYPE	COLSEQ	COLNAME
FK1INSTALL	F	1	COSTR
FK1INSTALL	F	2	SIGLA
PKINSTALL	P	1	COSTR
PKINSTALL	P	2	SIGLA
PKINSTALL	P	3	CODACC
PKINSTALL	P	4	ANNOPROD
SQL010509102242600	F	1	CODACC

- Come esempio conclusivo, la seguente interrogazione mostra come sia possibile ottenere la struttura delle foreign key:

```
SELECT R.CONSTNAME, C.COLSEQ, C.COLNAME, CR.TABNAME,
CR.COLNAME
FROM SYSCAT.REFERENCES R, SYSCAT.KEYCOLUSE C,
SYSCAT.KEYCOLUSE CR
WHERE R.TABNAME=C.TABNAME
AND R.CONSTNAME=C.CONSTNAME
AND R.REFKEYNAME=CR.CONSTNAME
AND R.REFTABNAME=CR.TABNAME
AND C.COLSEQ=CR.COLSEQ;
```

CONSTNAME	COLSEQ	COLNAME	TABNAME	COLNAME
SQL010509102242600	1	CODACC	ACCESS	CODACC
FK1INSTALL	1	COSTR	AUTO	COSTR
FK1INSTALL	2	SIGLA	AUTO	SIGLA

Cataloghi (DB2) : Informazioni statistiche

♢ Alcuni cataloghi mantengono **informazioni statistiche** usate dal modulo ottimizzatore per valutare i costi di diverse strategie di esecuzione di una istruzione SQL.

♢ DB2, utilizza diversi cataloghi per la memorizzazione delle statistiche memorizzati nello schema `SYSSTAT`. Le informazioni statistiche vengono create ed aggiornate tramite il comando `RUNSTATS`.

♢ I principali cataloghi utilizzati in DB2 per memorizzare le statistiche sono i seguenti:

SYSSTAT.COLUMNS: statistiche relative alle singole colonne definite per tabelle, viste o alias;

SYSSTAT.COLDIST: statistiche dettagliate relative alle singole colonne, ogni riga rappresenta l'n-esimo valore più frequente o l'n-esimo valore quantile della colonna;

SYSSTAT.INDEXES: statistiche relative agli indici;

SYSSTAT.TABLES: statistiche relative a tabelle, viste o alias;

SYSSTAT.ROUTINES: statistiche relative alle routine definite dall'utente;

4.5 Semplici esempi di interrogazioni SQL

Scopo di questa sezione è quello di presentare, commentare e risolvere alcune semplici interrogazioni SQL, evidenziando quando opportuno anche quelli che sono gli errori più comuni nella soluzione di determinate tipologie di interrogazioni. Le interrogazioni verranno riferite al seguente schema relazionale:

```
CLIENTE(NOME,CITTA,REGIONE, NAZIONE, TELEFONO)
PRODOTTO(CODPROD,PREZZO, TIPO,NUMERO)
ACQUISTO(NOMECLIENTE,CITTACLIENTE,CODPROD,DATA,)
   FK: NOMECLIENTE,CITTACLIENTE REFERENCES CLIENTE
   FK: CODPROD REFERENCES PRODOTTO
```

Es. "Elencare i clienti (tutti gli attributi di cliente) di nazionalità 'ITALIA"'

```
SELECT * FROM CLIENTE
WHERE NAZIONE = 'ITALIA'
```

Elencare i clienti (tutti gli attributi di cliente) di nazionalità 'ITALIA' significa elencare tutti i clienti (tutte le tuple) di nazionalità 'ITALIA' riportandone tutti gli attributi. Il fatto di volere tutti gli attributi di cliente spesso è sottointeso quindi scriveremo "Elencare i clienti di nazionalità 'ITALIA"'.

Es. "Elencare le regioni in cui ci sono clienti (ovvero c'è almeno un cliente)"

```
SELECT REGIONE
FROM CLIENTE
```

Es. "Elencare i clienti che hanno fatto degli acquisti (ovvero hanno fatto almeno un acquisto)"

```
SELECT CLIENTE.*
FROM CLIENTE, ACQUISTO
WHERE CLIENTE.NOME=ACQUISTO.NOMECLIENTE
AND CLIENTE.CITTA=ACQUISTO.CITTACLIENTE
```

Scriviamo la stessa query utilizzando gli alias

```
SELECT C.*
FROM CLIENTE  C, ACQUISTO A
WHERE C.NOME=A.NOMECLIENTE
AND C.CITTA=A.CITTACLIENTE
```

Es. "Elencare i clienti che hanno fatto degli acquisti di prodotti di tipo 'A' (ovvero hanno fatto almeno un acquisto di un prodotto di tipo 'A')"

```
SELECT C.*
FROM CLIENTE  C, ACQUISTO A, PRODOTTO P
WHERE C.NOME=A.NOMECLIENTE
AND C.CITTA=A.CITTACLIENTE
AND A.CODPROD = P.CODPROD
AND P.TIPO ='A'
```

Es. "Elencare gli acquisti fatti da clienti di 'PAPEROPOLI"'

```
SELECT *
FROM ACQUISTO
WHERE CITTACLIENTE = 'PAPEROPOLI'
```

Es. "Elencare gli acquisti fatti da clienti di nazionalità 'ITALIA"'

```
SELECT A.*
FROM CLIENTE  C, ACQUISTO A
WHERE C.NOME=A.NOMECLIENTE
AND C.CITTA=A.CITTACLIENTE
AND C.NAZIONE ='ITALIA'
```

Es. "Elencare gli acquisti fatti da clienti di nazionalità 'ITALIA' di un prodotto di tipo 'A"'

```
SELECT A.*
FROM CLIENTE  C, ACQUISTO A, PRODOTTO P
WHERE C.NOME=A.NOMECLIENTE
AND C.CITTA=A.CITTACLIENTE
AND A.CODPROD = P.CODPROD
AND P.TIPO ='A'
AND C.NAZIONE='ITALIA'
```

Es. "Elencare i prodotti non venduti, ovvero i prodotti per i quali non c'è nessun acquisto"

Occore fare una *differenza*, tra *tutti i prodotti* ed i *prodotti acquistati*:

```
SELECT P.*
FROM PRODOTTO P
WHERE P.CODPROD NOT IN ( SELECT CODPROD
                         FROM ACQUISTO  )
```

Si noti che la differenza tra i *tutti i prodotti* ed i *prodotti acquistati* viene fatta considerando la chiave della relazione `PRODOTTO`, ovvero `CODPROD`.

- Per ottenere i prodotti non venduti **non** si può usare la seguente query:

```
SELECT P.*
FROM ACQUISTO A, PRODOTTO P
WHERE A.CODPROD <>  P.CODPROD
```

  Infatti essa restituisce i prodotti per i quali c'è almeno un acquisto di un altro prodotto!

Es. "Elencare il tipo dei prodotti non venduti, ovvero i tipi X tale che c'è almeno un prodotto di tipo X che non è stato venduto."

Si utilizza la query precedente che forniva la differenza tra *tutti i prodotti* ed i *prodotti acquistati*, riportando in uscita solo il tipo dei prodotti:

```
SELECT TIPO
FROM PRODOTTO P
WHERE P.CODPROD NOT IN ( SELECT CODPROD
                         FROM ACQUISTO  )
```

Es. "Elencare i tipi non venduti, ovvero i tipi X tali che *nessun prodotto* di tipo X è stato venduto."

Diversamente dalle interrogazioni precedenti, dove si effettuava una *differenza* tra *tutti i prodotti* e i *prodotti acquistati*, in questo caso occorre fare una *differenza*, tra *tutti i tipi dei prodotti* e i *tipi dei prodotti acquistati*, ovvero la differenza è sull'attributo `TIPO`:

```
SELECT TIPO
FROM PRODOTTO P
WHERE TIPO NOT IN ( SELECT P.TIPO
                    FROM PRODOTTO P, ACQUISTO  A
                    WHERE A.CODPROD = P.CODPROD)
```

Un altro modo per risolvere questa interrogazione è quello di usare `NOT EXISTS`:

```
SELECT TIPO
FROM PRODOTTO X
WHERE NOT EXISTS( SELECT *
                    FROM PRODOTTO P, ACQUISTO  A
                    WHERE A.CODPROD = P.CODPROD
                    AND P.TIPO =  X.TIPO)
```

- In base a quanto discusso prima, per risolvere questa interrogazione è sbagliato scrivere

```
SELECT TIPO
FROM PRODOTTO P
WHERE P.CODPROD NOT IN ( SELECT CODPROD
                         FROM ACQUISTO  )
```

  Inoltre, come al solito, è sbagliato scrivere

```
SELECT P.TIPO
FROM ACQUISTO A, PRODOTTO P
WHERE A.CODPROD <>  P.CODPROD
```

  Infatti restituisce i tipi dei prodotti per i quali c'è almeno un acquisto di un altro prodotto!

Es. " Elencare i clienti che non hanno acquistato nessun prodotto"

In modo analogo a quanto fatto nelle interrogazioni precedenti, occore fare una *differenza* tra *tutti i clienti* e i *chienti che hanno acquistato*; siccome la chiave di `CLIENTE` è `(NOME, CITTA)` avremo:

```
SELECT *
FROM CLIENTE
WHERE (NOME,CITTA) NOT IN (SELECT NOMECLIENTE, CITTACLIENTE
                           FROM ACQUISTO)
```

In molti DBMS commerciali non è consentito esprimere condizioni su n-ple di attributi e quindi non possiamo esprimere la condizione `(NOME,CITTA) NOT IN ...`: tale condizione deve essere quindi riformulata in modo equivalente tramite `NOT EXISTS`:

```
SELECT *
FROM CLIENTE C
WHERE NOT EXISTS (SELECT *
                  FROM ACQUISTO A
                  WHERE A.NOMECLIENTE=C.NOME
                  AND A.CITTACLIENTE = C.CITTA        )
```

- Si noti che è ovviamente sbagliato esprimere la condizione `(NOME,CITTA) NOT IN ...` attraverso l'`AND` di due condizioni:

```
SELECT NOME
FROM CLIENTE
WHERE NOME NOT IN (SELECT NOMECLIENTE
                   FROM ACQUISTO)
AND CITTA NOT IN (SELECT CITTACLIENTE
                   FROM ACQUISTO)
```

Es. " Elencare il nome dei clienti che non hanno acquistato nessun prodotto"

Si utilizza la query precedente che forniva la differenza *tutti i clienti* e i *chienti che hanno acquistato*; riportando in uscita solo il nome dei clienti:

```
SELECT C.NOME
FROM CLIENTE C
WHERE NOT EXISTS (SELECT *
                  FROM ACQUISTO A
                  WHERE A.NOMECLIENTE=C.NOME
                  AND A.CITTACLIENTE = C.CITTA        )
```

Si noti che anche per ottenere solo il nome dei clienti, la differenza deve essere fatta considerando sempre la chiave di `CLIENTE`, quindi è sbagliato scrivere:

```
SELECT NOME
FROM CLIENTE
WHERE NOME NOT IN (SELECT NOMECLIENTE
                   FROM ACQUISTO)
```

Es. "Elencare la nazione dei clienti che non hanno acquistato alcun prodotto"

```
SELECT C.NAZIONE
FROM CLIENTE C
WHERE NOT EXISTS (SELECT *
                  FROM ACQUISTO A
                  WHERE A.NOMECLIENTE=C.NOME
                  AND A.CITTACLIENTE = C.CITTA        )
```

Es. "Elencare la nazione senza acquisti ovvero le nazioni X tale per cui NESSUN cliente della nazione X ha fatto un acquisto."

```
SELECT C.NAZIONE
FROM CLIENTE C
WHERE C.NAZIONE NOT IN (SELECT C1.NAZIONE
                        FROM CLIENTE C1, ACQUISTO A
                        WHERE A.NOMECLIENTE = C1.NOME
                        AND A.CITTACLIENTE = C1.CITTA)
```

Es. "Elencare, per ogni cliente, il numero totale dei suoi acquisti"

Essendo il cliente identificato da nome e città:

```
SELECT NOMECLIENTE, CITTACLIENTE, COUNT(*)
FROM ACQUISTO
GROUP BY NOMECLIENTE, CITTACLIENTE
```

D'altra parte se vogliamo ottenere anche altri attributi di cliente, quali il `TELEFONO`, si deve fare un join con la relazione `CLIENTE`, e raggruppare anche su `TELEFONO` (si consideri la discussione fatta a pagina 164):

```
SELECT C.NOMECLIENTE, C.CITTACLIENTE, C.TELEFONO, COUNT(*)
FROM ACQUISTO A, CLIENTE C
WHERE A.NOMECLIENTE = C.NOME
AND A.CITTACLIENTE = C.CITTA
GROUP BY C.NOMECLIENTE, C.CITTACLIENTE, C.TELEFONO
```

Es. " Elencare, per ogni CittaCliente, il numero totale degli acquisti (ovviamente intesi come gli acquisti dei clienti di quella citta) di prodotti con codice '1"'

```
SELECT CITTACLIENTE, COUNT(*)
FROM ACQUISTO
WHERE CODPROD = '1'
GROUP BY CITTACLIENTE
```

Es. " Elencare, per ogni Nazione, il numero totale degli acquisti (ovviamente intesi come gli acquisti dei clienti di quella Nazione)"

```
SELECT C.NAZIONE, COUNT(*)
FROM ACQUISTO A, CLIENTE C
WHERE A.NOMECLIENTE = C.NOME
AND A.CITTACLIENTE = C.CITTA
GROUP BY C.NAZIONE
```

Es. "Elencare NomeCliente e CittaCliente dei clienti con più di 10 acquisti"

```
SELECT NOMECLIENTE, CITTACLIENTE
FROM ACQUISTO
GROUP BY NOMECLIENTE, CITTACLIENTE
HAVING  COUNT(*) > 10
```

Es. " Elencare le nazioni con un numero totale degli acquisti (intesi come gli acquisti dei clienti di quella Nazione) maggiore di 13"

```
SELECT C.NAZIONE
FROM ACQUISTO A, CLIENTE C
WHERE A.NOMECLIENTE = C.NOME
AND A.CITTACLIENTE = C.CITTA
GROUP BY C.NAZIONE
HAVING COUNT(*) > 13
```

Es. " Elencare il tipo del prodotto più costoso"

```
SELECT TIPO
FROM PRODOTTO
WHERE PREZZO = (  SELECT MAX(PREZZO)
                  FROM PRODOTTO )
```

Es. "Elencare, per ogni tipo di prodotto, il Numero del prodotto più costoso"

```
SELECT P.TIPO, P.NUMERO
FROM PRODOTTO P
WHERE PREZZO = ( SELECT MAX(PREZZO)
                 FROM PRODOTTO  Q
                 WHERE Q.TIPO = P.TIPO)
```

Es. "Elencare, per ogni Data, il cliente (nome e città) che ha acquistato il prodotto più costoso"

```
SELECT A.DATA, A.NOMECLIENTE, A.CITTACLIENTE
FROM ACQUISTO A, PRODOTTO P
WHERE A.CODPROD=P.CODPROD
AND P.PREZZO = ( SELECT MAX(P1.PREZZO)
                 FROM ACQUISTO A1, PRODOTTO P1
                 WHERE A1.CODPROD=P1.CODPROD
                 AND A.DATA = A1.DATA)
```

4.6 SQL e linguaggi di programmazione

♢ È possibile eseguire enunciati SQL da un programma scritto in un linguaggio di programmazione quale COBOL, PL/1, RPG, C, ... ed avere una interazione tra variabili di programma e oggetti SQL.

♢ Per inserire statement SQL all'interno di un programma scritto in linguaggio ospite, la sintassi generica è: **exec sql** `SQL_statement ;`

Esempi:

```
exec sql  select  *
          from    STUDENTE
          where   MATRICOLA = '1234' ;
exec sql  update  STUDENTE
          set     NOME = 'Mario'
          where   MATRICOLA = '1234' ;
```

PROBLEMATICHE:

1. SQL opera in modo *orientato agli insiemi*, mentre i linguaggi di programmazione imperativi operano in modo *orientato al record*

   ♢ Soluzione: si introduce la nozione di *cursore:*

   - si associa un cursore ad una query di selezione
   - si apre il cursore come se si trattasse di un file sequenziale
   - si acquisisce un record alla volta dal cursore, con la possibilità di consultare i valori dei campi, aggiornare, cancellare
   - si chiude il cursore

2. È necessario poter usare variabili di programma per la composizione di statement SQL e poter copiare valori di campi in variabili di programma

   ♢ Soluzione: ogni implementazione di embedded SQL è dotata di adeguati strumenti sintattici.

3. È necessario disporre di strumenti per comunicare al programma lo stato delle esecuzioni SQL.

   ♢ Soluzione: c'è un parametro SQLCODE che restituisce al programma ospite un valore che indica l'esito dell'istruzione SQL eseguita:
   SQLCODE $= 0 \rightarrow$ ok
   SQLCODE $< 0 \rightarrow$ situazione di errore
   SQLCODE $= 100 \rightarrow$ non trovato

Operazioni su singolo record

♢ **Selezione:** selezione del livello `LIV` e della `CITTA` del fornitore `F` con codice `COD` pari a `:COD`

```
exec sql  select  LIV, CITTA
          into    :LIVELLO, :CITTA
          from    F
          where   COD = :COD ;
```

♢ **Aggiornamento:** aumento del livello `LIV` di una quantità pari a `VALORE` di tutti i fornitori `F` di `'Modena'`

```
exec sql  update  F
          set     LIV = LIV + :VALORE
          where   CITTA = 'Modena' ;
```

♢ **Cancellazione:** rimozione degli ordini `O` eseguiti dai fornitori `F` nella propria `CITTA`

```
exec sql  delete  from O
          where   CITTA = ( select  CITTA
                            from    F
                            where   F.COD = O.COD) ;
```

♢ **Inserimento:** inserimento di un articolo `A` con codice `:A_COD`, nome `:NOME` e peso `:PESO`

```
exec sql  insert
          into    A(COD, NOME, PESO)
          values  (:A_COD, :NOME, :PESO) ;
```

Operazioni con Cursore

♢ Un cursore è una variabile associata ad uno statement SELECT che assume come possibili valori le tuple risultanti dall'esecuzione dello statement stesso.

♢ Ogni cursore è dichiarato ed associato ad uno specifico statement SELECT:
DECLARE <cursor_name>
CURSOR FOR <select_clause>;

♢ La query *select_clause* non è eseguita all'atto della dichiarazione, ma quando il cursore viene aperto.
OPEN <cursor_name>;

♢ Quando è aperto, il cursore identifica una sequenza ordinata di righe, e una specifica posizione all'interno dell'ordinamento:
sopra una specifica riga
prima di una specifica riga
dopo l'ultima riga .
L'esecuzione dello statement OPEN posiziona il cursore prima della prima tupla.

♢ L'istruzione
FETCH <cursor_name> INTO <target_list>;
genera l'avanzamento del cursore alla riga successiva e la copia dei valori della tupla corrente nelle variabili della <target_list> corrispondenti alle colonne indicate nella <select_clause>.

♢ Dopo l'esecuzione di n FETCH, pari alla cardinalità del risultato, il cursore è posizionato dopo l'ultima tupla e SQLCODE assume il valore 100.

♢ Quando tutte le righe sono state esaminate possiamo disattivare il cursore con lo statement
CLOSE <cursor_name>;

Esempio di utilizzo del Cursore

Calcolare la media del valore di un ordine del fornitore avente codice `'1234'`, considerando solo i 30 di maggior importo.

Disponendo della tabella ORDINE(COD_ORD,CODF,VALORE), selezioniamo gli ordini (con i relativi valori), eseguiti dal fornitre richiesto.

```
select  COD_ORD, VALORE
from    ORDINE
where   CODF = '1234'
```

Volendo trasferire il risultato, una tupla alla volta, alle variabili di programma ORDINE e VALORE si introduce un cursore, Cur_Ordine, che permette di scandire le tuple secondo l'ordine con cui vengono prodotte in fase di esecuzione. Operando algebricamente sui VALORI si può calcolare la media.

	ORDINE.COD_ORD	**ORDINE.VALORE**
Cur_Ordine →	AB12	245,00
	AB14	3.472,67
	AF07	6.987,45
	...	

```
exec   sql declare Cur_Ordine cursor for
       select COD_ORD, VALORE
       from ORDINE
       where CODF = '1234'
       order by VALORE DESC;
...
for    (i=0;i<30;i++) {
       exec sql fetch Cur_Ordine into :ORDINE, :VALORE;
       somma_valore += VALORE;
}
       media_valore = somma_valore / 30;
...
```

Dichiarazioni di variabili

VARIABILI: Gli statement Embedded SQL utilizzano variabili C per trasferire dati dal database verso il programma (e viceversa).

exec sql begin declare section;
Dichiarazione di variabili e tipi in linguaggio C
exec sql end declare section;

♢ La definizione della variabili Embedded SQL può essere sia globale (esterna alle funzioni e procedure), sia locale, al pari delle variabili C. Le variabili utilizzate in statement embedded vengono definite in una sezione. I tipi di dati accettati dal preprocessore SQL sono i medesimi di quelli presenti nel linguaggio C (per ciascun DBMS esiste una tabella di compatibilità di tipi data tra Embedded SQL e C).

ETICHETTE: È possibile definire *etichette* referenziabili, aventi le seguenti caratteristiche: una label inizia con una lettera (o underscore '_'), è la prima parola di una riga, termina con i due punti **':'**.
Esempio: close_cursor: **exec sql close** cursor1;

♢ Una label non puó precedere una dichiarazione ed in generale è bene utilizzare le label solo in statement di comandi di esecuzione.

SQL Communications Area

SQLCA è la struttura dati predefinita per la gestione degli errori generati in ambiente SQL. L' istruzione **exec sql include sqlca;** specifica al preprocessore di includere la variabile **sqlca** che contiene le informazioni relative all'ultima istruzione SQL eseguita. **sqlca** contiene (tra gli altri) i campi:

sqlcode : specifica lo stato di ritorno di una istruzione SQL:
0 per esecuzione corretta, < 0 per situazione di errore e 100 risultato vuoto.

sqlwarn : struttura di 8 campi (sqlwarn0 ... sqlwarn7) con le warning relative alle ultime istruzioni eseguite;

sqlerrd[6] : il terzo campo [2] indica il numero di tuple coinvolte durante l'ultima esecuzione.

Error Handling con sqlca : **exec sql whenever** `condition action;`

`Condition` può assumere uno dei seguenti tre valori:

sqlwarning: indica che l'ultima istruzione SQL Embedded eseguita ha prodotto una warning. La variabile sqlwarn0 di SQLCA vale W. È quindi possibile specificare azioni condizionate a warning del DBMS.

sqlerror: indica che l'ultima istruzione SQL Embedded eseguita ha prodotto un errore. La variabile sqlcode di SQLCA ha valore negativo.

not found: indica che l'istruzione di select, update, insert, delete, ... non ha avuto effetto su alcuna riga. La variabile sqlcode di SQLCA è posta a 100.

`Action` può assumere uno dei seguenti quattro valori:

continue: continua l'esecuzione a partire dalla prossima istruzione. Nel caso di fatal error viene visualizzato un messaggio di errore e il programma termina.

stop: viene visualizzato un messaggio di errore e il programma termina. Nel caso in cui il database sia connesso quando la condizione é raggiunta la terminazione non esegue il commit delle operazioni compiute. L'azione di stop non può essere eseguita per la condizione di not found.

goto label: trasferisce il controllo del programma alla label specificata.

call `procedure`: chiama la procedura specificata. Nessun argomento può essere passato nella chiamata. Ad esempio, `call sqlprint` chiama una procedura che visualizza un messaggio riguardante l'errore o la warning occorsa.

♢ Ciascuna `action` ha effetto sulle istruzioni Embedded SQL fino alla successiva **whenever**

Compilare un programma Embedded C/SQL unix

I passi necessari in DB2 per eseguire la compilazione sono i seguenti.

♢ Eseguire la precompilazione da embedded SQL a C:

Il preprocessore è chiamato **prep**
prep <filename.sqc>
Preprocessa <filename>.sc e genera <filename>.c

♢ Compilare il codice C ottenuto

```
cc -c <filename>.c
```

♢ Linkare i codici oggetto con le librerie del DBMS

```
cc -o <filename>.o -I
$I_SYSTEM/db2inst/sqllib/include/ -L
$I_SYSTEM/db2inst/sqllib/lib/ -ldb2
<filename>.c
```

Esempio Completo

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sqlenv.h>
#include <sqlutil.h>

/* La section e' l'area di definizione delle variabili
   utilizzate dal DB */
EXEC SQL BEGIN DECLARE SECTION;
long code_c_ordine;
short n_ordini;
double somma_valore;
short cont;
double media_valore;
char codice_attuale[5];
struct {
 char cod_ord[5];
 char codf[5];
 double valore;
 } ordine_rec;
EXEC SQL END DECLARE SECTION;
```

Compilare un programma Embedded C/SQL unix (1)

```
main(){
/* Inclusione dellarea di comunicazione */
    EXEC SQL INCLUDE SQLCA;

/* Connessione al Database */
EXEC SQL CONNECT TO "sample";

/* Dichiarazione del cursore */
EXEC SQL DECLARE c_ordine CURSOR FOR
  select COD_ORD, CODF, VALORE
  from ORDINE
  order by CODF, VALORE DESC;

/* Apertura del cursore */
EXEC SQL OPEN c_ordine;
/* Caricamento del cursore nelle variabili */
EXEC SQL FETCH c_ordine INTO :ordine_rec.cod_ord,
 :ordine_rec.codf,
 :ordine_rec.valore;
code_c_ordine = SQLCODE;

/* scansione sequenzialmente il cursore */
while (code_c_ordine == 0) {
 somma_valore = 0;
 /* conteggio ordini effettuati da un fornitore */
 EXEC SQL select count(*) into :n_ordini
 from ordine
 where CODF = :ordine_rec.codf;
 strcpy(codice_attuale, ordine_rec.codf);
 cont = 1;
 /* totale esami scartando i due peggiori */
 while (strcmp(ordine_rec.codf, codice_attuale)==0 &&
        sqlca.sqlcode == 0 ) {
 /* sommatoria dei migliori 30 ordini */
 if (cont <= 30 && cont <= n_ordini)
 {
 cont++;
 somma_valore += ordine_rec.valore;
 }
```

Compilare un programma Embedded C/SQL unix (2)

```
 /* caricamento ordine successivo */
 EXEC SQL FETCH c_ordine INTO :ordine_rec.cod_ord,
 :ordine_rec.codf,
 :ordine_rec.valore;
 code_c_ordine = SQLCODE;
 }
 /* calcolo media ordini */
 media_valore = somma_valore/(cont-1);
 /* aggiornamento della media del fornitore */
 EXEC SQL update FORNITORE
 set MEDIA_ORDINI = :media_valore
 where CODF = :codice_attuale;
}

EXEC SQL CLOSE c_ordine;

/* Dichiarazione del cursore */
EXEC SQL DECLARE c_forn CURSOR for
select CODF, MEDIA_ORDINI
from FORNITORE
order by MEDIA_ORDINI desc;

/* Apertura del cursore */
EXEC SQL OPEN c_forn;

/* Caricamento del cursore nelle variabili */
EXEC SQL FETCH c_forn INTO :codice_attuale, :media_valore;
printf("Media dei migliori 30 ordini\tCODF\tVALORE\n");
while (sqlca.sqlcode == 0) {
printf("%s\t%f\n", codice_attuale, media_valore);
EXEC SQL FETCH c_forn INTO :codice_attuale, :media_valore;}

EXEC SQL CLOSE c_forn;

/* commit */
EXEC SQL COMMIT;
/* chiusura database */
EXEC SQL CONNECT RESET;
}
```

4.7 Stored Procedure

♢ Una *stored procedure* è una sequenza di istruzioni SQL con parametri memorizzata direttamente nel RDBMS:

```
procedure AssegnaAnno (:Stud varchar(20), :Anno int)
            update S
            set ACorso = :Anno
            where SNome = :Stud;
```

♢ Una stored procedure è eseguita dal RDBMS su esplicita richiesta degli utenti o delle applicazioni:

- Una stored procedure può essere invocata *internamente*, come una normale istruzione SQL:
  `execute procedure AssegnaAnno('Ugo Rossi', 3);`
- Una stored procedure può essere invocata *esternamente*, all'interno di un altro programma; ad esempio, se all'interno di un programma C si hanno le variabili `S` ed `A`, si può scrivere: `$ AssegnaAnno(:S, :A);`

♢ Lo standard **SQL92** prevede che una stored procedure sia composta da una sola istruzione SQL. In molti RDBMS questa limitazione è rimossa ed è consentito l'uso di variabili, di strutture di controllo, di sequenze di istruzioni ed altro: di fatto la definizione delle stored procedure non ha ancora raggiunto un livello accettabile di standardizzazione nei vari RDBMS ed ogni ogni sistema adotta un proprio linguaggio di programmazione proprietario, tra i quali citiamo PL/SQL, Informix4GL, Delphi e DB2 SQL/PL. Nel seguito vengono riportati alcuni esempi intuitivi per illustrare le principali caratteristiche delle stored procedure.

- È consentito utilizzare istruzioni SQL anche nella formulazione di condizioni delle strutture di controllo:

```
procedure CambiaAnno (:NomeStud varchar(20),
                      :NuovoAnno int)
    if    ((  select * from S
              where SNome = :NomeStud) = NULL)
                  insert into ErroriStud values (:NomeStud);
     else
           update S
           set ACorso = :NuovoAnno
           where Nome = :NomeStud;
    end if;
 end;
```

Esempio di Stored procedure in DB2 SQL/PL

♢ Consideriamo la seguente stored procedure, formulata in DB2 SQL/PL:

```
CREATE PROCEDURE ASSEGNAANNO(IN MATR VARCHAR ( 10 ) ,
                             OUT A_CORSO INTEGER ,
                             IN SOGLIA INTEGER )
 SPECIFIC ASSEGNAANNO
  BEGIN
    DECLARE n_esami INTEGER;
     SET n_esami = 0;
      SELECT COUNT(*) INTO n_esami
      FROM E
      WHERE E.MATR = Matr;
    IF n_esami >= Soglia
      THEN UPDATE S
        SET ACorso = ACorso + 1
        WHERE S.MATR = Matr;
    END IF;
    SELECT ACorso INTO A_CORSO
    FROM S
    WHERE S.MATR = Matr;
END
```

- Con `DECLARE` si dichiarano variabili locali per la procedure. Si noti che la variabile `n_esami` non è strettamente necessaria in quanto si può scrivere la condizione dell'`IF` utilizzando direttamente l'espressione SQL: `(SELECT COUNT(*) FROM E WHERE E.MATR = Matr >= Soglia)`.

- Con `OUT` vengono indicati i *parametri di output* della procedura, che vengono restituiti dall'esecuzione della procedura:

```
CALL AssegnaAnno(123,20,?)

Valore di parametri di output
--------------------------
Nome parametro  : ACORSO
Valore parametro : 2
```

4.8 Trigger

Regole Attive (Trigger SQL-99)

♢ Programmi attivati *automaticamente* dal DBMS al verificarsi di determinate condizioni e operazioni sulle tabelle
Da un punto di vista generale, in un trigger vengono specificati tre elementi (EVENT- CONDITION- ACTION): al seguito della modifica specificata in EVENT se è vera la condizione CONDITION vengono specificate le azioni di ACTION

```
1) CREATE TRIGGER <NOMETRIGGER>
2) <BEFORE|AFTER>
3) <INSERT|DELETE|UPDATE[OF <COLONNE>]>
            ON <NOMETABELLA> REFERENCING
4)          OLD [ROW] [AS] <NOMETABELLA>
           |NEW [ROW] [AS] <NOMETABELLA>
           |OLDTABLE [AS] <NOMETABELLA>
           |NEWTABLE [AS] <NOMETABELLA>
5) [WHEN (<CONDITION>)]
6) <ACTION>
```

1. Nome del trigger
2. Tipo del trigger: se deve essere attivato prima o dopo un'operazione
3. Operazioni che attivano il trigger (EVENT)
4. Trigger *set-oriented* (attivato un'unica volta) oppure *tuple-oriented* (attivato tante volte quante sono le tuple della tabella interessate dall'operazione)
5. Condizione eventuale che deve essere verificata per attivare il trigger
6. Azioni da eseguire

♢ In `<CONDITION>` ed `<ACTION>` si possono usare variabili particolari

- le tabelle speciali **OLD_TABLE** (contiene i record di `<NomeTabella>` *prima* dell'azione) e **NEW_TABLE** (contiene i record di `<NomeTabella>` *dopo* l'azione)
- riferimenti alle colonne di `<NomeTabella>` qualificate con **old** (valore prima dell'azione) oppure con **new** (valore dopo l'azione)

Uso dei Trigger

♢ Si possono classificare i trigger in base al loro uso:

Trigger passivi l'azione è sempre un abort, cioè serve a sollevare un fallimento sotto certe condizioni.
Sono utili per definire vincoli di integrità non esprimibili dichiarativamente in SQL, come ad esempio vincoli di **integrità dinamici** (riferiti a due stati del database)

Trigger attivi modificano lo stato della base di dati in corrispondenza di certi eventi.
Sono utili soprattutto per mantenere certi vincoli di integrità modificando i dati in maniera opportuna, come ad esempio nel caso di:
⇒ Vincolo di integrità referenziale
⇒ Per calcolare un dato derivato

♢ **Risoluzione dei conflitti**

- Quando una operazione attiva più di un trigger, sono possibili due alternative:
  Esecuzione parallela: esecuzione concorrente di tutti i trigger
  Esecuzione seriale: si eseguono i trigger in base ad un certo ordine, che puo essere:
  - **implicito:** in base all'ordine di definizione dei trigger
  - **esplicito:** si specifica l'ordine in base al quale i trigger vanno attivati
  - **stabilito** dal sistema

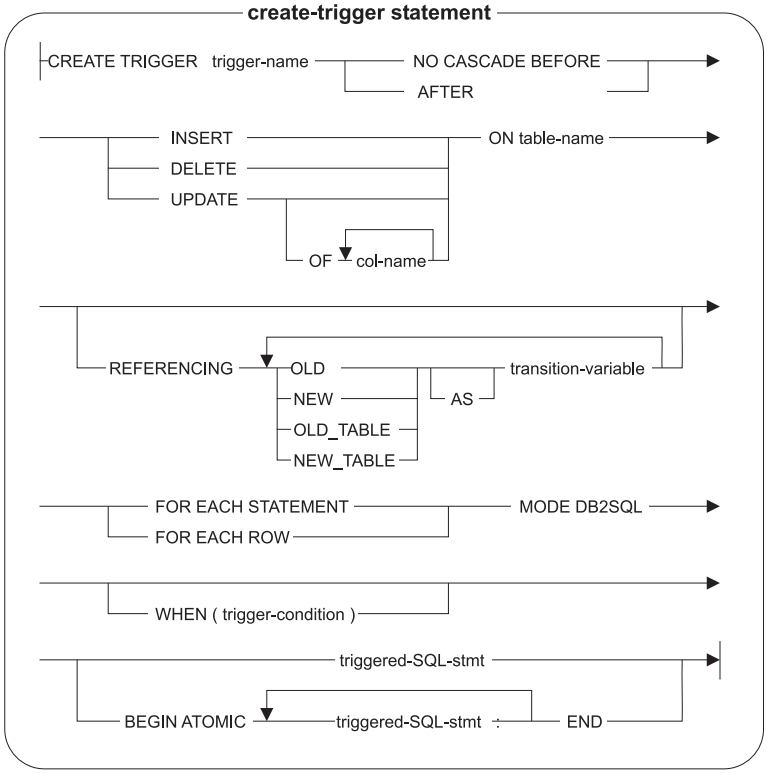
♢ **Interazione con le transazioni**

- Di solito l'azione di un trigger diventa parte della transazione che lo attiva: se l'azione abortisce, abortisce anche la transazione e viceversa.

In questa sezione useremo la sintassi del DBMS IBM DB2, che ha la sintassi più vicina allo standard SQL 99.

Trigger in DB2

♢ La sintassi dei trigger in DB2 è la seguente:



- l'EVENTO che attiva il trigger può essere solo un'operazione; se più trigger hanno lo stesso evento saranno eseguiti nell'ordine di creazione.
- Un trigger set-oriented è detto anche *row trigger*, mentre un trigger tuple-oriented è detto anche *statement trigger*. Si noti che un evento che opera su una tabella ma non modifica nessuna tupla attiva uno statement trigger ma non un row trigger.
- OLD e NEW si riferiscono, rispettivamente, ai valori prima e dopo l'evento.
- OLD_TABLE contiene *tutte e sole* le tuple modificate così come esse apparivano prima dell'evento.
- NEW_TABLE contiene *tutte e sole* le tuple modificate così come esse appaiono dopo l'evento.
- Le variabili OLD e NEW, OLD_TABLE e NEW_TABLE si possono utilizzare nella *trigger-condition*.

La seguente tabella mostra, per ogni tipo di trigger, le variabili che si possono utilizzare nel caso di row trigger e nel caso di statement trigger:

TIPO DI TRIGGER	ROW TRIGGER	STATEMENT TRIGGER
BEFORE INSERT	NEW	invalido
BEFORE UPDATE	OLD,NEW	invalido
BEFORE DELETE	OLD	invalido
AFTER INSERT	NEW, NEW_TABLE	NEW_TABLE
AFTER UPDATE	OLD, OLD_TABLE NEW, NEW_TABLE	OLD_TABLE, NEW_TABLE
AFTER DELETE	OLD, OLD_TABLE	OLD_TABLE

In particolare si noti che un trigger di tipo BEFORE può essere solo un row trigger: infatti i row trigger sono usati generalmente per "condizionare" i valori prima che essi vengano inseriti nel database. Un BEFORE TRIGGER non può effettuare una generica modifica al database (il NO CASCADE serve proprio per ricordare che un BEFORE TRIGGER non può attivare mai un altro BEFORE TRIGGER) ma può modificare solo le nuove (NEW) tuple.

TRIGGER in DB2 : Action

♢ La Action (*trigger body*) consiste di una o più istruzioni SQL;

Nel caso di più istruzioni si deve usare BEGIN ATOMIC e END e terminare tutte le istruzioni con ;
Un trigger body si comporta come una singola transazione : l'abort di una istruzione provoca l'abort di tutte le altre istruzioni che compongono il trigger body

Il trigger body dipende dal tipo di trigger :

- In un before trigger si può usare un'istruzione di Assignment tramite SET

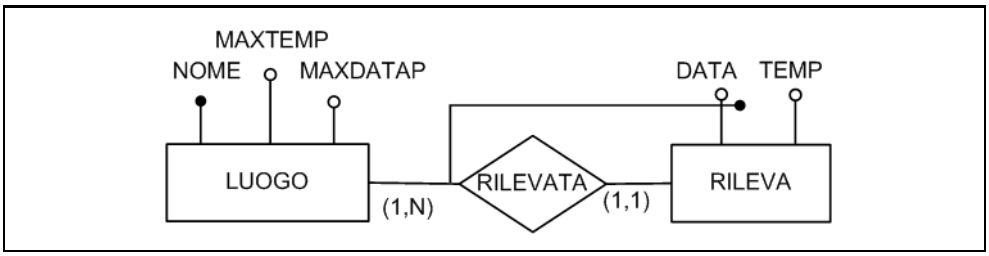
```
DIPENDENTE( COD ,NUMDIP,SALARIO)
DIPARTIMENTO( NUMERO ,MINSAL)
CREATE TRIGGER DIP1
NO CASCADE BEFORE INSERT ON DIPENDENTE
REFERENCING NEW AS N
FOR EACH ROW MODE DB2SQL
SET SALARIO = (SELECT MINSAL FROM DIPARTIMENTO
               WHERE NUMERO=N.NUMDIP)!
-- NOTA CHE SALARIO E' IMPLICITAMENTE QUALIFICATA CON NEW
```

- In un trigger (before o after) si può usare un'istruzione di SIGNAL per effettuare il rollback dell'evento che ha attivato il trigger

```
CREATE TRIGGER CARDMA_I
AFTER INSERT ON DIPENDENTE
REFERENCING NEW_TABLE AS NUOVAT
FOR EACH STATEMENT MODE DB2SQL
WHEN (10 <=ANY (SELECT COUNT(*) FROM DIPENDENTE,NUOVAT
                WHERE DIPENDENTE.COD=NUOVAT.COD
                GROUP BY DIPENDENTE.COD))
SIGNAL SQLSTATE '70000'('TROPPI DIPENDENTI')!
```

Esempio di TRIGGER in DB2 per i Dati Derivati

1. Uso degli after trigger per calcolare e mantenere aggiornato un dato derivato; un after trigger può effettuare una generica modifica su una qualsiasi tabella.



Nella tabella LUOGO abbiamo il dato derivato composto dalla massima temperatura rilevata (riportata in RILEVA) e dalla relativa data.

```
CREATE TABLE LUOGO(NOME VARCHAR(5) NOT NULL PRIMARY KEY ,
MAXTEMP INTEGER,
MAXDATA DATE)!
CREATE TABLE RILEVA(NOME VARCHAR(5) NOT NULL ,
DATA DATE NOT NULL,
TEMP INTEGER,
CONSTRAINT CHIAVERILEVA PRIMARY KEY(NOME,DATA),
CONSTRAINT FKTOLUOGO FOREIGN KEY(NOME)
           REFERENCES LUOGO ON DELETE CASCADE)!
```

- Dopo l'inserimento in RILEVA

```
AFTER INSERT ON RILEVA
REFERENCING NEW AS NUOVARIGA
FOR EACH ROW MODE DB2SQL
WHEN (NUOVARIGA.TEMP> (SELECT MAXTEMP FROM LUOGO
                       WHERE NOME=NUOVARIGA.NOME)
      OR (SELECT MAXTEMP FROM LUOGO
          WHERE NOME=NUOVARIGA.NOME) IS NULL)
UPDATE LUOGO
SET MAXTEMP=NUOVARIGA.TEMP, MAXDATA = NUOVARIGA.DATA
WHERE NOME=NUOVARIGA.NOME!
```

OPPURE, possiamo togliere la condizione e calcolarci sempre la max temperatura :

```
CREATE TRIGGER RILEVAMASSIMO_I
AFTER INSERT ON RILEVA
REFERENCING NEW AS NUOVARIGA
FOR EACH ROW MODE DB2SQL
UPDATE LUOGO
```

```
SET MAXTEMP= (SELECT MAX(TEMP) FROM RILEVA
              WHERE NOME=NUOVARIGA.NOME),
MAXDATA= (SELECT MAX(DATA) FROM RILEVA
          WHERE NOME=NUOVARIGA.NOME
AND TEMP = (SELECT MAX(TEMP) FROM RILEVA
            WHERE NOME=NUOVARIGA.NOME) )
WHERE NOME=NUOVARIGA.NOME!
```

Il valore massimo va calcolato su RILEVA che contiene tutte le tuple, anche quelle nuove inserite dall'insert che ha attivato il trigger, in quanto è un after insert. È conveniente tenere la condizione per limitare il numero di update sul database.

- Dopo la modifica della DATA e/o della TEMP su RILEVA:
  (1) Se modifico una DATA: devo verificare se la data modificata è quella di massima temperatura ed in tal caso modificare di conseguenza anche MAXDATA in LUOGO
  (2) Se modifico una TEMP: devo considerare due casi:
  (a) nel caso di incremento, la nuova temperatura può divenire quella massima
  (b) nel caso di decremento, la vecchia temperatura poteva essere quella massima (questo caso è individuato con il controllo sulla data) : in questo caso occorre ricalcolare Si potrebbero definire quindi definire tre trigger (V indica OLD e N indica NEW)

  Per il caso (1)

  ```
  EVENT : AFTER UPDATE OF DATA ON RILEVA
  CONDITION : (V.DATA=(SELECT MAXDATA FROM LUOGO
  WHERE NOME=V.NOME)
  ACTION : <MODIFICA LA DATA>
  ```

  Per il caso (2.a)

  ```
  EVENT : AFTER UPDATE OF TEMP ON RILEVA
  CONDITION : (V.TEMP< N.TEMP) AND
  N.TEMP> (SELECT MAXTEMP FROM LUOGO WHERE NOME=N.NOME))
  ACTION : <MODIFICA LA TEMPERATURA>
  ```

  Per il caso (2.b)

  ```
  EVENT : AFTER UPDATE OF TEMP ON RILEVA
  CONDITION : (V.TEMP> N.TEMP) AND
  (V.DATA=(SELECT MAXDATA FROM LUOGO WHERE NOME=V.NOME))
  ACTION : <RICALCOLARE LA TEMP. MASSIMA PER V.NOME >
  ```

Una soluzione più semplice è quella di considerare tutti i casi con un unico trigger e ricalcolare sempre tutto (la condition si può togliere)

```
CREATE TRIGGER RILEVAMASSIMO_U
AFTER UPDATE OF TEMP, DATA ON RILEVA
REFERENCING NEW AS NUOVARIGA OLD AS VECCHIARIGA
FOR EACH ROW MODE DB2SQL
WHEN (VECCHIARIGA.DATA=(SELECT MAXDATA FROM LUOGO WHERE
                        NOME=VECCHIARIGA.NOME)
OR NUOVARIGA.TEMP> (SELECT MAXTEMP FROM LUOGO
                    WHERE NOME=NUOVARIGA.NOME))
UPDATE LUOGO
SET MAXTEMP= (SELECT MAX(TEMP) FROM RILEVA
              WHERE NOME=NUOVARIGA.NOME),
MAXDATA= (SELECT MAX(DATA) FROM RILEVA
          WHERE NOME=NUOVARIGA.NOME
AND TEMP = (SELECT MAX(TEMP) FROM RILEVA
            WHERE NOME=NUOVARIGA.NOME) )
WHERE NOME=NUOVARIGA.NOME!
```

- Dopo una generica modifica su RILEVA: se si vuole considerare anche la modifica del NOME in RILEVA, occorre ricalcolare la massima temperatura e la data sia per il vecchio NOME che per il nuovo NOME
  In questo caso togliamo la condizione

```
CREATE TRIGGER RILEVAMASSIMO_U
AFTER UPDATE ON RILEVA
REFERENCING NEW AS NUOVARIGA OLD AS VECCHIARIGA
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
UPDATE LUOGO
SET MAXTEMP= (SELECT MAX(TEMP) FROM RILEVA
              WHERE NOME=NUOVARIGA.NOME),
MAXDATA= (SELECT MAX(DATA) FROM RILEVA
          WHERE NOME=NUOVARIGA.NOME
AND TEMP = (SELECT MAX(TEMP) FROM RILEVA
            WHERE NOME=NUOVARIGA.NOME) )
WHERE NOME=NUOVARIGA.NOME;
UPDATE LUOGO
SET MAXTEMP= (SELECT MAX(TEMP) FROM RILEVA
              WHERE NOME=VECCHIARIGA.NOME),
MAXDATA= (SELECT MAX(DATA) FROM RILEVA
          WHERE NOME=VECCHIARIGA.NOME
AND TEMP = (SELECT MAX(TEMP) FROM RILEVA
            WHERE NOME=VECCHIARIGA.NOME))
WHERE NOME=VECCHIARIGA.NOME;
END!
```

Si noti che MAX() restituisce il valore NULL se applicata ad un insieme vuoto.

- Dopo una cancellazione su RILEVA: nel caso in cui la tupla eliminata era quella di massima temperatura occorre ricalcolare

```
CREATE TRIGGER RILEVAMASSIMO_D
AFTER DELETE ON RILEVA
REFERENCING OLD AS V
FOR EACH ROW MODE DB2SQL
WHEN (V.DATA=(SELECT MAXDATA FROM LUOGO WHERE NOME=V.NOME)
UPDATE LUOGO
SET MAXTEMP= (SELECT MAX(TEMP) FROM RILEVA WHERE NOME= V.NOME),
MAXDATA= (SELECT MAX(DATA) FROM RILEVA WHERE NOME= V.NOME
AND TEMP = (SELECT MAX(TEMP) FROM RILEVA WHERE NOME= V.NOME) )
WHERE NOME=V.NOME!
```

- **Uso di uno Statement Trigger**

```
CREATE TRIGGER RILEVAMASSIMO_I
AFTER INSERT ON RILEVA
REFERENCING NEW_TABLE AS NUOVATAB
FOR EACH STATEMENT MODE DB2SQL
WHEN
(0 < (SELECT COUNT(*) FROM LUOGO,NUOVATAB
      WHERE LUOGO.NOME=NUOVATAB.NOME
AND (LUOGO.MAXTEMP<NUOVATAB.TEMP OR LUOGO.MAXTEMP IS NULL) ))
UPDATE LUOGO
SET MAXTEMP= (SELECT MAX(TEMP) FROM RILEVA
              WHERE NOME=LUOGO.NOME),
MAXDATA= (SELECT MAX(DATA) FROM RILEVA WHERE NOME=LUOGO.NOME
AND TEMP = (SELECT MAX(TEMP) FROM RILEVA
            WHERE NOME=LUOGO.NOME) )
WHERE NOME IN (SELECT NOME FROM NUOVATAB)!
```

Esempio di TRIGGER in DB2 per le Cardinalità

- max-card(LUOGO,RILEVA)=10

Le operazioni che possono violare la cardinalità massima sono ovviamente l'inserimento in `RILEVA` e la modifica del nome in `RILEVA`. In entrambi i casi le operazioni che violano la cardinalità massima devono essere abortite: si usano dei before trigger.

