

Basi di Dati

UniShare

Davide Cozzi
@dlcgold

Gabriele De Rosa
@derogab

Federica Di Lauro
@f_dila

Indice

1	Introduzione	2
2	Introduzione al Corso	3
3	Progettazione Concettuale:ER	11
4	Modello Relazionale	27
4.1	Vincoli d'integrità	29
4.1.1	In pratica	34
4.1.2	Relazione Matematica	35
4.1.3	Relazioni nel Modello Relazionale	36
5	SQL	43
5.0.1	Interrogazioni in SQL	54
5.1	Progettazione Concettuale, approfondimento	57
6	Progettazione Logica	60
6.1	Analisi dell'E-R	62
6.1.1	Ristrutturazione dell'E-R	65
6.1.2	Elaborazione dei Concetti	70
6.1.3	Scelta degli Identificatori Principali	74
6.2	Dall'E-R al Modello relazionale	75
6.2.1	Molti a Molti	75
6.2.2	Uno a Molti	76
6.2.3	Entità Esterna	76
6.2.4	Uno a Uno	77
6.2.5	Tabella Riassuntiva	78
7	Algebra Relazionale	80

Capitolo 1

Introduzione

Questi appunti sono presi a ldurante le esercitazioni in laboratorio. Per quanto sia stata fatta una revisione è altamente probabile (praticamente certo) che possano contenere errori, sia di stampa che di vero e proprio contenuto. Per eventuali proposte di correzione effettuare una pull request. Link: <https://github.com/dlcgold/Appunti>.

Grazie mille e buono studio!

Capitolo 2

Introduzione al Corso

Il corso di Basi di Dati affronta gli aspetti e i metodi per lo sviluppo di un database in maniera efficiente, aspetto fondamentale per un informatico e per lo sviluppo ottimale di software.

Il corso si divide in 7 parti:

1. introduzione generale
2. metodologie e modelli per il progetto delle basi di dati
3. progettazione concettuale
4. modello razionale
5. progettazione logica
6. linguaggio SQL
7. algebra relazionale

Le **informazioni** fanno parte delle risorse di un'azienda, soprattutto negli ultimi anni di clima globale, per cui la gestione efficiente ed ottimale dei dati è fondamentale, come ad esempio Facebook ed Amazon fanno ampio uso dei nostri dati, sia a scopi pubblicitari sia a scopi di marketing.

Un sistema informativo è una componente di un'organizzazione che gestisce le informazioni d'interesse, non per forza attraverso un'automatizzazione e/o supporto di un calcolatore, infatti sin dall'antichità le banche tenevano traccia dei depositi tramite un archivio cartaceo.

Una porzione automatizzata del sistema informativo si chiama sistema informatico, che si divide in:

- acquisizione e memorizzazione

- aggiornamento
- interrogazione
- elaborazione

Le informazioni vengono gestite in vari modi, attraverso il linguaggio naturale, graficamente con schemi e/o numeri, con il tempo si è arrivati a codifiche standard per quasi tutte le tipologie di informazioni, e sono rappresentate nei sistemi informatici dai *dati*, la cui differenza è che i dati sono valori senza alcun valore mentre le informazioni stabilisce un'interpretazione attribuendo una semantica ai valori.

I dati sono una risorsa strategica in quanto sono stabili nel tempo, infatti solitamente i dati sono immutati durante una migrazione tra un sistema e un altro, per questo lo sviluppo progettazione di un database rimane stabile in teoria, senza notevoli cambiamenti durante la durata di un sistema informativo.

Un **Data Base** è una collezione di dati usati per rappresentare le informazioni di interesse di un sistema informativo, definite solo una volta a cui un insieme di applicazioni ed utenti può accedere ad essi, mentre un **DBMS** è un software per la gestione di un database.

I dati presenti in un database sono molti, indipendenti dal programma in cui vengono utilizzati e cui si cerca di evitare la ridondanza, per garantire la consistenza delle informazioni.

Per la creazione di un database, al fine di garantire privacy, affidabilità, efficienza ed efficacia, sono presenti le seguenti tre fasi:

1. definizione
2. creazione e popolazione
3. manipolazione

Un'organizzazione è divisa in vari settori e ogni settore ha un suo sottosistema informativo, non necessariamente disgiunto, e i database solitamente sono condivisi, al fine di ridurre la ridondanza delle informazioni, per cui sono presenti meccanismi di autorizzazione e di controllo della concorrenza.

Un database deve essere conservato a lungo termine e si ha una gestione delle **transazioni**: insieme di operazioni da considerare indivisibile, atomico, corretto anche in presenza di concorrenza e con effetti definitivi.

La sequenza di operazioni nel database deve essere eseguita nella sua interezza, per cui l'effetto di transazioni concorrenti deve essere coerente, infatti la conclusione positiva di una transazione corrisponde ad un impegno,

commit, a mantenere traccia del risultato, anche in presenza di guasti e di esecuzioni concorrente.

Come tutti i software, i DBMS devono essere efficienti, utilizzando al meglio memoria e tempo, efficaci e produttivi.

Si hanno delle caratteristiche nell'approccio alla base dati:

- natura autodescrittiva di un sistema di basi di dati: il sistema di basi di dati memorizza i dati con anche una descrizione completa della sua struttura (metadati), per consentire ai DBMS di lavorare con qualsiasi applicazione.
- separazione tra programmi e dati, infatti è possibile cambiare la struttura dati senza cambiare i programmi.
- astrazione dei dati: si usa un modello dati per nascondere dettagli e presentare all'utente una visione concettuale del database.
- supporto di viste multiple dei dati, per cui ogni utente può usare una vista (view) differente del database, contenente solo i dati di interesse per quell'utente.
- condivisione dei dati e gestione delle transazioni con utenti multipli.

I DBMS estendono le funzionalità dei file system, fornendo più servizi ed in maniera integrata.

In ogni base di dati si ha:

- lo **schema**, sostanzialmente invariante nel tempo, che ne descrive la struttura, l'aspetto intensionale.
- l'**istanza**, i valori attuali, che possono cambiare anche molto rapidamente, l'aspetto estensionale "concreto".

Per lo sviluppo delle basi di dati si hanno due tipi di modelli, ambedue importanti:

- **modelli logici**, adottati nei DBMS esistenti per l'organizzazione dei dati, utilizzati dai programmi e sono indipendenti dalle strutture fisiche
- **modelli concettuali** che permettono di rappresentare i dati in modo indipendente da ogni sistema, con il fine di descrivere i concetti del mondo reali; sono usati nelle fasi preliminari di progettazione e il più diffuso è il modello **Entity-Relationship**.