```
CREATE TRIGGER CARDMAXLUOGO_I
NO CASCADE BEFORE INSERT ON RILEVA
REFERENCING NEW AS NUOVARIGA
FOR EACH ROW MODE DB2SQL
WHEN (10 <= (SELECT COUNT(*) FROM RILEVA
             WHERE NOME=NUOVARIGA.NOME))
SIGNAL SQLSTATE '70000'('INSERIMENTO IN RILEVA:
                         ERRORE CARDINALITA MASSIMA.')!

CREATE TRIGGER CARDMAXLUOGO_U
NO CASCADE BEFORE UPDATE OF NOME ON RILEVA
REFERENCING NEW AS NUOVARIGA
FOR EACH ROW MODE DB2SQL
WHEN (10 <= (SELECT COUNT(*) FROM RILEVA
             WHERE NOME=NUOVARIGA.NOME))
SIGNAL SQLSTATE '70000'('MODIFICA IN RILEVA:
                         ERRORE CARDINALITA MASSIMA.')!
```

Sarebbe conveniente un unico trigger con evento `INSERT ON RILEVA, UPDATE OF NOME ON RILEVA`.

- **Uso di uno Statement Trigger**

```
CREATE TRIGGER CARDMAXLUOGO_I
AFTER INSERT ON RILEVA
REFERENCING NEW_TABLE AS NUOVAT
FOR EACH STATEMENT MODE DB2SQL
WHEN (10 <=ANY (SELECT COUNT(*) FROM RILEVA,NUOVAT
                WHERE RILEVA.NOME=NUOVAT.NOME
                GROUP BY RILEVA.NOME))
SIGNAL SQLSTATE '70000'('INSERIMENTO IN RILEVA:
                         ERRORE CARDINALITA MASSIMA.')!
```

- min-card(LUOGO,RILEVA)=1 → Cancellazione

Prima della cancellazione su `RILEVA`, se il conteggio delle tuple relative a quel luogo ma in data differente è zero si genera un errore

```
DROP TRIGGER CARDMINRILEVA_D!
CREATE TRIGGER CARDMINRILEVA_D
NO CASCADE BEFORE DELETE ON RILEVA
REFERENCING OLD AS VECCHIARIGA
FOR EACH ROW MODE DB2SQL
WHEN (0 = (SELECT COUNT(*) FROM RILEVA
           WHERE NOME=VECCHIARIGA.NOME
AND DATA<>VECCHIARIGA.DATA))
SIGNAL SQLSTATE '70000'('CANCELLAZIONE IN RILEVA:
                         ERRORE CARDINALITA MINIMA.')!
```

Però questo modo non va bene perchè in `RILEVA` ci sono tutte le tuple (è un before trigger) quindi se ho due o più tuple, la cancellazione di tutte le tuple andrà sempre a buon fine. Allora viene effettuato un after trigger; si deve usare un Row Trigger (in quanto si deve controllare ciascun luogo separatamente)

```
DROP TRIGGER CARDMINRILEVA_D!
CREATE TRIGGER CARDMINRILEVA_D
AFTER DELETE ON RILEVA
REFERENCING OLD AS VECCHIARIGA
FOR EACH ROW MODE DB2SQL
WHEN (1 > (SELECT COUNT(*) FROM RILEVA
           WHERE NOME=VECCHIARIGA.NOME))
SIGNAL SQLSTATE '70000'('CANCELLAZIONE IN RILEVA:
                         ERRORE CARDINALITA MINIMA.')!
```

Problema: `RILEVA` ha `FOREIGN KEY(NOME) REFERENCES LUOGO ON DELETE CASCADE` quindi non si riesce più a cancellare, in quanto la cancellazione, propagandosi a cascata su tutte le tuple di `RILEVA`, fa sempre scattare il trigger `CARDMINRILEVA_D`. Allora si aggiunge alla condizione `VECCHIARIGA.NOME IN (SELECT NOME FROM LUOGO)`:

```
DROP TRIGGER CARDMINRILEVA_D!
CREATE TRIGGER CARDMINRILEVA_D
AFTER DELETE ON RILEVA
REFERENCING OLD AS VECCHIARIGA
FOR EACH ROW MODE DB2SQL
WHEN (1 > (SELECT COUNT(*) FROM RILEVA
           WHERE NOME=VECCHIARIGA.NOME) AND
VECCHIARIGA.NOME IN (SELECT NOME FROM LUOGO))
SIGNAL SQLSTATE '70000'('CANCELLAZIONE IN RILEVA:
                         ERRORE CARDINALITA MINIMA.')!
```

- min-card(LUOGO,RILEVA)=1 → Inserimento

In fase di inserimento su `LUOGO`, dato che `RILEVA` ha `FOREIGN KEY(NOME) REFERENCES LUOGO`, questo vincolo è soddisfatto *solo se* effettuiamo *contemporaneamente* almeno un inserimento anche in `RILEVA`: Il seguente trigger previene qualsiasi nuovo inserimento in `LUOGO`:

```
CREATE TRIGGER CARDMINLUOGO_I
AFTER INSERT ON LUOGO
REFERENCING NEW AS NUOVARIGA
FOR EACH ROW MODE DB2SQL
WHEN (1 > (SELECT COUNT(*) FROM RILEVA
           WHERE NOME=NUOVARIGA.NOME))
SIGNAL SQLSTATE '70000'('INSERIMENTO IN LUOGO:
                         ERRORE CARDINALITA MINIMA.')!
```

Per poter effettuare un inserimento *contemporaneo* sia in LUOGO che in RILEVA si potrebbe definire la seguente vista:

```
CREATE VIEW VISTALUOGO
AS SELECT LUOGO.NOME, DATA, TEMP
FROM LUOGO, RILEVA
WHERE LUOGO.NOME=RILEVA.NOME!
```

ed effettuare il seguente inserimento

```
INSERT INTO VISTALUOGO VALUES('CO',CURRENT DATE+18 DAYS,33)!
```

Però nella maggior parte dei sistemi (DB2 compreso) tutte le viste definite su un join sono considerate non aggiornabili e quindi il precedente inserimento fallisce.
Allora (1) si definisce una tabella *temporanea* al posto della vista

```
CREATE TABLE TEMPLUOGO(NOME VARCHAR(5) NOT NULL PRIMARY KEY,
DATA DATE NOT NULL, TEMP INTEGER)!
```

(2) l'inserimento viene fatto nella tabella *temporanea*

```
INSERT INTO TEMPLUOGO VALUES('CO',CURRENT DATE+18 DAYS,33)!
```

(3) e viene propagato alle tabelle interessate tramite un trigger

```
CREATE TRIGGER INSER_TEMP
AFTER INSERT ON TEMPLUOGO
REFERENCING NEW AS N
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
INSERT INTO LUOGO(NOME) VALUES(N.NOME);
INSERT INTO RILEVA VALUES(N.NOME,N.DATA,N.TEMP);
DELETE FROM TEMPLUOGO WHERE NOME=N.NOME;
END!
```

Si noti che in presenza del trigger CARDMINLUOGO_I l'inserimento:

```
INSERT INTO TEMPLUOGO VALUES('CO',CURRENT DATE+18 DAYS,33)!
```

non viene eseguito; allora:

```
DROP TRIGGER CARDMINLUOGO_I!
CREATE TRIGGER CARDMINLUOGO_I
AFTER INSERT ON LUOGO
REFERENCING NEW AS NUOVARIGA
FOR EACH ROW MODE DB2SQL
WHEN ((1 > (SELECT COUNT(*) FROM RILEVA
            WHERE NOME=NUOVARIGA.NOME))
AND (1 > (SELECT COUNT(*) FROM TEMPLUOGO
          WHERE NOME=NUOVARIGA.NOME)))
SIGNAL SQLSTATE '70000'('INSERIMENTO IN LUOGO:
                         ERRORE CARDINALITA MINIMA.')!
```

In definitiva, posso inserire in LUOGO solo passando tramite TEMPLUOGO.

- min-card(LUOGO,RILEVA)=1 → Modifica su NOME (key) di LUOGO

Essendo NOME la chiave, la modifica su LUOGO deve preservare tale proprietà, quindi ha senso considerare solo la rinomina di un certo luogo con un nuovo nome non già esistente nella tabella LUOGO Se in RILEVA mettiamo

```
FOREIGN KEY(NOME) REFERENCES LUOGO ON UPDATE CASCADE
```

allora la modifica del nome conserva la consistenza del database. Però in DB2 l'opzione ON UPDATE CASCADE non c'è. Riprendiamo:

```
CREATE TABLE RILEVA(NOME VARCHAR(5) NOT NULL ,
DATA DATE NOT NULL,
TEMP INTEGER,
CONSTRAINT CHIAVERILEVA PRIMARY KEY(NOME,DATA),
FOREIGN KEY(NOME) REFERENCES LUOGO ON DELETE CASCADE)!
```

e discutiamo l'implementazione dell'opzione ON UPDATE CASCADE: non si può usare un after trigger

```
CREATE TRIGGER CASCADELUOGO_U
AFTER UPDATE OF NOME ON LUOGO
REFERENCING NEW AS NUOVARIGA OLD AS VECCHIARIGA
FOR EACH ROW MODE DB2SQL
WHEN (0 < (SELECT COUNT(*) FROM RILEVA
           WHERE NOME=VECCHIARIGA.NOME))
UPDATE RILEVA
SET NOME=NUOVARIGA.NOME
WHERE NOME=VECCHIARIGA.NOME!
```

perchè la FOREIGN KEY(NOME) REFERENCES LUOGO fa comunque fallire la modifica. Non si può usare un before trigger, in quanto in esso non sono consentite modifiche di altre tabelle. Una possibile soluzione è quella di eliminare la
FOREIGN KEY(NOME) REFERENCES LUOGO e realizzare tutto tramite trigger.

Esempio di BEFORE TRIGGER in DB2

```
FIUME( CODLUOGO ,LUNGHEZZA, PROFONDITA)
NASCE ( FIUME , MONTE)
      FK: FIUME REFERENCES FIUME
      FK: MONTE REFERENCES MONTE
AFFLUENTE( FIUMEAFFLUENTE , FIUMEMAGGIORE)
      FK: FIUMEAFFLUENTE REFERENCES FIUME
      FK: FIUMEMAGGIORE REFERENCES FIUME
EMISSARIO ( FIUME , LAGO)
      FK: FIUME REFERENCES FIUME
      FK: LAGO REFERENCES LAGO
IMMISSARIO ( FIUME , LAGO)
      FK: FIUME REFERENCES FIUME
      FK: LAGO REFERENCES LAGO
```

Riportiamo i trigger per implementare i seguenti vincoli:

- Un fiume non può contemporaneamente nascere da un monte e da un lago
- Un fiume non può contemporaneamente sfociare in un fiume e in un lago

```
CREATE TRIGGER BINASCE
NO CASCADE BEFORE INSERT ON NASCE REFERENCING NEW AS NUOVO
FOR EACH ROW MODE DB2SQL
WHEN ( NUOVO.FIUME IN ( SELECT FIUME FROM EMISSARIO ) )
SIGNAL SQLSTATE '70000'('IMPOSSIBILE INSERIRE.IL FIUME E' GIA'
                        EMISSARIO')!

CREATE TRIGGER BIEMISSARIO
NO CASCADE BEFORE INSERT ON EMISSARIO REFERENCING NEW AS NUOVO
FOR EACH ROW MODE DB2SQL
WHEN( NUOVO.FIUME IN ( SELECT FIUME FROM NASCE ) )
SIGNAL SQLSTATE '70000'('IMPOSSIBILE INSERIRE.IL FIUME NASCE DA
                        UN MONTE')!

CREATE TRIGGER BIAFFLUENTE
NO CASCADE BEFORE INSERT ON AFFLUENTE REFERENCING NEW AS NUOVO
FOR EACH ROW MODE DB2SQL
WHEN ( NUOVO.FIUMEAFFLUENTE IN ( SELECT FIUME FROM IMMISSARIO ))
SIGNAL SQLSTATE '70000'('IMPOSSIBILE INSERIRE.IL FIUME E' GIA'
                        IMMISSARIO')!

CREATE TRIGGER BIIMMISSARIO
NO CASCADE BEFORE INSERT ON IMMISSARIO REFERENCING NEW AS NUOVO
FOR EACH ROW MODE DB2SQL
WHEN (NUOVO.FIUME IN(SELECT FIUMEAFFLUENTE FROM AFFLUENTE ))
SIGNAL SQLSTATE '70000'('IMPOSSIBILE INSERIRE.IL FIUME E' GIA'
                        AFFLUENTE')!
```

Esempio di TRIGGER in DB2 per una Gerarchia

```
DROP TABLE A!
CREATE TABLE A(K INTEGER NOT NULL PRIMARY KEY ,A INTEGER)!
DROP TABLE B!
CREATE TABLE B(K INTEGER NOT NULL PRIMARY KEY ,B INTEGER,
CONSTRAINT FKTO_A FOREIGN KEY(K) REFERENCES A
                  ON DELETE CASCADE)!
DROP TABLE D!
CREATE TABLE D(K INTEGER NOT NULL PRIMARY KEY ,D INTEGER,
CONSTRAINT FKTO_A FOREIGN KEY(K) REFERENCES A
                  ON DELETE CASCADE)!
```

- Gerarchia Esclusiva → Inserimento nelle sottoclassi

```
CREATE TRIGGER D_ESCL_I
NO CASCADE BEFORE INSERT ON D
REFERENCING NEW AS N
FOR EACH ROW MODE DB2SQL
WHEN (N.K IN (SELECT K FROM B))
SIGNAL SQLSTATE '70000' ('ERRORE: ELEMENTO E IN B.')!
CREATE TRIGGER B_ESCL_I
NO CASCADE BEFORE INSERT ON B
REFERENCING NEW AS N
FOR EACH ROW MODE DB2SQL
WHEN (N.K IN (SELECT K FROM D))
SIGNAL SQLSTATE '70000' ('ERRORE: ELEMENTO E IN D.')!
```

- Gerarchia Totale → Cancellazione dalle sottoclassi

```
CREATE TRIGGER A_GER_TOTALE_D_D
AFTER DELETE ON D
REFERENCING OLD_TABLE AS OT
FOR EACH STATEMENT MODE DB2SQL
DELETE FROM A WHERE A.K IN (SELECT K FROM OT)!
```

Analogo trigger `AFTER DELETE ON D`

- Gerarchia Totale → Inserimento nella superclasse

Come già discusso nel caso di cardinalità minima, in fase di inserimento sulla superclasse, è necessario effettuare *contemporaneamente* un inserimento anche in una delle sottoclassi (in una sola: supponiamo che sia anche esclusiva) Si usa una tabella temporanea con tutti gli attributi, con questa convenzione:

```
B NOT NULL, D NULL --> INSERIRE IN A, B
B NULL, D NOT NULL --> INSERIRE IN A, D
```

```
        B NOT NULL, D NOT NULL --> NON INSERIRE
        B NULL, D NULL --> NON INSERIRE

CREATE TABLE TEMP_A(
        K INTEGER NOT NULL PRIMARY KEY,
        A INTEGER,
        B INTEGER,
        D INTEGER)!

DROP TRIGGER A_GER_TOTALE_B_I!
CREATE TRIGGER A_GER_TOTALE_B_I
AFTER INSERT ON TEMP_A
REFERENCING NEW AS N
FOR EACH ROW MODE DB2SQL
WHEN (N.B IS NOT NULL AND N.D IS NULL)
BEGIN ATOMIC
INSERT INTO A VALUES(N.K,N.A);
INSERT INTO B VALUES(N.K,N.B);
DELETE FROM TEMP_A WHERE K=N.K;
END!

DROP TRIGGER A_GER_TOTALE_D_I!
CREATE TRIGGER A_GER_TOTALE_D_I
AFTER INSERT ON TEMP_A
REFERENCING NEW AS N
FOR EACH ROW MODE DB2SQL
WHEN (N.B IS NULL AND N.D IS NOT NULL)
BEGIN ATOMIC
INSERT INTO A VALUES(N.K,N.A);
INSERT INTO D VALUES(N.K,N.D);
DELETE FROM TEMP_A WHERE K=N.K;
END!

DROP TRIGGER A1!
CREATE TRIGGER A1
AFTER INSERT ON A
REFERENCING NEW AS N
FOR EACH ROW MODE DB2SQL
WHEN ((N.K NOT IN (SELECT K FROM B)) AND
(N.K NOT IN (SELECT K FROM D))
AND (N.K NOT IN (SELECT K FROM TEMP_A)))
SIGNAL SQLSTATE '70000'('INSERIMENTO IN A: ERRORE.')!
```

Trigger in cascata

♢ Quando l'esecuzione di una azione di un trigger T1 provoca l'attivazione di un altro trigger T2 si puo procedere in due modi:

- interrompere l'esecuzione di T1 per processare T2 (come in DB2,INGRES)
- concludere l'esecuzione di T1 e quindi processare l'altro trigger T2

Trigger ricorsivi L'esecuzione di un trigger T1 provoca, direttamente o indirettamente, l'attivazione dello stesso trigger T1

♢ In INGRES e DB2 le attivazioni ricorsive sono permesse e la loro terminazione e controllata run-time: se si eccede il massimo numero di attivazioni in cascata consentito (che, per default, e uguale a 20 in INGRES e 16 in DB2) si genera un errore che provoca l'abort di tutte le azioni effettuate fino a quel punto. In altri sistemi, quali ORACLE, le attivazioni ricorsive non sono permesse.

```
CREATE TABLE PERSON(
        NOME VARCHAR(5) NOT NULL PRIMARY KEY,
        SALARIO INTEGER )!

CREATE TRIGGER INC_SALAR_ROCCO
AFTER UPDATE OF SALARIO ON PERSON
REFERENCING NEW AS N
FOR EACH ROW MODE DB2SQL
WHEN ((N.SALARIO<5000) AND (N.NOME='ROCCO'))
UPDATE PERSON
SET SALARIO=1.5*N.SALARIO
WHERE NOME=N.NOME!

CREATE TRIGGER INC_SALAR_CIRO
AFTER UPDATE OF SALARIO ON PERSON
REFERENCING NEW AS N
FOR EACH ROW MODE DB2SQL
WHEN ((N.SALARIO<5000) AND (N.NOME='CIRO'))
BEGIN ATOMIC
UPDATE PERSON SET SALARIO=N.SALARIO WHERE NOME='ROCCO';
UPDATE PERSON
SET SALARIO = (SELECT SALARIO FROM PERSON WHERE NOME='ROCCO')
WHERE NOME='CIRO';
END!
```

♢ L'esecuzione del trigger INC_SALAR_CIRO è interrotta per eseguire INC_SAL_ROCCO; Ci possono essere problemi di esecuzione ...

Esempio: Trigger in cascata

```
CREATE TABLE MANAG(C INTEGER NOT NULL PRIMARY KEY ,
CM INTEGER REFERENCES MANAG, NSUB INTEGER)!
```

C	CM	NSUB
1	3	0
2	3	0
3	4	2
4	5	3
5		4

L'attributo NSUB è un dato derivato che indica, per un certo Manager, il numero di manager che esso dirige, ad ogni livello.
I seguenti trigger servono per calcolare e mantenere aggiornato il dato derivato:

```
CREATE TRIGGER MANAG_I AFTER INSERT ON MANAG
REFERENCING NEW AS N
FOR EACH ROW MODE DB2SQL
UPDATE MANAG
SET NSUB=NSUB+1
WHERE C=N.CM!

CREATE TRIGGER MANAG_D AFTER DELETE ON MANAG
REFERENCING OLD AS O
FOR EACH ROW MODE DB2SQL
UPDATE MANAG
SET NSUB=NSUB-1
WHERE C=O.CM!

CREATE TRIGGER MANAG_U AFTER UPDATE OF CM ON MANAG
REFERENCING OLD AS O NEW AS N
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
UPDATE MANAG
SET NSUB=NSUB+1
WHERE C=N.CM;