Un database è organizzato solitamente attraverso i tre schemi dell'architettura ANSI/SPARC:

- schema logico: descrizione dell'intera base di dati nel modello logico “principale” del DBMS, ossia si definisce la struttura concettuale del database, senza considerare l'implementazione fisica nel DBMS.
- schema fisico: rappresentazione dello schema logico per mezzo di strutture fisiche di memorizzazione
- schema esterno: descrizione di parte della base di dati in un modello logico (“viste” parziali, derivate, anche in modelli diversi)

L'accesso ai dati avviene solo mediante il livello esterno, il quale a volte coincide con quello logico, e si hanno 2 forme di indipendenza: quella fisica, in cui è possibile interagire con il DBMS senza conoscere la struttura fisica e quella logica, in cui si può accedere al livello esterno senza interagire con lo schema logico.

Per la definizione dei database ci sono quattro tipologie di linguaggi:

- **DLL**(Data Manipulation Languages) linguaggio per definire i dati
- **DML**(Data Manipulation Languages) linguaggio per la manipolazione dei dati
- **DCL**(Data Control Languages) linguaggio per il controllo degli accessi al database
- **DQL**(Data Query Languages) linguaggio per effettuare delle interrogazioni al database

Noi vedremo SQL per definire i database, linguaggio basato sull'algebra relazionale che implementa tutti e 4 le tipologie di linguaggi per la gestione di un database.

Si hanno due tipi di utenti:

1. **utenti finali (terminalisti)**: eseguono applicazioni predefinite (transazioni)
2. **utenti casuali**: eseguono operazioni non previste a priori, usando linguaggi interattivi

inoltre si hanno:

- progettisti e realizzatori di DBMS

- progettisti della base di dati e amministratori della base di dati (DBA), persona o gruppo di persone responsabile del controllo centralizzato del database, in tutti i suoi aspetti.
- progettisti e programmatori di applicazioni



Come visto anche nel corso di analisi e progettazione del software e nella figura X, durante lo sviluppo di un sistema informatico si hanno le seguente fasi:

- **studio di fattibilità**, definizione di costi, priorità e competenze per un progetto
- **raccolta e analisi dei requisiti**, ovvero lo studio delle proprietà del sistema
- **progettazione** di dati e funzioni
- **implementazione**
- **validazione e collaudo**, che comprendono anche test da parte del cliente
- **funzionamento**, ovvero lo stadio finale dove il sistema diventa effettivamente operativo

In questo corso ci occuperemo soltanto delle fasi di progettazione ed implementazione di base di dati, lasciando l'analisi delle altre fasi al corso ed ai libri di Ingegneria del Software.

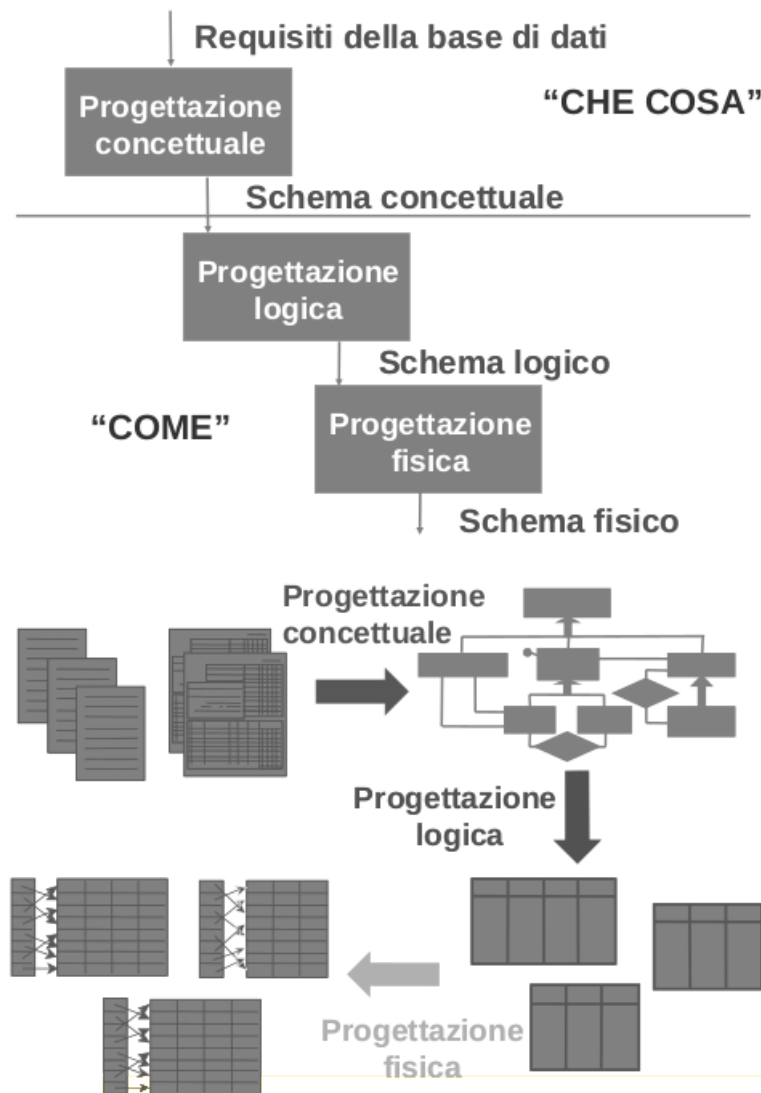
In questo processo a rimanere stabili sono prettamente i **dati** infatti prima si progetta la base dati, con una **metodologia di progetto**, e poi il software.

Ciclo di vita (modello a spirale)



Una **metodologia** è un'articolazione in fasi di guida ad un'attività di progettazione, in caso di una base dati la metodologia, con il fine di separare il cosa rappresentare dal come, è la seguente:

- suddivida la progettazione in fasi indipendenti
- fornisca strategie e criteri di scelta in caso di alternative
- fornisca modelli di riferimento (i linguaggi)
- garantisca generalità rispetto al problema
- garantisca qualità e facilità d'uso



Come si può notare nella figura X1, la metodologia corrente di modellazione di un database prevede la definizione e l'esecuzione delle seguenti fasi:

- la **progettazione concettuale** consiste nel tradurre i requisiti del sistema informatico in una descrizione formale, integrata e indipendente dalle scelte implementative (DBMS, SW e HW)
- la **progettazione concettuale** consiste nella traduzione dello schema concettuale nel modello dei dati scelto per la modellazione, ottenendo uno schema logico, espresso nel DDL del DBMS. In questa fase si considerano anche aspetti legati ai vincoli ed all'efficienza. Si hanno due sotto-fasi:

- ristrutturazione dello schema concettuale
- traduzione verso il modello logico
- la **progettazione fisica** completa lo schema logico ottenuto con le specifiche proprie del DBMS scelto. Il risultato è lo schema fisico che descrive le strutture di memorizzazione ed accesso ai dati

Incominciamo nel prossimo capitolo a considerare la progettazione concettuale, per poi analizzare nei successivi capitoli in dettaglio anche la fase di progettazione concettuale e il linguaggio SQL.