CREATE TRIGGER MANAG_P AFTER UPDATE OF NSUB ON MANAG
REFERENCING OLD AS O NEW AS N
FOR EACH ROW MODE DB2SQL
UPDATE MANAG
SET NSUB=NSUB+N.NSUB - O.NSUB
WHERE C=N.CM!
```

Quest'ultimo trigger serve per propagare la modifica a tutti i livelli; esso viene quindi attivato ricorsivamente e quindi può abortire se si ha un numero di attivazioni maggiore di 16.

Capitolo 5

Teoria della Normalizzazione

Questo capitolo riassume la teoria della normalizzazione e le tecniche di ragionamento sulle dipendenze funzionali sviluppate negli anni '70 dalla comunità di ricerca delle basi di dati. Una trattazione più approfondita di tali temi si può trovare in [37, 36, 6].

Il capitolo inizia introducendo il concetto di *dipendenza funzionale* tramite il quale è possibile modellare in maniera astratta le dipendenze tra dati. Vengono quindi presentate le principali tecniche per ragionare sulle dipendenze funzionali, ovvero per dedurre dalle dipendenze funzionali definite esplicitamente dal progettista della base di dati quelle implicate logicamente.

Vengono poi definite le proprietà di qualità di uno schema di relazione introducendo il concetto di *forma normale*. Uno schema di relazione che violi la forma normale desiderata può essere decomposto in sottoschemi allo scopo di raggiungere tale forma normale; si discuteranno le proprietà di tali decomposizioni, quali la preservazione dei dati e la preservazione delle dipendenze funzionali.

5.1 Dipendenze Funzionali

♢ Uno degli elementi di conoscenza a disposizione del progettista che deve modellare un'applicazione database è che esistono delle "dipendenze tra dati". In maniera astratta, questo viene modellato dal concetto di "dipendenza funzionale".

Convenzioni di Notazione :

1. Le prime lettere dell'alfabeto maiuscole (A, B, C, D, E) denotano un attributo;
2. Le ultime lettere dell'alfabeto maiuscole (U, V, X, Y, Z) denotano insieme di attributi;
3. R denota lo schema di una relazione, r l'istanza corrente dello schema R.
4. Dato un insieme di atttributi $A_1A_2 \dots A_n$, uno schema di relazione R avente tale insieme di attributi verrà denotato indifferentemente con $R(A_1A_2 \dots A_n)$ oppure con $R(A_1, A_2, \dots, A_n)$.

Definizione 1 (Dipendenza Funzionale) *Dato uno schema di relazione $R(X)$, una* dipendenza funzionale *(*FD*) su R è un vincolo di integrità espresso nella forma $Y \to Z$, dove Y e Z sono sottoinsiemi di X; in tal caso si dice che Y* determina funzionalmente *Z.*

♢ Un'istanza r di R soddisfa la dipendenza funzionale $Y \to Z$ se in ogni tupla di r il valore di Y determina univocamente il valore di Z. Formalmente, dato uno schema di relazione $R(T)$, un'istanza r di $R(T)$ soddisfa il vincolo di dipendenza funzionale $Y \to Z$ su $R(T)$ se e solo se

$$\forall t_1, t_2 \in r, t_1[Y] = t_2[Y] \to t_1[Z] = t_2[Z]$$

♢ È facile verificare che se $Y \subseteq Z$ allora $Y \to Z$. Queste dipendenze funzionali vengono dette **banali**.

♢ Uno schema $R(T)$ su cui è definito un insieme F di FD verrà denotato con $\langle R(T), F \rangle$.

Definizione 2 (Istanza Legale) *Dato uno schema $\langle R(T), F \rangle$, un'istanza r di $R(T)$ è detta* istanza legale *di $\langle R(T), F \rangle$ se soddisfa tutte le dipendenze funzionali in F.*

Dipendenze Funzionali implicate logicamente

♢ Uno schema $\langle R(T), F\rangle$ dà luogo a istanze legali che oltre alle *FD* in F *soddisfano necessariamente altre FD.*

Esempio : In ogni istanza legale dello schema $\langle R(ABC), F = A \to B, B \to C\rangle$, vale anche la dipendenza funzionale $A \to C$ (proprietà *transitiva*).

Infatti, non possono esistere due tuple $t1, t2 \in r$ tali che $t1[A] = t2[A]$ e $t1[C] \neq t2[C]$, in quanto se $t1[A] = t2[A]$, per la $A \to B$, si ha che $t1[B] = t2[B]$, e quindi per la $B \to C$ si deve avere necessariamente $t1[C] = t2[C]$.

Definizione 3 (Implicazione Logica) *Dato uno schema $\langle R(T), F\rangle$ e una dipendenza funzionale $X \to Y$, si dice che F* implica logicamente *$X \to Y$, denotata con $F \models X \to Y$, se ogni istanza legale r di $\langle R(T), F\rangle$ soddisfa anche $X \to Y$.*

Definizione 4 (Chiusura di un insieme di FD) *Dato uno schema $\langle R(T), F\rangle$, la* chiusura di *F, denotato con F^+, è l'insieme delle dipendenze funzionali implicate logicamente da F:*

$$F^+ = \{X \to Y \mid F \models X \to Y\}$$

♢ Usando la nozione di dipendenza funzionale diamo una nuova formulazione del concetto di *chiave* e *superchiave*. Si vedrà nel seguito come sarà possibile dotarsi di strumenti effettivi di verifica che un insieme di attributi sia una chiave o una superchiave di uno schema di relazione.

Definizione 5 (Chiave) *Dato $\langle R(T), F\rangle$, un insieme $K \subseteq T$ è detto* chiave *di $\langle R(T), F\rangle$ se*

1. *$K \to T$ è in F^+;*
2. *non esiste nessun sottinsieme proprio Y di K tale che $Y \to T$ è in F^+.*

♢ Con il termine **superchiave** denotiamo un qualsiasi soprainsieme di una chiave.

5.2 Ragionamento sulle dipendenze funzionali

♢ Per determinare le FD che sono logicamente implicate da quelle esplicite si usano regole di inferenza, note come assiomi di Armstrong [4].

Definizione 6 (Assiomi di Armstrong) *Dato uno schema $\langle R(T), F\rangle$, siano X, Y e Z insiemi di attributi contenuti in T. Le regole di inferenza sono:*

Riflessività $\{Y \subseteq X\} \vdash X \to Y$

Estensione $\{X \to Y\} \vdash XZ \to YZ$

Transitività $\{X \to Y, Y \to Z\} \vdash X \to Z$

♢ La derivazione sintattica di f da un insieme F di *FD* attraverso l'applicazione degli assiomi di Armstrong si indica con $F \vdash f$.

Esempio : Sia dato lo schema $\langle R(ABCD), F = \{A \to C, B \to D\}\rangle$; la *FD* $AB \to ABCD$ è derivabile da F attraverso gli assiomi:

1	$A \to C$	data
2	$AB \to ABC$	applicazione della estensione alla **1** con AB
3	$B \to D$	data
4	$ABC \to ABCD$	applicazione della estensione alla **3** con ABC
5	$AB \to ABCD$	applicazione della transitività alla **2** e **4**

♢ Nella derivazione di una FD da un insieme di dipendenze si possono usare, oltre agli assiomi di Armstrong, alcune regole di inferenza addizionali:

Unione : $\{X \to Y, X \to Z\} \vdash X \to YZ$

1	$X \to Y$	data
2	$X \to XY$	applicazione della estensione alla **1** con X
3	$X \to Z$	data
4	$XY \to YZ$	applicazione della estensione alla **3** con Y
5	$X \to YZ$	applicazione della transitività alla **2** e **4**

Pseudotransitività : $\{X \to Y, WY \to Z\} \vdash XW \to Z$

1	$X \to Y$	data
2	$XW \to WY$	applicazione della estensione alla **1** con W
3	$WY \to Z$	data
4	$XW \to Z$	applicazione della transitività alla **2** e **3**

Decomposizione : $\{X \to Y, Z \subseteq Y\} \vdash X \to Z$

1	$X \to Y$	data
2	$Y \to Z$	applicazione della riflessività alla $Z \subseteq Y$
3	$X \to Z$	applicazione della transitività alla **1** e **2**

♢ Dato un insieme di attributi $Y = A_1A_2 \ldots A_n$, una conseguenza notevole è che dalla regola di unione: $\{X \to Y\} \vdash \{X \to A_1, X \to A_2, \ldots, X \to A_n\}$, e da quella di decomposizione: $\{X \to A_1, X \to A_2, \ldots, X \to A_n\} \vdash \{X \to Y\}$.

Ragionamento sulle dipendenze funzionali

♢ Informalmente, dato uno schema $\langle R(T), F\rangle$, la chiusura di un insieme di attributi $X \subseteq T$ è l'insieme di attributi derivabili da F mediante gli assiomi di Armstrong.

Definizione 7 (Chiusura di un insieme di attributi) *Dato $\langle R(T), F\rangle$, sia $X \subseteq T$ un insieme di attributi; la chiusura di X rispetto a F, denotata con X^+, è l'insieme di attributi $A \in T$, tali che $X \to A$ deriva da F tramite gli assiomi di Armstrong:*

$$X^+ = \{A \in T \mid F \vdash X \to A\}$$

Esempio :

Dato $\langle R(ABCDE), F = \{A \to B, B \to C, CD \to E\}\rangle$, la chiusura di AD è $(AD)^+ = ABCDE$:

1 $AD \to A$ applicazione della riflessività
2 $AD \to B$ applicazione della transitività alla **1** e $A \to B$
3 $AD \to C$ applicazione della transitività alla **2** e $B \to C$
4 $AD \to D$ applicazione della riflessività
5 $AD \to E$
  5a $AD \to CD$ applicazione dell'unione alla **3** e **4**
  5b $AD \to E$ applicazione della transitività alla **5a** e $CD \to E$

♢ Il prossimo teorema stabilisce che il problema di determinare se $X \to A$ è derivabile, tramite gli assiomi di Armstrong, da un insieme F di FD è risolvibile calcolando la chiusura dell'insieme X.

Teorema 1 *Dato $\langle R(T), F\rangle$, siano X e Y insiemi di attributi contenuti in T,*

$$F \vdash X \to Y \quad \Leftrightarrow \quad Y \subseteq X^+$$

Dimostrazione : Supponiamo che sia $Y = A_1 A_2 \dots A_n$,

$F \vdash X \to Y \Rightarrow Y \subseteq X^+$: Se $F \vdash X \to Y$, allora per la regola di decomposizione si ha che, per ogni i, $F \vdash X \to A_i$, quindi $A_i \in X^+$ e pertanto $Y \subseteq X^+$.

$F \vdash X \to Y \Leftarrow Y \subseteq X^+$: Se $Y \subseteq X^+$, per definizione si ha che, per ogni i, $F \vdash X \to A_i$, quindi per la regola dell'unione $F \vdash X \to Y$.

Ragionamento sulle dipendenze funzionali

♢ Un semplice algoritmo (**XPIU**) per il calcolo di X^+ può essere definito:

Algoritmo XPIU

- **Input :** $\langle R(T), F\rangle$, con $F = \{V_1 \to W_1, \ldots, V_n \to W_n\}$ e $X \subseteq T$
- **Output :** **XPIU**(X, F), chiusura di X rispetto a F

begin
  XPIU := X ;
  repeat
    for $k := 1$ **to** n **do**
      if $V_k \to W_k \in F$ **and** $V_k \subseteq$ **XPIU** **and** $W_k \not\subseteq$ **XPIU**
        then XPIU := **XPIU** $\cup W_k$;
  until (XPIU non è cambiato);
end.

♢ Si può dimostrare che l'algoritmo **XPIU** termina e calcola correttamente X^+. Inoltre, l'algoritmo ha una buona complessità proporzionale alla lunghezza di tutte le dipendenze funzionali in F.

Esempio :

Dato $\langle R(ABCDE), F = \{A \to B, B \to C, CD \to E\}\rangle$, sia $X = AD$:

XPIU$^0 = AD$

XPIU^1 = **XPIU**$^0 \cup B = ADB$, in quanto $A \subseteq$ **XPIU**$^0 \wedge A \to B \in F$

XPIU^2 = **XPIU**$^1 \cup C = ADBC$, in quanto $B \subseteq$ **XPIU**$^1 \wedge B \to C \in F$

XPIU^3 = **XPIU**$^2 \cup E = ADBCE$, in quanto $CD \subseteq$ **XPIU**$^2 \wedge CD \to E \in F$

XPIU^4 = **XPIU**$^3 = X^+$

quindi **XPIU**$(AD, F) = ABCDE$.

Ragionamento sulle dipendenze funzionali

♢ Il prossimo teorema stabilisce che la derivazione sintattica $\vdash$ è equivalente all'implicazione logica $\models$.

Teorema 2 (Correttezza e completezza degli assiomi di Armstrong)

$F \vdash f \Rightarrow F \models f$ **:** gli assiomi di Armstrong sono **corretti**, cioè ogni FD derivata attraverso l'applicazione degli assiomi è implicata logicamente

$F \vdash f \Leftarrow F \models f$ **:** gli assiomi di Armstrong sono **completi**, cioè ogni FD implicata logicamente è derivabile attraverso l'applicazione degli assiomi

Dimostrazione

$F \vdash f \Rightarrow F \models f$ **:** la dimostrazione di correttezza è immediata. Verifichiamo, ad esempio, la correttezza della regola di Estensione, ovvero che, in ogni istanza r, se è valida $X \rightarrow Y$, vale anche $XZ \rightarrow YZ$. Infatti, non possono esistere due tuple $t1, t2 \in r$ tali che $t1[XZ] = t2[XZ]$ e $t1[YZ] \neq t2[YZ]$, in quanto se $t1[XZ] = t2[XZ]$, per la $X \rightarrow Y$, si ha che $t1[Y] = t2[Y]$, e quindi si deve avere necessariamente $t1[YZ] = t2[YZ]$.

$F \vdash f \Leftarrow F \models f$ **:** questo equivale a dimostrare che $F \nvdash f \Rightarrow F \not\models f$, ovvero, che se f non deriva da F tramite gli assiomi di Armstrong allora f non è implicata logicamente da F, cioè esiste un'istanza legale r di $\langle R(T), F\rangle$ in cui f non è soddisfatta.

Dato $\langle R(T), F\rangle$, supponiamo che $F \nvdash X \rightarrow Y$ cioè che dagli assiomi non si possa derivare $X \rightarrow Y$. L'istanza legale r in cui $X \rightarrow Y$ non è soddisfatta è costituita da due sole tuple che hanno gli stessi valori per tutti gli attributi in X^+ e valori differenti per gli altri attributi $(T - X^+)$:

$\overbrace{\text{Attributi di } X^+}$				$\overbrace{\text{Attributi di } T - X^+}$			
1	1	...	1	1	1	...	1
1	1	...	1	0	0	...	0

Ora occorre dimostrare r è una istanza legale e che $X \rightarrow Y$ non è soddisfatta in r.

Per dimostrare che r è una istanza legale di $\langle R(T), F\rangle$, si procede per assurdo, supponendo che esista $V \rightarrow W \in F$ non soddisfatta da r. Allora deve essere $V \subseteq X^+$ e $W \not\subseteq X^+$. Poichè $V \subseteq X^+$, si ha che $F \vdash X \rightarrow V$, e per transitività si ottiene che $F \vdash X \rightarrow W$. Di conseguenza $W \subseteq X^+$, e questo contraddice l'ipotesi. Pertanto r è legale.

Per dimostrare che $X \rightarrow Y$ non è soddisfatta in r, basta osservare che per l'ipotesi che $F \nvdash X \rightarrow Y$, si ha che $Y \not\subset X^+$ e quindi $X \rightarrow Y$non è soddisfatta da r.

Ragionamento sulle dipendenze funzionali

♢ Una importante conseguenza del teorema 2 è quella di poter calcolare F^+ in base alla seguente definizione: $F^+ = \{X \rightarrow Y \mid F \vdash X \rightarrow Y\}$

♢ Non è proponibile generare la chiusura F^+, dato l'elevato numero di *FD* in essa contenute (esponenziale nel numero di attributi).
Si preferisce quindi trovare una soluzione efficiente al problema di determinare se una FD $X \rightarrow A$ appartiene a F^+.
Tale problema è riconducibile, in base al Teorema 1, al calcolo della chiusura di un insieme di attributi: $X \rightarrow A \in F^+$ **se e solo se** $A \in \mathbf{XPIU}(X, F)$

♢ Pertanto, tramite **XPIU** si possono risolvere i problemi legati alle implicazioni di dipendenze funzionali che stanno alla base delle proprietà di normalizzazione di schemi di relazione: *test di superchiave*, *sintesi di una chiave*, *attributi primi*, *FD ridondanti* e *equivalenza di insiemi di FD.*

- **Test di superchiave** : K è superchiave di $\langle R(T), F\rangle$ **sse** $\mathbf{XPIU}(K, F) = T$;

- **Sintesi di una chiave** : Sia K una superchiave di $\langle R(T), F\rangle$; il seguente algoritmo sintetizza una chiave $X \subseteq K$ di $\langle R(T), F\rangle$.

  Algoritmo KEY
  - **Input:** $\langle R(T), F\rangle$, e $X \subseteq T$, con $X = A_1 \ldots A_n$ superchiave di $R(T)$
  - **Output:** KEY, chiave di $\langle R(T), F\rangle$ contenuta in X

  begin
  KEY := X **;**
  for $i := 1$ **to** n **do**
  if $A_i \in \mathbf{XPIU}(\mathsf{KEY} - A_i, F)$
  then $\mathsf{KEY} = \mathsf{KEY} - A_i$**;**
  end.

- **Attributo primo** : Verificare se l'attributo appartiene ad una delle chiavi calcolate con l'algoritmo KEY.

- **FD ridondanti** : Dato $\langle R(T), F\rangle$, una FD $X \rightarrow Y \in F$ è *ridondante* se è implicata logicamente dalle altre. In termini di **XPIU**: $Y \in \mathbf{XPIU}(X, F - \{X \rightarrow Y\})$.

- **Equivalenza di insiemi di FD** : Dato $R(T)$, due insiemi di dipendenze funzionali F, G definiti su $R(T)$ sono *equivalenti* se e solo se $F^+ = G^+$.

  Teorema 3 *Dato $R(T)$, due insiemi di dipendenze funzionali F, G definiti su $R(T)$ sono* equivalenti *se e solo se $F \subseteq G^+$ e $G \subseteq F^+$. In termini di **XPIU**: per ogni $X \rightarrow Y \in F$ si ha $Y \in \mathbf{XPIU}(X, G)$ e per ogni $X \rightarrow Y \in G$ si ha $Y \in \mathbf{XPIU}(X, F)$.*

  Dimostrazione : Si veda [37] oppure [14].

5.3 Forme Normali

♢ Le Forme Normali sono alla base della *teoria della normalizzazione* di schemi relazionali, il cui obiettivo principale è quello di definire formalmente la *qualità degli schemi*.

♢ La qualità di uno schema viene essenzialmente definita come l'**assenza** di:

- **ridondanza nei dati**
- **anomalie di aggiornamento dei dati.**

Prima Forma Normale - 1NF : La 1NF nasce in realtà dall'esigenza di introdurre un modello dei dati particolarmente semplice, *piatto*, che sta alla base di semplici linguaggi di interrogazione e manipolazione.

Definizione 8 (1NF) *Uno schema $R(X)$ è in **1NF** se e solo se i valori di tutti i domini degli attributi $A \in X$ sono atomici.*

♢ Normalmente, nel modello relazionale, si assume come implicito il vincolo di avere domini atomici per gli attributi.

♢ In altri modelli di dati, quali ad esempio i *modelli ad oggetti*, *modelli relazionali estesi* e *modelli relazionali ad oggetti*, il vincolo di avere domini atomici è rimosso e un attributo può avere come valore, oltre ad un valore elementare, anche un valore complesso, quale ad esempio una tupla, un insieme oppure una loro combinazione.

♢ **Esempio:** Consideriamo lo schema di relazione
ESAMI(MATR,CODCOR,VOTO,GIORNO,MESE,ANNO) e la sua istanza

MATR	CODCOR	VOTO	GIORNO	MESE	ANNO
Matr1	CodCor1	29	10	Luglio	1994
Matr1	CodCor3	24	15	Giugno	1995
Matr2	CodCor1	27	21	Febbraio	1997
Matr2	CodCor2	30	12	Luglio	1998

allora, in un modello relazionale esteso, la stessa informazione potrebbe essere espressa come:

MATR	ESAMI
	set of (CODCOR,VOTO,(GIORNO,MESE,ANNO))
Matr1	{ (CodCor1,29,(10,Luglio,1994)), (CodCor3,24,(15,Giugno,1995)) }
Matr2	{ (CodCor1,27,(21,Febbraio,1997)), (CodCor2,30,(12,Luglio,1998)), }

Seconda Forma Normale (2NF)

♢ **Esempio:** Consideriamo lo schema di relazione FREQUENZA

FREQUENZA(MATR,CODCOR,NUMEROORE,CODDOC)

e la dipendenza funzionale CODCOR → CODDOC

MATR	CODCOR	NUMEROORE	CODDOC
Matr1	CodCor1	102	CodDoc1
Matr1	CodCor3	98	CodDoc1
Matr2	CodCor1	111	CodDoc1
Matr2	CodCor2	101	CodDoc2

ridondanza: in tutte le frequenze di un corso si ripete lo stesso docente

anomalia di modifica: se un corso cambia docente si devono modificare tutte le tuple relative alla frequenza di quel corso

anomalia di inserimento: non si può inserire un corso senza inserire almeno uno studente che frequenta

anomalia di cancellazione: se vengono cancellate tutte le frequenze relative ad un corso si perde anche l'informazione sul docente del corso

- I problemi derivano da CODCOR → CODDOC che stabilisce la dipendenza di un attributo (CODDOC) **solo da una parte** (CODCOR) della chiave. Questo esempio è una violazione della Seconda Forma Normale.

♢ Informalmente, la 2NF serve per definire schemi esenti dai problemi derivanti dalla dipendenza *parziale* di un attributo dalla chiave.

> Nelle definizioni successive di forme normali si assume sempre, senza perdita di generalità, uno schema $\langle R(T), F\rangle$, dove F contiene solo dipendenze funzionali del tipo $X \to A$.

♢ Dato uno schema $\langle R(T), F\rangle$, un attributo $A \in T$ e un insieme di attributi $Y \subseteq T$, si dice che A *dipende completamente* da Y se $Y \to A \in F$ e non esiste nessun sottoinsieme proprio Z di Y, $Z \subset Y$, tale che $Z \to A$.

Definizione 9 (2NF) *Uno schema $\langle R(T), F\rangle$ è in **2NF** se e solo se ogni attributo non primo $A \in T$* dipende completamente *da ognuna delle chiavi di R.*

> In principio, le definizioni di forme normali dovrebbero essere date con riferimento a F^+, cioè anche alle dipendenze implicate logicamente, ma è possibile dimostrare che è sufficiente controllare la non violazione in F per garantirsi la non violazione in F^+.

Terza Forma Normale (3NF)

♢ **Esempio :** Consideriamo lo schema di relazione `CORSO`

`CORSO(CODCOR,NOME,CODDOC,CODDIP)`

e la dipendenza funzionale `CODDOC` → `CODDIP`

CODCOR	NOME	CODDOC	CODDIP
CodCor1	AnalisiI	CodDoc1	CodDipA
CodCor3	Biologia	CodDoc1	CodDipA
CodCor2	Chimica	CodDoc2	CodDipA
CodCor4	Fisica	CodDoc3	CodDipB

ridondanza: in tutti i corsi di un docente si ripete lo stesso dipartimento

anomalia di modifica: se un docente cambia dipartimento si devono modificare tutte le tuple relative ai corsi di quel docente

anomalia di inserimento: non si può inserire un dipartimento senza inserire almeno un corso di un suo docente

anomalia di cancellazione: se vengono cancellati tutti i corsi di un docente si perde anche l'informazione sul dipartimento del docente

- I problemi derivano da `CODDOC` → `CODDIP` che stabilisce che un attributo *non* superchiave (`CODDOC`) determina funzionalmente un altro attributo (`CODDIP`). Questo esempio è una violazione della 3NF.

♢ Informalmente, la 3NF serve per evitare problemi derivanti da attributi non superchiave che determinano funzionalmente altri attributi non primi.

Definizione 10 (3NF) *Uno schema $\langle R(T), F\rangle$ è in **3NF** se e solo se per ogni dipendenza funzionale non banale $X \to A \in F$, o X è una superchiave oppure A è primo.*

Teorema 4 (3NF ⇒2NF) *Uno schema $\langle R(T), F\rangle$ che è in **3NF** è anche in **2NF**.*

Dimostrazione : Equivale a dimostrare che **not 2NF** →**not 3NF**, ovvero che uno schema che non è in **2NF** allora non è in **3NF**. Se non è in **2NF** esiste una chiave K e un sottoinsieme proprio K' di K, $K' \subset K$, che determina un attributo non primo A, $K' \to A$. Allora lo schema non è in **3NF** in quanto si ha $K' \to A$, con K' non superchiave e A non primo.

♢ Uno schema in **3NF** non è esente da problemi di ridondanza e anomalie di aggiornamento dei dati.

La **3NF** definisce comunque schemi con un buon livello di qualità e costituisce quindi lo standard da noi adottato per il progetto logico.

Forma Normale di Boyce–Codd (BCNF)

♢ **Esempio :** Consideriamo lo schema di relazione `CAMPIONATO`

`CAMPIONATO(SQUADRA,PARTITA,GIOCATORE,RUOLO)`

con la dipendenza funzionale `GIOCATORE` → `SQUADRA`

SQUADRA	PARTITA	GIOCATORE	RUOLO
Inter	SerieA1132007	Materazzi	TerzinoDestro
Inter	SerieA28102006	Materazzi	Libero
Inter	SerieA1132007	Ibrahimovic	Centravanti
Palermo	SerieA1132007	Brienza	Centravanti
Fiorentina	SerieA28102006	Mutu	Centravanti

ridondanza: l'informazione sulla squadra di un giocatore viene ripetuta

anomalia di modifica: se un giocatore cambia squadra si devono modificare tutte le tuple relative al giocatore stesso

anomalia di inserimento: non si può inserire un giocatore in una squadra senza assegnargli almeno un ruolo in una certa partita;

anomalia di cancellazione: se vengono cancellate tutti i ruoli relativi ad un giocatore si perde anche l'informazione sulla squadra del giocatore

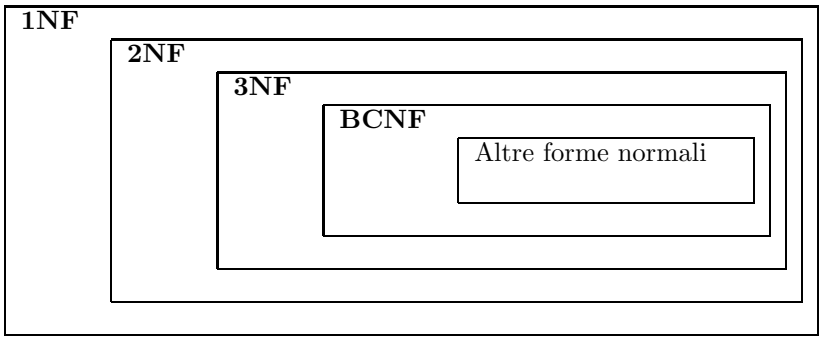
- I problemi derivano da `GIOCATORE` → `SQUADRA` che stabilisce che un attributo *non* superchiave (`GIOCATORE`) determina funzionalmente un attributo primo (`SQUADRA`). Questo esempio è una violazione della BCNF.

♢ Informalmente, la BCNF afferma che tutti gli attributi (anche quelli primi) dipendono funzionalmente solo dalle superchiavi.

Definizione 11 (BCNF) *Uno schema $\langle R(T), F\rangle$ è in **BCNF** se e solo se per ogni dipendenza funzionale non banale $X \rightarrow A \in F$, X è una superchiave.*

Teorema 5 (BCNF ⇒3NF) *Uno schema $\langle R(T), F\rangle$ che è in **BCNF** è anche in **3NF**.*

Dimostrazione : Segue immediatamente dalle definizioni.



5.4 Decomposizione di schemi

Uno schema di relazione che violi la forma normale desiderata può essere decomposto in sottoschemi allo scopo di raggiungere tale forma normale.

Definizione 12 (Decomposizione di uno schema) *Una decomposizione di uno schema $R(T)$ è un insieme di schemi $\rho = \{R_1(T_1), R_2(T_2), \ldots R_n(T_n)\}$, tale che $T = T_1 \cup T_2 \cup \ldots \cup T_n$.*

♢ A livello di istanze, decomporre uno schema significa memorizzare, al posto della istanza r dello schema iniziale $R(T)$, le sue proiezioni sui sottoschemi $\pi_{T_1}(r)$. Le proprietà auspicabili per una decomposizione sono sia quella della *preservazione dell'informazione* contenuta nell'istanza r che la *preservazione delle dipendenze funzionali.*

Esempio : Dato lo schema $\langle$`RUOLI(S,G,R)`$, F = \{\mathsf{SR} \rightarrow \mathsf{G}, \mathsf{G} \rightarrow \mathsf{S}\}\rangle$, dove `S`, `G` e `R` rappresentano rispettivamente gli attributi `SQUADRA`, `GIOCATORE` e `RUOLO`, consideriamo la decomposizione: $\rho_1 = \{R_1(S, R), R_2(G, S)\}$.
Intuitivamente, tale decomposizione non preserva i dati, in quanto se consideriamo una istanza legale r di `RUOLI` e le sue proiezioni sugli schemi decomposti:

r

SQUADRA	GIOCATORE	RUOLO
Italia	Materazzi	TerzinoDestro
Italia	Materazzi	Libero
Italia	Grosso	TerzinoSinistro
Brasile	Adriano	Centravanti
Argentina	Cambiasso	Centrocampista

$\pi_{\mathsf{SR}}(r)$

SQUADRA	RUOLO
Italia	TerzinoDestro
Italia	Libero
Italia	TerzinoSinistro
Brasile	Centravanti
Argentina	Centrocampista

$\pi_{\mathsf{GS}}(r)$

GIOCATORE	SQUADRA
Materazzi	Italia
Grosso	Italia
Adriano	Brasile
Cambiasso	Argentina

il join naturale $\pi_{\mathsf{SR}}(r) \bowtie \pi_{\mathsf{GS}}(r)$, effettuato per ricostruire l'istanza r, contiene *più* tuple di r, ad esempio (`Italia`,`Materazzi`,`TerzinoSinistro`) $\notin r$.
Inoltre, ρ_1 non preserva le dipendenze, infatti nell'istanza r di `RUOLI`, non è possibile inserire la tupla (`Argentina`,`Zanetti`,`CentroCampista`) in quanto verrebbe violata la dipendenza $\mathsf{SR} \rightarrow \mathsf{G}$ (infatti l'Argentina ha già un CentroCampista). Invece nello schema decomposto l'inserimento andrebbe a buon fine: infatti è possibile inserire le due tuple (`Argentina`,`CentroCampista`) e (`Zanetti`,`Argentina`) in $\pi_{\mathsf{SR}}(r)$ e $\pi_{\mathsf{GS}}(r)$ rispettivamente, in quanto non vengono violate le dipendenze dei singoli schemi.

Preservazione dei dati

♢ Una decomposizione preserva i dati se una qualunque istanza dello schema iniziale è ricostruibile a partire dalle istanze dei sottoschemi attraverso il join naturale. Formalmente:

Definizione 13 (Decomposizione lossless) *Dato uno schema $R(T)$, una sua decomposizione $\rho = \{R_1(T_1), R_2(T_2), \ldots R_n(T_n)\}$ viene detta* senza perdita di informazioni *o* lossless *se e solo se per ogni istanza r di R si ha che*

$$r = \pi_{T_1}(r) \bowtie \pi_{T_2}(r) \bowtie \ldots \bowtie \pi_{T_n}(r).$$

♢ Dato uno schema $R(T)$ e una sua qualsiasi decomposizione $\rho = \{R_1(T_1), R_2(T_2), \ldots R_n(T_n)\}$, si ha che, per ogni istanza legale r di R

$$r \subseteq \pi_{T_1}(r) \bowtie \pi_{T_2}(r) \bowtie \ldots \bowtie \pi_{T_n}(r)$$

quindi verificare che ρ sia *senza perdita di informazioni* equivale a verificare che in ρ non siano state aggiunte tuple.

♢ Verificare che una generica decomposizione n-arie sia lossless è un problema complesso. D'altra parte, per le decomposizioni binarie, che sono in generale quelle utilizzate in pratica, esiste invece un teorema molto semplice:

Teorema 6 (Decomposizioni Binarie Lossless) *Dato uno schema $R(T)$, una sua decomposizione binaria $\rho = \{R_1(T_1), R_2(T_2)\}$ è* lossless *se e solo se*

$$T_1 \cap T_2 \rightarrow T_1 \in F^+ \quad \textbf{\textit{oppure}} \quad T_1 \cap T_2 \rightarrow T_2 \in F^+$$

ovvero, se e solo se il join naturale è eseguito su una superchiave di uno dei due sottoschemi.

Esempio : Riprendiamo lo schema `RUOLI(S,G,R)` dell'esempio precedente; la sua decomposizione $\rho_1 = \{R_1(S, R), R_2(G, S)\}$ non è lossless in quanto il join naturale viene eseguito su `S` che non è superchiave in nessuno dei due sottoschemi.

Invece, la decomposizione $\rho_2 = \{R_1(G, R), R_2(G, S)\}$ è lossless in quanto il join naturale viene eseguito sull'attributo `G` che è superchiave in R_2.

Preservazione delle dipendenze

♢ Intuitivamente, una decomposizione preserva le dipendenze se dall'insieme delle dipendenze relative ai sottoschemi è possibile ottenere l'insieme delle dipendenze iniziali.

♢ Per definire formalmente la proprietà di preservazione delle dipendenze funzionali, si introduce il concetto di *proiezione di un insieme di dipendenze*:

Definizione 14 (Proiezione di F) *Dato uno schema $\langle R(T), F\rangle$ e un sottinsieme $T_i \subseteq T$, si definisce* proiezione *di F su T_i, denotato con $\pi_{T_i}(F)$, l'insieme di dipendenze funzionali:*

$$\pi_{T_i}(F) = \{X \to Y \mid X \to Y \in F^+, XY \subseteq T_i\}$$

♢ Si noti che $\pi_{T_i}(F)$ è definita considerando le dipendenze funzionali di F^+, non solo quelle di F, come è evidente dall'esempio che segue:

Esempio : Dato lo schema $\langle R(ABCD), F = \{A \to B, B \to C, C \to D\}\rangle$ e la sua decomposizione $\rho = \{R_1(AB), R_2(BC), R_3(CD)\}$, si ha che:

$\pi_{AB}(F) = \{A \to B, B \to A\}$

$\pi_{BC}(F) = \{B \to C, C \to B\}$

$\pi_{CD}(F) = \{C \to D, D \to C\}$

Definizione 15 (Decomposizione che preserva le dipendenze) *Dato uno schema $\langle R(T), F\rangle$, una sua decomposizione $\rho = \{R_1(T_1), R_2(T_2), \ldots R_n(T_n)\}$,* preserva le dipendenze *se e solo se:*

$$\Big(\pi_{T_1}(F) \cup \pi_{T_2}(F) \cup \ldots \cup \pi_{T_n}(F)\Big)^+ = F^+$$

♢ In base a tale definizione, per verificare che una decomposizione preservi le dipendenze, si dovrebbe calcolare F^+ e poi proiettarlo su ciascun T_i per ottenere $G = \pi_{T_1}(F) \cup \pi_{T_2}(F) \cup \ldots \cup \pi_{T_n}(F)$. Inoltre si dovrebbe calcolare la chiusura G^+ per verificare l'equivalenza con F^+.

♢ Per verificare che $G^+ = F^+$, siccome $G^+ \subseteq F^+$, è sufficiente verificare che $F^+ \subseteq G^+$, e a tale scopo è sufficiente controllare che ogni dipendenza in F sia anche in G^+. Questo equivale a verificare che, per ogni $X \to Y \in F$, la chiusura di X calcolata rispetto a G contenga Y.

Preservazione delle dipendenze: algoritmo **XPIUG**

♢ Per determinare la chiusura di un insieme di attributi X rispetto a G, senza calcolare G in modo esplicito, ma utilizzando F, è stato proposto il seguente algoritmo [37] che ha complessità polinomiale nella dimensione di F:

Algoritmo XPIUG

- **Input :** decomposizione di $\langle R(T), F\rangle$ con $T_1, T_2, \ldots T_n$ e $X \subseteq T$
- **Output : XPIUG**

begin
XPIUG := X ;
repeat
for $i := 1$ **to** n **do**
$\mathbf{XPIUG} = \mathbf{XPIUG} \cup \Big(\mathbf{XPIU}(\mathbf{XPIUG} \cap T_i, F) \cap T_i\Big)$;
until (XPIUG non è cambiato);
end.

♢ Si noti che $\mathbf{XPIU}(\mathbf{XPIUG} \cap T_i, F) \cap T_i$ individua gli attributi semplici A tali che $\mathbf{XPIUG} \cap T_i \rightarrow A \in \pi_{T_i}(F)$, cioè nell'algoritmo viene calcolata *ripetutamente* la chiusura di X rispetto alle proiezioni di F.

♢ Si può dimostrare che l'algoritmo **XPIUG** termina e calcola correttamente X^+ rispetto a G, cioè a $\cup\pi_{T_i}(F)$.

♢ Pertanto, se per ogni dipendenza funzionale $X \rightarrow Y \in F$, Y è un sottinsieme di X^+ rispetto a G (valutato con l'algoritmo **XPIUG**) ρ preserva le dipendenze, altrimenti non le preserva.

Esempio : Riprendiamo la decomposizione $\rho_1 = \{R_1(S, R), R_2(G, S)\}$ dello schema $\langle$RUOLI(S,G,R)$, F = \{\mathsf{SR} \rightarrow \mathsf{G}, \mathsf{G} \rightarrow \mathsf{S}\}\rangle$ e verifichiamo che non viene preservata la dipendenza funzionale SR $\rightarrow$ G:

$$\begin{aligned}
\mathbf{XPIUG}^0 &= SR \\
\mathbf{XPIUG}^1 &= SR \cup (\mathbf{XPIU}(SR \cap SR, F) \cap SR) \cup (\mathbf{XPIU}(SR \cap GS, F) \cap GS) \\
&= SR \cup (SRG \cap SR) \cup (\emptyset \cap GS) = SR
\end{aligned}$$

Poichè $G \notin \mathbf{XPIUG}(F, \rho_1, SR)$, SR $\rightarrow$ G non appartiene alla chiusura di SR rispetto a $\cup\pi_{T_i}(F)$ e quindi le dipendenze non sono preservate in ρ_1.

♢ Dato uno schema $\langle R(T), F\rangle$, è possibile verificare che se una decomposizione $\rho = \{R_1(T_1), R_2(T_2), \ldots R_n(T_n)\}$ *proietta* una dipendenza funzionale $X \rightarrow Y \in F$ su almeno un sottoschema, cioè se esiste i tale che $XY \subseteq T_i$, allora ρ preserva tale dipendenza funzionale. Quindi se ρ proietta ogni $f \in F$ su almeno un sottoschema, ρ preserva le dipendenze funzionali.

Osservazioni sulle decomposizioni

♢ Le proprietà di preservazione dei dati e di preservazione delle dipendenze sono *ortogonali*: vi sono decomposizioni che preservano i dati ma non le dipendenze, e viceversa.

Consideriamo ad esempio lo schema:

$$\langle R(ABCD), F = \{AB \to C, A \to D, DB \to C\}\rangle$$

e le seguenti decomposizioni:

- $\rho_1 = \{R_1(AD), R_2(BCD)\}$
  1. **non preserva i dati**, in quanto il join naturale viene eseguito su D che non è superchiave in nessuno dei due sottoschemi.
  2. **preserva le dipendenze**, in quanto $A \to D$ è preservata perchè proiettata su R_1, $DB \to C$ è preservata perchè proiettata su R_2, e queste due dipendenze implicano logicamente la dipendenza $AB \to C$ (tramite l'algoritmo **XPIUG**: $C \in$ **XPIUG**(F, ρ_1, AB)).

- $\rho_2 = \{R_1(AD), R_2(ABC)$
  1. **preserva i dati**, in quanto il join naturale viene eseguito su A che è una superchiave dello schema R_1.
  2. **non preserva le dipendenze**, in quanto $A \to D$ è preservata perchè proiettata su R_1, $AB \to C$ è preservata perchè proiettata su R_2, ma $DB \to C$ non è preservata in quanto $C \notin$ **XPIUG**(F, ρ_2, DB).

- $\rho_3 = \{R_1(ADB), R_2(BCD)\}$
  1. **preserva i dati**, in quanto il join naturale viene eseguito su DB che è superchiave in R_2.
  2. **preserva le dipendenze**, in quanto $A \to D$ è preservata perchè proiettata su R_1, $DB \to C$ è preservata perchè proiettata su R_2, e queste due dipendenze implicano logicamente $AB \to C$.

♢ In [3] viene presentato il seguente teorema che stabilisce una condizione sufficiente, ma non necessaria, affinchè una decomposizione che preserva le dipendenze preservi anche i dati:

Teorema 7 *Dato uno schema $\langle R(T), F\rangle$, sia $\rho = \{R_1(T_1), R_2(T_2), \ldots R_n(T_n)\}$ una sua decomposizione che preserva le dipendenze e tale che T_j, per qualche j, è una superchiave per $\langle R(T), F\rangle$. Allora ρ preserva i dati.*

5.5 Normalizzazione di uno schema relazionale

♢ Intuitivamente, la normalizzazione di uno schema avviene applicando ricorsivamente il seguente criterio di *decomposizione per proiezione*:

- Uno schema $\langle R(X, Y, A), F\rangle$ che non è nella forma normale desiderata a causa di $X \to A$ viene decomposto in $R1(X, Y)$ e $R2(X, A)$.

♢ La decomposizione complessiva è lossless ma la preservazione delle dipendenze non è garantita e dipende dall'ordine di applicazione delle singole decomposizioni.

Esempio : Riconsideriamo lo schema dell'esempio di pagina 236:

$\langle R(ABCD), F = \{AB \to C, A \to D, DB \to C\}\rangle$

L'unica chiave di R è $K_1 = AB$; lo schema quindi non è in BCNF a causa di $A \to D$. Pertanto consideriamo la decomposizione $R_1(AD)$ e $R_2(ABC)$.

Abbiamo visto che tale decomposizione è lossless, ma non preserva la dipendenza funzionale $DB \to C$. Siccome entrambi gli schemi di relazione ottenuti sono in BCNF, il processo di decomposizione termina.

D'altra parte, se avessimo considerato la dipendenza funzionale $DB \to C$, che rende lo schema iniziale non in BCNF, si sarebbe ottenuto $R_1(ADB)$ e $R_2(BCD)$. Abbiamo visto che tale decomposizione è lossless e preserva tutte le dipendenze. Lo schema $R_1(ADB)$ non è in BCNF a causa di $A \to D$; pertanto si decompone ulteriormente in $R_{11}(AD)$ e $R_{12}(AB)$; anche tale decomposizione è lossless e preserva le dipendenze. In definitiva si ottiene la decomposizione $R_{11}(AD), R_{12}(AB), R_2(BCD)$ che risulta essere lossless e preserva le dipendenze. I sottoschemi ottenuti sono in BCNF.

♢ Usando il criterio di decomposizione per proiezione sono stati realizzati algoritmi di decomposizione per la normalizzazione di schemi in BCNF (si veda [36, 3, 14]), tramite i quali:

qualunque schema può essere decomposto preservando i dati in un insieme di schemi in BCNF.

Gli algoritmi hanno complessità esponenziale in quanto nella decomposizione di R in $R1$ e $R2$ occorre calcolare le proiezioni di F su $R1$ e $R2$.

Normalizzazione di schemi

♢ Dato uno schema giustifichiamo con un esempio perchè *non è sempre possibile trovarne una decomposizione in BCNF che preserva le dipendenze.* Riprendiamo lo schema $\langle \texttt{RUOLI(S,G,R)}, F = \{\texttt{SR} \rightarrow \texttt{G}, \texttt{G} \rightarrow \texttt{S}\}\rangle$
Una decomposizione di `RUOLI`, per essere in BCNF, non deve presentare un sottoschema contenente tutti e tre gli attributi; pertanto la dipendenza funzionale $\texttt{SR} \rightarrow \texttt{G}$ non può essere in nessuna proiezione $\cup\pi_{T_i}(F)$. Inoltre $\texttt{SR} \rightarrow \texttt{G} \notin (\cup\pi_{T_i}(F))^+$, in quanto né `S` né `R`, presi separatamente, determinano `G`.

♢ Per la normalizzazione di schemi in 3NF, che stabilisce condizioni meno restrittive rispetto alla BCNF, sono stati realizzati algoritmi tramite i quali:

qualunque schema di relazione può essere decomposto preservando dati e dipendenze in un insieme di schemi di relazione in 3NF.

♢ Questi algoritmi (si veda [36, 3, 14]) sono detti di *sintesi*, in quanto partizionano l'insieme delle dipendenze dello schema secondo certi criteri e fanno corrispondere ad ogni elemento della partizione uno schema di relazione.

♢ Noi non presentiamo né gli algoritmi di decomposizione né gli algoritmi di sintesi ma nel seguito (soprattutto nella sezione 6.3 relativa agli esercizi) procederemo alla normalizzazione di uno schema $\langle R, F\rangle$ solo in maniera "euristica", applicando ricorsivamente il criterio di decomposizione per proiezione negli schemi $R1$ e $R2$. Inoltre verificheremo generalmente la violazione della forma normale desiderata sugli schemi $R1$ e $R2$ solo rispetto alle dipendenze di F proiettate sugli schemi di $R1$ e $R2$.

Normalizzazione e progettazione E/R

♢ La progettazione di basi di dati usando il modello E/R è molto più facile di quella relazionale a partire da un insieme dato di dipendenze funzionali.

♢ La traduzione di uno schema E/R "ben progettato" in schema logico relazionale ottenuta applicando le regole di progetto logico fornite nel capitolo 3 produce generalmente uno schema relazionale in 3NF.

♢ Per definire la nozione di schema E/R "ben progettato" la teoria della normalizzazione, sviluppata nel modello relazionale, si potrebbe estendere al modello E/R, definendo le forme normali sia per le entità che per le associazioni. In tali definizioni si dovrebbe fare uso del concetto di dipendenza funzionale tra gli attributi di un'entità oppure di una associazione.

La principale difficoltà da affrontare nell'estendere la teoria della normalizzazione dal modello relazionale al modello E/R è il differente modo con il quale si esprime nei due modelli il concetto di identificatore. Tale teoria è presentata in [10].

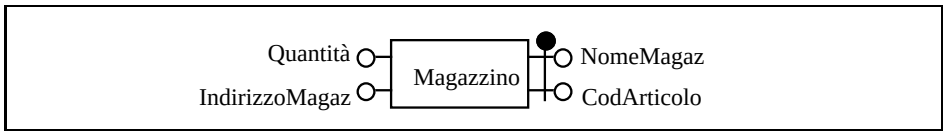
♢ Il modo usuale di procedere è quindi il seguente:

1. il progettista disegna uno schema E/R;
2. genera lo schema relazionale applicando le regole viste nel capitolo 3;
3. "controlla" in maniera euristica eventuali violazioni della 2NF, 3NF e BCNF e le risolve tramite decomposizione per proiezione;
4. modifica lo schema E/R iniziale in modo tale che vi sia corrispondenza con lo schema relazionale ottenuto al passo 3.

♢ Gli esempi seguenti mostrano come quando gli schemi E/R non sono "ben progettati" sorgono problemi di normalizzazione nello schema relazionale generato.

Esempi di schemi E/R che violano la 2NF

Per entità – Il progetto logico del seguente schema E/R:



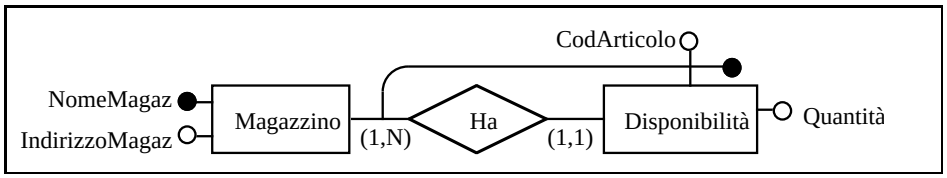
produce lo schema di relazione:

`Magazzino(NomeMagaz,CodArticolo,Quantità,IndirizzoMagaz)`

che non è in 2NF se `NomeMagaz` → `IndirizzoMagaz`; decomponendo:

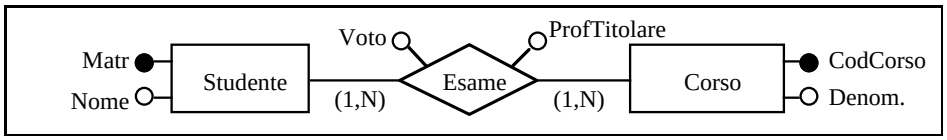
`Magazzino1(NomeMagaz,CodArticolo,Quantità)`
`Magazzino2(NomeMagaz,IndirizzoMagaz)`

Uno schema E/R ben progettato, se `NomeMagaz` → `IndirizzoMagaz`, è:



♢ Il progetto logico di questo schemo porta a schemi di relazione che sono in 3NF e non c'è quindi la necessità di effettuare la decomposizione.

Per associazioni – Il progetto logico del seguente schema E/R:



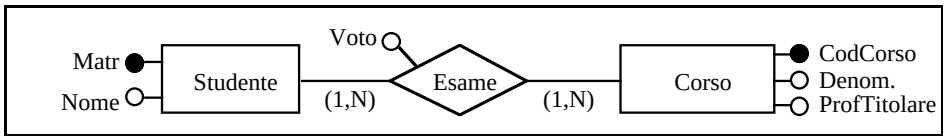
produce lo schema relazionale:

`Studente(Matr,Nome)`
`Corso(CodCor,Denom)`
`Esame(Matr,CodCor,Voto,ProfTitolare)`

`Esame` non è in 2NF se `CodCor` → `ProfTitolare`; decomponendo:

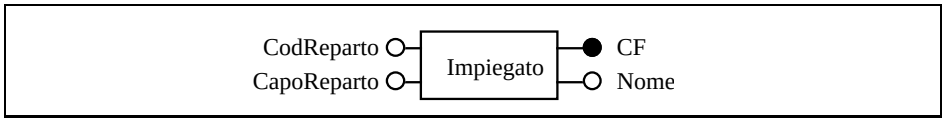
`Esame1(Matr,CodCor,Voto)`
`Esame2(CodCor,ProfTitolare)`

Uno schema E/R ben progettato, se `CodCor` → `ProfTitolare`, è:



Esempi di schemi E/R che violano la 3NF

Per entità – Il progetto logico del seguente schema E/R:



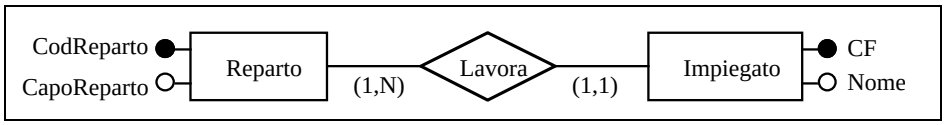
produce lo schema di relazione

`Impiegato(Cf,Nome,CodReparto,CapoReparto)`

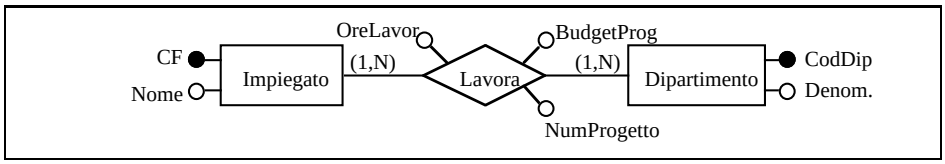
che non è in 3NF se `CodReparto` → `CapoReparto`; decomponendo:

`Impiegato1(Cf,Nome,CodReparto)`
`Impiegato2(CodReparto,CapoReparto)`

Uno schema E/R ben progettato, se `CodReparto` → `CapoReparto`, è:



Per associazioni – Il progetto logico del seguente schema E/R:



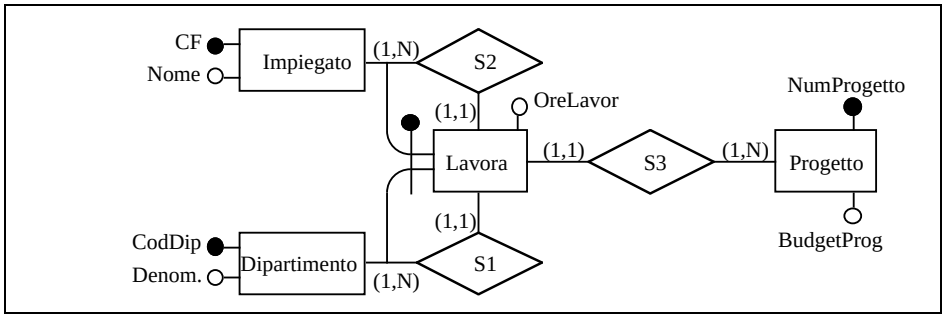
produce lo schema relazionale:

`Impiegato(CF,Nome)`
`Dipartimento(CodDip,Denom)`
`Lavora(CF,CodDip,OreLavor,NumProgetto,BudgetProg)`

`Lavora` non è in 3NF se `NumProgetto` → `BudgetProg`; decomponendo:

`Lavora1(CF,CodDip,OreLavor,NumProgetto)`
`Lavora2(NumProgetto,BudgetProg)`

Uno schema E/R ben progettato, se `NumProgetto` → `BudgetProg`, è:



Capitolo 6

Esercizi

In questo capitolo vengono presentati numerosi esercizi, integralmente risolti, su tutte le tematiche introdotte.

La prima parte contiene esercizi di progettazione concettuale a partire da requisiti in linguaggio naturale e conseguente progettazione logica. Nella soluzione di questi esercizi viene presentato uno tra i possibili schemi E/R corretti corrispondenti alle specifiche date e la relativa traduzione in schema relazionale. In alcune soluzioni, soprattutto quelle relative ai primi esercizi, viene riportato un breve commento allo schema E/R per giustificare le scelte effettuate. Questa prima parte termina proponendo alcuni esercizi relativi ai dati derivati.

La seconda parte contiene esercizi in cui è richiesto di esprimere interrogazioni sia in algebra relazionale che in SQL. Anche nella soluzione di questi esercizi viene presentato una possibile espressione, sia in SQL sia in algebra relazionale, che risolve l'interrogazione richiesta. I commenti sono limitati ad alcune interrogazioni particolarmente complicate.

La terza parte contiene esercizi sulla normalizzazione in cui viene generalmente richiesto, dato uno schema relazionale e un insieme di dipendenze funzionale (esplicite oppure descritte a parole), di determinare se lo schema è in 2NF, 3NF e BCNF e di discutere eventuali decomposizioni.

6.1 E/R e Progetto Logico

6.1.1 Esercizi commentati

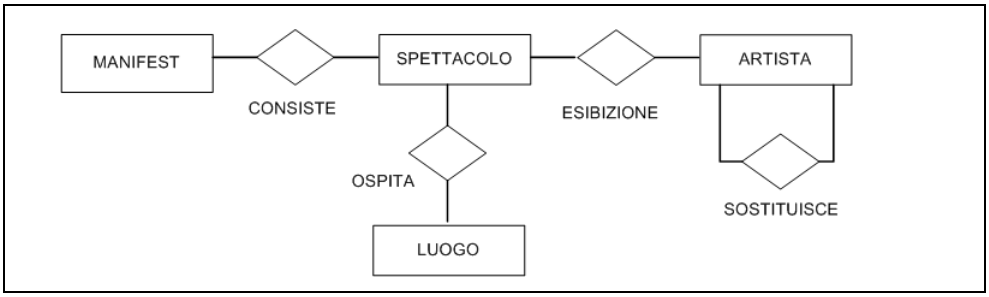
Per ciascun esercizio, viene richiesto di:

1. Progettare lo schema E/R
2. Tradurre lo schema E/R in schema relazionale in terza forma normale.

♢ **Esercizio 1**
Si vogliono rappresentare informazioni relative alla gestione di manifestazioni artistiche durante l'estate. Una manifestazione, descritta da un codice e da un nome, consiste di due o più spettacoli; ogni spettacolo è descritto da un numero univoco all'interno della manifestazione nella quale è inserito e dall'ora di inizio. Durante uno spettacolo si esibiscono uno o più artisti (un artista si può esibire al massimo una volta durante lo stesso spettacolo) ricevendo un certo compenso. Un artista è descritto dal codice SIAE e dal nome d'arte. Per ogni artista si deve indicare necessariamente un altro artista che lo sostituisca in caso di indisponibilità; un artista può essere indicato come sostituto di più artisti. Per ospitare gli spettacoli vengono adibiti opportuni luoghi; inoltre, in una certa data, un luogo può ospitare al massimo tre spettacoli, sia della stessa manifestazione sia di manifestazioni differenti.

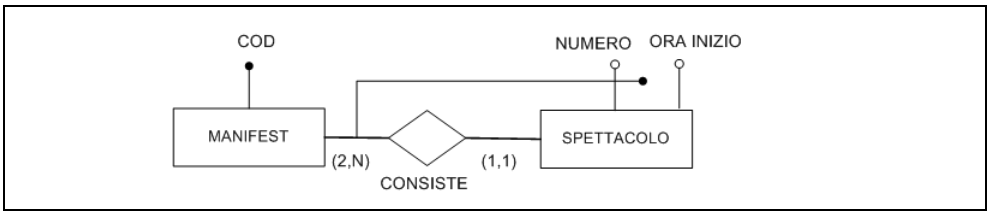
Soluzione

Dall'analisi del testo si individuano i concetti fondamentali che devono essere modellati. In particolare occorre modellare i concetti di `MANIFESTAZIONE`, `SPETTACOLO`, `ARTISTA`, `LUOGO` come entità e le seguenti associazioni binarie: `CONSISTE` tra `MANIFESTAZIONE` e `SPETTACOLO`, `OSPITA` tra `SPETTACOLO` e `LUOGO`, `ESIBIZIONE` tra `ARTISTA` e `SPETTACOLO`, `SOSTITUISCE` tra `ARTISTA` e se stesso.
Lo schema scheletro che si ottiene è il seguente:

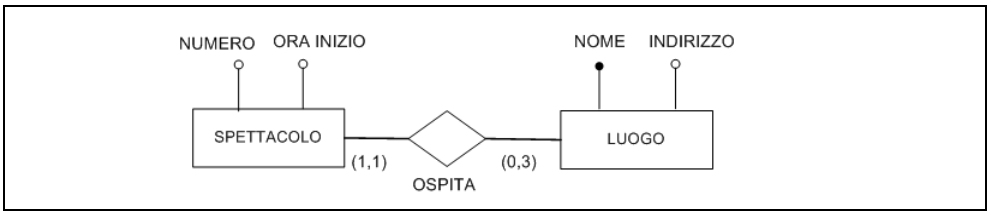


L'associazione `CONSISTE` permette di modellare due vincoli: il fatto che una manifestazione consiste di due o più spettacoli attraverso la cardinalità dell'associazione card(`MANIFEST`,`CONSISTE`)=(2,N) e il fatto che uno

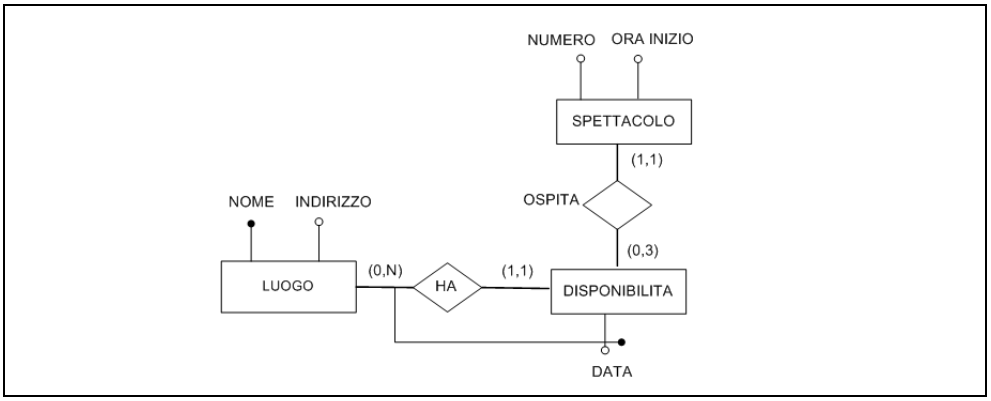
spettacolo sia identificato da un numero univoco all'interno della manifestazione attraverso l'identificazione di spettacolo mediante un numero e l'entità `MANIFEST`.



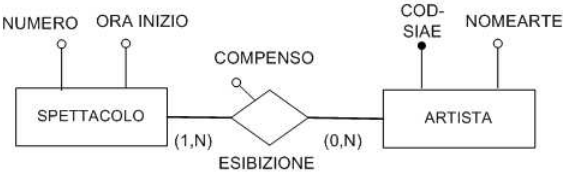
L'associazione `OSPITA` deve modellare il vincolo che un certo luogo in una certa data possa ospitare al massimo tre spettacoli, sia della stessa manifestazione sia di manifestazioni differenti. Lo schema seguente modella una rappresentazione non corretta della specifica: in quel modo si vincola un luogo ad ospitare in tutto tre spettacoli, mentre questa condizione è valida unicamente per una certa data.



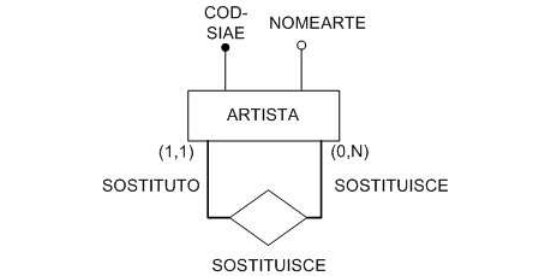
Per modellare in maniera corretta il vincolo occorre reificare l'associazione `OSPITA` introducento l'entità `DISPONIBILITA` che rappresenta la disponibilità in un certo luogo di ospitare in una certa data uno spettacolo.



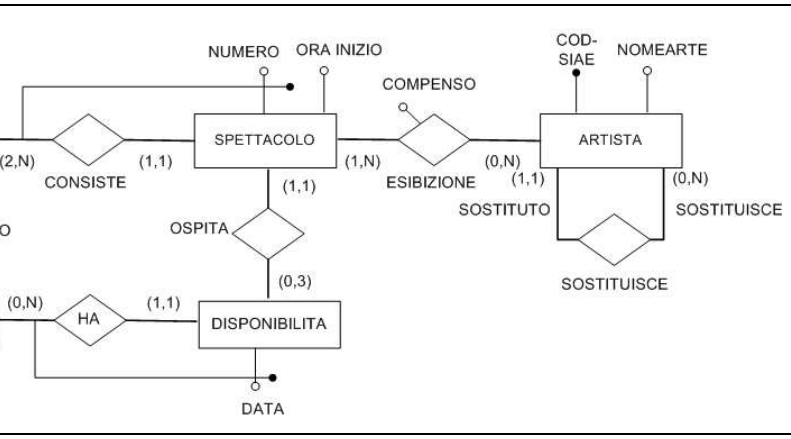
Il testo impone che un artista si esibisca una sola volta all'interno di uno spettacolo. Questo vincolo è già rappresentato dalla associazione binaria esibizione.



L'associazione ad anello `SOSTITUISCE` permette di indicare l'artista sostitutivo in caso di indisponibilità. Sono state introdotte due label `SOSTITUISCE` e `SOSTITUTO` per esplicitare il vincolo espresso dalla cardinalità.



Lo schema completo che si ottiene è il seguente.



Schema Relazionale

```
MANIFEST(CODICE)
SPETTACOLO(NUMERO,CODICE,ORA-INIZIO,NOMELUOGO,DATADISP)
     FK: CODICE REFERENCES MANIFEST
     FK: NOMELUOGO,DATADISP REFERENCES DISPONIBILITA NOT NULL
ESIBIZIONE(COD-MANIF,NUM-SPETTACOLO,COD-SIAE,COMPENSO)
     FK: COD-MANIF,NUM-SPETTACOLO REFERENCES SPETTACOLO
     FK: COD-SIAE REFERENCES ARTISTA
DISPONIBILITA(DATA,LUOGO)
     FK: LUOGO REFERENCES LUOGO
LUOGO(NOME,INDIRIZZO)
ARTISTA(COD-SIAE,NOME-ARTE,COD-SIAE-SOSTITUTO)
     FK: COD-SIAE-SOSTITUTO REFERENCES ARTISTA NOT NULL
```

Le associazioni `CONSISTE` e `OSPITA` non sono state tradotte in maniera esplicita, ma attraverso le foreign key `CODICE` e `NOMELUOGO,DATADISP` nell'entità `SPETTACOLO`. Stessa cosa per l'associazione `HA` che è rappresentata mediante la forign key `NOME` in `DISPONIBILITA`.
L'associazione ad anello `SOSTITUISCE` è stata poi tradotta all'interno della relazione `ARTISTA`.

♢ **Esercizio 2**

Si vuole rappresentare l'attività di una agenzia immobiliare con più filiali.
Ciascuna filiale è caratterizzata da partita I.V.A., denominazione, indirizzo, città e regione in cui è ubicata. Il personale dipendente è caratterizzato dagli usuali dati anagrafici, dallo stipendio, ed è ripartito in tre categorie: 1) responsabili di filiale: ciascuna filiale ha un responsabile e un responsabile è associato ad esattamente una filiale; 2) segretari: una segretario lavora in una filiale; una filiale può avere fino a 10 segretari; 3) agenti: un agente collabora con cinque filiali al massimo e non è fissato il numero di agenti che collaborano con una certa filiale.
I clienti della agenzia, con i consueti dati anagrafici, sono registrati presso una singola agenzia e sono di due tipi:

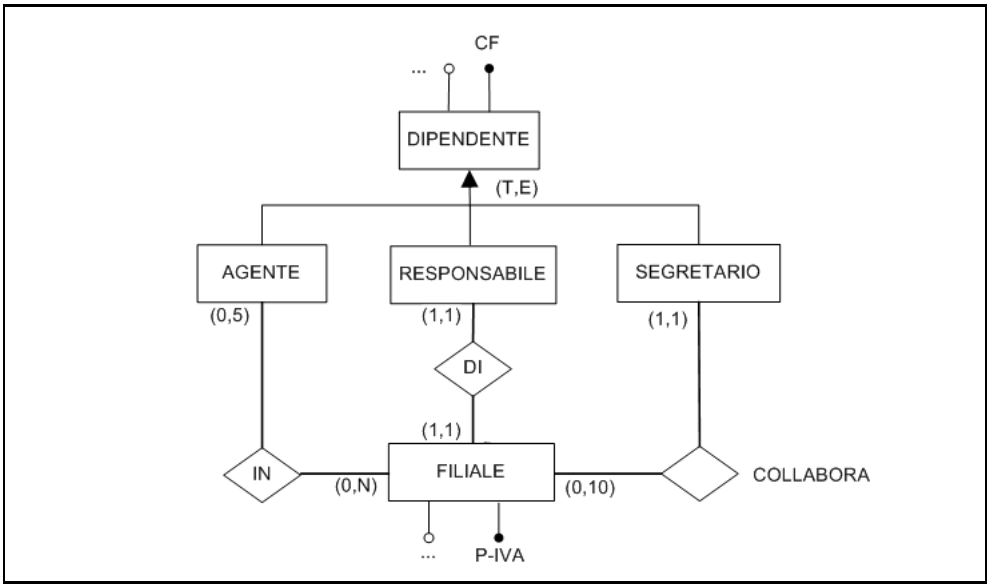
- offerente: che mette a disposizione la sua proprietà per la vendita. Le proprietà, individuate tramite un codice univoco, hanno una tipologia (ad esempio, terreno, appartamento, villino, villa, ...) ed una dimensione; una proprietà è di un solo offerente, un offerente può avere più proprietà.

- acquirente, per il quale devono essere riportate le tipologie alle quali è interessato con la relativa dimensione: ad esempio, l'acquirente "Tizio" è interessato sia alla tipologia "terreno" con dimensione di 1000 mq sia alla tipologia "villa" con dimensione di 200 mq.

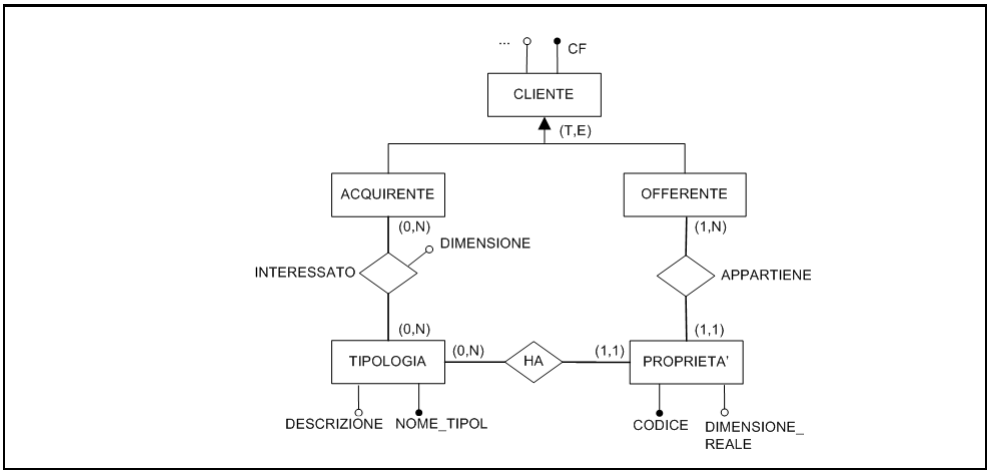
Il lavoro degli agenti consiste nel mostrare agli acquirenti le proprietà in vendita; tale lavoro è organizzato nel seguente modo: in una certa data, un agente mostra una singola proprietà, ad uno o più acquirenti e riporta, per ciascuno di questi acquirenti, una o più note. Ad esempio, durante la visita della proprietà "VillaParco" del "12Dicembre2006" per l'acquirente "Caio" vengono riportate le seguenti note: "è interessato alla proprietà", "ritiene il prezzo eccessivo" e "vuole valutare anche altre proprietà". In una certa data, una proprietà è mostrata da un singolo agente.
Vincolo aggiuntivo: modificare lo schema, inserendo il vincolo che "alla visita di una proprietà possono partecipare da uno a 10 acquirenti."

Soluzione

Si riportano di seguito i punti fondamentali della soluzione. Nella rappresentazione di alcune entità gli attributi semplici sono omessi.
1) Le specifiche relative alla strutturazione dei dipendenti sono realizzabili con una gerarchia: per le parti non esplicitate nella descrizione si possono fare opportune scelte: ad esempio, si suppone che la generalizzazione sia totale ed esclusiva.



2) I clienti dell'agenzia possono essere modellati come segue:



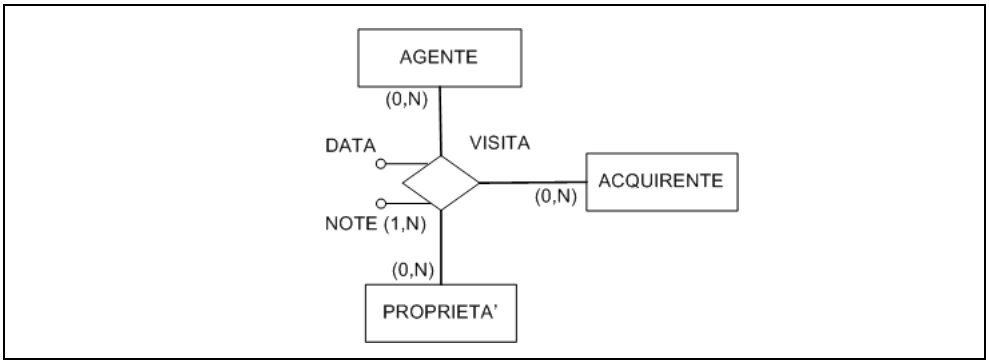
Si noti che è possibile introdurre una gerarchia di generalizzazione con superclasse `PERSONA` e con classi specializzanti `CLIENTE` e `DIPENDENTE`; tale gerarchia ha copertura totale e sovrapposta.
Si noti che nella soluzione si è introdotta l'entità `TIPOLOGIA` per rappresentare le tipologie di proprietà cui è interessato un acquirente. Sono possibili rappresentazioni alternative a quella proposta:

1. L'attributo `DIMENSIONE` può essere attributo multiplo (si noti però che questo non è esplicitamente richiesto dalle specifiche)

2. L'attributo `DIMENSIONE` può essere riferito anche all'entità `TIPOLOGIA` (in questo caso deve essere eliminata dall'associazione `INTERESSATO`)

3. Non introdurre l'entità `TIPOLOGIA` e rappresentare le tipologie alle quali è interessato l'acquirente tramite un attributo multiplo composto (con componenti tipologia e dimensione)

3) E' possibile rappresentare il lavoro degli agenti tramite una associazione ternaria:



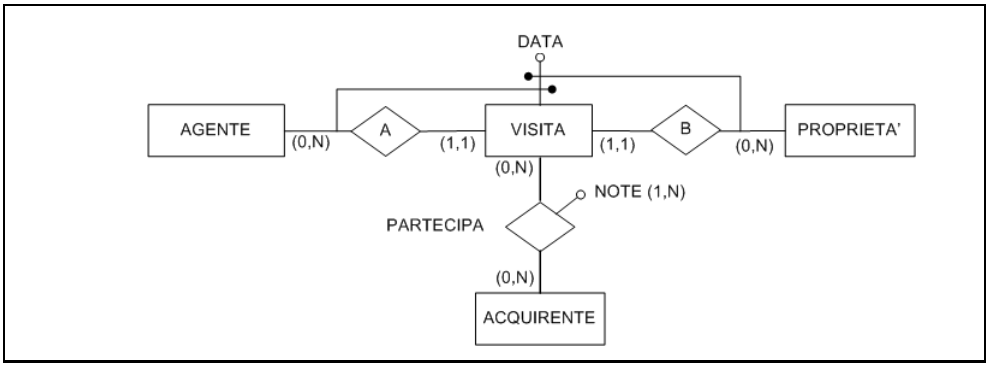
Questa rappresentazione non permette di modellare completamente le specifiche del testo:

1. Non viene espresso il vincolo che "in una certa data, un agente mostra una singola proprietà"

2. Non viene espresso che "In una certa data, una proprietà è mostrata da un singolo agente"

3. Si aggiunge il vincolo che "un agente mostra una proprietà ad un acquirente solo una volta"

Per modellare tali vincoli è necessario reificare l'associazione come segue:



In questo modo:

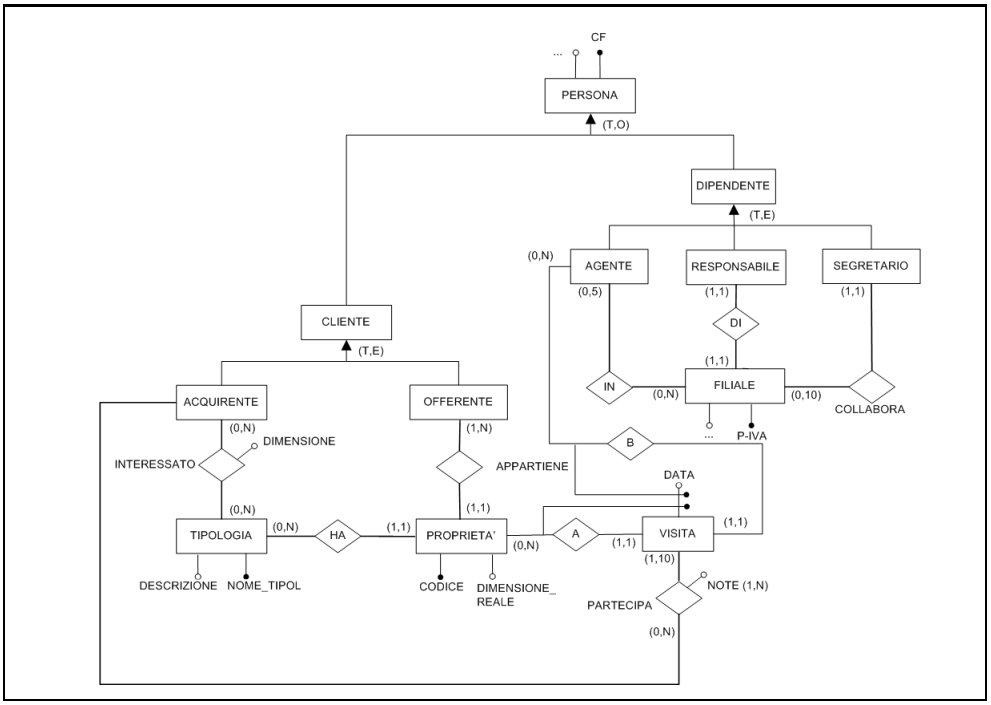
1. Il vincolo che "in una certa data, un agente mostra una singola proprietà" è espresso dall'identificatore `(DATA,AGENTE)`

2. Il vincolo che "In una certa data, una proprietà è mostrata da un singolo agente" è dato dall'identificatore `(DATA,PROPRIETA)`

3. E' stato eliminato il vincolo che "un agente mostra una proprietà ad un acquirente solo una volta"

4. È possibile modellare il vincolo facoltativo "alla visita di una proprietà possono partecipare da uno a 10 acquirenti" considerando come cardmax(`MOSTRA`,`PARTECIPA`)=10.

4) Lo schema che segue propone una possibile soluzione comprensiva del vincolo aggiuntivo.



Schema Relazionale

Si è preferito tradurre la gerarchia di `PERSONA` con un collasso verso il basso nelle entità `CLIENTE` e `DIPENDENTE`. Gli attributi di `PERSONA` sono in numero relativamente basso e quindi la duplicazione delle eventuali istanze nelle due sotto-entità non dovrebbe generare problemi nella dimensione del database. Le gerarchie di `CLIENTE` e `DIPENDENTE` sono invece state tradotte mantenendo le entità con associazioni. In questo modo è stato possibile garantire nel progetto logico la modellazione della associazioni che pongono in relazione le entità figlie. l'entità `RESPONSABILE` ha come chiave alternativa l'attributo `P-IVA` dell'entità `FILIALE`. L'entità `RESPONSABILE` e `FILIALE`

sono infatti collegate da una associazione uno a uno, e quindi a una istanza di `RESPONSABILE` corrisponde una e una sola istanza di `FILIALE` e viceversa.

```
CLIENTE(CF,NOME,COGNOME)
DIPENDENTE(CF,NOME,COGNOME,STIPENDIO)
ACQUIRENTE(CF)
     FK: CF REFERENCES CLIENTE
OFFERENTE(CF)
     FK: CF REFERENCES CLIENTE
TIPOLOGIA(NOME_TIPOLOGIA,DESCRIZIONE)
INTERESSATO(CF-ACQUIRENTE,TIPOLOGIA,DIMENSIONE)
     FK: CF-ACQUIRENTE REFERENCES ACQUIRENTE
     FK: TIPOLOGIA REFERENCES TIPOLOGIA NOT NULL
PROPRIETA(CODICE,DIMENSIONE_REALE,TIPOLOGIA,CF-PROPRIETARIO)
     FK: CF-PROPRIETARIO REFERENCES OFFERENTE NOT NULL
     FK: TIPOLOGIA REFERENCES TIPOLOGIA
AGENTE(CF)
     FK: CF REFERENCES DIPENDENTE
RESPONSABILE(CF,P-IVA)
     FK: CF REFERENCES DIPENDENTE
     FK: P-IVA REFERENCES FILIALE
     AK: P-IVA
SEGRETARIO(CF,P-IVA)
     FK: CF REFERENCES DIPENDENTE
     FK: P-IVA REFERENCES FILIALE NOT NULL
FILIALE(P-IVA,INDIRIZZO,CITTA,REGIONE)
VISITA(DATA,COD-PROPRIETA,CF-AGENTE)
     FK: CF-AGENTE REFERENCES AGENTE
     FK: COD-PROPRIETA REFERENCES PROPRIETA
     AK: CF-AGENTE,DATA
PARTECIPA(DATA,COD-PROPRIETA,CF-ACQUIRENTE)
     FK: DATA,COD-PROPRIETA REFERENCES VISITA
     FK: CF-ACQUIRENTE REFERENCES ACQUIRENTE
NOTE(DATA,COD-PROPRIETA,CF-ACQUIRENTE,NOTE)
     FK: DATA,COD-PROPRIETA,CF-ACQUIRENTE REFERENCES PARTECIPA
```

♢ **Esercizio 3**
Si vuole rappresentare l'attività di una cooperativa di vendita con più negozi. Ciascun *negozio* è caratterizzato da una partita I.V.A., denominazione, indirizzo, città e regione in cui è ubicato.
I *soci* della cooperativa, caratterizzati dai consueti dati anagrafici, sono suddivisi in tre categorie:

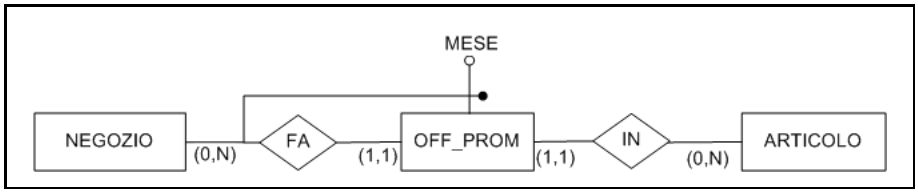
- *fondatore*: ciascun negozio ha un unico socio fondatore e, viceversa, un socio fondatore è associato ad un solo negozio;
- *dipendente*: lavora presso cinque negozi al massimo; un negozio può avere fino ad un massimo di 20 dipendenti;
- *cliente*: un cliente è associato ad un singolo negozio.

Gli articoli trattati dalla cooperativa di vendita hanno un codice identificativo, un prezzo ed una descrizione. La cooperativa effettua offerte promozionali, organizzate nel seguente modo: in *un certo mese*, un *negozio* effettua la promozione di *un solo articolo* e, viceversa, in *un certo mese*, un *articolo* è offerto in promozione da *un solo negozio*. Tutti i soci della cooperativa possono usufruire di cento offerte promozionali al massimo, una offerta promozionale può essere usufruita da più soci. Ogni volta che un socio usufruisce di una promozione occorre riportare la relativa data.
La cooperativa gestisce anche *liste di regali* nel seguente modo: una *lista di regali* ha un codice univoco ed una data; una lista viene richiesta da un socio (un socio può richiedere una o più liste). Una lista di regali è composta da uno o più (massimo 100) articoli: per ciascun articolo si riporta la (eventuale) persona che ha scelto tale articolo come regalo con i relativi messaggi augurali. Ad esempio, la lista "LISTANOZZE123" è composta dagli articoli "TVCOLOR" e "DECODER": in tale lista l'articolo "TVCOLOR" viene scelto da "Tizio" che riporta i seguenti messaggi augurali "Auguri e congratulazioni" e "Cento anni di felicita"' mentre l'articolo "DECODER" viene scelto da "Caio" che riporta il seguente messaggio augurale "Tantissimi auguri".
Vincolo aggiuntivo: modificare lo schema E/R, inserendo il vincolo che "uno stesso articolo non può essere offerto in promozione due o più volte dallo stesso negozio.

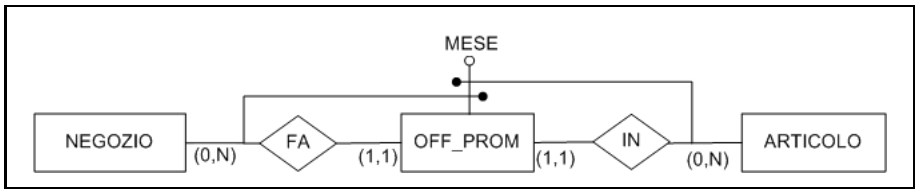
Soluzione

Si riportano di seguito i punti fondamentali della soluzione. Nella rappresentazione di alcune entità gli attributi semplici sono omessi.
1) La gestione delle offerte promozionali può essere modellata in questo modo:

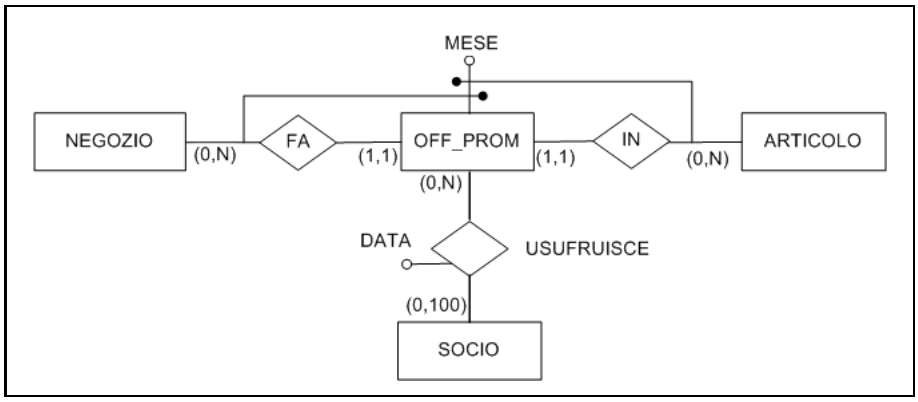
- La chiave mista tra l'attributo `MESE` e l'entità `NEGOZIO` modella la specifica "in *un certo mese*, un *negozio* effettua la promozione di *un solo articolo* ..."



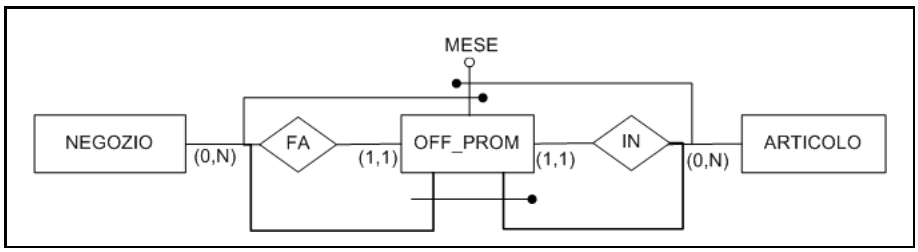
- Analogamente la chiave mista tra `ARTICOLO` e `MESE` modella il fatto che "e, viceversa, in *un certo mese*, un *articolo* è offerto in promozione da *un solo negozio*".



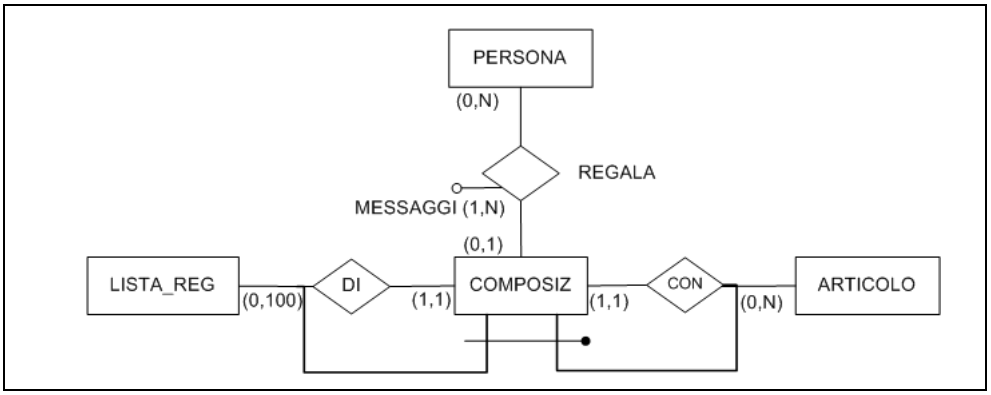
- Infine l'uso di un'offerta promozionale da parte di un socio è rappresentata da un'associazione binaria tra `OFF_PROM` e `SOCIO`. Si noti che la card-max(`SOCIO`,`USUFRUISCE`)= 100 coerentemente con le specifiche.

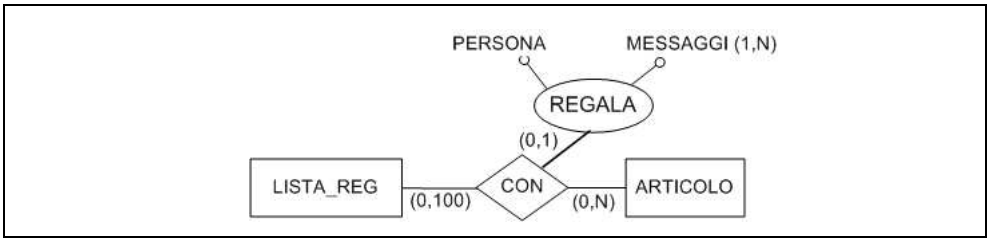


- Il vincolo aggiuntivo può essere poi soddisfatto attraverso un ulteriore identificatore di `OFF_PROM`

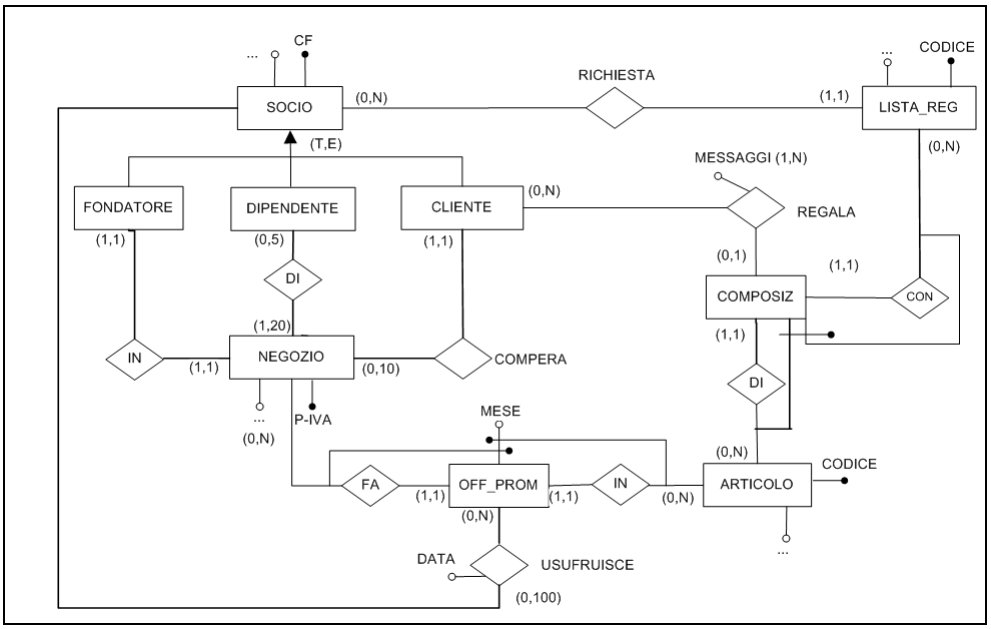


2) Si propongono due diverse rappresentazioni per modellare la gestione della lista di regali. la seconda soluzione è più compatta, ma deve essere trasformata per potere essere rappresentata in relazionale.





3) In figura viene rappresentata la soluzione completa dell'esercizio.



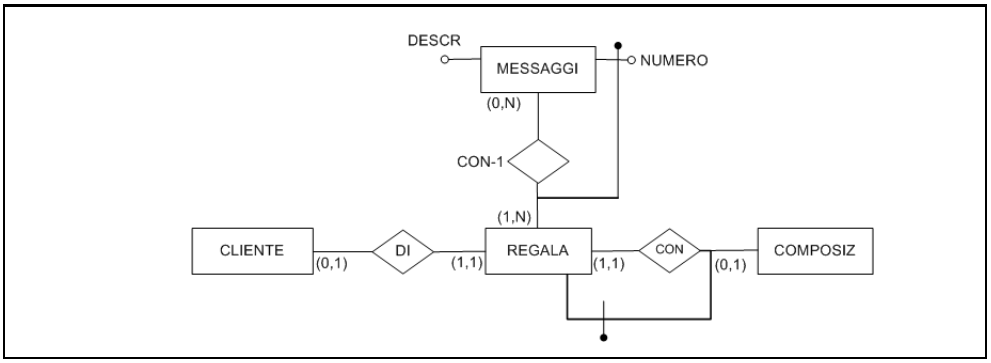
Schema Relazionale

```
SOCIO(CF,DATI_ANAGRAFICI)
FONDATORE(CF,P-IVA)
     FK: CF REFERENCES SOCIO
     FK: P-IVA REFERENCES NEGOZIO
     AK: P-IVA
CLIENTE(CF,P-IVA)
     FK: CF REFERENCES SOCIO
     FK: P-IVA REFERENCES NEGOZIO NOT NULL
DIPENDENTE(CF)
     FK: CF REFERENCES SOCIO
DI(CF,P-IVA)
     FK: CF REFERENCES DIPENDENTE
     FK: P-IVA REFERENCES NEGOZIO
LISTA_REGALI(CODICE,DATA)
NEGOZIO(P-IVA,DENOMINAZIONE,INDIRIZZO,CITTA,REGIONE)
COMPOSIZIONE(CODICE_ART,CODICE_LISTA)
     FK: CODICE_ART REFERENCES ARTICOLO
     FK: CODICE_LISTA REFERENCES LISTA_REGALI
ARTICOLO(CODICE,PREZZO,DESCRIZIONE)
OFF_PROM(P-IVA,MESE,CODICE-ART)
     FK: CODICE-ART REFERENCES ARTICOLO
     FK: P-IVA REFERENCES NEGOZIO
     AK: CODICE-ART,MESE
USUFRUISCE(CF,P-IVA,MESE,DATA)
     FK: P-IVA,MESE REFERENCES OFF_PROM
     FK: CF REFERENCES SOCIO
REGALA(CODICE_ART,CODICE_LISTA,CF)
     FK: CODICE_ART,CODICE_LISTA REFERENCES COMPOSIZIONE
     FK: CF REFERENCES CLIENTE NOT NULL
MESSAGGI(NUMERO,CODICE_ART,CODICE_LISTA,DESCR)
     FK: CODICE_ART,CODICE_LISTA REFERENCES REGALA
```

La gerarchia di SOCIO è stata tradotta mantendo le entità come associazioni. Un eventuale collasso verso il basso avrebbe introdotto ridondanze nella traduzione della associazione RICHIESTA. Un traduzione mediante il collasso verso l'alto avrebbe invece impedito una modellazione esplicita delle associazioni tra NEGOZIO e le sotto-entità di SOCIO.

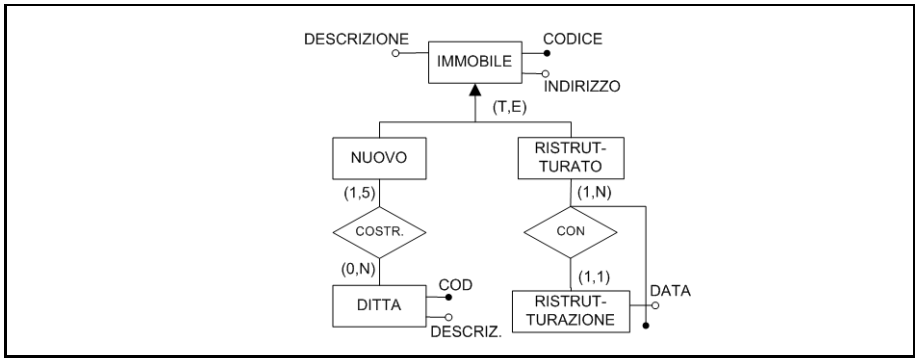
La chiave primaria e la chiave alternativa della relazione OFF_PROM permettono di modellare il vincolo aggiuntivo: la coppia P-IVA,MESE garantisce che per ogni negozio ci sia un solo prodotto in offerta ogni mese, la coppia CODICE-ART,MESE impedisce che un articolo compaia in offerte promozionali di più mesi.

Per la traduzione dell'attributo multiplo `MESSAGGI` è stato necessario reificare l'associazione `REGALA` in una omonima entità identificata dallo stesso identificatore di `COMPOSIZIONE` in modo da mantenere la stessa cardinalità dell'associazione. La figura seguente rappresenta lo schema semplificato che si è tradotto.



♢ **Esercizio 4**
Si vuole rappresentare l'attività di un'agenzia immobiliare che gestisce la vendita di immobili. Per un immobile viene riportato un codice univoco, un indirizzo ed una descrizione. Gli immobili possono essere di nuova costruzione oppure ristrutturati. Per gli immobili di nuova costruzione si riportano le ditte che lo hanno costruito (al massimo 5 ditte; per ogni ditta indicare una descrizione della stessa). Per gli immobili ristrutturati si devono riportare le ristrutturazioni effettuate; in una data, su un immobile può essere fatta una sola ristrutturazione. Per rappresentare lo stato dell'immobile, vengono riportati inoltre tutti gli interventi di riparazione e di ristrutturazione effettuati. Per ogni intervento occorre memorizzare la data in cui è stato effettuato, la ditta che lo ha effettuato, le parti dell'immobile interessate dall'intervento, con i relativi lavori effettuati. La vendita degli immobili avviene tramite atti di vendita. Un atto di vendita riguarda un solo immobile ed è stipulato da una o più persone giuridiche, ciascuna con un proprio ruolo (venditore, acquirente, intermediario, etc.), rispettando i seguenti vincoli: a) su un immobile può essere fatto uno ed un solo atto di vendita; b) una persona giuridica può partecipare al massimo una volta a un singolo atto di vendita, ricoprendo un solo ruolo. Le persone giuridiche sono rappresentate dal codice fiscale, dallo stato giuridico e da una descrizione.
Vincolo aggiuntivo: modificare lo schema ottenuto, inserendo il vincolo che "in un atto di vendita non ci sono due persone giuridiche che ricoprono lo stesso ruolo, cioè c'è una sola persona che vende, una sola persona che acquista.
Soluzione
Si riportano di seguito i punti fondamentali della soluzione. Nella rappresentazione di alcune entità gli attributi semplici sono omessi.
1) Si consideri la modellazione degli immobili:

- "Per gli immobili ristrutturati occorre riportare le ristrutturazioni effettuate; in una data, su un immobile può essere fatta una sola ristrutturazione; l'identificatore della ristrutturazione effettuata è `DATA,RISTRUTTURATO` in quanto, in una data, su un immobile può essere fatta una sola ristrutturazione;



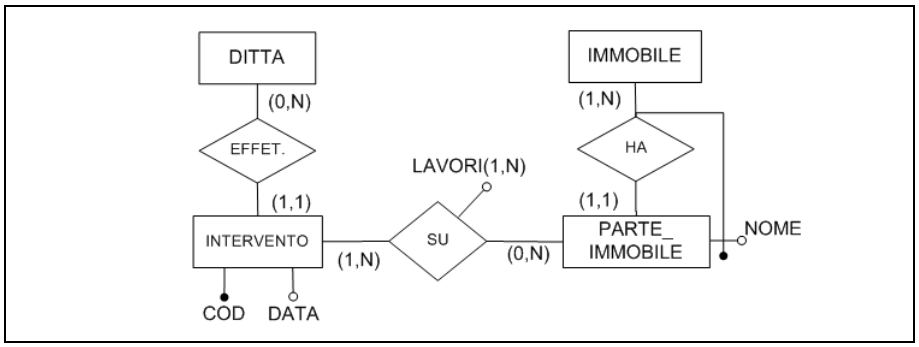
- "Per rappresentare lo stato dell'immobile, vengono riportati inoltre tutti gli interventi di riparazione e di ristrutturazione effettuati."

  Si tratta di una frase ambigua che presuppone che tutti gli immobili abbiano subito delle ristrutturazioni, contraddicendo il punto precedente. In questo caso, occorre considerare unicamente gli interventi di riparazione, ovvero la specifica viene *riscritta* come: "Per rappresentare lo stato dell'immobile, vengono riportati tutti gli interventi di riparazione, chiamati semplicemente *interventi*".

  Avendo l'intervento una descrizione complessa, si ritiene opportuno modellarlo come entità, introducendo un codice identificatore. Un intervento è relativo a una parte dell'immobile. Per rappresentare questa specifica si introduce una nuova entità `PARTE_IMMOBILE`, identificata dal nome della parte (ad esempio, "scalaA", "scalaB", "tetto", ...) e dall'immobile: si suppone che in un immobile ci sia una sola "scalaA", una sola "scalaB", ...

  E' possibile poi collegare `INTERVENTO` a `PARTE_IMMOBILE` tramite una associazione molti-a-molti; tale associazione rappresenta "le parti dell'immobile interessate dall'intervento". Per riportare i " relativi lavori effettuati", si introduce un attributo multiplo su tale associazione.

  Infine, l'entità `INTERVENTO` è poi collegata alla "ditta che lo ha effettuato".

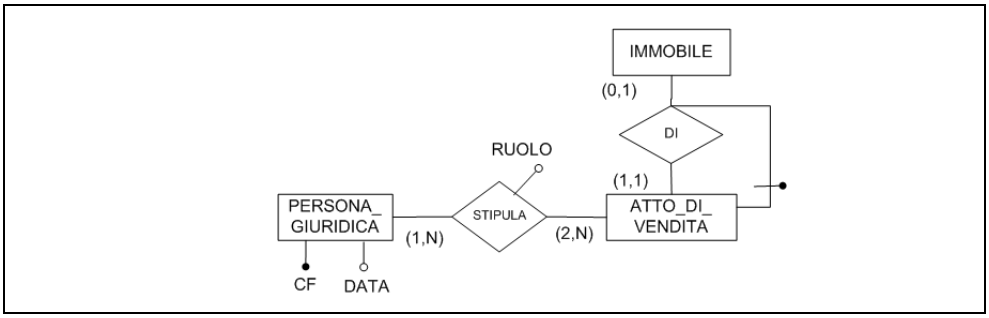


3) Si considerino gli atti di vendita. E' stata introdotta l'entità `ATTO_DI_VENDITA`, collegata ad `IMMOBILE` tramite l'associazione binaria `DI` e collegata a `PERSONA-GIURIDICA` tramite `STIPULA`. Le specifiche del testo impongono i seguenti vincoli:

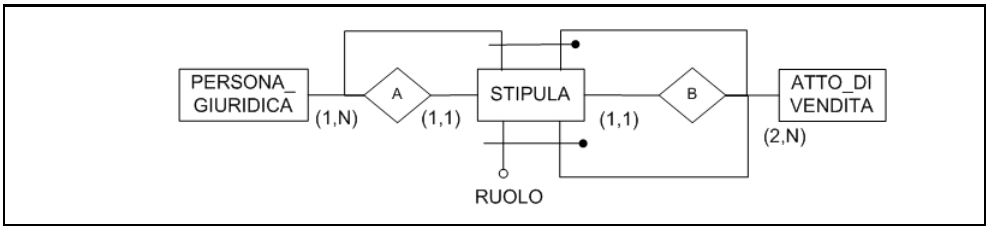
- "su un immobile può essere fatto uno ed un solo atto di vendita": `IMMOBILE` partecipa all'associazione `HA` con (0,1). Inoltre, considerato che un atto di vendita riguarda un solo immobile, l'entità `ATTO_DI_VENDITA` partecipa all'associazione `HA` con cardinalità (1,1). Conseguentemente `ATTO_DI_VENDITA` può essere identificato con `IMMOBILE`. Si noti che

questo equivale a dire che ATTO_DI_VENDITA è una specializzazione di IMMOBILE.

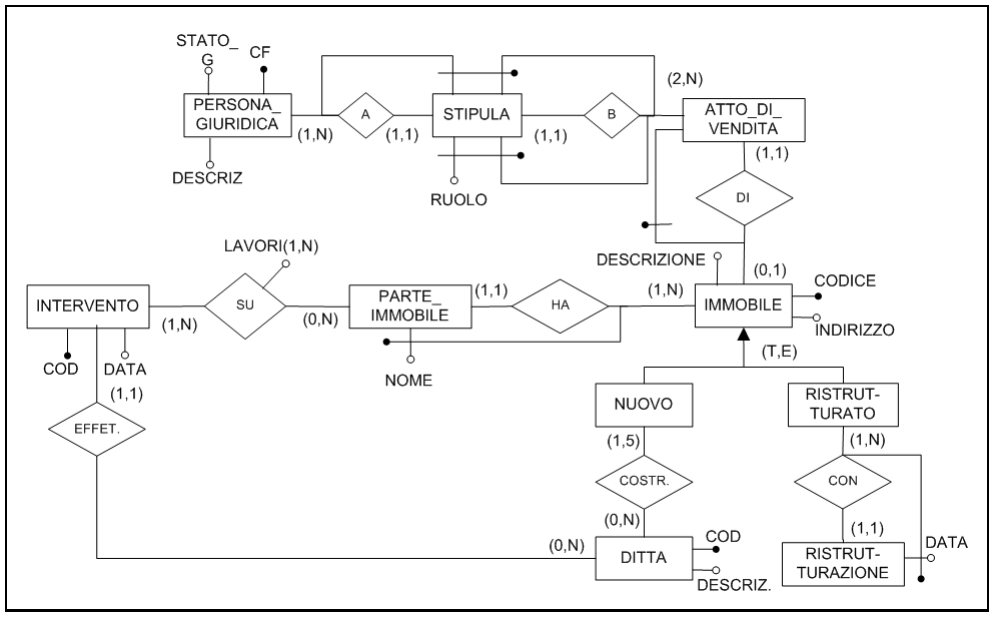
- "in un atto di vendita una persona giuridica può partecipare al massimo una volta, ricoprendo un solo ruolo": l'associazione STIPULA tra ATTO_DI_VENDITA e PERSONA_GIURIDICA è molti-a-molti: infatti "l'atto è stipulato da una o più persone giuridiche, ciascuna con un proprio ruolo"; inoltre si suppone che un atto di vendita sia stipulato tra almeno due persone giuridiche (si può non aggiungere questa ipotesi e considerare ATTO_DI_VENDITA in STIPULA con cardinalità (1,N)). Il vincolo in oggetto è già assicurato dall'associazione binaria STIPULA, in quanto in essa non posso ripetere la stessa coppia atto-persona. Il ruolo viene espresso tramite un semplice attributo RUOLO sull'associazione STIPULA.



4) Si consideri il vincolo aggiuntivo: "in un atto di vendita non ci sono due persone giuridiche che ricoprono lo stesso ruolo, cioè c'è una sola persona che vende, una sola persona che acquista".
Il vincolo implica che ATTO_DI_VENDITA e RUOLO debbano *determinare* la PERSONA_GIURIDICA, quindi devono essere un identificatore di STIPULA; pertanto l'entità STIPULA deve essere reificata e identificata da tali attributi



5) Si propone di seguito una possibile soluzione completa dell'esercizio comprensiva del vincolo aggiuntivo.



Schema Relazionale