Capitolo 3

Progettazione Concettuale:ER

Come abbiamo già introdotto nel precedente capitolo, affrontiamo ora la progettazione concettuale di una base dati attraverso il modello **ER**(Entity Relationships), modello concettuale che fornisce una serie di strutture atte a descrivere in maniera semplice e facile la realtà di interesse da modellare.

Si hanno dei vantaggi con la progettazione concettuale, prevale infatti l'aspetto intensionale indipendente dalla tecnologia ed è una rappresentazione grafica ed è utile per la documentazione, in quanto facilmente comprensibile anche da persone poco avvezze alla tecnologia e ai database.

In uno schema ER si hanno i seguenti costrutti, come si nota nella figura Y, la cui rappresentazione effettiva varia in quanto vi sono più versioni di ER:

- **entità**: classe di oggetti con proprietà comune ed esistenza "autonoma", della quale si vogliono specificare fatti specifici;ogni entità ha un nome univoco, espressivo e al singolare.

- **relazione**: rappresentano legami logici, significativi per la realtà da modellare, tra due o più entità.

Un'occorrenza di relazione è un n-upla costituita da occorrenze di entità, una per ciascuna delle entità coinvolte, e ogni relazione ha un nome univoco, in cui è preferibile assegnare un sostantivo per evitare di stabilire un verso alla relazione.

Essendo una relazione matematica tra le entità coinvolte non è possibile avere delle n-uple identiche, con conseguenze per la realtà da rappresentare, per esempio non è possibile attraverso una relazione il fatto che è possibile ripetere un esame, obbligando a rappresentare l'esame come entità e non più come relazione.

- **attributo semplice:** associa ad ogni istanza di entità o associazione un valore, definito su un dominio di valori, specificato nella documentazione associata, con il fine di descrivere le proprietà elementari di entità e/o relazioni disegnate per rappresentare la realtà d'interesse.
- **attributo composto:** raggruppamento di attributi di una medesima entità/relazione con affinità di significato e/o uso come ad esempio possiamo raggruppare gli attributi Via, Numero Civico e Cap dall'entità persona per formare l'attributo composto Indirizzo.
- **cardinalità delle relazioni:** vengono specificate per ogni relazione e descrivono il numero minimo e massimo di occorrenze di relazione, a cui una occorrenza dell'entità può partecipare alla relazione, ossia quante volte un'occorrenza di un'entità può essere legata ad occorrenze delle altre entità coinvolte.
È possibile assegnare un qualunque intero non negativo, con l'unico vincolo che la cardinalità minima sia minore o uguale alla cardinalità massima e di solito si usano i valori $0, 1eN$, indicanti zero, una o molte occorrenze, senza preoccuparsi in caso di N del numero effettivo di occorrenze.
- **cardinalità di un attributo:** descrivono il numero minimo e massimo di valori dell'attributo associati all'entità e/o relazione, con la cardinalità $(1, 1)$ stabilita come default, che può essere vista come funzione che associa ad ogni occorrenza di entità un solo valore dell'attributo; si hanno le stesse consuetudini delle cardinalità delle relazioni.
- **identificatore interno:** permette di identificare in maniera univoca un'entità ed un identificatore è interno in caso sia uno o più attributi di un'entità, tutti con cardinalità $(1, 1)$.
- **identificatore esterno:** un identificatore è esterno, in caso un'entità E viene identificata da un'attributo di un'entità F , cui esiste una relazione uno a uno tra l'entità E e F .
È possibile, anche se molto raro, avere la definizione dell'identificatore usando entità di entità, ossia l'identificatore dell'entità E viene definito nell'entità G , cui esiste una relazione con l'entità F

che a sua volta ha una relazione con l'entità E , ma si può capire da quanto è contorto il ragionamento qual'è la sua percentuale d'utilizzo nella modellazione dello schema ER.

- **generalizzazione:** rappresentano legami logici tra un'entità E , detta padre, e una serie di entità E_1, E_2, \dots, E_n , dette figlie, di cui l'entità E rappresenta un caso generale della serie di entità figlie.

Tra le entità coinvolte in una generalizzazione valgono le seguenti proprietà:

- ogni occorrenza di un'entità figlia è anche un'occorrenza dell'entità genitore.
- ogni proprietà dell'entità genitore è anche una proprietà delle entità figlie, quindi nello schema ER non devono essere rappresentate, e ciò prende il nome di ereditarietà.




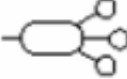




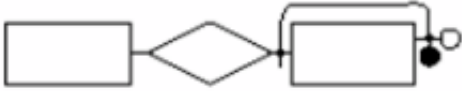


Le generalizzazioni possono essere classificate sulla base di due proprietà ortogonali:

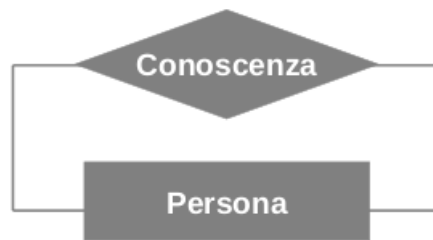
- una generalizzazione è totale se ogni occorrenza del genitore è un'occorrenza di almeno uno dei figli, altrimenti è parziale.
 - una generalizzazione è esclusiva se ogni occorrenza del genitore è al più un'occorrenza di una delle entità figlie, altrimenti è sovrapposta.
- **sottoinsieme:** generalizzazione con soltanto un'entità figlia, di cui solitamente rappresenta una parte dell'entità genitore come ad esempio gli studenti sono un sottoinsieme delle persone.

Un'occorrenza, o istanza, di un'entità, è un oggetto della classe che l'entità rappresenta, ma noi rappresentiamo le entità dello schema concettuale non le singole istanze, in quanto esse sono variabili nel tempo a differenza della struttura, concetto fondamentale per definire un modello funzionale.

Nessuno impone un tipo per un certo attributo, possono essere anche complessi di cui però non ci interessano le informazioni che lo rappresentano, come ad esempio una foto può essere un attributo, ma se ho bisogno dell'autore della foto, non sarà più un attributo ma un'altra entità.