```
IMMOBILE(CODICE,INDIRIZZO,DESCRIZIONE)
NUOVO(CODICE)
     FK: CODICE REFERENCES IMMOBILE
RISTRUTTURATO(CODICE)
     FK: CODICE REFERENCES IMMOBILE
RISTRUTTURAZIONE(CODICE,DATA)
     FK: CODICE REFERENCES RISTRUTTURATO
DITTA(CODICE,DESCRIZIONE)
COSTR(COD-DITTA,COD-IMM)
     FK: COD-DITTA REFERENCES DITTA
     FK: COD-IMM REFERENCES NUOVO
PARTE_IMMOBILE(CODICE,NOME)
     FK: CODICE REFERENCES IMMOBILE
INTERVENTO(CODICE,DATA,COD-DITTA)
     FK: COD-DITTA REFERENCES DITTA NOT NULL
SU(COD-INT,COD-IMM, NOME)
     FK: COD-INT REFERENCES INTERVENTO
     FK: CCOD-IMM, NOME REFERENCES PARTE_IMMOBILE
LAVORI(DESCR-LAVORO,COD-INT,COD-IMM, NOME)
     FK: COD-INT,COD-IMM, NOME REFERENCES SU
ATTO_DI_VENDITA(CODICE)
     FK: CODICE REFERENCES IMMOBILE
PERSONA_GIURIDICA(CF,STATO-GIURIDICO,DESCRIZIONE)
STIPULA(CF,CODICE,RUOLO)
     FK: CODICE REFERENCES ATTO_DI_VENDITA
     FK: CF REFERENCES PERSONA_GIURIDICA
     AK: CODICE,RUOLO
```

L'entità `ATTO_DI_VENDITA` è identificata dallo stesso identificatore di `IMMOBILE` coerentemente con quanto rappresentato dallo schema E/R. Tale traduzione è analoga a quella che si avrebbe avuto nel caso di un subset.

♢ **Esercizio 5**
Si intendono memorizzare dati sugli articoli e gli annunci sulle edizioni giornaliere di una testata giornalistica, secondo le seguenti specifiche.
Ogni edizione giornaliera della testata giornalistica, chiamata semplicemente *giornale*, è identificata dal giorno e dall'anno di pubblicazione ed è caratterizzata dal prezzo. Ogni giornale è strutturato in pagine (minimo 10, massimo 20 pagine); una pagina è identificata da un numero univoco all'interno del giornale; per ogni pagina viene riportato il numero di colonne scritte. Ogni articolo è pubblicato su un unico giornale ed è identificato da un codice univoco all'interno del giornale in cui è pubblicato. Un articolo è descritto dal titolo, dal sottotitolo ed dal testo; un articolo può occupare fino a 3 pagine del giornale; una pagina del giornale può contenere fino a 15 articoli.
Ogni articolo è scritto da un solo giornalista; un giornalista può scrivere al massimo un articolo nello stesso giornale. I giornalisti sono descritti dagli usuali dati anagrafici.
Il giornale contiene anche degli annunci che sono descritti da un codice univoco e da un testo. Gli annunci sono di due tipi distinti: avvisi e pubblicità: 1) un avviso è inserito in un solo giornale, in una precisa pagina; su una pagina di un giornale possono essere inseriti fino ad un massimo di 4 avvisi; 2) una pubblicità è inclusa in una pagina del giornale; una stessa pubblicità può essere inclusa più volte nello stesso giornale, ma in pagine differenti; una pagina di un giornale può contiene una ed un sola pubblicità.
Vincolo aggiuntivo 1: modificare lo schema E/R, eliminando il vincolo che "in uno stesso giornale, un giornalista può scrivere al massimo un articolo e considerando che in uno stesso giornale, un giornalista può scrivere al massimo tre articoli".
Vincolo aggiuntivo 2: modificare lo schema E/R, inserendo il vincolo che "una stessa pubblicità può essere inclusa una ed una sola volta nello stesso giornale, in una determinata pagina".

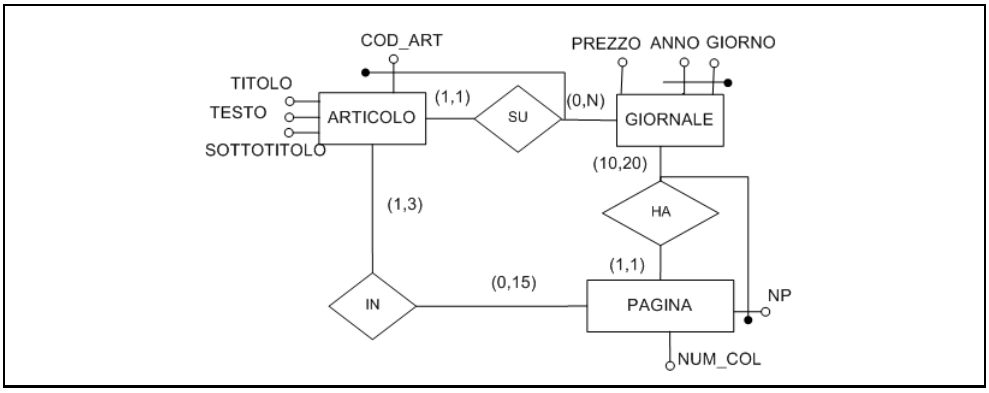
Soluzione

Si riportano di seguito i punti fondamentali della soluzione. Nella rappresentazione di alcune entità gli attributi semplici sono omessi.
1) Si consideri il primo paragrafo, relativo alle pagine ed agli articoli:

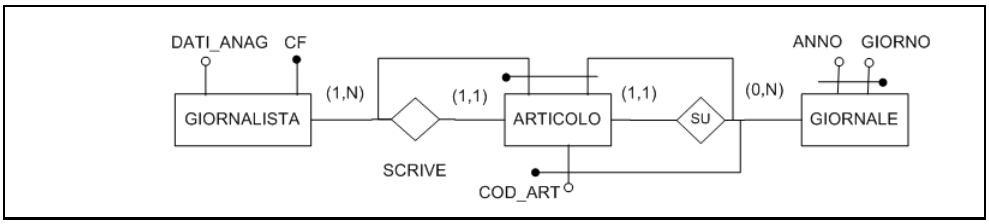
- Sebbene nel testo non sia esplicitato il numero massimo di articoli di un giornale, si può ricavare facilmente (ed indicare nello schema) il valore puntuale (`300`) : infatti una pagina può avere al massimo 15 articoli ed un giornale può avere al massimo 20 pagine. Nella soluzione proposta è possibile quindi indicare max-card(`GIORNALE`,`SU`)=300.

- Nello schema è possibile individuare un *ciclo*: l'entità `ARTICOLO` è collegabile a `GIORNALE` sia direttamente tramite l'associazione `SU` sia in-

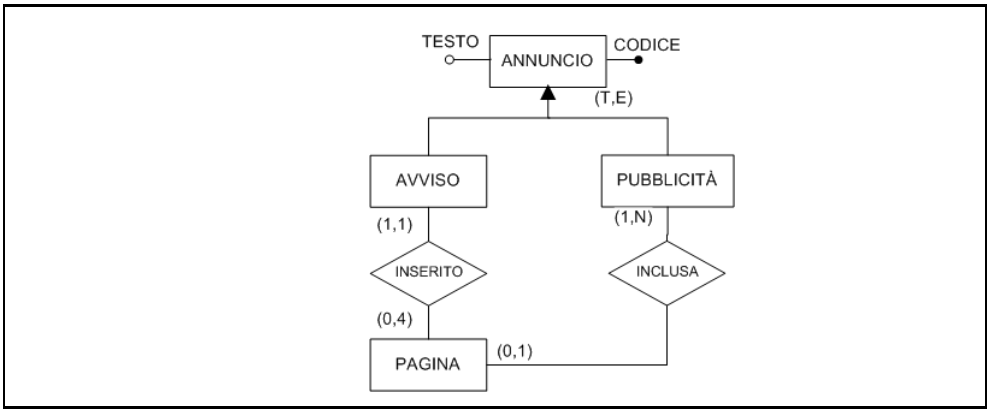
direttamente tramite il percorso `IN - PAGINA - HA`. Si consideri un articolo a, un giornale g e una pagina p; sia a associato tramite `SU` a g; sia a associato tramite `IN` a p: lo schema *non impone il vincolo* che tale pagina p sia associata tramite `HA` allo stesso giornale g, ovvero l'articolo a sul giornale g può essere in una pagina p di un altro giornale.
Non è possibile imporre tale vincolo con la modellazione E/R.



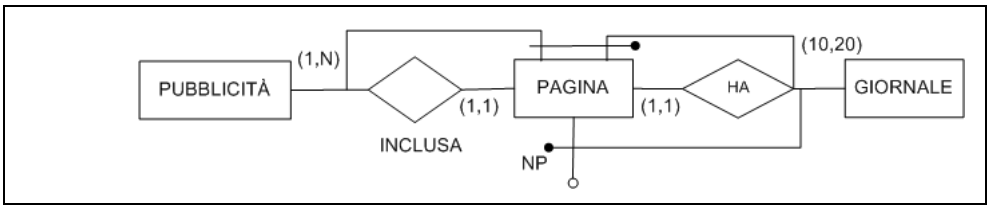
2) Si consideri il vincolo imposto dalla frase "Ogni articolo è scritto da un solo giornalista; un giornalista può scrivere al massimo un articolo nello stesso giornale". Di conseguenza `ARTICOLO` è identificato da `GIORNALISTA` e da `GIORNALE`, quindi è necessario aggiungere un altro identificatore ad `ARTICOLO` ottenendo:



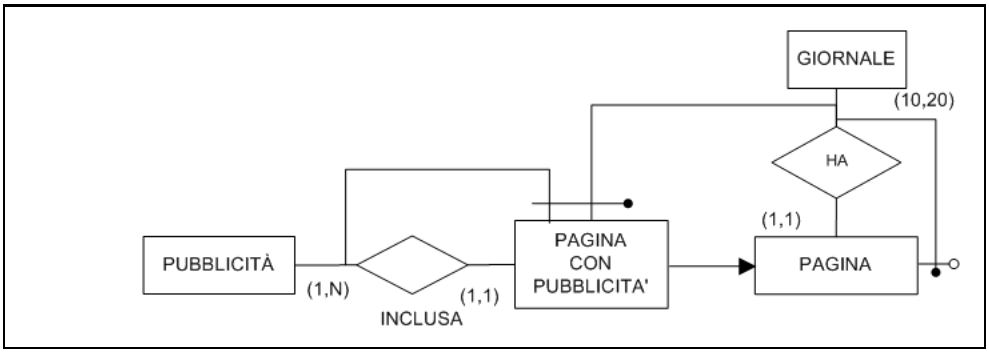
3) Si considerino i vincoli espressi relativamente agli annunci; in particolare "una pubblicità è inclusa in una pagina del giornale; una stessa pubblicità può essere inclusa più volte nello stesso giornale, ma in pagine differenti; una pagina di un giornale contiene una ed un sola pubblicità". Si noti che questa descrizione è in ridondante, infatti se "su una pagina di un giornale può essere inclusa una ed un sola pubblicità" allora ogni "pubblicità" deve essere necessariamente su una pagina differente. Pertanto il vincolo è imposto semplicemente dal fatto che `PAGINA` partecipa al massimo una volta in `INCLUSA` ("su una pagina di un giornale può essere inclusa una ed un sola pubblicità"):



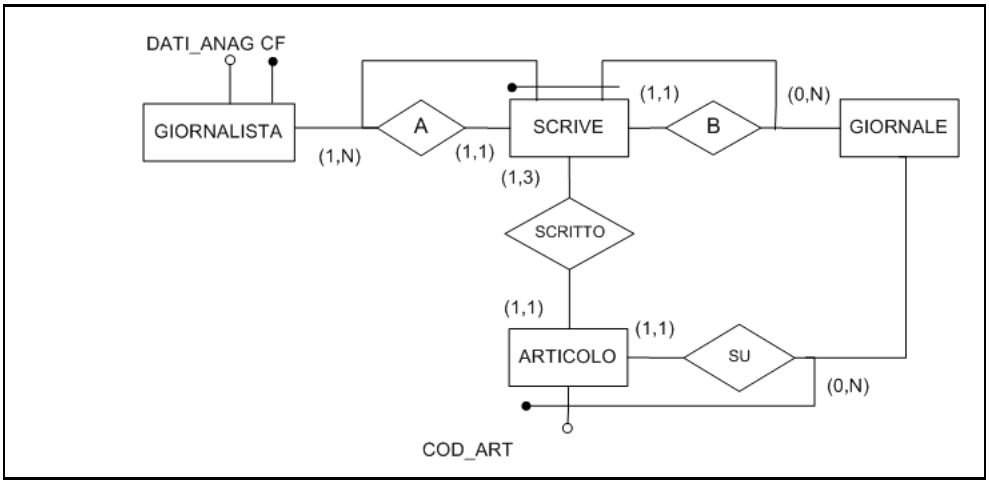
4) Si consideri il vincolo aggiuntivo 2: "una stessa pubblicità può essere inclusa una ed una sola volta nello stesso giornale, in una determinata pagina". Questo significa che data una pubblicità e dato un giornale, si identifica una sola pagina, ovvero che `PAGINA` dovrebbe essere identificata da `GIORNALE` e `PUBBLICITÀ`. Occorre osservare che `PAGINA` è legata a `PUBBLICITÀ` dall'associazione `INCLUSA` con card`(PAGINA,INCLUSA) = (0,1)`, mentre l'identificatore esterno necessita l'obbligatorietà dell'associazione, card`(PAGINA,INCLUSA) = (1,1)`. In altre parole, per identificare una pagina con la pubblicità, tutte le pagine devono avere una pubblicità. La soluzione più semplice è quindi imporre card(`PAGINA,INCLUSA`)= (1,1):



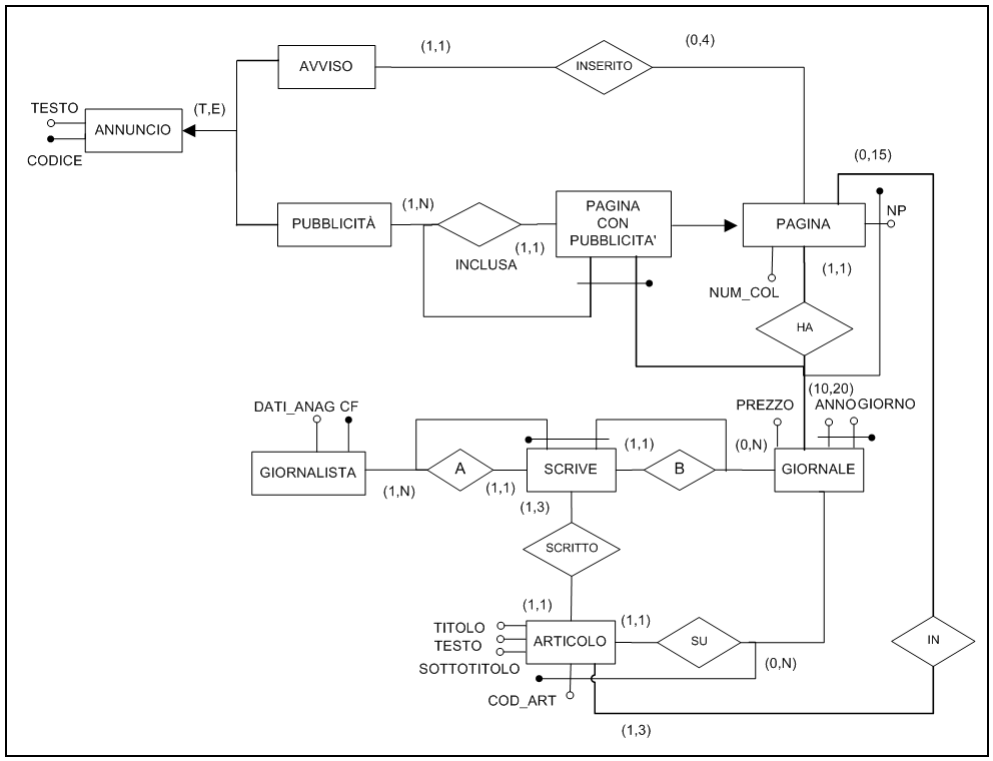
La soluzione proposta è *parziale* perchè non permette di rappresentare le pagine senza pubblicità. Per risolvere questo problema, si può introdurre un *subset* per rappresentare `PAGINA-CON-PUBBLICITA'` e l'identificatore `GIORNALE` e `PUBBLICITÀ` è relativo unicamente a tale entità.
Si noti che l'entità `PAGINA-CON-PUBBLICITA'` può usare `GIORNALE` come componente di identificazione esterna in quanto eredita l'associazione `HA` da `PAGINA`.



5) Si consideri il Vincolo aggiuntivo 1 che indica di eliminare il vincolo "in uno stesso giornale, un giornalista può scrivere al massimo un articolo" e considerando che "in uno stesso giornale, un giornalista può scrivere al massimo tre articoli". Per soddisfare il vincolo si può togliere l'associazione diretta tra articolo e giornalista. Infatti "in uno stesso giornale, un giornalista può scrivere più articoli" quindi non può più essere identificato da `GIORNALISTA` e `GIORNALE`.
Occorre quindi introdurre una nuova entità in modo definire una certa disponibilità giornaliera che consenta a un `GIORNALISTA` di scrivere in un `GIORNALE` fino a tre articoli. Chiamando tale disponibilità giornaliera semplicemente `SCRIVE` si ottiene il seguente schema:



6) Lo schema completo che si ottiene considerando anche il vincolo aggiuntivo è il seguente:



Schema Relazionale

ANNUNCIO(CODICE,TESTO)
AVVISO(CODICE)
FK: CODICE **REFERENCES** ANNUNCIO
PUBBLICITA(CODICE)
FK: CODICE **REFERENCES** ANNUNCIO
PAGINA_PUBBLICITA(NP,ANNO,GIORNO,CODICE)
FK: NP,ANNO,GIORNO **REFERENCES** PAGINA
FK: CODICE **REFERENCES** PUBBLICITA
AK: CODICE,ANNO,GIORNO
PAGINA(NP,ANNO,GIORNO,NUM_COL)
FK: ANNO,GIORNO **REFERENCES** GIORNALE
GIORNALISTA(CF,DATI_ANAGRAFICI)
SCRIVE(CF,ANNO,GIORNO)
FK: ANNO,GIORNO **REFERENCES** GIORNALE
FK: CF **REFERENCES** GIORNALISTA
GIORNALE(ANNO,GIORNO,PREZZO)
ARTICOLO(COD_ART,ANNO,GIORNO,TITOLO,SOTTOTITOLO,TESTO,CF)
FK: ANNO,GIORNO **REFERENCES** GIORNALE NOT NULL
FK: CF,ANNO,GIORNO **REFERENCES** SCRIVE NOT NULL
IN(COD_ART,ANNO,GIORNO,NP)
FK: COD_ART,ANNO,GIORNO **REFERENCES** GIORNALE
FK: NP,ANNO,GIORNO **REFERENCES** PAGINA

Si noti che lo schema E/R mediante l'associazione IN non riesce a esprimere il vincolo che l'articolo di un giornale sia incluso in una pagina appartenente allo stesso giornale dell'articolo. Questa limitazione appare evidente dalla traduzione standard dell'associazione IN:

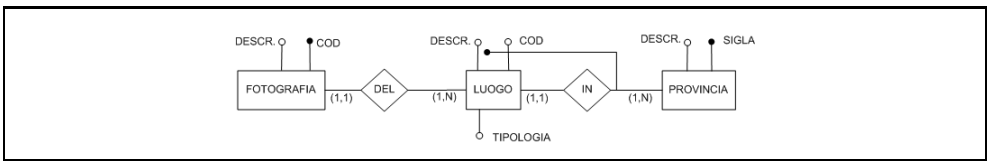
IN(COD_ART,ANNO-GIORNALE,GIORNO-GIORNALE,NP,
ANNO-PAGINA,GIORNO-PAGINA)
FK: COD_ART,ANNO-GIORNALE,GIORNO-GIORNALE
REFERENCES GIORNALE
FK: NP,ANNO-PAGINA,GIORNO-PAGINA **REFERENCES** PAGINA

La soluzione che si è proposta utilizza una sola coppia di attributi ANNO,GIORNO implementando in questo modo la condizione che l'articolo che appare in un giornale sia contenuto in una pagina dello stesso giornale. Un analogo ragionamento è stato fatto per la traduzione della foreign key su SCRIVE dell'entità ARTICOLO.

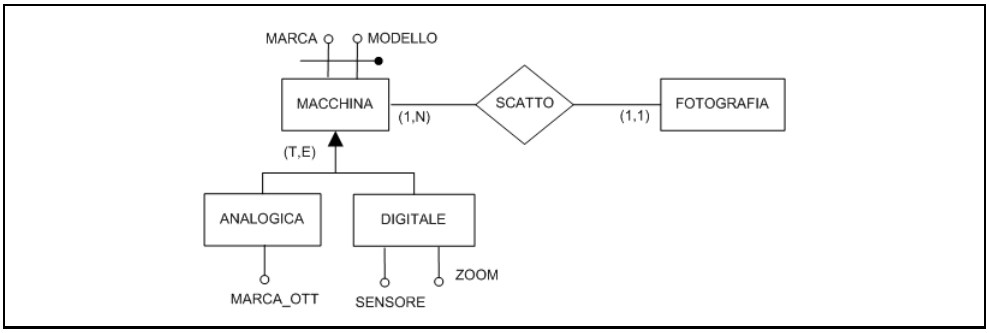
♢ Esercizio 6

Un sistema informativo memorizza informazioni su un archivio di fotografie, secondo le seguenti specifiche.
Le fotografie hanno un codice identificativo e una descrizione. Una fotografia è relativa ad un luogo. Ogni luogo, che ha una descrizione ed una tipologia, è identificato da un codice che è univoco all'interno della provincia alla quale il luogo appartiene. La provincia ha una sigla e una descrizione.
Una fotografia è scattata con una macchina fotografica, identificata dalla marca e dal modello, che può essere analogica o digitale. Per le macchine digitali occorre indicare la capacità del sensore e la potenza dello zoom ottico; per le macchine analogiche la marca dell'ottica montata .
Una fotografia fa parte di un servizio fotografico. Un servizio fotografico è realizzato da un fotografo, ha una data ed una descrizione; in una certa data, un fotografo può realizzare un solo servizio fotografico; un servizio fotografico contiene da 5 a 100 fotografie.
Il sistema informativo permette l'accesso ad utenti che possono esprimere un voto per le fotografie, con il vincolo che un utente voti una sola volta per una fotografia. Ogni voto ha la data in cui viene espresso e può essere corredato da più motivazioni.
I fotografi e gli utenti sono descritti dagli usuali dati anagrafici; inoltre, gli utenti hanno anche un nomeutente, ovviamente unico.
Vincolo aggiuntivo 1: modificare lo schema E/R aggiungendo il vincolo: che "in una certa data, un utente può esprimere al massimo un solo voto."
Vincolo aggiuntivo 2: modificare lo schema E/R aggiungendo il vincolo: "Il sistema informativo permette anche a critici di giornali di effettuare recensioni sui servizi fotografici. Un critico è caratterizzato dagli usuali dati anagrafici. Il critico scrive una recensione di servizio fotografico che può fare riferimento ad altre recensioni.

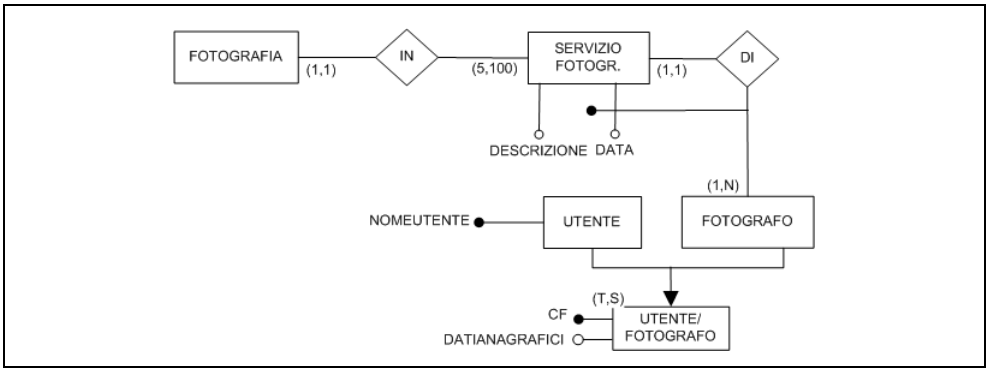
Soluzione
Si riporta la discussione relativa ai soli punti fondamentali; in alcune entità gli attributi semplici sono omessi.
1) "Le fotografie hanno un codice identificativo e una descrizione. Una fotografia è relativa ad un luogo. Ogni luogo, che ha una descrizione ed una tipologia, è identificato da un codice che è univoco all'interno della provincia alla quale il luogo appartiene. La provincia ha una sigla e una descrizione". Conviene introdurre *PROVINCIA* come entità in quanto ha un identificatore (sigla) ed una descrizione.



2) Attraverso una gerarchia è possibile rappresentare le macchine fotografiche.

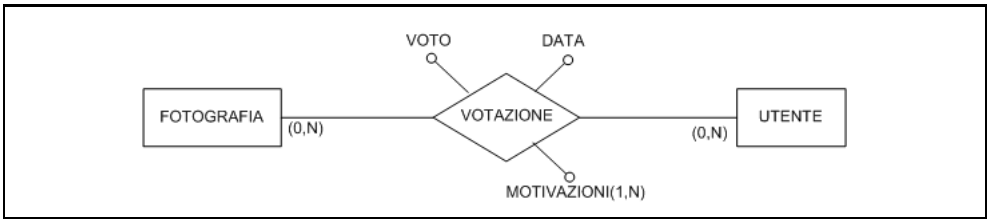


3) "Una fotografia fa parte di un servizio fotografico. Un servizio fotografico è realizzato da un fotografo, ha una data ed una descrizione; in una certa data, un fotografo può realizzare un solo servizio fotografico; un servizio fotografico contiene da 5 a 100 fotografie. I fotografi e gli utenti sono descritti dagli usuali dati anagrafici; inoltre, gli utenti hanno anche un nomeutente, ovviamente unico." Per rappresentare i fotografi e gli utenti si introduce una entità comune `UTENTE/FOTOGRAFO`. Tale entità è specializzata in `UTENTE` (identificata anche da `nomeutente`) e in `FOTOGRAFO` entità che pur non avendo attributi propri è utilizzata nella descrizione del servizio fotografico.
L'entità `UTENTE/FOTOGRAFO` può essere anche chiamata `PERSONA`: il fatto che i membri della classe PERSONA siano fotografi ed utenti è indicato dalla gerarchia di generalizzazione.



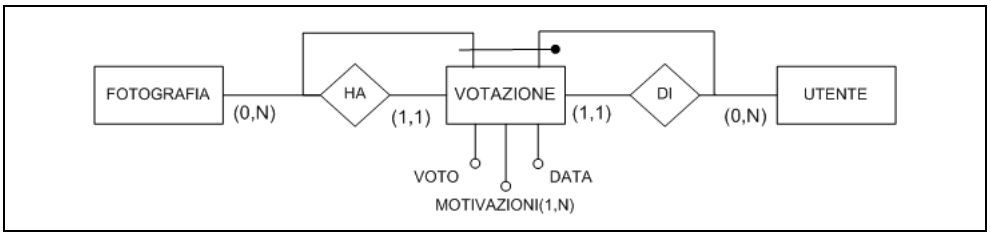
4) "Il sistema informativo permette l'accesso ad utenti che possono esprimere un voto per le fotografie, con il vincolo che un utente voti una sola volta per una fotografia. Ogni voto ha la data in cui viene espresso e può essere corredato da più motivazioni." Il voto riguarda un utente ed una fotografia, quindi viene espresso tramite l'associazione binaria `VOTAZIONE` tra `UTENTE` e `FOTOGRAFIA`. Si noti che tale associazione descrive solo il legame logico tra una foto ed un utente (ovvero solo il fatto che un utente ha votato per una

certa foto); per esprimere il *valore del voto* occorre mettere un attributo `voto` per tale associazione; inoltre si aggiunge su tale associazione un attibuto `data` e `motivazione` (come attributo multiplo). Si può assumere che `VOTAZIONE` rappresenti un'associazione *molti-a-molti*: un utente vota più fotografie e, viceversa, su una foto verranno fatte più votazioni; assumiamo inoltre che entrambe le entità partecipino in modo opzionale.

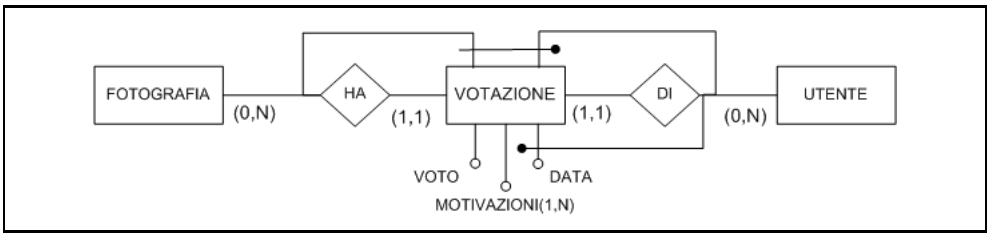


L'associazione binaria `VOTAZIONE` esprime implicitamente il vincolo che "un utente voti una sola volta per una fotografia": è inutile reificare tale associazione.

5) Il vincolo aggiuntivo 1 è equivalente alla specifica "in una certa data, un utente può effettuare al massimo una votazione", ovvero impone il fatto che `VOTAZIONE` sia identificata da `data` e da `UTENTE`. E' quindi necessario reificare l'associazione (si noti che per definizione l'identificatore di `VOTAZIONE` è `UTENTE` e `FOTOGRAFIA`: tale identificatore deve restare perchè deve continuare ad essere valido il vincolo che "un utente voti una sola volta per una fotografia"):

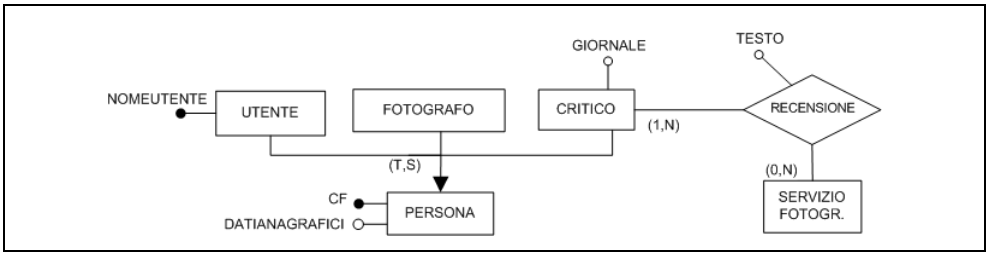


all'identificatore proposto è necessario aggiungere un altro identificatore formato da `data` e da `UTENTE`.

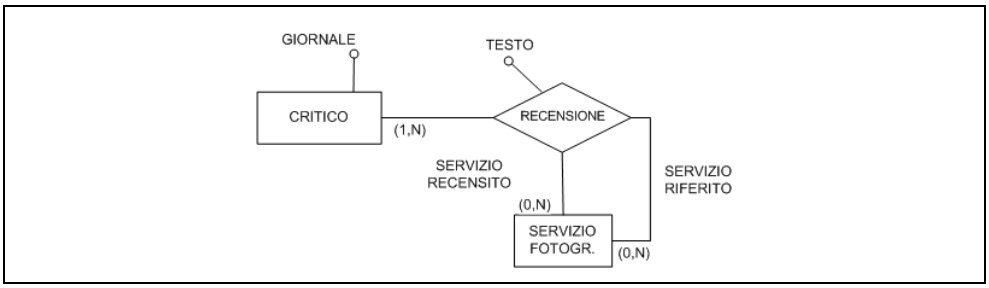


6) Vincolo aggiuntivo 2: "Il sistema informativo permette anche a critici di giornali di effettuare recensioni sui servizi fotografici. Un critico è caratterizzato dagli usuali dati anagrafici. Il critico scrive una recensione di servizio fotografico che può fare riferimento ad altre recensioni."

Il critico può essere rappresentato come ulteriore specializzazione dell'entità `UTENTE/FOTOGRAFO` che a questo punto dovrà essere denominata `PERSONA`. L'entità `CRITICO` ha come attributo il giornale (si noti che non è necessario rappresentare il giornale come un'entità, in quanto nel testo non viene richiesto di rappresentare alcuna proprietà del giornale). Le recensioni sono associazioni binarie tra `SERVIZIO/FOTOGRAFICO` e `CRITICO`; si può assumere che tale associazione sia *molti-a-molti*: un critico scrive più di una recensione e, viceversa, su un servizio fotografico sono scritte più recensioni. Inoltre, anche se non richiesto esplicitamente, all'associazione *RECENSIONE* occorre aggiungere l'attributo `testo` che rappresenta il testo della recensione.

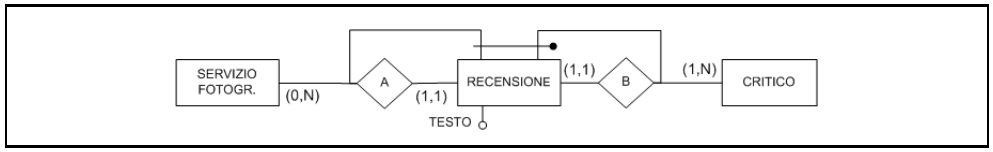


Lo schema deve essere modificato per rappresentare che "una recensione di un servizio fotografico può fare riferimento ad altre recensioni"; in altre parole si vuole rappresentare una sorta di bibliografia della recensione, nella quale vengono citate altre recensioni. Si noti che i riferimenti sono ad altre recensioni e non ad altri servizi fotografici, quindi una associazione ternaria come quella in figura non è corretta.

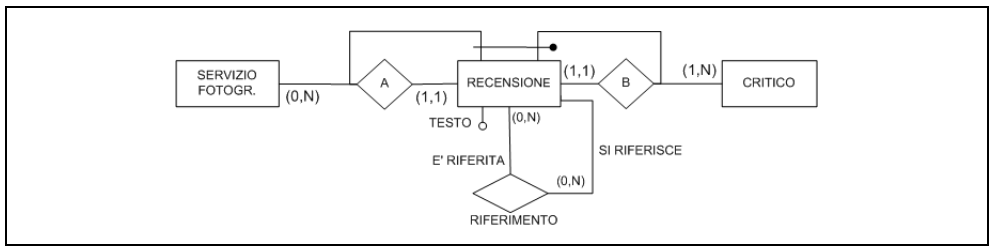


Inoltre con tale associazione ternaria, per scrivere una recensione su un articolo fotografico si deve necessariamente fare un riferimento ad un altro servizio fotografico.

Per modellare una soluzione corretta occorre partire dal fatto che un riferimento è un legame logico tra due recensioni, quindi deve essere espresso come una associazione tra `RECENSIONE` e se stessa, ovvero come un anello su `RECENSIONE`. Con il modello E/R non è possibile definire un'associazione tra associazioni, cioè non è possibile modellare una associazione tra `RECENSIONE` e se stessa. Occorre quindi reificare l'associazione `RECENSIONE`



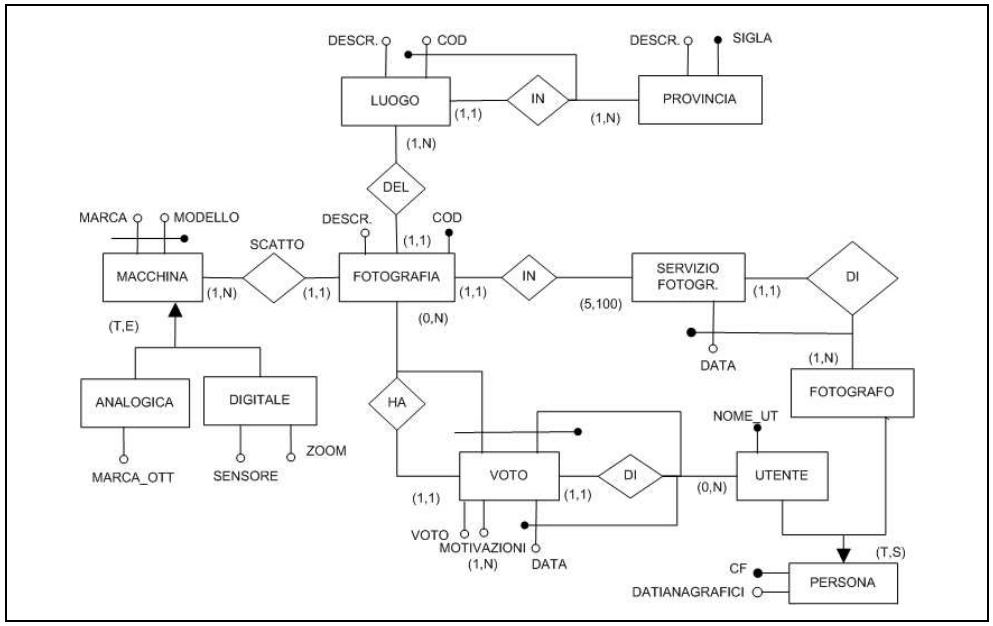
e conseguentemente definire l'anello su `RECENSIONE`:



Si osservi che:

- `card-min(E'-RIFERITA, RIFERIMENTO) = 0` : una recensione può non essere citata
- `card-max(E'-RIFERITA, RIFERIMENTO) = N` : una recensione può essere citata più volte
- `card-min(SI-RIFERISCE, RIFERIMENTO) = 0` : una recensione può non citare altre recensioni (non ha bibliografia)
- `card-max(SI-RIFERISCE, RIFERIMENTO) = N` : una recensione può citare più recensioni.

7) Nello schema che segue è modellato unicamente il vincolo aggiuntivo 1:



Schema Relazionale

```
MACCHINA(MARCA,MODELLO)
ANALOGICA(MARCA,MODELLO,MARCA_OTTICA)
     FK: MARCA,MODELLO REFERENCES MACCHINA
DIGITALE(MARCA,MODELLO,SENSORE,ZOOM)
     FK: MARCA,MODELLO REFERENCES MACCHINA
PROVINCIA(SIGLA,DESCRIZIONE)
LUOGO(CODICE,SIGLA,DESCRIZIONE)
     FK: SIGLA REFERENCES PROVINCIA
FOTOGRAFIA(COD_FOTO,DESCRIZIONE,MARCA,MODELLO,
                CODICE-L,PROVINCIA,DATA,CF)
     FK: MARCA,MODELLO REFERENCES MACCHINA NOT NULL
     FK: DATA,CF REFERENCES SERVIZIO_FOTO NOT NULL
     FK: CODICE-L,PROVINCIA REFERENCES LUOGO NOT NULL
SERVIZIO_FOTO(DATA,CF,DESCRIZIONE)
     FK: CF REFERENCES FOTOGRAFO
PERSONA(CF,DATI)
     FK: CF REFERENCES DIPENDENTE
FOTOGRAFO(CF)
     FK: CF REFERENCES PERSONA
UTENTE(CF,NOMEUTENTE)
     FK: CF REFERENCES PERSONA
     AK: NOMEUTENTE
VOTO(CODICE-FOTO,CF,DATA,VOTO)
     FK: CF REFERENCES UTENTE
     FK: CODICE-FOTO REFERENCES FOTOGRAFIA
     AK: CF,DATA
VOTO(CODICE-FOTO,CF,DATA,VOTO)
     FK: CF REFERENCES UTENTE
     FK: CODICE-FOTO REFERENCES FOTOGRAFIA
     AK: CF,DATA
```

La gerarchia MACCHINA è stata tradotta mantenendo le entità con associazioni. Sarebbe stata possibile una traduzione mediante il collasso verso l'alto, considerato che le sottoentità differiscono unicamente per la presenza di alcuni attributi.
L'associazione IN tra le entità FOTOGRAFIA e SERVIZIO_FOTO viene tradotta all'interno della relazione FOTOGRAFIA. Il vincolo espresso dala cardinalità card(SERVIZIO_FOTO,IN)=(5,100) non può essere espresso in maniera analoga in relazionale. L'associazione viene trasformata in una generica associazione uno a molti.

♢ Esercizio 7

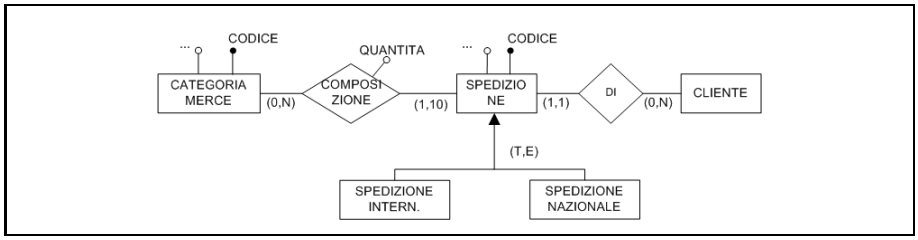
Una società di spedizioni nazionali ed internazionali intende creare un sistema informativo secondo le seguenti specifiche.
Le categorie di merce trattate dall'agenzia sono descritte da un codice univoco, da un tipo e da un peso.
Una spedizione è descritta da un codice univoco, da una data di richiesta e da una descrizione. Una spedizione è richiesta da un cliente; un cliente può richiedere più spedizioni. Per ogni spedizione si deve riportare la sua composizione, ovvero elenco delle categorie di merce spedite con la relativa quantità. In una composizione ci possono essere fino ad un massimo di 10 categorie. Le spedizioni sono di due tipi distinti: nazionali ed internazionali. Per le spedizioni nazionali, la società organizza trasporti giornalieri con vetture ed autisti propri: ogni trasporto è descritto dalla data di trasporto, dalla località di destinazione, dalla vettura con la quale viene effettuato e dall'autista; in una certa data di trasporto, un autista guida una sola vettura e, viceversa, una vettura è guidata da un solo autista. Una spedizione nazionale è effettuata tramite un trasporto giornaliero; tramite un trasporto giornaliero si possono effettuare fino ad un massimo di 20 spedizioni.
Per le spedizioni internazionali la società utilizza mezzi di trasporto, quali ad esempio treno, aereo, nave e corriere. Una spedizione internazionale è costituita da *tratte*, ciascuna delle quali è identificata da un numero univoco all'interno della spedizione ed ha una località di partenza ed un mezzo di trasporto.
I clienti e gli autisti sono descritti dagli usuali dati anagrafici; per i clienti si riporta anche la eventuale percentuale di sconto; per gli autisti invece si riportano informazioni sulla patente di guida. Le località, nazionali ed internazionali, sono descritte da una sigla univoca, da un nome, dallo Stato/Provincia e dalla nazione.
Vincolo Aggiuntivo 1: modificare lo schema E/R, inserendo il vincolo che "in una certa data di trasporto, non ci possono essere due o più trasporti giornalieri nella stessa località di destinazione".
Vincolo Aggiuntivo 2: modificare lo schema E/R, inserendo il vincolo che "in una certa spedizione internazionale, non ci possono essere due o più tratte con la stessa località di partenza".

Soluzione

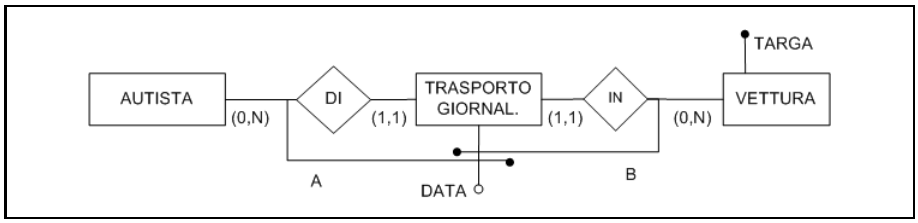
Si riportano di seguito i punti fondamentali della soluzione. Nella rappresentazione di alcune entità gli attributi semplici sono omessi.

1) Le diverse spedizioni sono modellate attraverso una gerarchia. Si noti che nella composizione si indica la categoria della merce, non il singolo articolo spedito: se in una spedizione ho 5 Alberi di Natale, si dirà che

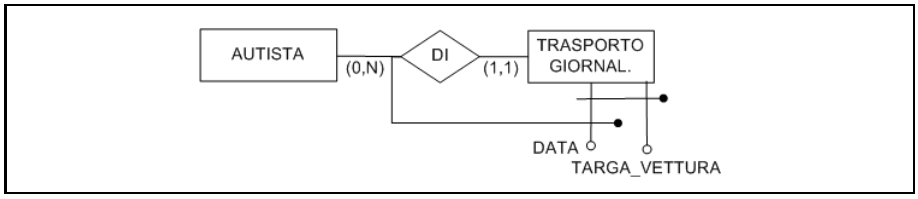
la spedizione è composta da merce di categoria "Legno21" con quantità 5.



2) Attraverso un doppio identificatore sull'entità `TRASPORTO-GIORNALIERO` è possibile modellare il vincolo che in una certa data un autista guidi una sola vettura (identificatore A) e che in una certa data una vettura sia guidata da un solo autista (identificatore B).

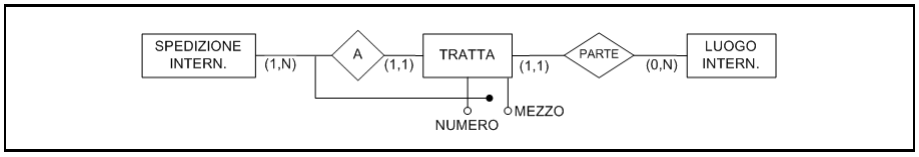


Non essendo riportate altre specifiche di vettura, si può semplicemente riportare la vettura come attributo di `TRASPORTO GIORNALIERO`:

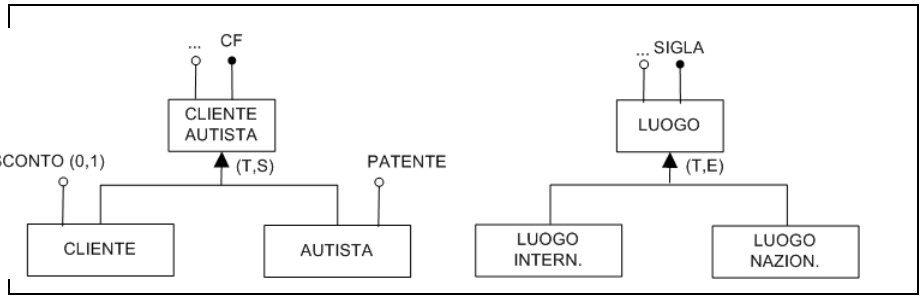


3) Nello schema proposto si introduce la rappresentazione della località di destinazione; coerentemente con le specifiche, la cardinalità massima di `TRASPORTO-GIORNALIERO` nella associazione `IN` è uguale a 20.

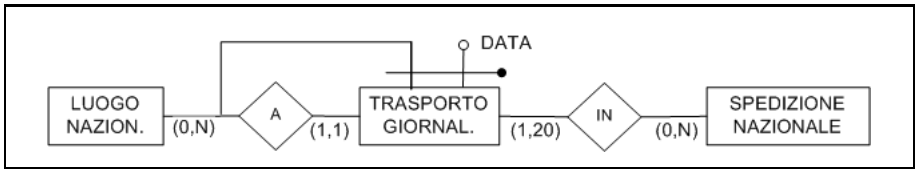
4) Le spedizioni internazionali sono modellate dallo schema seguente: si noti che ogni tratta è identificata da un numero univoco all'interno della spedizione coerentemente con le specifiche.



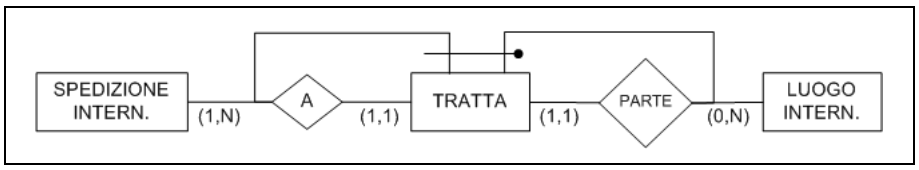
5) Attraverso due gerarchie è possibile rappresentare i clienti/autisti e i luoghi. Per la gerarchia cliente/autista, si è ipotizzata una copertura di sovrapposizione non essendo espresso nulla nel testo.



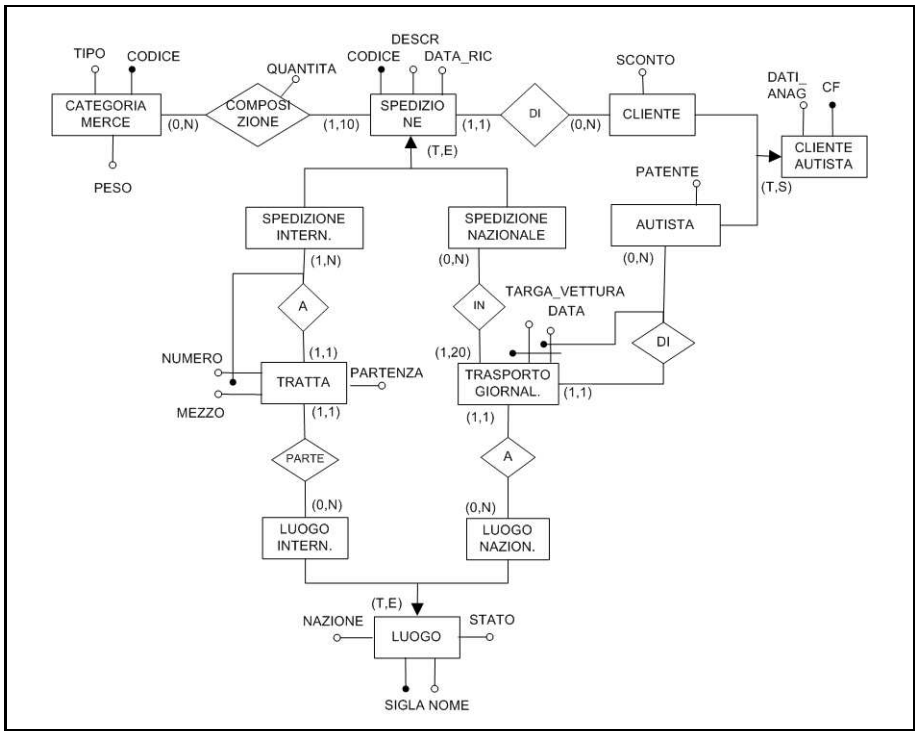
6) Il vincolo aggiuntivo 1 è implementabile imponendo come identificatore di `TRASPORTO-GIORNALIERO`la data e il luogo nazionale di spedizione.



7) Il vincolo aggiuntivo 2 è modellabile imponendo la spedizione internazionale e il luogo internazionale come identificatore di tratta.



8) Lo schema che segue rappresenta la soluzione completa ottenuta senza l'applicazione dei vincoli aggiuntivi.



Schema Relazionale

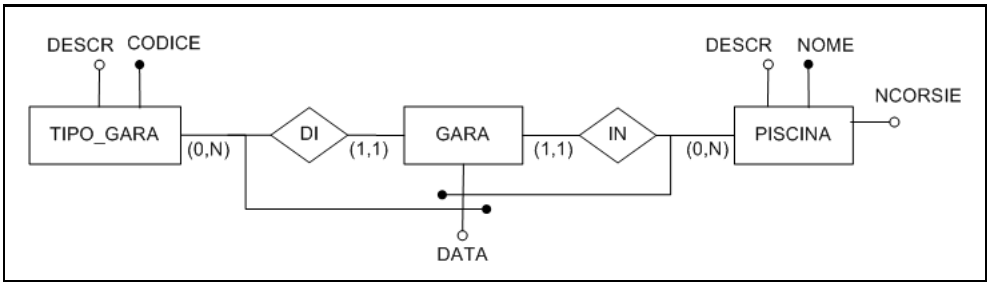
CLIENTE/AUTISTA(CF,DATI-ANAG)
CLIENTE(CF,SCONTO)
FK: CF **REFERENCES** CLIENTE/AUTISTA
AUTISTA(CF,PATENTE)
FK: CF **REFERENCES** CLIENTE/AUTISTA
SPEDIZIONE(CODICE,CF,DESCRIZIONE,DATA_RICHIESTA)
FK: CF **REFERENCES** CLIENTE NOT NULL
INTERNAZIONALE(CODICE)
FK: CODICE **REFERENCES** SPEDIZIONE
NAZIONALE(CODICE)
FK: CODICE **REFERENCES** SPEDIZIONE
CATEGORIA-MERCE(CODICE-M,TIPO,PESO)
COMPOSIZIONE(CODICE-M,CODICE-S,QUANTITA)
FK: CODICE-S **REFERENCES** SPEDIZIONE
FK: CODICE-M **REFERENCES** CATEGORIA-MERCE
TRATTA(NUMERO,CODICE,PARTENZA,MEZZO,LUOGO-I)
FK: CODICE **REFERENCES** INTERNAZIONALE
FK: LUOGO-I **REFERENCES** LUOGO-INTERNAZIONALE NOT NULL
TRASPORTO-GIORNALIERO(DATA,TARGA,CF,LUOGO-N)
FK: CF **REFERENCES** AUTISTA
FK: LUOGO-N **REFERENCES** LUOGO-NAZIONALE NOT NULL
AK: DATA,CF
IN(CODICE,DATA,TARGA)
FK: CODICE **REFERENCES** NAZIONALE
FK: DATA,TARGA **REFERENCES** TRASPORTO-GIORNALIERO
LUOGO(SIGLA,NOME,STATO/PROVINCIA,NAZIONE)
LUOGO-NAZIONALE(SIGLA)
FK: SIGLA **REFERENCES** LUOGO
LUOGO-INTERNAZIONALE(SIGLA)
FK: SIGLA **REFERENCES** LUOGO

♢ Esercizio 8

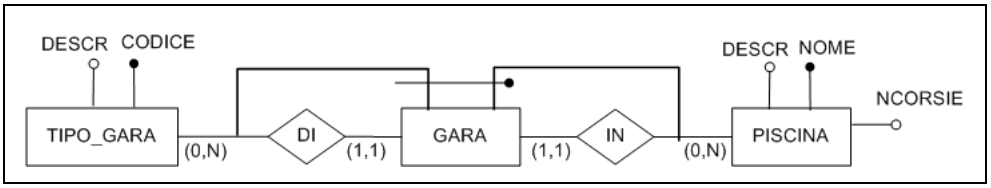
La federazione nazionale nuoto vuole memorizzare dati sulle piscine, sulle gare di nuoto e sugli atleti secondo le seguenti specifiche.
Una piscina è rappresentata da un nome (univoco), una descrizione e da un numero di corsie.
Le tipologie di gare (200 st. libero, 50 dorso, ...) sono rappresentate da un codice univoco e da una descrizione.
Per ogni gara di nuoto si riporta la sua tipologia e la data in cui si svolge: non si possono svolgere due gare della stessa tipologia nella stessa data.
Ogni gara si svolge in una ed una sola piscina; in una certa data in una piscina non si può svolgere più di una gara.
Per ogni gara devono essere riportati gli atleti partecipanti: ad una gara partecipano da uno a otto atleti; un atleta può partecipare a più gare. Per ogni partecipazione di un atleta ad una gara si deve indicare il numero della corsia occupata (ovviamente, in una gara, un atleta è in una ed una sola corsia e, viceversa, in una corsia c'è uno ed un solo atleta) e la posizione finale (in una gara, un atleta ha una ed una sola posizione finale ma una posizione finale può essere occupata da più atleti).
Le gare possono fare parte di campionati. Un campionato è descritto da un codice univoco, un anno ed un tipo. Una gara fa parte di uno ed un solo campionato; il campionato prevede un minimo di quindici competizioni.
Ogni gara è arbitrata da uno ed un solo arbitro; un arbitro può arbitrare più competizioni.
Atleti ed arbitri sono descritti dagli usuali dati anagrafici. Per gli atleti vengono riportate informazioni sulla carriera nel seguente modo: per ogni atleta e ogni tipologia di gara viene riportato il tempo migliore (record) ottenuto.
Vincolo aggiuntivo 1: modificare lo schema E/R aggiungendo il seguente vincolo: "un arbitro può arbitrare più competizioni ma non nello stesso campionato, ovvero in un certo campionato, un arbitro può arbitrare una ed una sola gara".
Vincolo aggiuntivo 2 : modificare lo schema E/R , **togliendo** il vincolo che "in una certa data in una piscina non si può svolgere più di una gara e considerando che in una certa data in una piscina si svolgono da una a tre gare.

Soluzione

Si riportano di seguito i punti fondamentali della soluzione. Nella rappresentazione di alcune entità gli attributi semplici sono omessi.
1) Un doppio identificatore sull'entità GARA garantisce che in ogni piscina si svolga una gara al giorno e, allo stesso tempo, che per ogni giorno si svolga un solo tipo di gara.

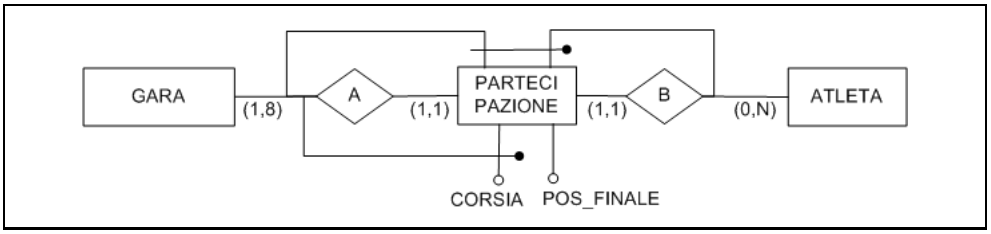


Si noti che aggiungere il seguente identificatore a GARA :

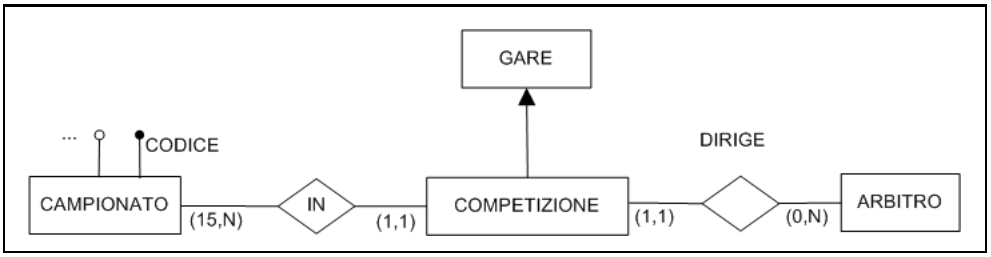


è errato in quanto si impone il fatto che in una piscina un tipo di gara si possa svolgere una sola volta.

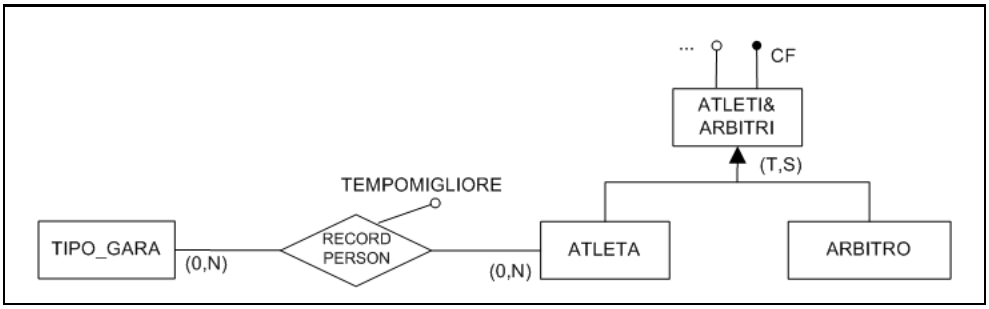
2) La gestione della partecipazione degli atleti alle gare è modellata attraverso l'entità PARTECIPAZIONE. Tale entità presenta due indentificatori garantendo che un atleta non partecipi due volte alla stessa gara e che una corsia della piscina per una gara sia impegnata una sola volta.



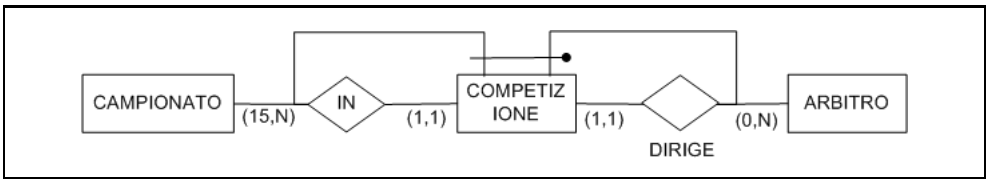
3) Attraverso un subset di GARE vengono modellate le competizioni



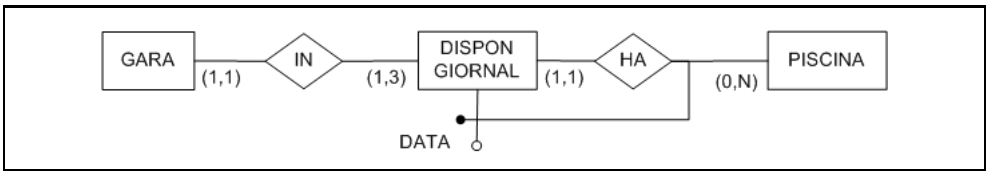
4) Gli atleti e gli arbitri sono rappresentati con una gerarchia. La gerarchia è sovrapposta in quanto non si esclude che un arbitro partecipi a una gara.



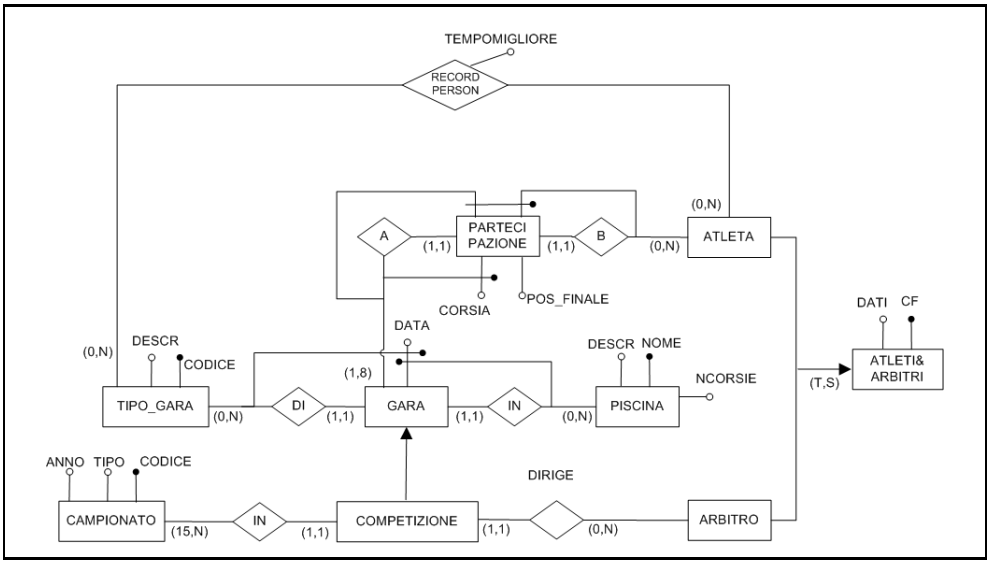
5) Il vincolo aggiuntivo 1 viene modellato imponendo che una `COMPETIZIONE` sia ulteriormente identificata dalle entità `CAMPIONATO` e ARBITRO. In questo modo si impediscono due competizioni con lo stesso arbitro nello stesso campionato.



6) Per soddisfare il vincolo aggiuntivo 2 è necessario reificare l'entità `IN` del punto 1. Si introduce quindi una nuova entità `DISPONIBILITA-GIORNALIERA` identificata dalla data e dall'entità `PISCINA`.



7) Lo schema che segue rappresenta la soluzione completa ottenuta non considerando i vincoli aggiuntivi.



Schema Relazionale

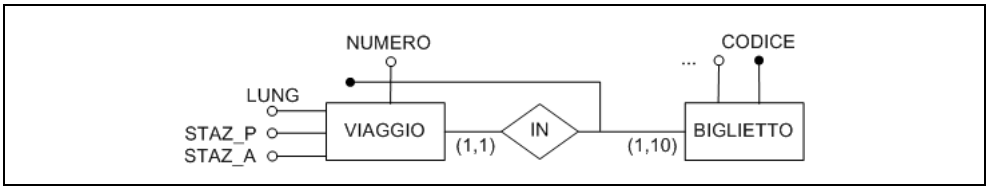
ATLETA/ARBITRO(<u>CF</u>,DATI-ANAG)
ATLETA(<u>CF</u>)
FK: CF **REFERENCES** ATLETA/ARBITRO
ARBITRO(<u>CF</u>)
FK: CF **REFERENCES** ATLETA/ARBITRO
PISCINA(<u>NOME</u>,DESCRIZIONE,N-CORSIE)
TIPO_GARA(<u>CODICE</u>,DESCRIZIONE)
RECORD(<u>CODICE,CF</u>,TEMPOMIGLIORE)
FK: CODICE **REFERENCES** TIPO_GARA
FK: CF **REFERENCES** ATLETA
GARA(<u>CODICE,DATA</u>,NOME)
FK: CODICE **REFERENCES** TIPO_GARA
FK: NOME **REFERENCES** PISCINA
AK: DATA,NOME
PARTECIPAZIONE(<u>DATA,CODICE,CF</u>,CORSIA,POS_FINALE)
FK: DATA,CODICE **REFERENCES** GARA
FK: CF **REFERENCES** ATLETA
AK: DATA,CODICE,CORSIA
COMPETIZIONE(<u>DATA,CODICE</u>,CF,COD-CAMPIONATO)
FK: DATA,CODICE **REFERENCES** GARA
FK: CF **REFERENCES** ARBITRO NOT NULL
FK: COD-CAMPIONATO **REFERENCES** CAMPIONATO NOT NULL
CAMPIONATO(<u>CODICE</u>,TIPO,ANNO)

♢ Esercizio 9

Si vuole realizzare il sistema informativo di una compagnia ferroviaria privata. Il sistema gestisce l'emissione di biglietti, che hanno un codice univoco, una data di emissione, un periodo di validità e un prezzo.
Un biglietto è composto da almeno un viaggio su una tratta delle rete ferroviaria. Un viaggio, identificato da un numero univoco all'interno del biglietto, ha una stazione di partenza, una stazione di arrivo e una lunghezza. Un biglietto può avere al massimo 10 tratte. Per una certo viaggio di un biglietto si può effettuare la prenotazione di un posto a sedere: in tal caso si deve specificare la data del viaggio, il numero del treno, il numero del vagone e il numero del posto. Per una data, per un numero di treno, un numero di vagone e un numero di posto c'è al massimo una prenotazione.
I biglietti si dividono in biglietti base e supplementi. I supplementi si riferiscono a un unico biglietto, e un biglietto può avere un unico supplemento associato.
La compagnia vuole memorizzare anche l'anagrafica dei clienti VIP. I Clienti VIP, identificati dagli usuali dati anagrafici, possono acquistare biglietti a tariffe agevolate: per ciascuno di tali acquisti occorre memorizzare lo sconto percepito.
Inoltre la compagnia ferroviaria spedisce mensilmente proprie offerte promozionali ai clienti VIP. Una offerta è caratterizzata da un nome, da una descrizione, da una tipologia e da una data di scadenza. Il sistema deve memorizzare tali spedizioni e in particolare la stessa offerta non può essere spedita più volte allo stesso cliente e a un cliente non deve arrivare più di una offerta per mese.
Vincolo aggiuntivo modificare lo schema E/R aggiungendo il seguente vincolo: in una stessa settimana, un cliente VIP può acquisitare tre biglietti al massimo a tariffe agevolate.

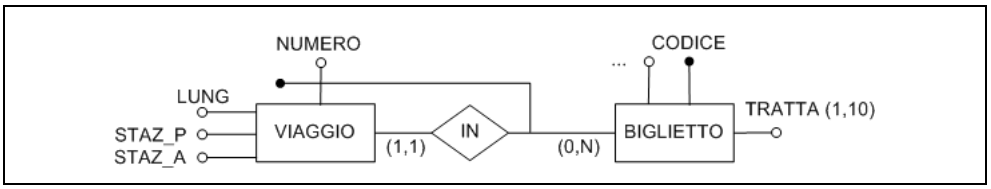
Soluzione

Si riportano di seguito i punti fondamentali della soluzione. Nella rappresentazione di alcune entità gli attributi semplici sono omessi.
1) La nozione di *tratta* può essere interpretata come una proprietà di `VIAGGIO` e quindi dalla specifica che "Un biglietto può avere al massimo 10 tratte." si può derivare che "un biglietto può avere al massimo 10 viaggi":

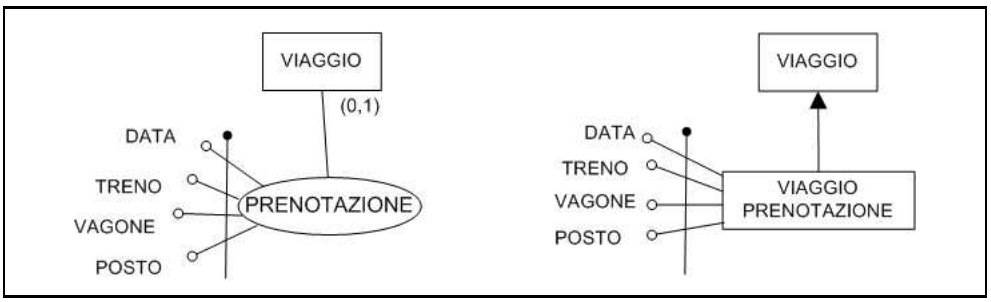


oppure, la nozione di *tratta* può essere interpretata come una proprietà di

`BIGLIETTO` scollegata da `VIAGGIO` e quindi rappresentare che "un biglietto può avere al massimo 10 tratte" tramite un attributo multiplo su `BIGLIETTO`:

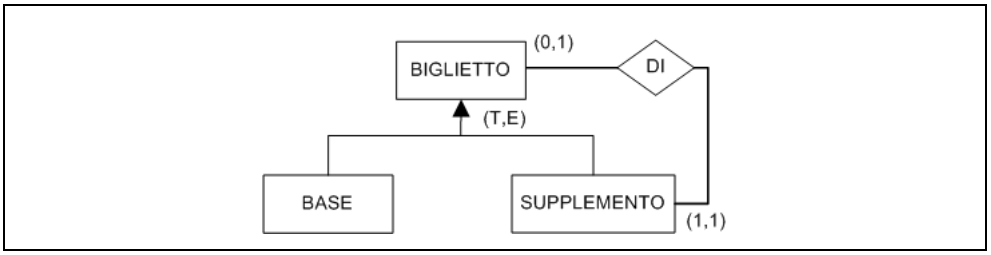


2) Per descrivere la prenotazione di un posto a sedere si propongono due possibili schemi:

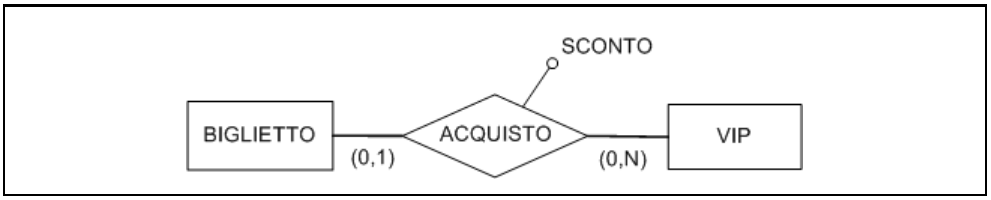


Lo schema di sinistra non è corretto infatti gli identificatori devono essere per definizione attributi obbligatori. Lo schema di destra è corretto: per i viaggi con prenotazione è previsto un ulteriore identificatore.

3) Le tipologie di biglietti sono rappresentabili con una gerarchia:

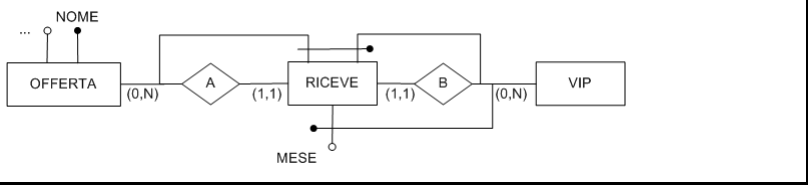


4) Il fatto che clienti VIP possano acquistare biglietti a tariffe agevolate è modellabile con una associazione. Nella specifica dell'esercizio viene sottointeso il fatto che un `BIGLIETTO` può essere acquistato *solo una volta*, e quindi può partecipare al massimo una volta all'acquisto :

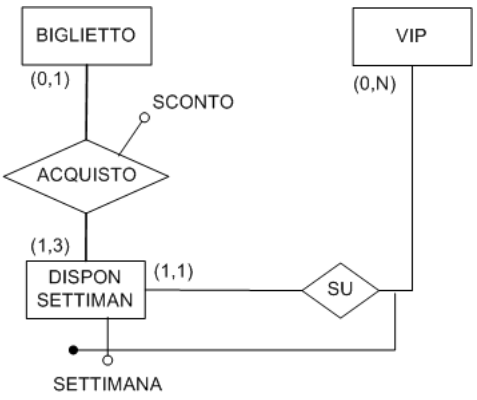


5) Per implementare la gestione delle offerte promozionali, occorre creare una associazione tra `VIP` e `OFFERTA`. Tale associazione deve essere reificata in

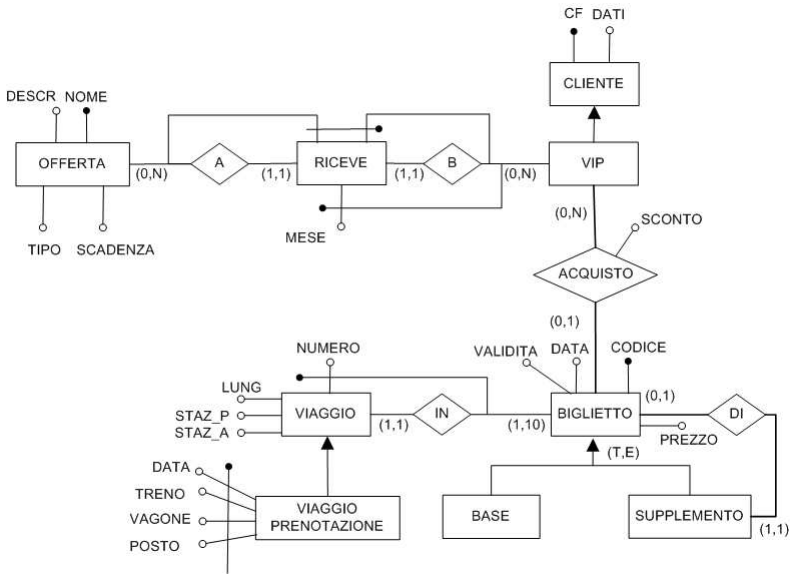
modo da implementare le specifiche imposte:



6)
Il vincolo aggiuntivo viene implementato introducendo un'entità per rappresentare la disponibilità di acquisto dei biglietti di un utente `VIP`:



7) Lo schema seguente propone una possibile soluzione dell'esercizio che trascura il vincolo aggiuntivo.



Schema Relazionale

```
CLIENTE(CF,DATI-ANAG)
VIP(CF)
     FK: CODICE REFERENCES BIGLIETTO
OFFERTA(NOME,DESCRIZIONE,TIPO,SCADENZA)
RICEVE(NOME,CF,MESE)
     FK: NOME REFERENCES OFFERTA
     FK: CF REFERENCES VIP
     AK: CF,MESE
BIGLIETTO(CODICE,PREZZO,DATA,VALIDITA,CF-VIP)
     FK: CF-VIP REFERENCES VIP
BASE(CODICE)
     FK: CODICE REFERENCES BIGLIETTO
SUPPLEMENTO(CODICE,CODICE-BIGLIETTO)
     FK: CODICE REFERENCES BIGLIETTO
     FK: CODICE-BIGLIETTO REFERENCES BIGLIETTO
     AK: CODICE-BIGLIETTO
VIAGGIO(NUMERO,CODICE,LUNG,STAZ_P,STAZ_A)
     FK: CODICE REFERENCES BIGLIETTO
VIAGGIO-PRENOTAZIONE(NUMERO,CODICE,DATA,TRENO,VAGONE,POSTO)
     FK: NUMERO,CODICE REFERENCES VIAGGIO
     AK: DATA,TRENO,VAGONE,POSTO
```

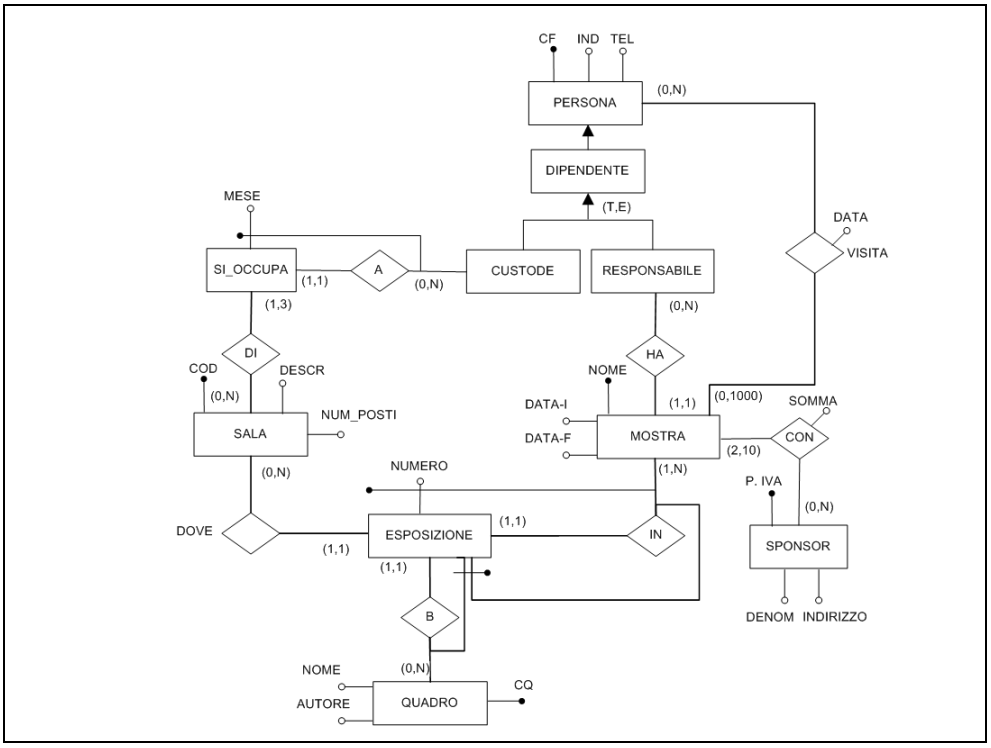
Si noti che nella traduzione del subset è stato mantenuto il doppio identificatore espresso nella modellazione E/R: le istanze di `VIAGGIO-PRENOTAZIONE` possono essere identificate sia attraverso la chiave di `VIAGGIO`, entità che specializzano, sia attraverso una nuova chiave specifica.

♢ **Esercizio 10**

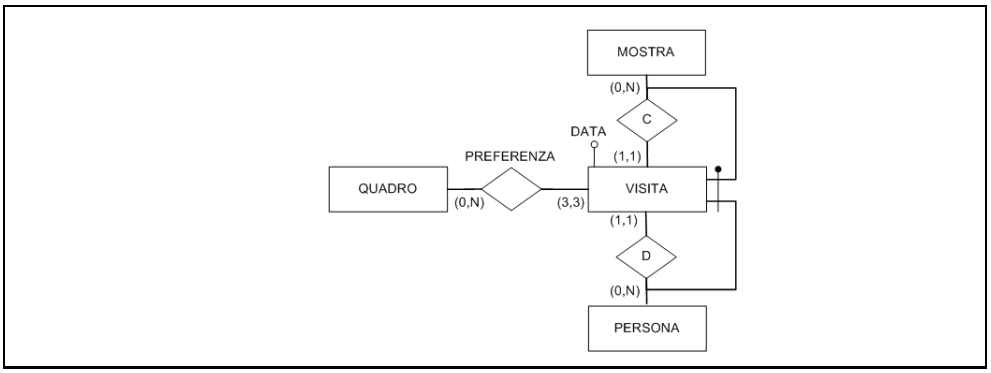
Si vuole rappresentare l'attività di organizzazione di mostre di una galleria d'arte. Un quadro ha un codice identificativo, un nome e un autore. Una mostra ha un nome univoco, una data di inizio e una data di fine. Una sala ha un codice univoco, un numero posti ed una descrizione. I dipendenti della galleria d'arte, caratterizzati dai consueti dati anagrafici, sono ripartiti in due categorie: 1) responsabile: ogni mostra ha un responsabile, che può svolgere tale ruolo anche in altre mostre; 2) custode: ogni mese un custode si occupa di minimo una e massimo tre sale; una sala può avere molti custodi.
Devono essere memorizzate le persone, caratterizzate dai consueti dati anagrafici, che visitano le mostre, riportando la data in cui è stata effettuata la visita, con la limitazione che una persona non possa visitare due o più volte la stessa mostra; per una mostra sono accettati fino ad un massimo di mille visitatori.
Durante una certa mostra, ad un quadro viene assegnato un numero univoco e il quadro viene esposto in una sola sala. In una sala possono essere esposti un numero arbitrario di quadri.
Ogni mostra ha da due a dieci sponsor: per ciascuna di queste sponsorizzazioni si riporta la somma in euro versata.
Uno sponsor è rappresentato tramite una partita IVA, una denominazione ed un indirizzo.
Vincolo aggiuntivo modificare lo schema E/R aggiungendo il vincolo che "durante la visita di una persona ad una mostra, occorre indicare anche i tre quadri esposti nella mostra che gli sono piaciuti di più."

Soluzione

Di seguito si riporta lo schema E/R complessivo.



Per esprimere il vincolo aggiuntivo è sufficiente reificare l'associazione `VISITA` e collegare ad essa l'entità `QUADRO` attraverso una nuova associazione `VISITA`. Da notare che card-min(`VISITA`,`PREFERENZA`) può essere anche espressa come opzionale in modo da modellare visitatori che abbiano indicato meno di 3 quadri come preferiti.



Schema Relazionale

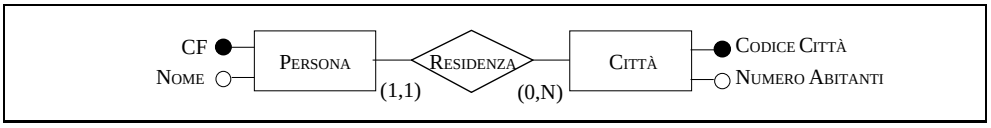
```
PERSONA(CF,INDIRIZZO,TELEFONO)
DIPENDENTE(CF)
     FK: CF REFERENCES PERSONA
CUSTODE(CF)
     FK: CF REFERENCES DIPENDENTE
RESPONSABILE(CF)
     FK: CF REFERENCES DIPENDENTE
SI_OCCUPA(MESE,CF)
     FK: CF REFERENCES DIPENDENTE
SALA(CODICE,NUM_POSTI,DESCRIZIONE)
DI(CODICE,CF,MESE)
     FK: CODICE REFERENCES SALA
     FK: CF,MESE REFERENCES SI_OCCUPA
MOSTRA(NOME,CF,DATA-I,DATA-F)
     FK: CF REFERENCES RESPONSABILE NOT NULL
VISITA(CF,NOME,SOMMA)
     FK: NOME REFERENCES MOSTRA
     FK: CF REFERENCES PERSONA
SPONSOR(P-IVA,DENOMINAZIONE,INDIRIZZO)
CON(P-IVA,NOME,SOMMA)
     FK: P-IVA REFERENCES SPONSOR
     FK: NOME REFERENCES MOSTRA
ESPOSIZIONE(NUMERO,NOME,COD-QUADRO,COD-SALA)
     FK: NOME REFERENCES MOSTRA
     FK: COD-QUADRO REFERENCES QUADRO
     FK: COD-SALA REFERENCES SALA NOT NULL
     AK: COD-QUADRO,NOME
QUADRO(CODICE,NOME,AUTORE)
```

Si osservi che la traduzione dell'associazione `VISITA` non mantiene i vincoli espressi dalle cardinalità: nella fattispecie i vincoli espressi dalla cardinalitaà card(`MOSTRA`,`VISITA`)=(0,1000) non sono esprimibili in relazionale. Analogo discorso può essere fatto considerando l'associazione `CON`.

6.1.2 Dati Derivati

♢ **Esercizio 1**

Dato il seguente schema E/R, volume dei dati e operazioni, decidere se è conveniente conservare nello schema l'attributo derivato `NUMEROABITANTI` calcolato contando il numero delle persone che risiedono in una certà città. Si trascuri l'occupazione di memoria del dato derivato.



Operazione 1) Dato il codice di una città, visualizzarne tutti i suoi dati.
Operazione 2) Dato il codice di una città e di una persona, memorizza la relativa residenza (incrementando anche l'eventuale dato derivato).

Tavola dei volumi

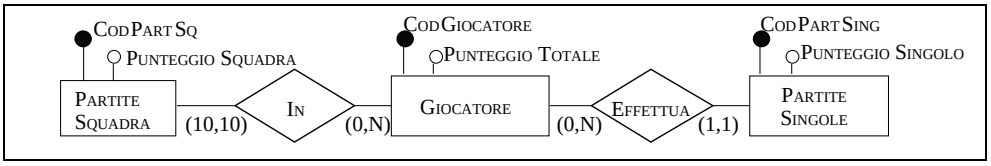
CONCETTO	TIPO	VOL.
Città	E	200
Persona	E	1000000

Tavola delle operazioni

OPER.	TIPO	FREQ.
Oper . 1	I	2/Giorno
Oper . 2	I	500/Giorno

♢ **Esercizio 2**

Dato il seguente schema E/R, volume dei dati e operazioni, decidere se è conveniente conservare nello schema l'attributo derivato `PUNTEGGIOTOTALE`, che per una certo giocatore è calcolato sommando sia i punteggi delle partite singole che quelli delle partite in squadra. Si noti che il punteggio di una partita in squadra deve essere sommato a tutti i giocatori della squadra. Si trascuri l'occupazione di memoria del dato derivato.



Operazione 1) Inserimento di una nuova partita singola (si suppone noto e valido il codice del giocatore che effettua la partita);
Operazione 2) Inserimento di una nuova partita di squadra (si suppongono noti e validi i codici dei giocatori che effettuano la partita);
Operazione 3) Visualizzare tutti i dati di un giocatore, compreso il dato derivato.

Tavola dei volumi

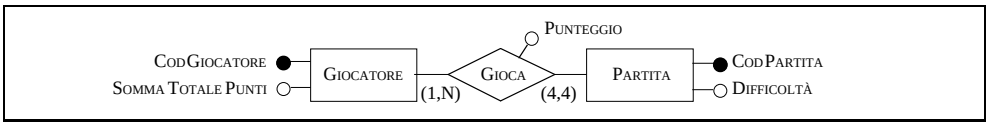
CONCETTO	TIPO	VOL.
Giocatore	E	100
PartiteSquadra	E	500
PartiteSingole	E	1000

Tavola delle operazioni

OPER.	TIPO	FREQ.
Oper . 1	I	100/Giorno
Oper . 2	I	50/Giorno
Oper . 3	I	15/Giorno

♢ Esercizio 3

Dato il seguente schema E/R, con il seguente volume dei dati e le seguenti operazioni, decidere se è conveniente conservare nello schema l'attributo derivato `SOMMATOTALEPUNTI`, che per un certo `GIOCATORE` è calcolato come la somma di `PUNTEGGIO*DIFFICOLTÀ` di tutte le partite giocate. Si trascuri l'occupazione di memoria del dato derivato.



Operazione 1) Introduzione di una nuova partita;
Operazione 2) Visualizzare i dati di un giocatore.

Tavola dei volumi

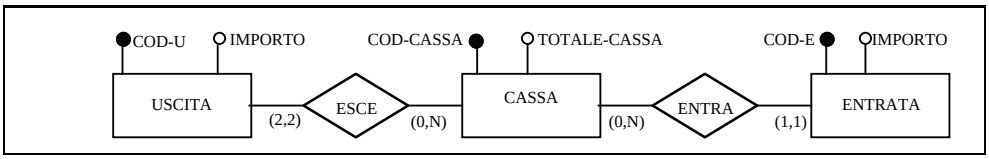
CONCETTO	TIPO	VOL.
Giocatore	E	100
Partita	E	1000

Tavola delle operazioni

OPER.	TIPO	FREQ.
Oper . 1	I	40/Giorno
Oper . 2	I	10/Giorno

♢ Esercizio 4

Dato il seguente schema E/R, volume dei dati e operazioni, decidere se è conveniente conservare nello schema l'attributo derivato `TOTALECASSA`, che per una certa cassa é calcolato come la differenza tra la somma degli importi delle entrate e la somma degli importi delle uscite. Si supponga che l'importo di una entrata venga aggiunto al `TOTALECASSA` di un'unica cassa mentre l'importo di una uscita venga sottratto al `TOTALECASSA` di due casse definite. Si trascuri l'occupazione di memoria del dato derivato.



Operazione 1) Inserimento di una nuova entrata (si suppone noto e valido il codice della cassa sulla quale l'entrata deve essere registrata);
Operazione 2) Visualizzare tutti i dati di un cassa. Si noti che nel caso in cui si deve determinare il `TOTALECASSA` occorre leggere tutte le entrate e tutte le uscite di quella cassa;
Operazione 3) Inserimento di una nuova uscita (si suppongono noti e validi i codici delle due casse sulle quali l'uscita deve essere registrata).

Tavola dei volumi

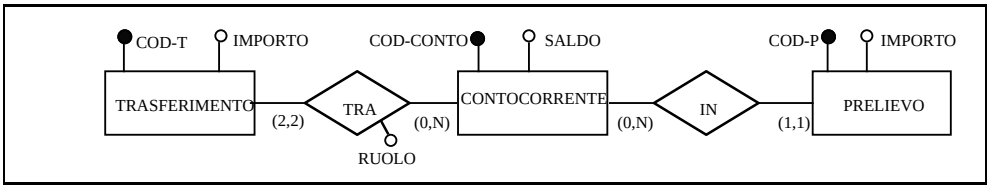
CONCETTO	TIPO	VOL.
Cassa	E	100
Entrata	E	3000
Uscita	E	1000

Tavola delle operazioni

OPER.	TIPO	FREQ.
Oper . 1	I	100/Giorno
Oper . 2	I	10/Giorno
Oper . 3	I	1/Giorno

♢ **Esercizio 5**

Dato il seguente schema E/R, volume dei dati e operazioni, decidere se è conveniente conservare nello schema l'attributo derivato `SALDO`, che per una certo contocorrente è calcolato tenendo in considerazione sia gli importi dei prelievi sia gli importi dei trasferimenti. Si supponga che l'importo di un prelievo venga sottratto al `SALDO` di un unico contocorrente mentre l'importo di un trasferimento venga sottratto dal `SALDO` di un contocorrente (`RUOLO='Origine'`) ed aggiunto al `SALDO` di un altro contocorrente (`RUOLO='Destinazione'`). Si trascuri l'occupazione di memoria del dato derivato.



Operazione 1) Inserimento di un nuovo prelievo (si suppone noto e valido il codice del contocorrente sul quale viene effettuato il prelievo);

Operazione 2) Inserimento di un nuovo trasferimento (si suppongono noti e validi i codici dei due conticorrenti che interessano il trasferimento);

Operazione 3) Visualizzare tutti i dati di un contocorrente, anche il `SALDO`.

Tavola dei volumi

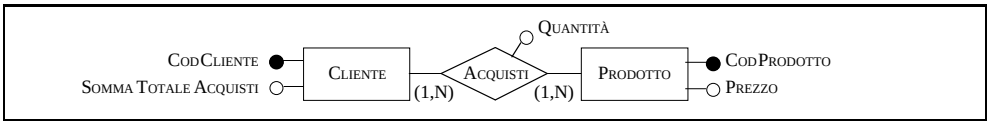
CONCETTO	TIPO	VOL.
ContoCorrente	E	100
Prelievo	E	3000
Trasferimento	E	1000

Tavola delle operazioni

OPER.	TIPO	FREQ.
Oper . 1	I	100/Giorno
Oper . 2	I	10/Giorno
Oper . 3	I	4/Giorno

♢ **Esercizio 6**

Dato il seguente schema E/R, con il seguente volume dei dati e le seguenti operazioni, decidere se è conveniente conservare nello schema l'attributo `SOMMATOTALEACQUISTI`, trascurando l'occupazione di memoria di tale dato.



Operazione 1) Dati i codici di un cliente e di un prodotto già esistenti, inserire l'acquisto del prodotto da parte del cliente;

Operazione 2) Visualizzare i dati di un cliente;

Operazione 3) Modificare il prezzo di un prodotto.

Tavola dei volumi

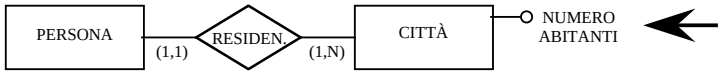
CONCETTO	TIPO	VOL.
Prodotto	E	600
Cliente	E	300
Acquisti	R	12000

Tavola delle operazioni

OPER.	TIPO	FREQ.
Oper . 1	I	400/Giorno
Oper . 2	I	10/Giorno
Oper . 3	I	1/Giorno

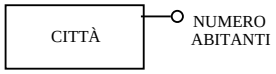
Soluzioni

Esercizio 1

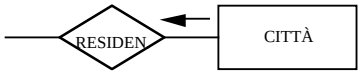


operazione 1:

Con il dato derivato

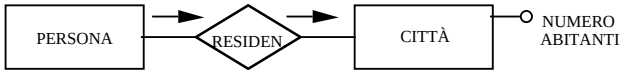


Senza il dato derivato

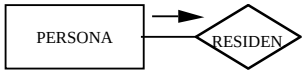


operazione 2:

Con il dato derivato



Senza il dato derivato



Con il dato derivato:

Operazione 1	**CONCETTO**	**ACC.**	**TIPO**
1 accesso in lettura	Città	1	L
1*2 = **2**/giorno			
Operazione 2	Persona	1	S
1 accesso in lettura	Residenza	1	S
3 accessi in scrittura	Città	1	L
7*500 = **3500**/giorno	Città	1	S

Senza il dato derivato:

Operazione 1	**CONCETTO**	**ACC.**	**TIPO**
5001 accessi in lettura	Città	1	L
5001*2 = **10002**/giorno	Residenza	5000	L
Operazione 2	Persona	1	S
2 accessi in scrittura	Residenza	1	S
4*500 = **2000**/giorno			

Conclusione: Conviene tenere il dato derivato.

Esercizio 2

Con il dato derivato :

	CONCETTO	ACC.	TIPO
Operazione 1	PartitaSingola	1	S
1 accesso in lettura	In	1	S
3 accessi in scrittura	Giocatore	1	L
(1+3*2)*100 = 700/giorno	Giocatore	1	S
Operazione 2	PartitaSquadra	1	S
10 accessi in lettura	Effettua	10	S
21 accessi in scrittura	Giocatore	10	L
(10+21*2)*50 = 2600/giorno	Giocatore	10	S
Operazione 3	Giocatore	1	L
1 accesso in lettura			
1*15 = 15/giorno			

Totale : 3315/giorno

Senza il dato derivato :

	CONCETTO	ACC.	TIPO
Operazione 1	PartitaSingola	1	S
2 accessi in scrittura	In	1	S
(2*2)*100 = 400/giorno			
Operazione 2	PartitaSquadra	1	S
3 accessi in scrittura	Effettua	10	S
(3*2)*50 = 300/giorno			
Operazione 3	Giocatore	1	L
121 accessi in lettura	In	50	L
121*15 = 1815/giorno	PartitaSquadra	50	L
	Effettua	10	L
	PartitaSingola	10	L

Totale : 3315/giorno

Conclusione : Dal punto di vista del numero di accessi, è indifferente tenere o meno il dato derivato.

Esercizio 3

Con il dato derivato :

	CONCETTO	**ACC.**	**TIPO**
Operazione 1	Partita	1	S
4 accessi in lettura	Gioca	4	S
9 accessi in scrittura	Giocatore	4	L
22*40 = **880**/giorno	Giocatore	4	S
Operazione 2	Giocatore	1	L
1 accesso in lettura			
1*100 = **10**/giorno			

Totale : 890/giorno

Senza il dato derivato :

	CONCETTO	**ACC.**	**TIPO**
Operazione 1	Partita	1	S
5 accessi in scrittura	Gioca	4	S
10*40 = **400**/giorno			
Operazione 2	Giocatore	1	L
81 accessi in lettura	Gioca	40	L
81*100 = **810**/giorno	Partita	40	L

Totale : 1210/giorno

Conclusione : Conviene tenere il dato derivato.

Esercizio 4

Con il dato derivato :

	CONCETTO	ACC.	TIPO
Operazione 1	Entrata	1	S
1 accesso in lettura	Entra	1	S
3 accessi in scrittura	Cassa	1	L
7*100 = **700**/giorno	Cassa	1	S
Operazione 2	Cassa	1	L
1 accesso in lettura			
1*10 = **10**/giorno			
Operazione 3	Uscita	1	S
2 accesso in lettura	Esce	2	S
5 accessi in scrittura	Cassa	2	L
12*1 = **12**/giorno	Cassa	2	S

Totale : 722/giorno

Senza il dato derivato :

	CONCETTO	ACC.	TIPO
Operazione 1	Entrata	1	S
2 accessi in scrittura	Entra	1	S
4*100 = **400**/giorno			
Operazione 2	Cassa	1	L
121 accesso in lettura	Entra	40	L
121*10 = **1210**/giorno	Entrata	40	L
	Esce	20	L
	Uscita	20	L
Operazione 3	Uscita	1	S
3 accessi in scrittura	Esce	2	S
3*1 = **3**/giorno			

Totale : 1613/giorno

Conclusione : Conviene tenere il dato derivato.

Esercizio 5

Con il dato derivato :

	CONCETTO	ACC.	TIPO
Operazione 1	Cliente	1	L
2 accessi in lettura	Acquisti	1	S
2 accessi in scrittura	Prodotto	1	L
6*100 = **600**/giorno	Cliente	1	S
Operazione 2	Cliente	1	L
1 accesso in lettura			
1*10 = **10**/giorno			
Operazione 3	Prodotto	1	L
41 accessi in lettura	Prodotto	1	S
21 accessi in scrittura	Acquisti	20	L
83*1 = **83**/giorno	Cliente	20	L
	Cliente	20	S

Totale : 693/giorno

Senza il dato derivato :

	CONCETTO	ACC.	TIPO
Operazione 1	Acquisti	1	S
1 accessi in scrittura			
2*100 = **200**/giorno			
Operazione 2	Cliente	1	L
81 accessi in lettura	Acquisti	40	L
81*10 = **1210**/giorno	Prodotto	40	L
Operazione 3	Prodotto	1	L
1 accesso in lettura	Prodotto	1	S
1 accesso in scrittura			
3*1 = **3**/giorno			

Totale : 1413/giorno

Conclusione : Conviene tenere il dato derivato.

Esercizio 6

Con il dato derivato :

	CONCETTO	**ACC.**	**TIPO**
Operazione 1	Cliente	1	L
2 accessi in lettura	Acquisti	1	S
2 accessi in scrittura	Prodotto	1	L
6*400 = **2400**/giorno	Cliente	1	S
Operazione 2	Cliente	1	L
1 accesso in lettura			
1*10 = **10**/giorno			
Operazione 3	Prodotto	1	L
41 accessi in lettura	Prodotto	1	S
21 accessi in scrittura	Acquisti	20	L
83*1 = **83**/giorno	Cliente	20	L
	Cliente	20	S

Totale : 2493/giorno

Senza il dato derivato :

	CONCETTO	**ACC.**	**TIPO**
Operazione 1	Acquisti	1	S
1 accessi in scrittura			
2*400 = **800**/giorno			
Operazione 2	Cliente	1	L
81 accessi in lettura	Acquisti	40	L
81*10 = **1210**/giorno	Prodotto	40	L
Operazione 3	Prodotto	1	L
1 accesso in lettura	Prodotto	1	S
1 accesso in scrittura			
3*1 = **3**/giorno			

Totale : 2093/giorno

Conclusione : Conviene eliminare il dato derivato.

6.2 SQL e Algebra Relazionale

Per ciascun esercizio, viene richiesto di scrivere in SQL tutte le interrogazioni riportate; le interrogazioni da scrivere anche in Algebra Relazionale sono esplicitamente indicate.

♢ **Esercizio 1**

Schema Relazionale

`DOCENTE(CFDOC,NOME,COGNOME)`
`STUDENTE(CFSTUD,NOME,COGNOME)`
`ARGOMENTO(CODARG,DESCRIZIONE)`
`LEZIONE(CODARG,DATA,CFDOC,NUMSTUDENTI)`
AK: `DATA,CFDOC`
FK: `CODARG` **REFERENCES** `ARGOMENTO`
FK: `CFDOC` **REFERENCES** `DOCENTE`
dove `NUMSTUDENTI` è il numero di studenti presenti alla lezione
`INTERROGAZIONE(CODARG,DATA,CFSTUD,VOTO)`
FK: `CODARG,DATA` **REFERENCES** `LEZIONE`
FK: `CFSTUD` **REFERENCES** `STUDENTE`
Interrogazione e voto relativo dello studente `CFSTUD` durante una lezione.

Interrogazioni (Algebra Relazionale **a)**, **b)** e **c)**).

a) selezionare il codice fiscale, il nome ed il cognome degli studenti che non sono mai stati interrogati su un argomento con descrizione 'Fisica';

b) selezionare il codice fiscale del docente che ha svolto lezioni su tutti gli argomenti con descrizione 'Fisica';

c) selezionare il codice fiscale del docente che ha sempre interrogato, cioè che durante ogni sua lezione ha fatto almeno una interrogazione;

d) selezionare, per ogni argomento, la media dei voti riportati dagli studenti interrogati sull'argomento, considerando solo gli studenti che sono stati interrogati almeno tre volte sull'argomento in questione;

e) selezionare, per ogni studente, il codice fiscale del docente con il quale ha effettuato il maggior numero di interrogazioni.

♢ **Esercizio 2**

Schema Relazionale

CICLISTA(NOMECICLISTA, NAZIONALITÀ, ETÀ)

GARA(NOMECORSA, ANNO, PARTENZA, ARRIVO)

edizione di un certo ANNO della corsa ciclistica NOMECORSA; PARTENZA e ARRIVO sono la città di partenza e di arrivo.

PARTECIPA(NOMECORSA, ANNO, NOMECICLISTA, POSIZIONE)
FK: NOMECORSA, ANNO **REFERENCES** GARA
FK: NOMECICLISTA **REFERENCES** CICLISTA

NOMECICLISTA ha partecipato alla gara (NOMECORSA,ANNO) classificandosi in una certa POSIZIONE o ritirandosi (POSIZIONE='R').

Interrogazioni (Algebra Relazionale **a)**, **b)** e **c)**).

a) selezionare i ciclisti che si sono classificati in prima posizione in una gara ciclistica partita da Milano;

b) selezionare il nome dei ciclisti che non si sono mai ritirati al Giro (corsa con nome Giro);

c) selezionare le corse per le quali in ogni edizione c'è stato almeno un ritirato;

d) selezionare, per ogni corsa ciclistica, l'anno in cui c'è stato il maggior numero di ciclisti ritirati.

♢ **Esercizio 3**

Schema Relazionale

QUADRO(CQ, AUTORE, PERIODO)

MOSTRA(CM, NOME, ANNO, ORGANIZZATORE)

ESPONE(CM, CQ, SALA) **FK:** CM **REFERENCES** MOSTRA
FK: CQ **REFERENCES** QUADRO

Nella mostra CM, il quadro CQ è stato esposto in una certa SALA.

Interrogazioni (Algebra Relazionale **a)**, **b)** e **c)**).

a) selezionare le sale nelle quali è stato esposto, nell'anno 1997, un quadro di Picasso;

b) selezionare tutti i dati dei quadri di Picasso che non sono mai stati esposti nell'anno 1997;

c) selezionare tutti i dati dei quadri che non sono mai stati esposti insieme ad un quadro di Picasso, cioè nella stessa mostra in cui compariva anche un quadro di Picasso;

d) selezionare tutti i dati delle mostre in cui sono stati esposti quadri di almeno 5 autori distinti;

e) selezionare, per ogni mostra, l'autore di cui si esponevano il maggior numero di quadri.

♢ **Esercizio 4**

Schema Relazionale

PRODOTTO(CP,DESCRIZIONE)
SOCIO(CS,NOME,COGNOME)
OFFERTA(CO,VALIDITÀ)
COMPRENDE(CO,CP,QUANTIT **FK: CO REFERENCES OFFERTA**
FK: CP REFERENCES PRODOTTO

QUANTITÀ è il numero di unità del prodotto CP comprese nell'offerta CO

RITIRA(CO,CS,DATA) **FK: CO REFERENCES OFFERTA**
FK: CS REFERENCES SOCIO

DATA è il giorno in cui il socio CS ritira l'offerta CO

Interrogazioni (Algebra Relazionale **a)**, **b)** e **c)**).

a) selezionare il codice e la descrizione dei prodotti che non sono mai stati offerti insieme ad un prodotto con descrizione ='Uva';

b) selezionare il codice, nome e cognome dei soci che non hanno ritirato alcuna offerta che comprende un prodotto con descrizione 'Uva'.

c) selezionare il codice, nome e cognome dei soci che hanno ritirato tutte le offerte che comprendono un prodotto con descrizione ='Uva'.

d) selezionare, per ogni socio, il numero delle offerte ritirate che comprendono un prodotto con descrizione ='Uva'.

♢ **Esercizio 5**

Schema Relazionale

VIA(CV,NOME,QUARTIERE,LUNGHEZZA)
INC(CVA,CVB,N-VOLTE)
FK: CVA REFERENCES VIA
FK: CVB REFERENCES VIA

La via CVA incrocia N-VOLTE la via CVB

si assume che se nella relazione INC è presente la tupla (codviax,codviay,5) non sia presente la tupla simmetrica (codviay,codviax,5);

Interrogazioni (Algebra Relazionale **a)** e **b)**).

a) selezionare le vie che incrociano almeno una via del quartiere 'Pastena';

b) selezionare le vie che non incrociano via 'Marco Polo';

c) selezionare le coppie (CODICE1, CODICE2) tali che le vie con codice CODICE1 e CODICE2 abbiano la stessa lunghezza;

d) selezionare il quartiere che ha il maggior numero di vie;

e) selezionare, per ogni quartiere, la via di lunghezza maggiore;

f) selezionare le vie che incrociano tutte le vie del quartiere 'Pastena'.

♢ **Esercizio 6**

Schema Relazionale

CAMPO(NCAMPO,TIPO,INDIRIZZO)
TENNISTA(CF,NOME,NAZIONE)
INCONTRO(CI,NCAMPO,GIOC1,GIOC2,SET1,SET2)
FK: NCAMPO **REFERENCES** CAMPO
FK: GIOC1 **REFERENCES** TENNISTA
FK: GIOC2 **REFERENCES** TENNISTA

Incontro, svolto nel campo NCAMPO, tra i tennisti GIOC1 e GIOC2: in SET1 e SET2 sono riportati il numero di set vinti da GIOC1 e GIOC2 rispettivamente.

Interrogazioni (Algebra Relazionale **a)**, **b)**, **c)** e **d)**).

a) selezionare gli incontri disputati sull'erba (campo con tipo 'erba');

b) selezionare i campi in erba sui quali non c'è stato nessun incontro;

c) selezionare i dati dei tennisti vincitori di almeno una partita sull'erba;

d) selezionare i dati delle nazioni in cui tutti i giocatori hanno sempre vinto le partite disputate;

e) selezionare il campo in erba che ha ospitato il maggior numero di incontri.

♢ **Esercizio 7**

Schema Relazionale

OPERATORE(CODOP,INDIRIZZO,QUALIFICA,COSTO-ORARIO)
ARTICOLO(CODART,DESCRIZIONE)
LOTTO(CODART,COPOP,TOTESEM)
FK: CODART **REFERENCES** ARTICOLO
FK: CODOP **REFERENCES** OPERATORE

dove TOTESEM è il numero di pezzi dell'articolo CODART.

RECLAMO(CODART,COPOP,NESEMPLARE,NOMECL)
FK: CODART,CODOP **REFERENCES** LOTTO

Reclamo effettuato dal cliente NOMECL sull'esemplare NESEMPLARE del lotto confezionato dall'operatore CODOP relativo all'articolo CODART.

Interrogazioni (Algebra Relazionale **a)**, **b)** e **c)**).

a) selezionare il codice e il nome degli operatori per i quali non esiste alcun reclamo, cioè per i quali nessun esemplare di nessun lotto da essi confezionato ha ricevuto un reclamo;

b) selezionare il codice degli operatori per i quali ogni lotto da essi confezionato contiene almeno un esemplare al quale si riferisce un reclamo;

c) selezionare il nome del cliente che ha fatto reclami per tutti gli operatori;

d) selezionare, per ogni articolo, il codice dell'operatore che ha confezionato il lotto con il maggior numero di esemplari, senza considerare i lotti con un numero di esemplari TOTESEM non specificato.

e) selezionare il lotto che ha ricevuto più reclami.

♢ **Esercizio 8**

Schema Relazionale

MANIFESTAZIONE(CM,NOME)
LUOGO(NOME-LUOGO,INDIRIZZO,CITTÀ)
SPETTACOLO(CM,NUM,ORA-INIZIO,NOME-LUOGO,DATA)
FK: CM **REFERENCES** MANIFESTAZIONE
FK: NOME-LUOGO **REFERENCES** LUOGO

NUM è il numero dello spettacolo all'interno della manifestazione CM

Interrogazioni (Algebra Relazionale **a)** e **b)**).

a) selezionare il codice e il nome delle manifestazioni che non hanno interessato luoghi della città di Modena;

b) selezionare i nomi dei luoghi che hanno ospitato tutte le manifestazioni (hanno ospitato almeno uno spettacolo di ciascuna manifestazione);

c) selezionare il nome dei luoghi che, in una certa data, ospitano più di tre spettacoli dopo le ore 15;

d) selezionare, per ogni luogo, il numero totale delle manifestazioni e il numero totale degli spettacoli ospitati;

e) Descrivere sinteticamente a parole e riportare in SQL l'interrogazione descritta dalla seguente espressione dell'algebra relazionale:
$\pi_{\texttt{CM}}(\texttt{SPETTACOLO}) -$
$\pi_{\texttt{CM}}\big(\pi_{\texttt{CM,NUM}}(\texttt{SPETTACOLO}) - \pi_{\texttt{CM,NUM}}\big(\sigma_{\texttt{ORA-INIZIO}>15}(\texttt{SPETTACOLO})\big)\big)$

♢ **Esercizio 9**

Schema Relazionale

FORNITORE(CODF,NOMEFORNITORE,CITTÀ)
PRODOTTO(CODP,DESCRIZIONE,PREZZO)
FORNISCE(ANNO,CODP,CODF,QTY)
FK: CODP **REFERENCES** PRODOTTO
FK: CODF **REFERENCES** FORNITORE

Nell'ANNO specificato, il prodotto CODP è stato fornito dal fornitore CODF in quantità QTY

Interrogazioni (Algebra Relazionale **a)**, **b)** e **c)**).

a) selezionare i prodotti che nell'anno 1995 sono stati forniti da almeno un fornitore di Modena;

b) selezionare i prodotti non forniti da nessun fornitore di Modena;

c) selezionare i prodotti che nell'anno 1994 sono stati forniti esclusivamente da fornitori di Modena;

d) selezionare, per ogni anno, la quantità totale dei prodotti forniti dai fornitori di Modena;

e) selezionare, per ogni anno, il codice del fornitore che ha fornito in totale la maggiore quantità di prodotti.

♢ **Esercizio 10**

Schema Relazionale

GARA(CG,NOMECAMPO,LIVELLO)
GIOCATOREGOLF(CF,NOME,NAZIONE)
PARTECIPA(CG,CF,PUNTEGGIO) **FK: CG REFERENCES GARA**
FK: CF REFERENCES GIOCATOREGOLF

Il giocatore CF partecipa alla gara CG totalizzando un certo PUNTEGGIO. La gara è vinta dal giocatore che totalizza il punteggio più basso.

Interrogazioni (Algebra Relazionale **a)**, **b)** e **c)**).

a) selezionare i dati dei giocatori di golf che hanno partecipato ad almeno una gara disputata a livello 'nazionale';

b) selezionare le nazioni in cui tutti i giocatori hanno ottenuto un punteggio minore o uguale a 0 nelle gare disputate;

c) selezionare i dati dei giocatori di golf che hanno partecipato a tutte le gare disputate a livello 'nazionale';

d) selezionare i dati dei giocatori di golf che hanno vinto almeno una gara disputata a livello 'internazionale';

e) selezionare, per ogni nazione che nelle gare di livello 'internazionale' ha schierato più di 5 giocatori distinti, il punteggio medio ottenuto dai giocatori in tali gare; si ordini il risultato in modo decrescente rispetto al punteggio medio.

6.2.1 Soluzioni

Di alcuni esercizi viene omessa la soluzione in SQL dell'interrogazione 1), in quanto si tratta in genere di un semplice join.
Inoltre, per semplicità, dati due schemi di relazioni

$R_1(\underline{A}, \ldots)$
$R_2(\ldots, B, \ldots)$
FK: B **REFERENCES** R_1

il theta-join che collega R_1 e R_2 in base alla FK, $R_1 \bowtie_{R_1.A=R_2.B} R_2$, verrà indicato con il simbolo del join naturale: $R_1 \bowtie R_2$

Esercizio 1

1a)
```
SELECT *
FROM   STUDENTE
WHERE  NOT EXISTS
         ( SELECT *
           FROM   INTERROGAZIONE,ARGOMENTO
           WHERE  STUDENTE.CFSTUD = INTERROGAZIONE.CFSTUD
           AND    ARGOMENTO.CODARG = INTERROGAZIONE.CODARG
           AND    ARGOMENTO.DESCRIZIONE = 'Fisica')
```

2a) $\texttt{STUDENTE} \bowtie \left(\pi_{\texttt{CFSTUD}}\,(\texttt{STUDENTE}) - \pi_{\texttt{CFSTUD}}\,\left(\texttt{INTERROGAZIONE} \bowtie \sigma_{\texttt{DESCRIZIONE='Fisica'}}\,(\texttt{ARGOMENTO})\right)\right)$

1b)
```
SELECT distinct CFDOC
FROM   LEZIONE X
WHERE  NOT EXISTS
         ( SELECT *
           FROM   ARGOMENTO
           WHERE  DESCRIZIONE = 'Fisica'
           AND    NOT EXISTS
                    ( SELECT *
                      FROM   LEZIONE Y
                      WHERE  X.CFDOC = Y.CFDOC
                      AND    ARGOMENTO.CODARG = Y. CODARG))
```

2b) $\pi_{\texttt{CFDOC,CODARG}}(\texttt{LEZIONE}) \div \pi_{\texttt{CODARG}}\left(\sigma_{\texttt{DESCRIZIONE='Fisica'}}(\texttt{ARGOMENTO})\right)$

1c) L'interrogazione può essere riformulata come "i docenti per i quali *non esiste* una loro lezione *senza alcuna* interrogazione";