Un'**istanza di associazione** è una combinazione o aggregazione di istanze di entità che prendono parte all'associazione (per esempio "prof. Schettini")

Construct	Graphical representation
Entity	
Relationship	
Simple attribute	
Composite attribute	
Cardinality of a	
Cardinality of an attribute	
Internal identifier	 
External identifier	
Generalization	
Subset	

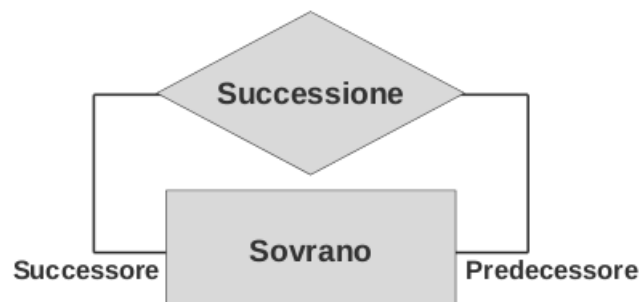


è istanza di associazione per l'entità docente).

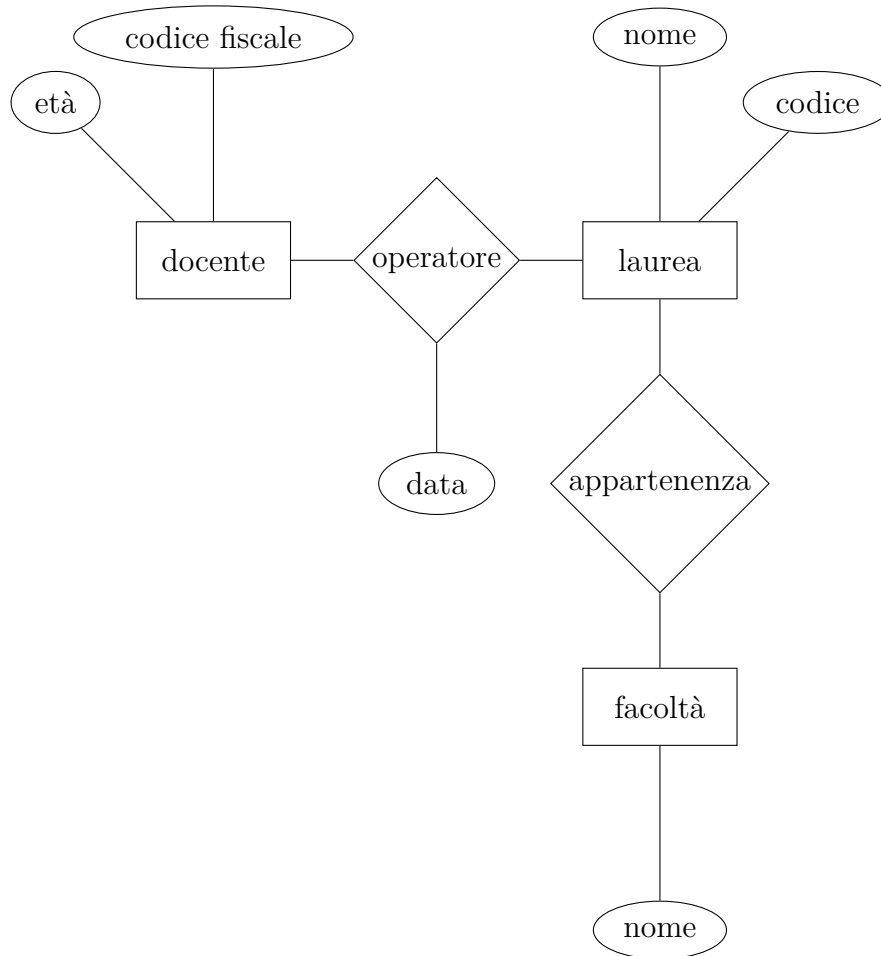
Le relazioni possono avere attributi, con valori specificati in un certo dominio, che modella una proprietà del legame tra tutte le entità rappresentato dalla relazione.

Una associazione può coinvolgere “due o più volte” la stessa entità e ciò si chiama *associazione ricorsiva o ad anello*, come si nota nella figura XXXX, ed è molto utile per definire le relazioni subordinazione e/o comunicazioni tra istanze di una stessa entità, come nel caso della relazione sovrano, indicante la lista dei sovrani con i predecessori e successori; in queste associazioni è necessario aggiungere la specifica dei **ruoli**, come nell'esempio SDERR, per specificare in maniera chiara e non ambigua quale è il verso della relazione.

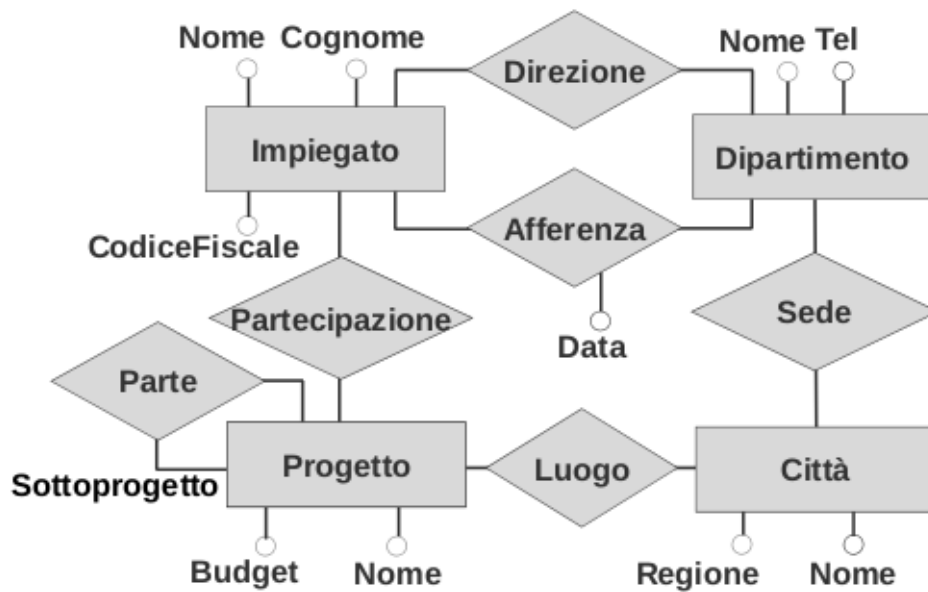
Un'associazione ad anello, ovviamente essendo una relazione, può avere delle proprietà come ad esempio la relazione della figura SJDFHJF è irreflessiva, intransitiva e asimetrica.



Descrivere lo schema concettuale della seguente realtà: I docenti hanno un codice fiscale ed una età. I docenti operano nei corsi di laurea (si dice che afferiscono ai corsi di laurea). Interessa la data di afferenza dei docenti ai corsi di laurea. I corsi di laurea hanno un codice ed un nome, ed appartengono alle facoltà. Ogni facoltà ha un nome



Esempio 1. Descrivere lo schema concettuale della seguente realtà: Degli impiegati interessa il codice fiscale, il nome, il cognome, i dipartimenti ai quali afferiscono (con la data di afferenza), ed i progetti ai quali partecipano. Dei progetti interessa il nome, il budget, e la città in cui hanno luogo le corrispondenti attività. Alcuni progetti sono parti di altri progetti, e sono detti loro sottoprogetti. Dei dipartimenti interessa il nome, il numero di telefono, gli impiegati che li dirigono, e la città dove è localizzata la sede. Delle città interessa il nome e la regione:



Nella scelta del costrutto ideale, per rappresentare in maniera fedele e corretta la realtà, si usano le seguenti "regole":

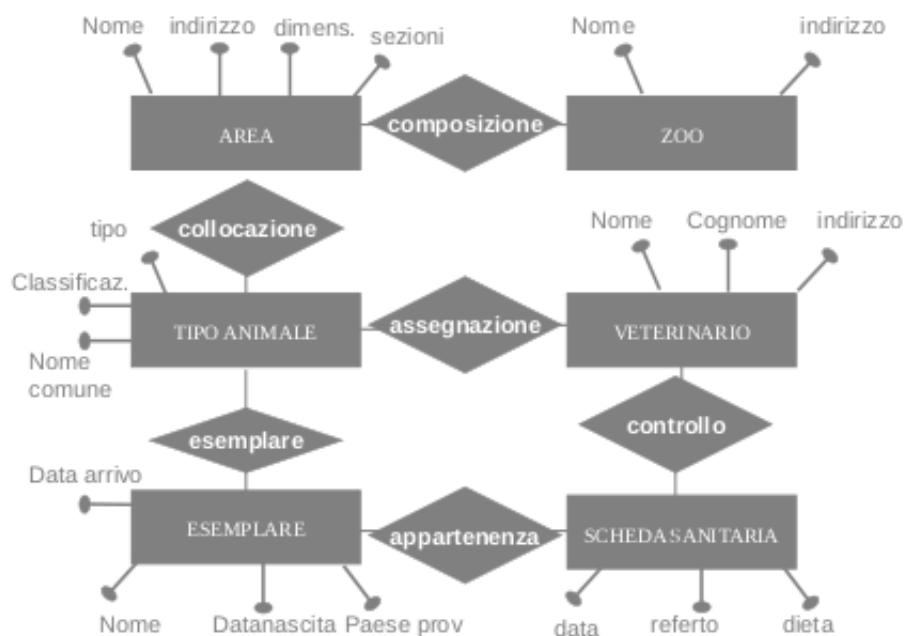
- **entità:** si usa un'entità in caso vengano rispettate le seguenti proprietà: le sue istanze sono significative indipendentemente dalle altre istanze, se si ha o si può avere delle proprietà indipendenti dagli altri concetti oppure se il concetto da rappresentare è importante per la realtà da modellare.
- **attributo:** si usa un'attributo in caso succedono i seguenti avvenimenti: non ha senso considerare una sua istanza a se stante ad altre istanze, le istanze non sono concettualmente significative oppure serve soltanto rappresentare una proprietà locale di un altro concetto.
- **relazione:** si dovrebbe usare una relazione in caso non ha senso pensare alla partecipazione delle sue istanze ad altre relazioni e

le sue istanze non sono significative indipendentemente da altre istanze, ossia una sua istanza ha senso se messa in relazione con altre istanze, provenienti da altri costrutti.

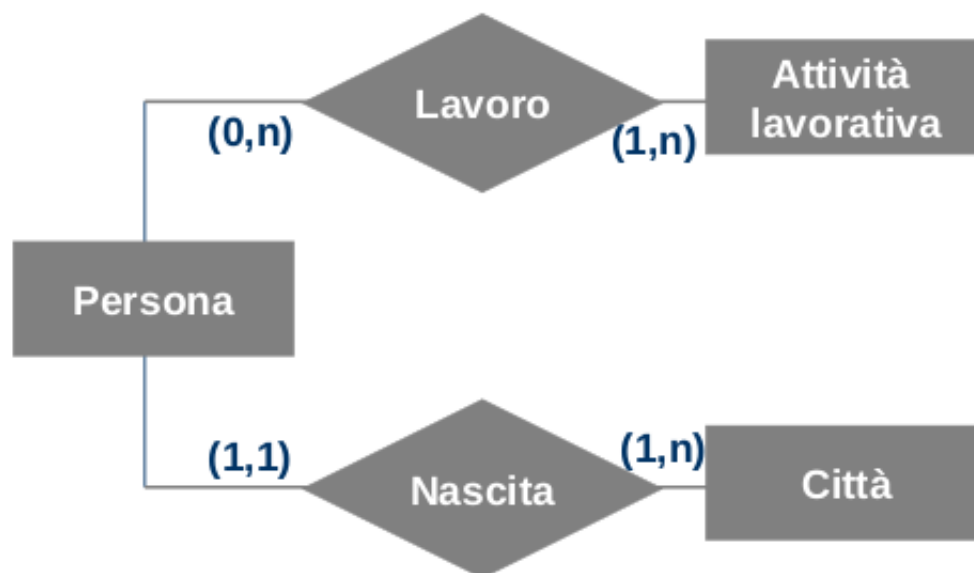
•Calcio: un giocatore gioca in una squadra