```
SELECT LX.CFDOC
FROM   LEZIONE LX
WHERE  NOT EXISTS
         ( SELECT *
           FROM   LEZIONE LY
           WHERE  LX.CFDOC = LY.CFDOC
           AND    NOT EXISTS
                    ( SELECT *
                      FROM   INTERROGAZIONE I
                      WHERE  LY.DATA=I.DATA
                      AND    LY.CODARG=I.CODARG ))
```

2c) Prima di tutto determiniamo, indicandole con R_1, le (chiavi delle) lezioni nelle quali non c'è stata *nessuna* interrogazione

$$R_1 = \pi_{\texttt{CODARG,DATA}}(\texttt{LEZIONE}) - \pi_{\texttt{CODARG,DATA}}\big(\texttt{INTERROGAZIONE}\big)$$

Indichiamo con R_2 i codici fiscali dei docenti delle lezioni R_1, ovvero i docenti che hanno svolto almeno una lezione senza effettuare alcuna interrogazione:

$$R_2 = \pi_{\texttt{CFDOC}}\big(\texttt{LEZIONE} \bowtie R_1\big)$$

Sottraendo tale insieme dai codici fiscali dei docenti che hanno tenuto delle lezioni, si ottengono i codici fiscali dei docenti che hanno sempre interrogato:

$$\pi_{\texttt{CFDOC}}(\texttt{LEZIONE}) - R_2$$

Riassumendo:

$$\pi_{\texttt{CFDOC}}\,(\texttt{LEZIONE}) - \pi_{\texttt{CFDOC}}\,\big(\texttt{LEZIONE} \bowtie \big(\pi_{\texttt{CODARG,DATA}}(\texttt{LEZIONE}) - \pi_{\texttt{CODARG,DATA}}(\texttt{INTERROGAZIONE})\,\big)\big)$$

d)
```
SELECT   X.CODARG, avg(X.VOTO)
FROM     INTERROGAZIONE X
WHERE    3 <=  (  SELECT COUNT(*)
                  FROM   INTERROGAZIONE Y
                  WHERE  Y.CODARG = X.CODARG
                  AND    Y.CFSTUD = X.CFSTUD)
GROUP BY X.CODARG
```

e)
```
SELECT   IX.CFSTUD, LEZIONE.CFDOC
FROM     LEZIONE , INTERROGAZIONE IX
WHERE    IX.CODARG = LEZIONE.CODARG
AND      IX.DATA = LEZIONE.DATA
GROUP BY IX.CFSTUD, LEZIONE.CFDOC
HAVING   COUNT(*) >= ALL  ( SELECT   COUNT(*)
                            FROM     LEZIONE L, INTERROGAZIONE I
                            WHERE    L.CODARG = I.CODARG
                            AND      L.DATA = I.DATA
                            AND      IX.CFSTUD = I.CFSTUD
                            GROUP BY I.CFSTUD, L.CFDOC)
```

Esercizio 2

1a) $\texttt{CICLISTA} \ltimes \big(\sigma_{\texttt{PARTENZA='Milano'}}(\texttt{GARA}) \bowtie \sigma_{\texttt{POSIZIONE='Primo'}}(\texttt{PARTECIPA})\big)$

1b)
```
SELECT *
FROM   CICLISTA
WHERE  NOMECICLISTA NOT IN  (  SELECT NOMECICLISTA
                               FROM   PARTECIPA
                               WHERE  NOMECORSA='Giro'
                               AND    POSIZIONE='R')
```

2b) $\texttt{CICLISTA} - \texttt{CICLISTA} \ltimes \big(\sigma_{\texttt{NOMECORSA='Giro' and POSIZIONE='R'}}(\texttt{PARTECIPA})\big)$

1c)
```
SELECT NOMECORSA
FROM   GARA GX
WHERE  NOT EXISTS
         ( SELECT *
           FROM   GARA GY
           WHERE   GX.NOMECORSA = GY.NOMECORSA
           AND    NOT EXISTS
                    ( SELECT *
                      FROM   PARTECIPA P
                      WHERE  GY.NOMECORSA=P.NOMECORSA
                      AND    GY.ANNO=P.ANNO
                      AND    P.POSIZIONE='R'))
```

2c) Indichiamo con R_1 le gare ciclistiche (limitandoci alla chiave) per le quali non c'è stato nessun ciclista ritirato

$R_1 = \pi_{\texttt{NOMECORSA,ANNO}}(\texttt{GARA}) - \pi_{\texttt{NOMECORSA,ANNO}}\big(\sigma_{\texttt{POSIZIONE='R'}}(\texttt{PARTECIPA})\big)$

Proiettando R_1 su NOMECORSA si ottengono i nomi delle corse per le quali c'e' stata almeno una edizione *senza* ritirati. Sottraendo tale insieme da tutti i nomi delle corse, specificati in GARA, si ottengono i nomi delle corse per le quali, in ogni edizione, c'è stato almeno un ritirato:

$$\pi_{\texttt{NOMECORSA}}(\texttt{GARA}) \; - \; \pi_{\texttt{NOMECORSA}}(R_1)$$

1d)
```
SELECT   NOMECORSA,ANNO
FROM     PARTECIPA PX
WHERE    POSIZIONE='R'
GROUP BY NOMECORSA,ANNO
HAVING   COUNT(*) >= ALL
           ( SELECT   COUNT(*)
             FROM     PARTECIPA PY
             WHERE    POSIZIONE='R'
             AND      PY.NOMECORSA=PX.NOMECORSA
             GROUP BY ANNO)
```

Esercizio 3

2a) $\pi_{\texttt{SALA}}\big(\sigma_{\texttt{ANNO}='1997'}(\texttt{MOSTRA}) \bowtie \texttt{ESPONE} \bowtie \sigma_{\texttt{AUTORE}='\texttt{Picasso}'}(\texttt{QUADRO})\big)$

1b)
```
SELECT *
FROM   QUADRO
WHERE  AUTORE='Picasso'
AND    CQ NOT IN  (  SELECT CQ
                     FROM   MOSTRA, ESPONE
                     WHERE  MOSTRA.CM=ESPONE.CM
                     AND    ANNO='1997'  )
```

2b) $\sigma_{\texttt{AUTORE}='\texttt{Picasso}'}(\texttt{QUADRO}) - \texttt{QUADRO} \ltimes \big(\texttt{ESPONE} \bowtie \sigma_{\texttt{ANNO}='1997'}(\texttt{MOSTRA})\big)$

1c)
```
SELECT *
FROM   QUADRO
WHERE  CQ NOT IN  (  SELECT E2.CQ
                     FROM   QUADRO, ESPONE E1, ESPONE E2
                     WHERE  QUADRO.CQ=E1.CQ
                     AND    E1.CM=E2.CM
                     AND    AUTORE='Picasso')
```

2c) Se indichiamo con R_1 i codici delle mostre nelle quali è stato esposto almeno un quadro di Picasso

$$R_1 = \pi_{\texttt{CM}}\big(\texttt{ESPONE} \bowtie \sigma_{\texttt{AUTORE}='\texttt{Picasso}'}(\texttt{QUADRO})\big)$$

i codici dei quadri esposti in tali mostre si ottengono dal join di ESPONE con R_1. Sottraendo tali quadri dalla relazione QUADRO si ottengono quelli richiesti dall'interrogazione:
$\texttt{QUADRO} - \texttt{QUADRO} \ltimes \big(\texttt{ESPONE} \bowtie \pi_{\texttt{CM}}\big(\texttt{ESPONE} \bowtie \sigma_{\texttt{AUTORE}='\texttt{Picasso}'}(\texttt{QUADRO})\big)\big)$

d)
```
SELECT *
FROM   MOSTRA
WHERE  5 >=  (  SELECT COUNT(DISTINCT AUTORE)
                FROM   QUADRO,ESPONE
                WHERE  QUADRO.CQ=ESPONE.CQ
                AND    MOSTRA.CM=ESPONE.CM)
```

e)
```
SELECT   CM,AUTORE
FROM     QUADRO Q1,ESPONE E1
WHERE    Q1.CQ=E1.CQ
GROUP BY CM,AUTORE
HAVING   COUNT(*) >= ALL  ( SELECT   COUNT(*)
                            FROM     QUADRO Q2,ESPONE E2
                            WHERE    Q2.CQ=E2.CQ
                            AND      E2.CM=E1.CM
                            GROUP BY AUTORE)
```

Esercizio 4

1a)
```
SELECT *
FROM   PRODOTTO
WHERE  PRODOTTO.CP NOT IN
         ( SELECT C2.CP
           FROM   COMPRENDE C1, PRODOTTO, COMPRENDE C2
           WHERE  C1.CP=PRODOTTO.CP
           AND    C2.CO=C1.C0
           AND    PRODOTTO.DESCRIZIONE='Uva')
```

2a) $\texttt{PRODOTTO} - \texttt{PRODOTTO} \ltimes \big(\texttt{COMPRENDE} \bowtie \pi_{\texttt{CO}}\big(\texttt{COMPRENDE} \bowtie \sigma_{\texttt{DESCRIZIONE}='\texttt{Uva}'}(\texttt{PRODOTTO})\big)\big)$

1b)
```
SELECT *
FROM   SOCIO
WHERE  SOCIO.CS NOT IN
         ( SELECT RITIRA.CS
           FROM   COMPRENDE, PRODOTTO, RITIRA
           WHERE  COMPRENDE.CP=PRODOTTO.CP
           AND    COMPRENDE.CO= RITIRA.CO
           AND    PRODOTTO.DESCRIZIONE='Uva')
```

2b) $\texttt{SOCIO} - \texttt{SOCIO} \ltimes \big(\texttt{RITIRA} \bowtie \pi_{\texttt{CO}}\big(\texttt{COMPRENDE} \bowtie \sigma_{\texttt{DESCRIZIONE}='\texttt{Uva}'}(\texttt{PRODOTTO})\big)\big)$

1c)
```
SELECT *
FROM   SOCIO
WHERE  NOT EXISTS
         ( SELECT *
           FROM   COMPRENDE,PRODOTTO
           WHERE  COMPRENDE.CP=PRODOTTO.CP
           AND    PRODOTTO.DESCRIZIONE ='Uva'
           AND    NOT EXISTS
                    ( SELECT *
                      FROM   RITIRA
                      WHERE  SOCIO.CS= RITIRA.CS
                      AND    COMPRENDE.CO= RITIRA. CO))
```

2c) $\texttt{SOCIO} \bowtie \big(\pi_{\texttt{CS,CO}}\big(\texttt{RITIRA}\big) \div \pi_{\texttt{CO}}\big(\texttt{COMPRENDE} \bowtie \sigma_{\texttt{DESCRIZIONE}='\texttt{Uva}'}(\texttt{PRODOTTO})\big)\big)$

d)
```
SELECT   RITIRA.CS, COUNT(*)
FROM     COMPRENDE, PRODOTTO, RITIRA
WHERE    COMPRENDE.CP = PRODOTTO.CP
AND      COMPRENDE.CO = RITIRA. CO
AND      PRODOTTO.DESCRIZIONE ='Uva'
GROUP BY RITIRA.CS
```

Esercizio 5

1a)
```
SELECT  *
FROM    VIA
WHERE   CV IN ( SELECT CVA
                FROM   INC,VIA
                WHERE  INC.CVB=VIA.CV
                AND    QUARTIERE='Pastena'
OR      CV IN ( SELECT CVB
                FROM   INC,VIA
                WHERE  INC.CVA=VIA.CV
                AND    QUARTIERE='Pastena'
```

2a) $R_1 = \pi_{\texttt{CV}}(\sigma_{\texttt{QUARTIERE='Pastena'}}(\texttt{VIA}))$

$\texttt{VIA} \ltimes_{\texttt{CV=CVB}} \pi_{\texttt{CVB}}(\texttt{INC} \bowtie_{\texttt{CVA=CV}} (R_1)) \cup \texttt{VIA} \ltimes_{\texttt{CV=CVA}} \pi_{\texttt{CVA}}(\texttt{INC} \bowtie_{\texttt{CVB=CV}} (R_1))$

1b)
```
SELECT  *
FROM    VIA
WHERE   CV NOT IN ( SELECT CVA
                    FROM   INC,VIA
                    WHERE  INC.CVB=VIA.CV
                    AND    NOME='MarcoPolo')
AND     CV NOT IN ( SELECT CVB
                    FROM   INC,VIA
                    WHERE  INC.CVA=VIA.CV
                    AND    NOME='MarcoPolo')
```

2b) $R_2 = \pi_{\texttt{CV}}(\sigma_{\texttt{QUARTIERE='MarcoPolo'}}(\texttt{VIA}))$

$\texttt{VIA} - (\texttt{VIA} \ltimes_{\texttt{CV=CVB}} \pi_{\texttt{CVB}}(\texttt{INC} \bowtie_{\texttt{CVA=CV}} (R_2)) \cup \texttt{VIA} \ltimes_{\texttt{CV=CVA}} \pi_{\texttt{CVA}}(\texttt{INC} \bowtie_{\texttt{CVB=CV}} (R_2)))$

1c)
```
SELECT V1.CV AS CODICE1, V2.CV AS CODICE2
FROM   VIA V1, VIA V2
WHERE  V1.LUNGHEZZA=V2.LUNGHEZZA
AND    V1.CV > V2.CV
```

d)
```
SELECT   QUARTIERE
FROM     VIA
GROUP BY QUARTIERE
HAVING   COUNT(*) >= ALL  ( SELECT   COUNT(*)
                            FROM     VIA
                            GROUP BY QUARTIERE)
```

e)
```
SELECT QUARTIERE,CV
FROM   VIA V1
WHERE  LUNGHEZZA =  (  SELECT MAX(LUNGHEZZA)
                       FROM   VIA V2
                       WHERE  V1.QUARTIERE=V2.QUARTIERE)
```

```
f)  SELECT *
    FROM   VIA X
    WHERE  NOT EXISTS
             ( SELECT *
               FROM   VIA Y
               WHERE  QUARTIERE='Pastena'
               AND    NOT EXISTS (  SELECT *
                                    FROM   INC
                                    WHERE  (X.CV=CVA AND Y.CV=CVB)
                                    OR     (X.CV=CVB AND Y.CV=CVA))
```

La soluzione in SQL di **a)**, **b)** e **f)** può essere semplicficata usando la view INCSIM che contiene le tuple di `INC` e le rispettive tuple simmetriche:

```
CREATE  VIEW INCSIM(CV,CVINC)
AS      SELECT CVA,CVB
        FROM   INC
        UNION
        SELECT CVB,CVA
        FROM   INC
```

Con tale view, possiamo risolvere le interrogazioni a), b) e f) come segue:

```
1a)  SELECT V1.*
     FROM   VIA V1, INCSIM, VIA V2
     WHERE  V1.CV=INCSIM.CV
     AND    INCSIM.CVINC=V2.CV
     AND    V2.QUARTIERE='Pastena'
```

```
1b)  SELECT *
     FROM   VIA
     WHERE  CV NOT IN  (   SELECT INCSIM.CV
                           FROM   VIA,INCSIM
                           WHERE  INCSIM.CVINC=VIA.CV
                           AND    NOME='MarcoPolo')
```

```
f)  SELECT *
    FROM   VIA X
    WHERE  NOT EXISTS
             ( SELECT *
               FROM   VIA Y
               WHERE  QUARTIERE='Pastena'
               AND    NOT EXISTS
                        ( SELECT *
                          FROM   INCSIM
                          WHERE  X.CV=INCSIM.CV
                          AND    Y.CV=INCSIM.CVINC))
```

Esercizio 6

1a)
```
SELECT INCONTRO.*
FROM   INCONTRO,CAMPO
WHERE  INCONTRO.NCAMPO= CAMPO.NCAMPO
AND    TIPO='erba'
```

2a) $\text{Ra}= \texttt{INCONTRO} \bowtie \pi_{\texttt{NCAMPO}}\big(\ \sigma_{\texttt{TIPO='erba'}}\ (\ \texttt{CAMPO})\big)$

1b)
```
SELECT *
FROM   CAMPO
WHERE  TIPO='erba'
AND    NCAMPO NOT IN
         ( SELECT NCAMPO
           FROM   INCONTRO)
```

2b) $\text{Rb}= \sigma_{\texttt{TIPO='erba'}}\big(\texttt{CAMPO}\big) - \texttt{CAMPO} \bowtie \pi_{\texttt{NCAMPO}}\big(\texttt{INCONTRO}\big)$

1c)
```
SELECT  *
FROM    TENNISTA
WHERE   CF IN ( SELECT GIOC1
                FROM   INCONTRO I,CAMPO C
                WHERE  I.NCAMPO= C.NCAMPO
                AND    TIPO='erba'
                AND    SET1 > SET2)
OR      CF IN ( SELECT GIOC2
                FROM   INCONTRO I, CAMPO C
                WHERE  I.NCAMPO= C.NCAMPO
                AND    TIPO='erba'
                AND    SET1 < SET2)
```

2c) $\text{Rc}= \texttt{TENNISTA} \bowtie_{\texttt{CF=GIOC1}} \pi_{\texttt{GIOC1}}\ \big(\sigma_{\texttt{SET1>SET2}}\big(\ \text{Ra}\ \big)\big) \cup$
$\texttt{TENNISTA} \bowtie_{\texttt{CF=GIOC2}} \pi_{\texttt{GIOC2}}\ \big(\sigma_{\texttt{SET1<SET2}}\big(\ \text{Ra}\ \big)\big)$

1d)
```
SELECT  DISTINCT NAZIONE
FROM    TENNISTA
WHERE   NAZIONE NOT IN ( SELECT NAZIONE
                         FROM   INCONTRO,TENNISTA
                         WHERE  GIOC1= CF
                         AND    SET1 < SET2)
AND     NAZIONE NOT IN ( SELECT NAZIONE
                         FROM   INCONTRO,TENNISTA
                         WHERE  GIOC2= CF
                         AND    SET1 > SET2)
```

2d) Se indichiamo con $R1$ i giocatori che hanno perso almeno una partita:

$R1 = \ \texttt{TENNISTA} \bowtie_{\texttt{CF=GIOC1}} \pi_{\texttt{GIOC1}}\ \big(\sigma_{\texttt{SET1<SET2}}\big(\ \texttt{INCONTRO}\ \big)\big) \cup$
$\texttt{TENNISTA} \bowtie_{\texttt{CF=GIOC2}} \pi_{\texttt{GIOC2}}\ \big(\sigma_{\texttt{SET1>SET2}}\big(\ \texttt{INCONTRO}\ \big)\big)$

allora le nazioni dei giocatori che hanno sempre vinto sono:
$\pi_{\texttt{NAZIONE}}\big(\ \texttt{TENNISTA}\ \big) - \pi_{\texttt{NAZIONE}}\big(\ \text{R1}\ \big)$

```
e) SELECT   CAMPO.NCAMPO
   FROM     INCONTRO,CAMPO
   WHERE    INCONTRO.NCAMPO= CAMPO.NCAMPO
   AND      TIPO='erba'
   GROUP BY CAMPO.NCAMPO
   HAVING   COUNT(*) >= ALL  ( SELECT   COUNT(*)
                               FROM     INCONTRO I, CAMPO C
                               WHERE    I.NCAMPO=C.NCAMPO
                               AND      TIPO='erba'
                               GROUP BY CAMPO.NCAMPO)
```

La soluzione in SQL delle interrogazioni c) e d) può essere essere formulata in modo più semplice introducendo la seguente view:

```
CREATE  VIEW INCTEN(CI,GIOC,AVV,SETG,SETA,NCAMPO)
AS      SELECT CI,GIOC1,GIOC2,SET1,SET2,NCAMPO
        FROM   INCONTRO
        UNION
        SELECT CI,GIOC2,GIOC1,SET2,SET1,NCAMPO
        FROM   INCONTRO
```

che riporta, per ogni coppia incontro - giocatore (`CI,GIOC`), l'avversario `AVV`, il numero di set `SETG,SETA` vinti da `GIOC` e `AVV` rispettivamente, e il nome del campo. Con tale view, possiamo risolvere c) e d) come segue:

```
1c) SELECT TENNISTA.*
    FROM   TENNISTA,INCTEN,CAMPO
    WHERE  TENNISTA.CF=INCTEN.GIOC
    AND    INCTEN.NCAMPO=CAMPO.NCAMPO
    AND    SETG > SETA
    AND    TIPO='erba'
```

```
1d) SELECT DISTINCT NAZIONE
    FROM   TENNISTA
    WHERE  NAZIONE NOT IN  (  SELECT NAZIONE
                              FROM   INCTEN,TENNISTA
                              WHERE  GIOC= CF
                              AND    SETG < SETA)
```

Nota) Se nell'interrogazione d) vogliamo escludere gli incontri tra due giocatori della stessa nazione:

```
SELECT DISTINCT NAZIONE
FROM   TENNISTA
WHERE  NAZIONE NOT IN  (  SELECT NAZIONE
                          FROM   INCTEN,TENNISTA T1,TENNISTA T2,
                          WHERE  GIOC=T1.CF
                          AND    AVV=T2.CF
                          AND    T1.NAZIONE <>T2.NAZIONE
                          AND    SETG < SETA)
```

Esercizio 7

1a)
```
SELECT CODOP,NOME FROM OPERATORE
WHERE CODOP NOT IN (SELECT CODOP FROM RECLAMO)
```

2a) $\pi_{\texttt{CODOP,NOME}}(\texttt{OPERATORE}) - \pi_{\texttt{CODOP,NOME}}\left(\texttt{OPERATORE} \bowtie \pi_{\texttt{CODOP}}(\texttt{RECLAMO})\right)$

1b)
```
SELECT LX.CODOP
FROM   LOTTO LX
WHERE  NOT EXISTS
         ( SELECT *
           FROM   LOTTO LY
           WHERE  LX.CODOP = LY.CODOP
           AND    NOT EXISTS
                    ( SELECT *
                      FROM   RECLAMO
                      WHERE  LY.CODOP = RECLAMO.CODOP
                      AND    LY.CODART = RECLAMO.CODART ))
```

2b) $\pi_{\texttt{CODOP}}(\texttt{LOTTO}) - \pi_{\texttt{CODOP}}\left(\pi_{\texttt{CODART,CODOP}}(\texttt{LOTTO}) - \pi_{\texttt{CODART,CODOP}}(\texttt{RECLAMO})\right)$

1c)
```
SELECT NOMECL
FROM   RECLAMO RX
WHERE  NOT EXISTS
         ( SELECT *
           FROM   OPERATORE
           WHERE  NOT EXISTS
                    ( SELECT *
                      FROM   RECLAMO RY
                      WHERE  RX.NOMECL = RY.NOMECL
                      AND    OPERATORE.CODOP = RY.CODOP))
```

2c) $\pi_{\texttt{NOMECL,CODOP}}\,(\texttt{RECLAMO}) \div \pi_{\texttt{CODOP}}(\texttt{OPERATORE})$

d)
```
SELECT LX.CODART, LX.CODOP
FROM   LOTTO LX
WHERE  LX.TOTESEM IS NOT NULL
AND    LX.TOTESEM >= ALL  (  SELECT LY.TOTESEM
                             FROM   LOTTO LY
                             WHERE  LY.TOTESEM is not null
                             AND    LX.CODART = LY.CODART)
```

e)
```
SELECT   LOTTO.CODART, LOTTO.CODOP, LOTTO.TOTESEM
FROM     LOTTO,RECLAMO
WHERE    LOTTO.CODART = RECLAMO.CODART
AND      LOTTO.CODOP = RECLAMO.OP
GROUP BY LOTTO.CODART, LOTTO.CODOP,LOTTO.TOTESEM
HAVING   COUNT(*) >= ALL  ( SELECT   COUNT(*)
                            FROM     RECLAMO
                            GROUP BY CODART,CODOP)
```

Esercizio 8

1a)
```
SELECT *
FROM   MANIFESTAZIONE M
WHERE   CM NOT EXISTS  (  SELECT *
                          FROM   LUOGO L, SPETTACOLO S
                          WHERE  M.CM = S.CM
                          AND    L.NOME-LUOGO= S.NOME-LUOGO
                          AND    L.CITTÀ = 'Modena')
```

2a) $\texttt{MANIFESTAZIONE} \bowtie \big(\pi_{\texttt{CM}}\big(\texttt{MANIFESTAZIONE}\big) - \pi_{\texttt{CM}}\big((\texttt{SPETTACOLO} \bowtie \sigma_{\texttt{CITTÀ='Modena'}}\big(\texttt{LUOGO}\big))\big)$

1b)
```
SELECT DISTINCT NOME-LUOGO
FROM   SPETTACOLO X
WHERE  NOT EXISTS
         ( SELECT *
           FROM   MANIFESTAZIONE M
           WHERE  NOT EXISTS
                    ( SELECT *
                      FROM   SPETTACOLO Y
                      WHERE  X.NOME-LUOGO = Y.NOME-LUOGO
                      AND    M.CM=Y.CM))
```

2b) $\pi_{\texttt{NOME-LUOGO,COD-MAN}}\big(\texttt{SPETTACOLO}\big) \div \pi_{\texttt{COD-MAN}}\ \big(\texttt{MANIFESTAZIONE}\big)$

1c)
```
SELECT DISTINCT NOME-LUOGO
FROM   SPETTACOLO SX
WHERE  ORA-INIZIO > 15
AND    3 <  (  SELECT COUNT(*)
               FROM   SPETTACOLO SY
               WHERE  ORA-INIZIO > 15
               AND    SX.NOME-LUOGO = SY.NOME-LUOGO
               AND    SX.DATA = SY.DATA)
```

d)
```
SELECT  NOME-LUOGO, COUNT(distinct CM) as N-MAN,
        COUNT(*) as N-SPET
FROM     SPETTACOLO
GROUP BY NOME-LUOGO
```

e) Seleziona i codici delle manifestazioni i cui spettacoli sono iniziati sempre dopo le ore 15. In SQL:
```
SELECT DISTINCT X.CM
FROM   SPETTACOLO X
WHERE  NOT EXISTS (  SELECT *
                     FROM   SPETTACOLO Y
                     WHERE  X.CM = Y.CM
                     AND    (Y.ORA-INIZIO<=15) )
```

Esercizio 9

1a)
```
SELECT PRODOTTO.*
FROM   PRODOTTO,FORNISCE,FORNITORE
WHERE  PRODOTTO.CODP=FORNISCE.CODP
AND    FORNISCE.CODF=FORNITORE.CODF
AND    ANNO=1994
AND    CITTÀ='MO'
```

2a) $R_a = \texttt{PRODOTTO} \ltimes \left(\sigma_{\texttt{ANNO=1994}}(\texttt{FORNISCE}) \bowtie \sigma_{\texttt{CITTÀ='MO'}}(\texttt{FORNITORE})\right)$

1b)
```
SELECT *
FROM   PRODOTTO
WHERE  CODP NOT IN  (  SELECT CODP
                       FROM   FORNISCE,FORNITORE
                       WHERE  FORNISCE.CODF=FORNITORE.CODF
                       AND    CITTÀ='MO')
```

2b) $\texttt{PRODOTTO} - \texttt{PRODOTTO} \ltimes \left(\texttt{FORNISCE} \bowtie \sigma_{\texttt{CITTÀ='MO'}}(\texttt{FORNITORE})\right)$

1c)
```
SELECT PRODOTTO.*
FROM   PRODOTTO,FORNISCE,FORNITORE
WHERE  PRODOTTO.CODP=FORNISCE.CODP
AND    FORNISCE.CODF=FORNITORE.CODF
AND    ANNO=1994
AND    CITTÀ='MO'
AND    PRODOTTO.CODP NOT IN
         ( SELECT PRODOTTO.CODP
           FROM   PRODOTTO,FORNISCE,FORNITORE
           WHERE  PRODOTTO.CODP=FORNISCE.CODP
           AND    FORNISCE.CODF=FORNITORE.CODF
           AND    ANNO=1994
           AND    CITTÀ <> 'MO')
```

2c) $R_a - \texttt{PRODOTTO} \ltimes \left(\sigma_{\texttt{ANNO=1994}}(\texttt{FORNISCE}) \bowtie \sigma_{\texttt{CITTÀ<>'MO'}}(\texttt{FORNITORE})\right)$

d)
```
SELECT   ANNO,SUM(QTY)
FROM     FORNISCE,FORNITORE
WHERE    FORNISCE.CODF=FORNITORE.CODF
AND      CITTÀ = 'MO'
GROUP BY ANNO
```

e)
```
SELECT   ANNO,CODF
FROM     FORNISCE FX
GROUP BY ANNO,CODF
HAVING   SUM(qty) >=ALL  ( SELECT   SUM(qty)
                           FROM     FORNISCE FY
                           WHERE    FY.ANNO=FX.ANNO
                           GROUP BY CODF)
```

Esercizio 10

2a) $\texttt{GIOCATOREGOLF} \ltimes \left(\texttt{PARTECIPA} \bowtie \sigma_{\texttt{LIVELLO='nazionale'}}(\texttt{GARA}) \right)$

1b)
```
SELECT NAZIONE
FROM   GIOCATOREGOLF
WHERE  NAZIONE NOT IN
         ( SELECT NAZIONE
           FROM   GIOCATOREGOLF G, PARTECIPA P
           WHERE  G.CF= P.CF AND PUNTEGGIO > 0 )
```

2b) $\pi_{\texttt{NAZIONE}}(\texttt{GIOCATOREGOLF}) - \pi_{\texttt{NAZIONE}} \left(\texttt{GIOCATOREGOLF} \bowtie \sigma_{\texttt{LIVELLO='nazionale'}} (\texttt{PARTECIPA}) \right)$

1c)
```
SELECT *
FROM   GIOCATOREGOLF G
WHERE  NOT EXISTS
          ( SELECT *
            FROM   GARA
            WHERE  LIVELLO='nazionale'
            AND    NOT EXISTS
                     ( SELECT *
                       FROM   PARTECIPA P
                       WHERE  GARA.CG=P.CG
                       AND    G.CF = P.CF))
```

2c) $\texttt{GIOCATOREGOLF} \bowtie \left(\pi_{\texttt{CF,CG}} (\texttt{PARTECIPA}) \div \pi_{\texttt{CG}} \left(\sigma_{\texttt{LIVELLO='nazionale'}} (\texttt{GARA}) \right) \right)$

d)
```
SELECT *
FROM   GIOCATOREGOLF G
WHERE  CF IN  (  SELECT P.CF-GIOCATOREGOLF
                 FROM   GARA, PARTECIPA P
                 WHERE  GARA.CG=P.CG
                 AND    LIVELLO='internazionale'
                 AND    PUNTEGGIO =
                            ( SELECT MIN(PUNTEGGIO)
                              FROM   PARTECIPA P1
                              WHERE  P1.CG=P.CG) )
```

e)
```
SELECT    G.NAZIONE, AVG(P.PUNTEGGIO)
FROM      GIOCATOREGOLF G,GARA, PARTECIPA P
WHERE     G.CF= P.CF
AND       GARA.CG=P.CG
AND       LIVELLO='internazionale'
GROUP BY  G.NAZIONE
HAVING    COUNT(distinct P.CF) > 5
ORDER BY  2 DESC
```

6.3 Normalizzazione

♢ Esercizio 1

Dato il seguente schema di relazione:

`R(A,B,C,D,E)`

e considerando le seguenti dipendenze funzionali:

- `(FD1) A B` $\rightarrow$ `C E`
- `(FD2) C` $\rightarrow$ `D`
- `(FD3) D` $\rightarrow$ `B`

viene richiesto di:

1. Determinare la chiave o le chiavi dello schema di relazione;
2. Determinare se lo schema di relazione è in 2NF, 3NF e BCNF;
3. Produrre eventuali decomposizioni e discutere la preservazione dei dati e delle dipendenze funzionali;
4. Produrre uno schema E/R che descriva lo schema di relazione e soddisfi le dipendenze funzionali date.

♢ Esercizio 2

Dato il seguente schema di relazione:

`Magazzino(Locale,Prodotto,Stanza,Scaffale)`

e considerando i seguenti vincoli:

- Un prodotto è immagazzinato in uno ed un solo locale;
- un prodotto può essere immagazzinato in uno o più stanze e in uno o più scaffali;
- in una stanza di un locale, uno scaffale immagazzina un preciso prodotto.

viene richiesto di:

1. Determinare le dipendenze funzionali (non banali) insite nello schema di relazione.
2. Determinare la chiave o le chiavi dello schema di relazione.
3. Determinare se lo schema di relazione è in 2NF, 3NF e BCNF.
4. Produrre eventuali decomposizioni e discutere la preservazione dei dati e delle dipendenze funzionali.

Esempio di istanza r dello schema di relazione `Magazzino` che soddisfa i vincoli:

Locale	Prodotto	Stanza	Scaffale
Pelletteria	Scarpa	A	15
Pelletteria	Scarpa	B	6
Pelletteria	Cintura	A	16
Abbigliamento	Pantalone	A	14
Abbigliamento	Pantalone	A	15

♢ **Esercizio 3**

Dato il seguente schema di relazione:

`Tornei(Torneo,Squadra,Categoria,Capitano)`

e considerando i seguenti vincoli:

- Un capitano gioca in una precisa squadra di una precisa categoria;
- Per ogni squadra e categoria c'è un solo capitano;
- Per ogni torneo, una squadra partecipa con una sola categoria;

viene richiesto di:

1. Determinare le dipendenze funzionali (non banali) insite nello schema di relazione.
2. Determinare la chiave o le chiavi dello schema di relazione.
3. Determinare se lo schema di relazione è in 2NF, 3NF e BCNF.
4. Produrre eventuali decomposizioni e discutere la preservazione dei dati e delle dipendenze funzionali.

Esempio di istanza r dello schema di relazione `Tornei` che soddisfa i vincoli:

Torneo	Squadra	Categoria	Capitano
TorneoEstivo1997	Modena	Ragazzi	Neri
TorneoTopolino1998	Modena	Pulcini	Bianchi
TorneoDiNatale1997	Modena	Juniores	Pergola
TorneoTopolino1997	Modena	Pulcini	Bianchi
TorneoEstivo1997	Bologna	Pulcini	Verdi
TorneoDiNatale1997	Bologna	Ragazzi	Russo
TorneoTopolino1998	Bologna	Pulcini	Verdi
TorneoEstivo1997	Firenze	Pulcini	Rossi

♢ Esercizio 4

Dato il seguente schema di relazione:

`R(A,B,C,D)`

e considerando le seguenti dipendenze funzionali:

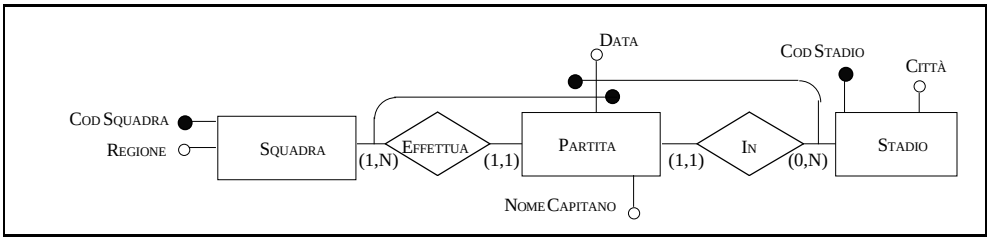
- `(FD1)` $A \rightarrow B$
- `(FD2)` $BC \rightarrow D$
- `(FD3)` $A \rightarrow C$

viene richiesto di:

1. Determinare la chiave o le chiavi dello schema di relazione;
2. Determinare se lo schema di relazione è in 2NF, 3NF e BCNF;
3. Produrre eventuali decomposizioni e discutere la preservazione dei dati e delle dipendenze funzionali.

♢ Esercizio 5

Dato il seguente schema E/R:



si consideri la seguente traduzione in schema relazionale:

`Squadra(CodSquadra,Regione)`
`Stadio(CodStadio,Città)`
`Partita(Data,CodSquadra,NomeCapitano,CodStadio)`

e si consideri sullo schema di relazione `Partita` la seguente dipendenza funzionale aggiuntiva (che esprime il vincolo che un capitano gioca in una sola squadra): `(FD)` `NomeCapitano` $\rightarrow$ `CodSquadra`.

Viene richiesto di

1. Determinare la chiave o le chiavi dello schema di relazione `Partita`, prendendo in considerazione sia i vincoli dello schema E/R sia la dipendenza funzionale aggiunta;
2. Determinare se lo schema di relazione `Partita` è in 2NF, 3NF e BCNF;
3. Produrre eventuali decomposizioni dello schema di relazione `Partita` che preservono i dati ma non le dipendenze e scrivere una query SQL che ne controlli la violazione.

♢ **Esercizio 6**

Dato il seguente schema di relazione:

`R(A,B,C,D)`

e considerando le seguenti dipendenze funzionali:

- `(FD1)` $AB \rightarrow C$
- `(FD2)` $AB \rightarrow D$
- `(FD3)` $C \rightarrow A$
- `(FD4)` $D \rightarrow B$

viene richiesto di:

1. Determinare la chiave o le chiavi dello schema di relazione.
2. Determinare se lo schema di relazione è in 2NF, 3NF e BCNF.
3. Produrre eventuali decomposizioni e discutere la preservazione dei dati e delle dipendenze funzionali.

♢ **Esercizio 7**

Dato il seguente schema di relazione:

`Libretto(Matricola,NomeStudente,Corso,Professore,Voto)`

e considerando i seguenti vincoli:

- Ad ogni studente viene attribuito un numero di matricola unico;
- Ogni studente può registrare un unico voto per ogni corso seguito;
- Ogni professore tiene un unico corso;
- Ogni corso può essere tenuto da più professori (es. Rossi e Neri tengono entrambi un corso di Informatica I); gli studenti vengono assegnati ai corsi sulla base del loro nome (es. Bianchi ha seguito Informatica I con Rossi mentre Verdi lo ha seguito con Neri);
- Possono essere attivati corsi anche senza studenti iscritti.

viene richiesto di:

1. Determinare le dipendenze funzionali (non banali) insite nello schema di relazione;
2. Determinare la chiave o le chiavi dello schema di relazione;
3. Determinare se lo schema di relazione è in 2NF, 3NF e BCNF;
4. Produrre eventuali decomposizioni e discutere la preservazione dei dati e delle dipendenze funzionali;

5. Discutere brevemente se le relazioni in BCNF ottenute presentano comunque problemi.

♢ **Esercizio 8**

Dato il seguente schema di relazione:

`Progetti(Anno,Progetto,CapoProgetto,Reparto,Responsabile)`

e considerando le seguenti dipendenze funzionali:

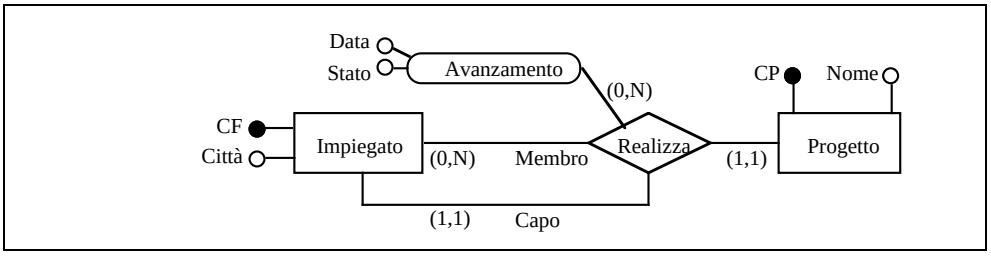
- (FD1) `Anno Progetto` → `CapoProgetto`
- (FD2) `CapoProgetto` → `Reparto`
- (FD3) `Reparto` → `Responsabile`
- (FD4) `Anno Progetto` → `Responsabile`

viene richiesto di:

1. Determinare la chiave o le chiavi dello schema di relazione;
2. Determinare se lo schema di relazione è in 2NF, 3NF e BCNF;
3. Produrre eventuali decomposizioni e discutere la preservazione dei dati e delle dipendenze funzionali;
4. Produrre uno schema E/R che descriva lo schema di relazione e soddisfi le dipendenze funzionali date.

♢ **Esercizio 9**

Dato il seguente schema E/R:



viene richiesto di

1. Tradurre lo schema E/R in schema relazionale in terza forma normale;
2. Aggiungere allo schema relazionale ottenuto al punto 1) il vincolo che in una certa Data lo Stato di Avanzamento di un certo Progetto è unico;
3. Tradurre lo schema E/R in un'unico schema di relazione e
   (a) determinare le dipendenze funzionali (non banali) insite nello schema di relazione;
   (b) determinare la chiave o le chiavi dello schema di relazione;
   (c) Determinare se lo schema di relazione è in 2NF, 3NF e BCNF;
   (d) Determinare una eventuale decomposizione almeno in 3NF che sia lossless.

6.3.1 Soluzioni

Nella soluzione di questi esercizi la normalizzazione di uno schema $\langle R, F\rangle$ viene effettuata in maniera "euristica", applicando ricorsivamente il criterio di decomposizione:

> Uno schema $\langle R(X, Y, A), F\rangle$ che non è nella forma normale desiderata a causa di $X \to A$ viene decomposto in $R1(X, Y)$ e $R2(X, A)$.

Inoltre la violazione della forma normale desiderata sugli schemi $R1$ e $R2$ verrà generalmente verificata solo rispetto alle dipendenze di F proiettate sugli schemi di $R1$ e $R2$.

Esercizio 1

1. Le chiavi dello schema di relazione sono K_1=AB, K_2=AC e K_3=AD.
   Ad esempio, dimostriamo che AC → ADBCE:

1	C → B	applicazione della transitività alla FD2 e FD3
2	C → DB	applicazione dell'unione alla **1** e FD2
3	AC → ADB	applicazione della estensione alla **2** con A
4	AC → ADBCE	applicazione dell'unione alla **3** e FD1

   Quindi AC è superchiave; è facile verificare che è anche chiave.
2. Lo schema di relazione non è in BCNF a causa sia della FD2 che della FD3. Lo schema di relazione è in 3NF nonostante la presenza di tali dipendenze funzionali, in quanto gli attributi B e D sono attributi primi.
3. Consideriamo le seguenti decomposizioni:
   - Siccome lo schema non è in BCNF a causa della FD2, consideriamo la decomposizione binaria:

     R1(C,D), con dipendenza funzionale FD2

     R2(A,B,C,E), con dipendenza funzionale FD1

     Tale decomposizione è lossless in quanto il join naturale riguarda l'attributo C che è chiave in R1, ma non preserva la dipendenza FD3. Lo schema di relazione R2 ha come chiavi K_1= AB e K_2= AC.
     Entrambi i sottoschemi relazionali ottenuti risultano essere in BCNF.
   - Se si decompone per proiezione sulla base della dipendenza funzionale FD3 che viola la BCNF si ottiene:

     R1(D,B), con dipendenza funzionale FD3

     R2(A,C,D,E), con dipendenza funzionale FD2

     Tale decomposizione è lossless in quanto il join naturale riguarda l'attributo D che è chiave in R1, ma non preserva la dipendenza

FD3. Lo schema di relazione R2 ha come chiavi K_1= AB e K_2= AC.
Entrambi i sottoschemi relazionali ottenuti risultano essere in BCNF.

Esercizio 2

1. Dato che un prodotto è immagazzinato in uno ed un solo locale, è valida la (FD1) Prodotto → Locale;

   inoltre, dato che in una stanza di un locale, uno scaffale immagazzina un preciso prodotto, si ha che (FD2) Locale Stanza Scaffale → Prodotto.
2. Le chiavi dello schema di relazione sono K_1 = Locale Stanza Scaffale e K_2= Prodotto Stanza Scaffale.
3. Lo schema di relazione non è in BCNF a causa della FD1. Lo schema di relazione è in 3NF nonostante la presenza di tale dipendenza funzionale, in quanto l'attributo Locale è un attributo primo.
4. Siccome lo schema non è in BCNF a causa della FD1, consideriamo la decomposizione binaria:

   Magazzino1(Prodotto,Locale), con dipendenza funzionale FD1

   Magazzino2(Prodotto,Stanza,Scaffale)

   Tale decomposizione è lossless ma non preserva la dipendenza FD2.
   Entrambi i sottoschemi risultano essere in BCNF.

Esercizio 3

1. Un capitano gioca in una precisa squadra di una precisa categoria:
   (FD1) Capitano → Squadra e (FD2) Capitano → Categoria;
   per ogni squadra e categoria c'è un solo capitano:
   (FD3) Squadra Categoria → Capitano;
   per ogni torneo, una squadra partecipa con una sola categoria:
   (FD4) Squadra Torneo → Categoria.
2. Le chiavi sono K_1 = Torneo Squadra e K_2 = Torneo Capitano.
3. Lo schema di relazione non è in 2NF sia a causa della FD1 che della FD2; Inoltre, lo schema di relazione non è in BCNF a causa della FD3.
4. Consideriamo le seguenti decomposizioni:
   (a) per proiezione sulla base di FD1 e FD2 che violano la 2NF:
   Tornei1(Capitano,Squadra,Categoria), con FD1 e FD2
   Tornei2(Torneo,Capitano)
   (b) proiezione sulla base di FD3 che viola la BCNF:
   Tornei1(Capitano,Squadra,Categoria), con FD1 e FD2
   Tornei3(Torneo,Squadra,Capitano), con FD4

Entrambe queste decomposizioni sono lossless, preservano le dipendenze funzionali e generano sottoschemi che risultano essere in BCNF.

Esercizio 4

1. Lo schema di relazione ha come unica chiave $\texttt{K}_1$ = A.
2. Lo schema di relazione è in 2NF ma non è in 3NF a causa della FD2.
3. La decomposizione R1(A,B,C), R2(B,C,D) è lossless, preserva le dipendenze funzionali e genera sottoschemi che risultano essere in BCNF.

Esercizio 5

1. Dallo schema E/R si ha che Partita ha come chiavi $\texttt{K}_1$=Data CodSquadra e $\texttt{K}_2$=Data CodStadio. Dalla (FD) aggiuntiva, risulta che un'ulteriore chiave di Partita è $\texttt{K}_3$ = Data NomeCapitano.
2. Partita non è in BCNF a causa della FD aggiuntiva (è in 3NF in quanto l'attributo CodSquadra è primo).
3. Siccome lo schema non è in BCNF a causa di FD, si considera:

   Capitano(NomeCapitano,CodSquadra)

   Partita1(Data,NomeCapitano,CodStadio), con chiavi $\texttt{K}_2$ e $\texttt{K}_3$

   Entrambi i sottoschemi risultano essere ora in BCNF. Tale decomposizione è lossless ma non preserva la dipendenza (dovuta alla chiave $\texttt{K}_1$) (FD1) Data CodSquadra $\rightarrow$ CodStadio NomeCapitano.
   Tale dipendenza vieta l'introduzione di una tupla (X, Y, Z, W) in una istanza di Partita, se l'istanza contiene già una tupla (partita) con data X e squadra Y. Per controllare che nello schema decomposto tale dipendenza non sia violata, prima di inserire (Z, Y) in un'istanza r_1 di Capitano e (X, Z, W) in un'istanza r_2 di Partita1, occorre verificare che nella data X per la squadra Y di Z non ci sia già in r_2 una partita, ovvero che il risultato della seguente query sia zero:

```
SELECT count(*)
FROM   Partita1, Capitano
WHERE  Partita1.NomeCapitano=Capitano.NomeCapitano
AND    Capitano.CodSquadra=Y
AND    Partita1.Data=X
```

   Se invece si inserisce la sola tupla (X, Z, W) in r_2 occorre determinare quale sia la squadra di Z (self-join su Capitano) e quindi verificare che nella data X non ci sia già in r_2 una partita per tale squadra:

```
SELECT count(*)
FROM   Partita1, Capitano C1, Capitano C2
WHERE  C1.NomeCapitano=Z
AND    C1.CodSquadra=C2.CodSquadra
AND    Partita1.NomeCapitano=C2.NomeCapitano
AND    Capitano.CodSquadra=Y
AND    Partita1.Data=X
```

Esercizio 6

1. Le chiavi dello schema di relazione sono:
   $\mathtt{K}_1$ = AB, $\mathtt{K}_2$ = BC, $\mathtt{K}_3$ = CD e $\mathtt{K}_4$ = DA.

2. Lo schema di relazione non è in BCNF a causa sia della FD3 che della FD4 (è in 3NF, in quanto gli attributi A e B sono attributi primi).

3. Essendo lo schema non in BCNF a causa della FD3, consideriamo la decomposizione binaria:

   R1(AC) con FD3, R2(BCD) con FD4

   Tale decomposizione è lossless ma non preserva le dipendenze FD1 e FD2. Lo schema R2 non è in BCNF a causa di FD4, quindi:

   R21(CD), R22(BD) con FD4

   che risulta essere lossless e preserva le dipendenze funzionali.
   In definitiva, si ottiene la decomposizione R1(AC), R21(CD) e R22(BD), che risulta essere lossless ma non preserva le dipendenze FD1 e FD2. Lo schema relazionale risultante è in BCNF.

Esercizio 7

1. Le dipendenze funzionali sono:

   - (FD1) Matricola → NomeStudente
   - (FD2) Professore → Corso
   - (FD3) NomeStudente Corso → Professore
   - (FD4) Matricola Corso → Voto

2. Le chiavi dello schema di relazione sono $\mathtt{K}_1$ = Matricola Professore e K_2= Matricola Corso.

3. Lo schema di relazione non è in 2NF a causa della FD1. Lo schema di relazione non è in BCNF sia a causa della FD2 che della FD3.

4. Siccome lo schema non è in 2NF a causa della FD1:

   Libretto1(Matricola,NomeStudente), con FD1

`Libretto2(Matricola,Corso,Professore,Voto)`, con `FD2` e `FD4`

Tale decomposizione è lossless ma non preserva la dipendenza `FD3`. Lo schema di relazione `Libretto2` non è in BCNF a causa `FD2`, quindi:

`Libretto21(Professore,Corso)`, con `FD2`

`Libretto22(Matricola,Professore,Voto)`

Tale decomposizione è lossless ma non preserva la dipendenza `FD4`. Entrambi i sottoschemi risultano essere ora in BCNF.

Esercizio 8

È facile verificare la dipendenza `FD4` è ridondante rispetto alle altre dipendenze e quindi non viene presa in considerazione.

1. Lo schema di relazione ha un'unica chiave K_1=`Anno Progetto`.
2. Lo schema di relazione non è in 3NF a causa sia della `FD2` che della `FD3`.
3. Decomponiamo lo schema sulla base della `FD2` che viola la 3NF:

   `R1(CapoProgetto,Reparto)`, con `FD2`

   `R2(Anno,Progetto,CapoProgetto,Responsabile)`, con `FD1`

   Tale decomposizione è lossless ma non preserva la dipendenza `FD3`.
   Invece se decomponiamo lo schema sulla base della `FD3` che viola la 3NF:

   `R1(Reparto,Responsabile)`, con `FD3`

   `R2(Anno,Progetto,CapoProgetto,Reparto)`, con `FD1` e `FD2`

   Tale decomposizione è lossless e preserva le dipendenze. Lo schema `R2` non è in 3nf a causa della `FD2`, quindi viene decomposto in:

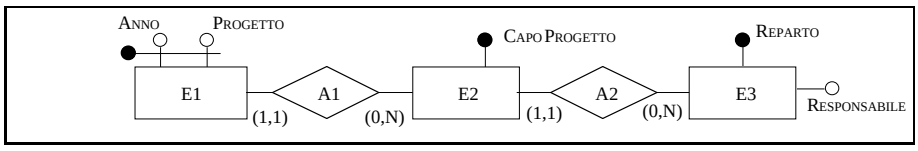
   `R21(CapoProgetto,Reparto)`, con `FD2`

   `R22(Anno,Progetto,CapoProgetto)`, con `FD1`

   In definitiva si ottiene la decomposizione che è lossless e preserva le dipendenze:

   `R1(Reparto,Responsabile)`, con `FD3`
   `R21(CapoProgetto,Reparto)`, con `FD2`
   `R22(Anno,Progetto,CapoProgetto)`, con `FD1`

   Tutti gli schemi di relazione sono ora in BCNF.
4. Uno schema E/R equivalente allo schema di relazione e che soddisfa le dipendenze funzionali date è il seguente:



Esercizio 9

1. `Impiegato(CF,Città)`
   `Progetto(CP,Nome)`
   `Realizza(CP,CFCapo,CFMembro)`
   AK: `CFCapo`
   FK: `CFCapo` **REFERENCES** `Impiegato`
   FK: `CFMembro` **REFERENCES** `Impiegato`
   `Avanzamento(CP,Data,Stato)`
   FK: `CP` **REFERENCES** `Realizza`
2. È sufficiente modificare la chiave della relazione `Avanzamento`:

   `Avanzamento(CP,Data,Stato)`
3. La soluzione è riferita allo schema E/R di figura, senza prendere in considerazione il vincolo introdotto nel punto 2):

   `R(CP,Nome,CFCapo,CittàCapo,CFMembro,CittàMembro,Data,Stato)`

   (a) Le dipendenze funzionali sono le seguenti:
   - `(FD1)` `CFCapo` $\rightarrow$ `CP`
   - `(FD2)` `CP` $\rightarrow$ `CFCapo`
   - `(FD3)` `CFCapo` $\rightarrow$ `CittàCapo`
   - `(FD4)` `CFMembro` $\rightarrow$ `CittàMembro`
   - `(FD5)` `CP` $\rightarrow$ `Nome`
   - `(FD6)` `CFCapo` $\rightarrow$ `CFMembro`
   - `(FD7)` `CP` $\rightarrow$ `CFMembro`

   (b) Le chiavi dello schema di relazione `R` sono K_1 = `CFCapo Data Stato` e K_2= `CP Data Stato`;

   (c) Chiaramente lo schema non è in 2NF dato che gli attributi non primi `Nome`, `CittàCapo`, `CFMembro` e `CittàMembro` dipendono parzialmente dalle chiavi. Si considera quindi:

   `R1(CP,Data,Stato,CFCapo)` **AK:** `CFCapo,Data,Stato`
   `R2(CP,Nome,CFMembro)`
   `R3(CFCapo,CittàCapo)`
   `R4(CFMembro,CittàMembro)`

   Tutte questi schemi di relazione sono in 3NF, però `R1` non è in BCNF a causa delle dipendenze `(FD1)` e `(FD2)`.

   `R11(CP,Data,Stato)`
   `R12(CP,CFCapo)` **AK:** `CFCAPO`

   Questa decomposizione, oltre ad essere lossless, preserva anche le dipendenze. Lo schema relazionale risultante è in BCNF.

6.4 SQL Avanzato e Trigger

◊ **Esercizio 1**

Dato il seguente schema relazionale:
AVVOCATO(CODA,NOME,PROVINCIA)
CAUSA(CODC,DESCRIZIONE,VERDETTO,CODA)
FK: CODA **REFERENCES** AVVOCATO
FASCICOLO(CODF,CODC)
FK: CODC **REFERENCES** CAUSA

Il verdetto finale di una causa assume i valori "assolto", "colpevole", "in corso".

Scrivere in SQL le seguenti interrogazioni

1) Selezionare, per ciascuna provincia, l'avvocato che ha vinto il maggior numero di cause (con verdetto = "assolto");
2) Selezionare, per ciascuna provincia, il numero medio di fascicoli associato a ciascuna causa.

Scrivere in embedded SQL la seguente interrogazione:

3) Mostrare, per ciascun avvocato, la percentuale di cause vinte, quelle perse e quelle non concluse.

Soluzione

1) Selezionare, per ciascuna provincia, l'avvocato che ha vinto il maggior numero di cause (con verdetto = "assolto");

```
SELECT       A.PROVINCIA, A.CODA
FROM         AVVOCATO A, CAUSA C
WHERE        A.CODA = C.CODA
AND          C.VERDETTO = 'assolto'
GROUP BY     A.PROVINCIA, A.CODA
HAVING COUNT(*)>= ALL    (
                   SELECT COUNT(*)
                   FROM  AVVOCATO A1, CAUSA C1
                   WHERE A1.CODA = C1.CODA
                   AND   C1.VERDETTO = 'assolto'
                   AND   A1.PROVINCIA = A.PROVINCIA
                   GROUP BY A1.CODA
                   )
```

2) Selezionare, per ciascuna provincia, il numero medio di fascicoli associato a ciascuna causa.

```
CREATE VIEW V1 AS
SELECT      A.PROVINCIA, C.CODC, COUNT(*) AS NFASCICOLI
FROM        AVVOCATO A, CAUSA C, FASCICOLO F
WHERE       A.CODA = C.CODA
AND         C.CODC = F.CODC
GROUP BY    A.PROVINCIA, C.CODC

SELECT      PROVINCIA, AVG(NFASCICOLI)
FROM        V1
GROUP BY    PROVINCIA
```

3) Mostrare, per ciascun avvocato, la percentuale di cause vinte, quelle perse e quelle non concluse.

```
Q1:   SELECT      A.CODA, COUNT(*) AS N_CAUSE
      FROM        AVVOCATO A, CAUSA C
      WHERE       A.CODA = C.CODA
      GROUP BY    A.CODA

Q2:   SELECT      A.CODA, COUNT(*) AS N_SEL
      FROM        AVVOCATO A, CAUSA C
      WHERE       A.CODA = C.CODA
      AND         C.VERDETTO = :VERDETTO
      GROUP BY    A.CODA


Declare Cursor "C1" For  Q1
Declare Cursor "C2" For  Q2

open C1;
fetch C1 into :COD_A, :N_CAUSE;
while (SQLCODE == 0){
      strcpy(VERDETTO, 'assolto');
      open C2;
      fetch C2 into :COD_A, :N_VINTE;
      if (SQLCODE != 0) N_VINTE = 0;
      close C2;
      strcpy(VERDETTO, 'colpevole');
      open C2;
      fetch C2 into :COD_A, :N_PERSE;
```

```
      if (SQLCODE != 0) N_PERSE = 0;
      close C2;
      printf("COD_A %s, vinte %f, perse %f, in corso %f \n",
                  COD_A, N_VINTE/ N_CAUSE * 100 ,
                  N_PERSE/ N_CAUSE * 100,
                  (1 - (N_VINTE + N_PERSE)/ N_CAUSE) * 100 );
      fetch C1 into :COD_A, :N_CAUSE; }
close C1;
```

◊ **Esercizio 2**

VERIFICA DIPENDENZE FUNZIONALI ATTRAVERSO TRIGGER

Supponiamo di avere il seguente schema di relazione:

```
R(A,B,C,D)
```

e la dipendenza funzionale

```
B -> C
```

Definire il trigger che verifichi il rispetto della dipendenza funzionale.

1. **Inserimento/modifica di singole tuple**

In questo caso, semplificato rispetto al caso generale, il trigger di verifica è il seguente:

```
CREATE TRIGGER DipendenzaBdeterminaC ON [dbo].[R]
FOR INSERT, UPDATE
AS
IF UPDATE(B) OR UPDATE (C)
BEGIN
      declare @cont as int
      SELECT @cont = COUNT(DISTINCT R.C)
           FROM R, Inserted I
            WHERE R.B = I.B
      IF (@cont) > 1
         BEGIN
            raiserror('Violazione della dipendenza funzionale
                  B determina C. Numero di elementi C
                  distinti: %d',16,1, @cont)
            rollback transaction
         END
END
```

Ad esempio, partendo dal seguente stato per la relazione R:

A	B	C	D
A2	B1	C2	D2
A4	B3	C1	D3
A1	B1	C2	D3

Supponiamo di applicare il seguente comando SQL:

```
INSERT INTO R VALUES('A3','B1','C2','D3')
```

Viene inserita correttamente, mentre il seguente commando:

```
INSERT INTO R VALUES('A3','B1','C1','D3')
```

viene rifiutato dal DBMS che invia il seguente messaggio:

```
Server: messaggio 50000, livello 16, stato 1, procedura
DipendenzaBdeterminaC, riga 16
Violazione della dipendenza funzionale B determina C. Numero
di elementi C distinti: 2
```

Analogamente il comando:

```
UPDATE R SET C='C3' WHERE A='A4'
```

viene eseguito correttamente, portando il DB nel seguente stato:

A	B	C	D
A2	B1	C2	D2
A4	B3	C3	D3
A1	B1	C2	D3
A3	B1	C2	D3

2. Inserimento/modifica simultanea di un insieme di tuple

In questo caso generale il trigger proposto per modifiche di singole tuple non è corretto.

Supponiamo, partendo dallo stato raggiunto di inserire il seguente comando:

```
UPDATE R SET C='C' + SUBSTRING(B,2,1)
```

che dovrebbe portare lo schema nel seguente stato corretto:

A	B	C	D
A2	B1	C1	D2
A3	B1	C1	D3
A1	B3	C3	D3
A4	B1	C1	D3

In realtà il comando viene rifiutato con il seguente messaggio:

```
Server: messaggio 50000, livello 16, stato 1, procedura
DipendenzaBdeterminaC, riga 16
Violazione della dipendenza funzionale B determina C. Numero
di elementi C distinti: 2
```

L'errore deriva dal conteggio globale dei valori distinti di C indipendentemente dai valori distinti di B. Infatti, in questo caso si hanno 2 valori distinti di C ('C1' e 'C3') correlati a 2 valori distinti di B ('B1' e 'B3'), mentre l'interrogazione conta in modo indifferenziato i valori di C.

Una prima soluzione è quella di aggiungere alla query di selezione della variabile @cont un raggruppamento sull'attributo B:

```
CREATE TRIGGER DipendenzaBdeterminaC ON [dbo].[R]
FOR INSERT, UPDATE
AS

IF UPDATE(B) OR UPDATE (C)
BEGIN
      declare @cont as int
      SELECT @cont = COUNT(DISTINCT R.C)
           FROM R, Inserted I
            WHERE R.B = I.B
           GROUP BY R.B
      IF (@cont) > 1
         BEGIN
            raiserror('Violazione della dipendenza funzionale
            B determina C. Numero di elementi C distinti:
            %d',16,1, @cont)
            rollback transaction
         END
         ELSE
        PRINT 'Numero di elementi C distinti:' +  STR(@cont)
END
```

Utilizzando questo nuovo trigger e riapplicando il comando:

```
UPDATE R SET C='C' + SUBSTRING(B,2,1)
```

si ottiene la modifica di stato attesa ed il seguente messaggio:

```
Numero di elementi C distinti:          1
(righe interessate: 4)
```

Il messaggio indica che il controllo sul valore di @cont è eseguito una sola volta, anche se gli insiemi raggruppati su valori distinti di B sono due: questo significa che il controllo non è efficace in tutti i casi possibili poiché il valore @cont verificato è uno solo rispetto a tutti quelli che si possono presentare.

Infatti, inserendo il seguente comando:

```
UPDATE R SET C='C3'
WHERE D='D3'
```

che dovrebbe essere rifiutato, perchè porterebbe lo schema nel seguente stato non ammissibile:

A	B	C	D
A2	B1	C1	D2
A3	B1	C3	D3
A1	B3	C3	D3
A4	B1	C3	D3

viene eseguito dal sistema visualizzando il seguente messaggio:

```
Numero di elementi C distinti:          1
(righe interessate: 3)
```

L'errore deriva appunto dal fatto che il numero di elementi di C distinti è 1 per la tupla con B='B3', mentre è 2 per le tuple con B='B1'.
Il fatto che la variabile @cont viene valutata una sola volta porta all'errore.

Per poter contemplare tutti i casi occorre poter verificare tutti i valori associati alla variabile @cont.
Per fare questo è necessario utilizzare un cursore che analizzi sequenzialmente tutti i valori di @cont.

Il trigger corretto è il seguente:

```
CREATE TRIGGER DipendenzaBdeterminaC ON [dbo].[R]
FOR INSERT, UPDATE
AS

IF UPDATE(B) OR UPDATE (C)
BEGIN
      declare @cont as int

        --- Dichiaro il cursore
            DECLARE R_cursor CURSOR FOR
            SELECT  COUNT(DISTINCT R.C)
            FROM R, Inserted I
            WHERE R.B = I.B
            GROUP BY R.B

         --- Apro e carico il cursore
            OPEN R_cursor
            FETCH NEXT FROM R_cursor INTO @cont

            WHILE @@FETCH_STATUS = 0
            BEGIN
                  IF (@cont) > 1
                  BEGIN
                  raiserror('Violazione della dipendenza
                  funzionale B determina C. Numero di
                  elementi C distinti: %d)',16,1, @cont)
                  rollback transaction
            END
                  ELSE
                  PRINT 'Numero di elementi C distinti:' +
                        STR(@cont)
            FETCH NEXT FROM R_cursor INTO @cont
            END
END
```

Riapplicando il comando:

```
UPDATE R SET C='C3'
WHERE D='D3'
```

Esso viene rifiutato mostrando il seguente messaggio di errore:

```
Server: messaggio 50000, livello 16, stato 1, procedura
DipendenzaBdeterminaC, riga 27
Violazione della dipendenza funzionale B determina C. Numero
di elementi C distinti: 2)
Numero di elementi C distinti:         1
```

Da cui si evince che sono stati valutati, correttamente, due insiemi di valori di C associati a due distinti valori di B.

◊ Esercizio 3

Si completi lo schema database con le primary e foreign key (ogni tabella deve avere la primary key), e si scrivano i trigger che preservano le dipendenze funzionali elencate nel seguito:

Optional	
CodA	CodM
001	1
002	2
002	4
005	4
004	1

Moto	
CodM	Nome
1	‘Aaaaaa’
2	‘Bbbbb’
3	‘Ccccc’
4	‘Dddd’

Accessorio		
CodA	Nome	Tipo
001	‘Aaaaaaaaa’	Tipo1
002	‘Bbbbbb’	Tipo1
003	‘Ccccc’	Tipo2
004	‘Dddddd’	Tipo1
005	‘Eeeeee’	Tipo3

Dipendenze funzionali:

1. Il codice della Moto (*CodM*) determina il nome (*Nome*).
2. Il codice accessorio (*CodA*) determina il nome (*Nome*) e il tipo (*Tipo*).
3. Optional definisce gli accessori di una moto.
4. Una moto può contenere al più due accessori dello stesso tipo.

Soluzione

Dipendenza n. 1: chiave primaria sulla tabella Moto su *CodM*.
Dipendenza n. 2: chiave primaria sulla tabella Accessorio su *CodA*.
Dipendenza n. 3: Foreign Key su Optional:

FK:	CodA	REFERENCES	Accessorio
FK:	CodM	REFERENCES	Moto

Dipendenza n. 4: Definire la chiave primaria sulla tabella Optional sulla coppia *CodA,CodM*.
Dipendenza n. 5: Definire il seguente trigger:

```
CREATE TRIGGER Controllo_Numero_Accessori
ON Optional
FOR INSERT, UPDATE
AS
--- Dichiaro il contatore
Declare @cont int
Select @cont = (count(*)), @cont2 = Max(@cont)
from  Optional, inserted
where Optional.CodM = inserted.CodM
And   Optional.CodA IN (SELECT CodA
                        FROM   Accessori
                        WHWERE Tipo = inserted.Tipo)
GROUP BY Optional.CodM

if @cont > 2
begin
      raiserror('Una moto puo' contenere al piu' due accessori
                  dello stesso tipo.',16,1)
      rollback transaction
end
```

◊ Esercizio 4

Dato il seguente schema relazionale:

```
MEDICO(CODM, Nome, NumAssistiti)
PAZIENTE(CF, CODM)
FK: CODM REFERENCES MEDICO
```

Scrivere il Trigger (secondo la sintassi IBM DB2, MS SQLServer o ORACLE) che preservi il dato derivato NumAssistiti a fronte di inserimenti, aggiornamenti o cancellazioni nella relazione PAZIENTE.

Soluzione

```
CREATE TRIGGER Aggiorna_Num_Pazienti ON [dbo].[Paziente]
FOR INSERT, UPDATE, DELETE
AS

declare @codm as varchar(5),  @cont as int

DECLARE patient_inserted CURSOR FOR
SELECT  CodM, count(*)
FROM    Inserted
GROUP BY CodM

   OPEN patient_inserted
   FETCH NEXT FROM patient_inserted INTO @codm, @cont

   WHILE @@FETCH_STATUS = 0
   BEGIN
      UPDATE Medico SET NumAssistiti = NumAssistiti + @cont
      WHERE CodM = @codm
      FETCH NEXT FROM patient_inserted INTO @codm, @cont
   END


CLOSE patient_inserted
DEALLOCATE patient_inserted

DECLARE patient_deleted CURSOR FOR
SELECT CodM, count(*)
FROM Deleted
GROUP BY CodM

   OPEN patient_deleted
   FETCH NEXT FROM patient_deleted INTO @codm, @cont

   WHILE @@FETCH_STATUS = 0
   BEGIN
      UPDATE Medico SET NumAssistiti = NumAssistiti - @cont
      WHERE  CodM = @codm
      FETCH NEXT FROM patient_deleted INTO @codm, @cont
   END
```

```
CLOSE patient_deleted
DEALLOCATE patient_deleted
```

Soluzione alternativa (piu' onerosa in termini di risorse di esecuzione)

```
CREATE TRIGGER Aggiorna_Num_Pazienti ON [dbo].[Paziente]
FOR INSERT, UPDATE, DELETE
AS

UPDATE    Medico
SET       NumAssistiti =
          (SELECT  COUNT(CF)
           FROM    Paziente AS P
           WHERE   P.CodM = Medico.CodM)
```

Bibliografia

[1] A. ALBANO, "Basi di dati: strutture ed algoritmi", Addison-Wesley Masson, Milano, 1992.

[2] A. ALBANO, "Costruire Sistemi per Basi di Dati", Addison Wesley Longman, Milano, 2001

[3] A. ALBANO, G. GHELLI, R. ORSINI, "Basi di dati relazionali e a oggetti", Zanichelli, 2005.

[4] W. W. ARMSTRONG, "Dependency structures on data base relationships", Atti del Congresso IFIP, 1974 North Holland, Armsterdam.

[5] P. ATZENI, C. BATINI, V. DE ANTONELLIS, "La Teoria Relazionale dei Dati", Bollati Boringhieri, 1985

[6] P. ATZENI, S. CERI, S. PARABOSCHI, R. TORLONE, "Basi di dati: modelli e linguaggi di interrogazione", Seconda edizione, McGraw-Hill Italia, 2006

[7] P. ATZENI, S. CERI, P. FRATERNALI, S. PARABOSCHI, R. TORLONE, "Basi di dati: architetture e linee di evoluzione", Seconda edizione, McGraw-Hill Italia, 2007

[8] P.ATZENI, V.DE ANTONELLIS, "Relational Database Theory", Benjamin-Cummings, 1993

[9] C. BATINI, G. DEPETRA, M. LENZERINI, G. SANTUCCI, "La progettazione concettuale dei dati, VII Edizione" Franco Angeli, 2002.

[10] C. BATINI, S. CERI, S. NAVATHE, "Conceptual Database Design- an Entity-Relationship approach", Benjiamin Cummings, 1992.

[11] S. CANNAN, G. OTTEN, "Il manuale SQL", McGraw Hill, 1994.

[12] S.CERI, "Methodology and Tools for Data Base Design ", Elsevier, 1983

[13] D.D.CHAMBERLIN, "A Complete Guide to DB2 Universal Database", Morgan Kaufmann, 1998

[14] P. CIACCIA, D. MAIO, "Lezioni di Basi di Dati", Società Editrice Esculapio, Bologna, 2002.

[15] C.J. DATE, "An Introduction to Database Systems", Eighth Edition, Addison-Wesley, 2003.

[16] C.J. DATE, "Relational Database: selected writings", Addison Wesley, 1986.

[17] C.J.DATE, "The Database Relational Model, a Retrospective Review and Analysis", Addison Wesley, 2000

[18] C.J. DATE, H. DARWEN, "A guide to the SQL standard", Fourth Edition, Addison Wesley, 1996.

[19] C.J. DATE, MC GOVERAN, "A guide to SYBASE and SQL-SERVER", Addison Wesley, 1992.

[20] C.J. DATE, C.J. WHITE, "A guide to DB2", Reading, Mass.: Addison-Wesley, 1993.

[21] R.ELMASRI, S.B.NAVATHE, "Fundamentals of Database Systems", Fifth Edition, Addison Wesley, 2006

[22] C.C.FLEMING, B.VON HALLE, , "Handbook of Relational Database Design", Addison-Wesley Professional, 1989

[23] G.GARDARIN, P. VALDURIEZ, "Relational Databases and Knowledge Bases", Addison Wesley, 1992.

[24] J. GRAY, A. REUTER, "Transaction Processing: concepts and techiques", Morgan Kauffman Publishers, San Francisco, California, 1993.

[25] J.M.HELLERSTEIN, M.STONEBRAKER, "Readings in Database Systems", Fourth Edition, The MIT Press, 2005

[26] IBM's DB2 Version 9 for Linux, Unix and Windows, SQL Reference Volume 1, http://www.ibm.com/support

[27] IBM's DB2 Version 9 for Linux, Unix and Windows, SQL Reference Volume 2, http://www.ibm.com/support

[28] H.MANNILA,K.J.RAIHA "The Design of Relational Databases ", Addison-Wesley, 1992

[29] P.O'NEIL, E.O'NEIL, "Database: Principles, Programming, and Performance", Second Edition, Morgan Kaufmann, 2000

[30] J.PAREDAENS, P. DE BRA, M.GYSSENS, D.VAN GUCHT, "The Structure of the Relational Database Model", Springer, 1989

[31] R.RAGHU, G.JOHANNES, "Sistemi di Basi di Dati (a cura di Tiziana Catarci)", McGraw-Hill, 2004

[32] R.RAMAKRISHNAN,G.GEHRKE, "Database Management Systems", Third Edition, McGraw-Hill, 2002

[33] A.SILBERSCHATZ, H.F.KORTH, S.SUDARSHAN, "Database Systems Concepts", Fifth Edition, McGraw-Hill, 2005

[34] G. STURIALE, G.M. VERARDI, "Oracle", Jackson, 1989

[35] T.J.TEOREY, "Database Modeling & Design", Third Edition,Morgan Kaufmann, 1995

[36] J. D. ULLMAN, "Basi di Dati e Basi di Conoscenza", Jackson, 1991.

[37] J. D. ULLMAN, "Principles of Database and Knowledge-Base Systems, Vol. 1", Jackson, Computer Science Press, 1988.

[38] J.D.ULLMAN, J.D.WIDOM, "A First Course in Database Systems", Second Edition, Prentice Hall, 2001

[39] R.F. VAN DER LANS, "Introduzione a SQL", Pearson Education Italia, 2001.

[40] G.WIEDERHOLD, "Database Design", Second Edition, Mcgraw-Hill, 1983

Appendice A

Lucidi Date

In questa appendice vengono riportati i primi tre lucidi del seminario "An Introduction to Relational Data Base" tenuto nel 1976 da C. J. Date ai Laboratori IBM di San Jose. Due anni dopo a me li dette M. W. Blasgen perché avevo bisogno di usarli per un seminario. I lucidi rappresentano in modo sintetico i primissimi concetti espressi da E. F. Codd nel suo famoso lavoro "A Relational Model of Data for Large Shared Data Banks" del 1970.

Questi concetti sono stati alla base di tantissimo lavoro di ricerca sulle basi di dati al quale negli anni hanno partecipato con successo anche numerosi ricercatori italiani. Successivamente al lavoro di ricerca sono seguite importanti realizzazioni industriali di successo che oggi sono leader di mercato.

Mi fa piacere che Sonia e Domenico me li abbiano chiesti per questa terza edizione. Il loro libro sinteticamente, ma con efficacia e rigore, descrive l'evoluzione ed il funzionamento dei database relazionali e costituisce un ottimo testo per introdurre gli studenti a questo complesso e così importante settore dell'informatica.

Prof. Paolo Tiberio

Preside della Facoltà di Ingegneria di Modena dellUniversità di Modena e Reggio Emilia

AN INTRODUCTION TO

RELATIONAL DATABASE

C. J. DATE

IBM General Products Division

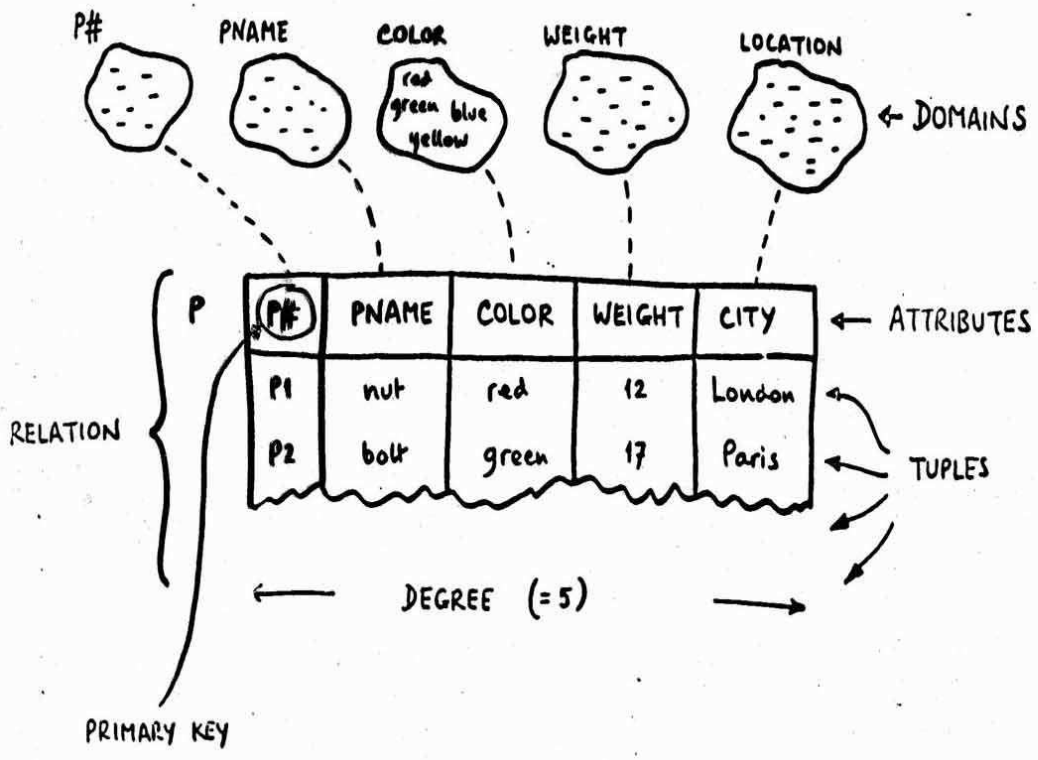
Reference : DATAMATION, April 1976.

DEFINITION: RELATION

GIVEN SETS $D_1, D_2, \ldots, D_n$ (NOT NECESSARILY DISTINCT), R IS A RELATION ON THESE DOMAINS IF IT IS A SET OF ORDERED n-TUPLES $\langle d_1, d_2, \ldots, d_n \rangle$ SUCH THAT d_1 BELONGS TO D_1, d_2 BELONGS TO D_2, ..., d_n BELONGS TO D_n.

THE VALUE n IS THE DEGREE OF R.

TERMINOLOGY



PROPERTIES OF RELATIONS

WITHIN ANY ONE RELATION :

1. NO DUPLICATE ROWS
2. ROW ORDER INSIGNIFICANT
3. COLUMN ORDER INSIGNIFICANT (usually)
4. ALL VALUES ATOMIC