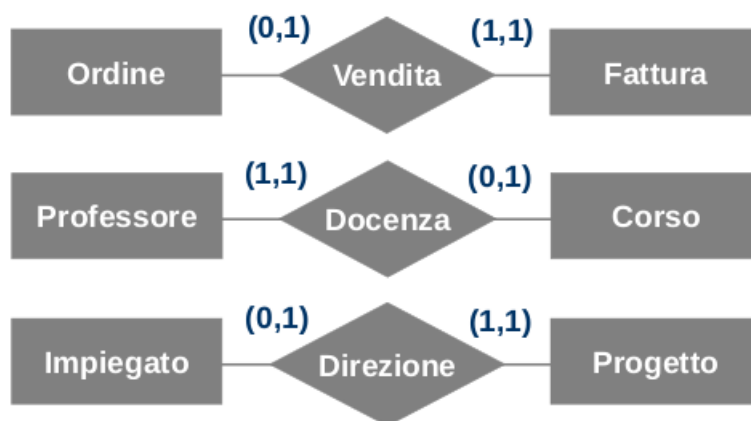
- Esempio 2.**
- ogni zoo è diviso in aree diverse a seconda che si tratti di rettili, pesci, uccelli, scimmie, grandi mammiferi, ... Ogni area è dotata di: nome, indirizzo, dimensione, numero di sezioni.
 - per ogni tipo di animale ci sono informazioni che riguardano: classificazione zoologica, nome comune (giraffa, elefante, serpente, tartaruga, ...), habitat, alimentazione, ... Per ogni tipo di animale c'è un diverso veterinario specialista, dipendente dello zoo.
 - ogni tipo di animale è rappresentato da esemplari e relativi dati anagrafici: nome proprio (giraffa Enrico, giraffa Giulia, ...), data di nascita, Paese di provenienza, data di arrivo allo zoo, ...
 - ogni esemplare è dotato di più schede sanitarie contenenti ognuna: la data della visita, referto, dieta, nome del veterinario, ...



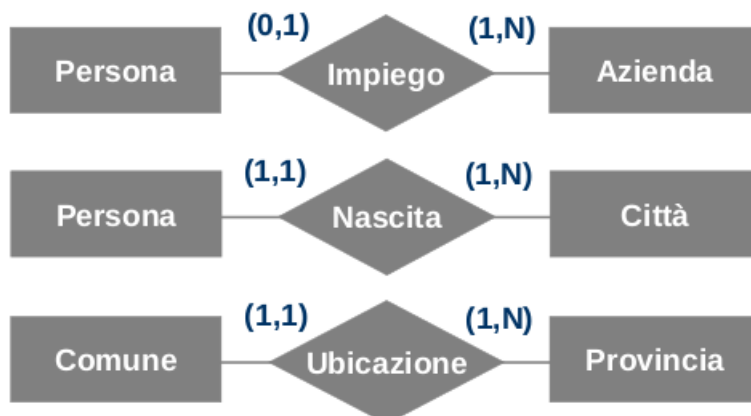
lo zoo ha degli attributi che vanno indicati anche se non richiesti dalla traccia per evitare ambiguità. Dato che ogni scheda è collegata ad un veterinario, di cui ho un'entità la collego a veterinario con una relazione e non ne faccio un attributo



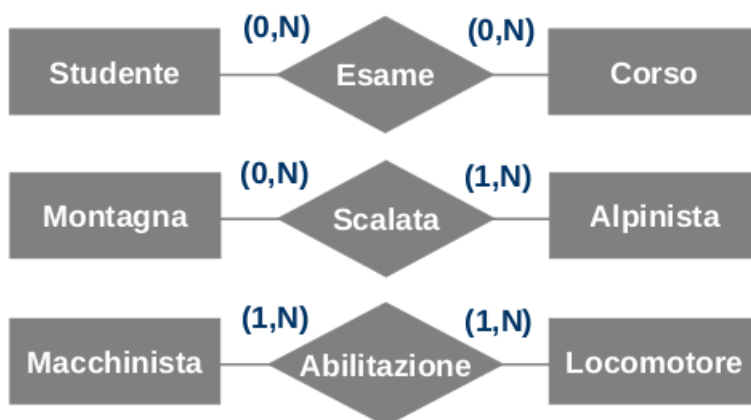
Relazioni “uno a uno”



Relazioni “uno a molti”



Relazioni “molti a molti”



vediamo un esempio:

Esempio 3. *si ha:*



- $\text{min-card}(\text{Automobile}, \text{Proprietario}) = 0$: esistono automobili non possedute da alcuna persona
- $\text{min-card}(\text{Persona}, \text{Proprietario}) = 0$: esistono persone che non posseggono alcuna automobile
- $\text{max-card}(\text{Persona}, \text{Proprietario}) = n$: ogni persona può essere proprietaria di un numero arbitrario di automobili
- $\text{max-card}(\text{Automobile}, \text{Proprietario}) = 1$: ogni automobile può avere al più un proprietario



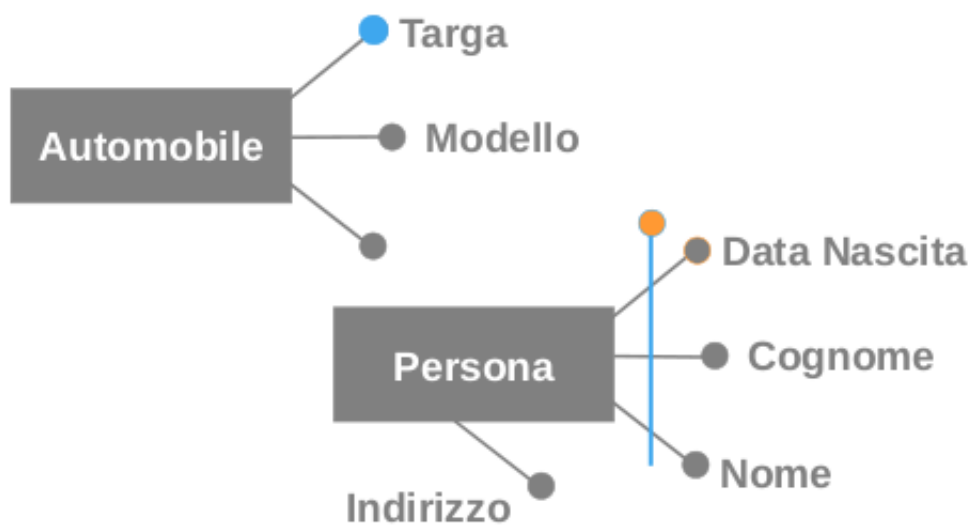
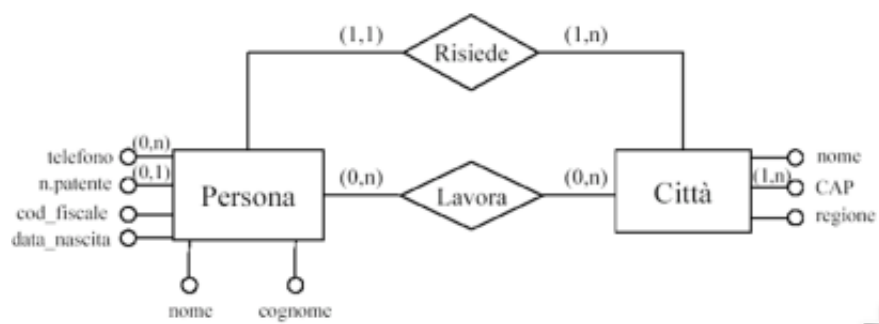
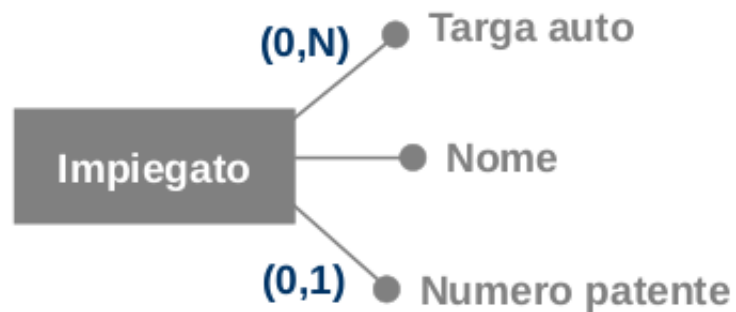
Istanza dello schema:

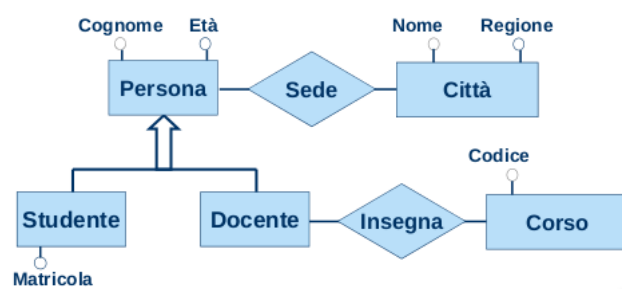
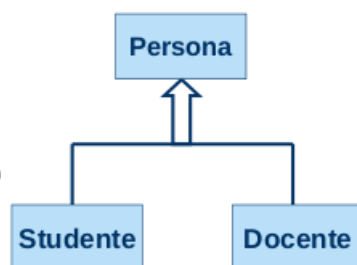
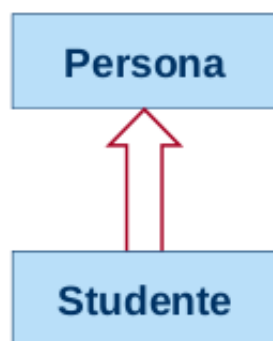
$\text{Istanze}(\text{Impiegato}) = \{ a, b, c \}$

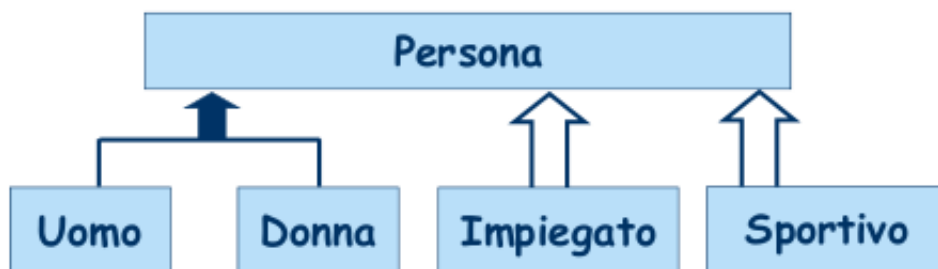
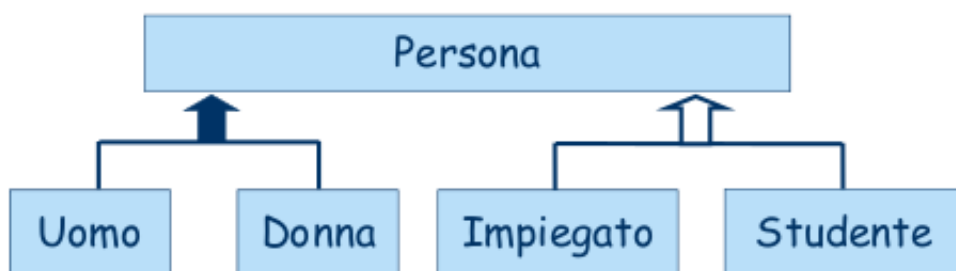
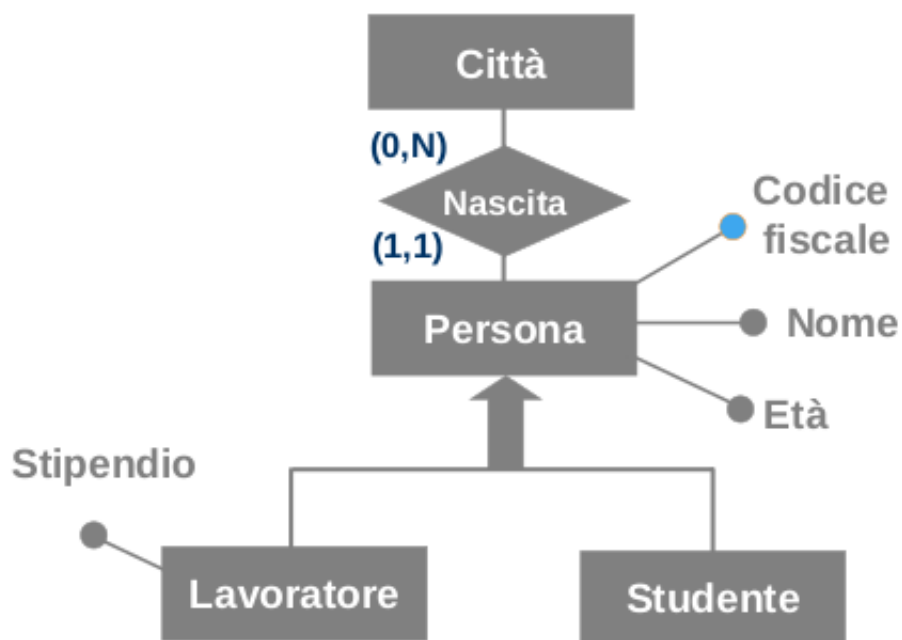
$\text{istanze}(\text{Progetto}) = \{ x, y, v, w, z \}$

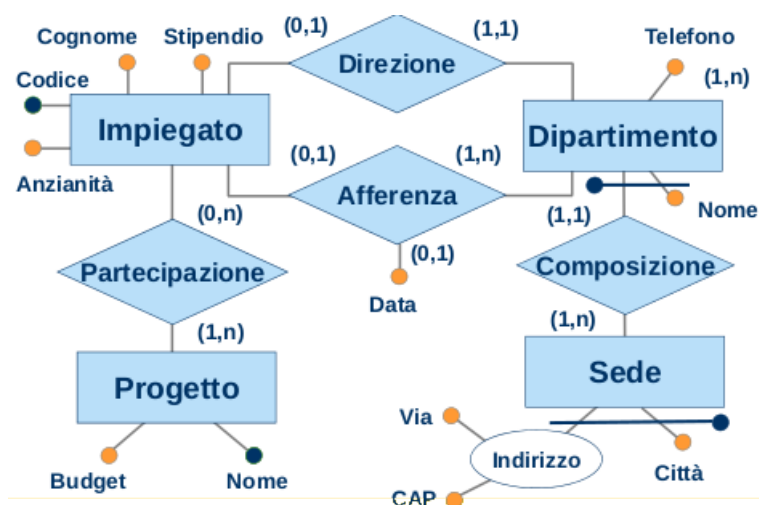
$\text{istanze}(\text{Assegnazione}) = \{ (a, w), (b, v), (b, w), (c, y), (c, w), (c, z) \}$

Ad ogni impiegato sono assegnati da 1 a 5 progetti e ogni progetto è assegnato ad al più 50 impiegati. a, b, c compaiono in almeno una istanza di Assegnazione. x non compare nelle istanze di Assegnazione. Infine ci sono progetti (ad esempio lanciati da poco tempo) che possono non essere assegnati a nessun impiegato









Vincoli di integrità esterni	
(1)	Il direttore di un dipartimento deve afferire a tale dipartimento da almeno 5 anni
(2)	Un impiegato non deve avere uno stipendio maggiore del direttore del dipartimento al quale afferisce
(3)	Un dipartimento con sede a Roma deve essere diretto da un impiegato con più di dieci anni di anzianità
(4)	Un impiegato non può partecipare ad un numero di progetti maggiore di due volte il numero di dipartimenti ai quali afferisce



Capitolo 4

Modello Relazionale

Dopo aver introdotto il modello concettuale, specifichiamo ed analizziamo ora il modello logico necessario per rappresentare in maniera efficace ed efficiente la base dati modellata.

Analizziamo il modello relazionale, il più diffuso dei modelli dei dati, in cui si utilizzano le tabelle e le relazioni per ottenere la rappresentazione voluta, introdotto negli anni '70 con l'obiettivo di rendere indipendenti i dati, ossia separare la modellazione “concettuale” dall'effettiva implementazione fisica sul DBMS.

Il modello relazionale, presenta una notevole differenza dai modelli precedenti, dato che i riferimenti fra dati in strutture diverse sono rappresentati per mezzo dei valori stessi, garantendo così l'indipendenza dei dati.

Il concetto di relazione, ovviamente proviene dalla teoria degli insiemi, rappresenta un sottoinsieme del prodotto cartesiano ma nel modello relazionale si utilizza una variante di esso, ossia ad ogni dominio, su cui è definito il prodotto cartesiano, viene associato un nome, chiamato *attributo*, al fine di superare la struttura posizionale della relazione matematica.

Questo aspetto di utilizzare un'attributo con una notazione non posizionale ci garantisce di poter accedere ai campi di una tupla mediante un nome e soprattutto ci evita di stabilire un ordine tra i diversi campi, ad esempio chi stabilisce che la matricola sia prima del nome e non viceversa.

Forniamo ora un pò di formalismo al fine di definire in maniera non ambigua il concetto di base di dati:

Definizione 1. *Si definisce dominio una funzione $dom : X \rightarrow D$ in cui si associa ogni attributo $A \in X$ un dominio $dom(A) \in D$ mentre si definisce tupla su un insieme di attributi X una funzione t che associa ogni elemento di X un elemento appartenente al dominio $dom(X)$.*

Definizione 2. Si definisce schema di relazione $R(X)$ formato da R indicante il nome della relazione mentre X indica un insieme di attributi, a cui ad ogni attributo viene associato un dominio, come abbiamo definito nella definizione precedente.

Si definisce invece istanza di relazione su uno schema $R(X)$ è un insieme r di tuple definite su X , a cui si può accedere ad un attributo della tupla attraverso r.a, con a nome di un attributo.

Definizione 3. Si definisce schema di base di dati un insieme $R = \{R_1(X_1), R_2(X_2), \dots, R_n(X_n)\}$ di schemi di relazione in cui risulta $R_i \neq R_j$ con $i \neq j$.

Si definisce inoltre istanza di base di dati su uno schema $R = \{R_1(X_1), \dots, R_n(X_n)\}$, un insieme di relazioni $r = \{r_1, r_2, \dots, r_n\}$ dove ogni r_i , con $1 \leq i \leq n$, è una relazione sullo schema $R_i(X_i)$.

Nel modello relazione i riferimenti tra i dati di relazioni diverse viene fatto attraverso i valori delle tuple, a differenza di altri modelli, come ad esempio il modello reticolare, in cui i riferimenti si effettuano attraverso i puntatori; il vantaggio di questo modo di gestire i riferimenti permette di ottenere l'indipendenza dei dati, evitando i riferimenti alla rappresentazione fisica ma ciò non significa che nel livello fisico non si possa usare i puntatori per gestire i riferimenti ma solo che si evita di renderlo visibile agli utenti e/o al livello logico.

Secondo le definizioni formali fatte degli schemi di relazioni e/o basi di dati, sono ammissibili relazioni su un solo attributo e ciò ha senso soprattutto in basi di dati con molte relazioni, in cui la relazione su un solo attributo contiene valori che appaiono come valori di un attributo di un'altra relazione.

La struttura del modello relazionale discussa ed analizzata nei precedenti paragrafi è indubbiamente molto semplice e potente ma essa impone un certo grado di rigidità, in quanto le informazioni devono essere rappresentate attraverso tuple appartenenti allo schema di relazione stesso ma ciò non sempre può avvenire.

Ad esempio in una relazione *Persona*(Cognome, Nome, Indirizzo, Telefono) non sempre il valore del telefono risulta definito per cui sarebbe scorretto usare un valore del dominio per rappresentare questa assenza di informazione in quanto esso genererebbe confusione ed ambiguità.

Per risolvere sto problema e rappresentare in maniera semplice e chiara la mancanza di valori si è deciso di estendere il concetto di relazione, permettendo di usare il valore speciale *null*, non facente parte il dominio degli attributi.

Questo valore nullo può rappresentare tre diverse tipologie di informazioni:

- indicare un valore sconosciuto, in cui si sa che il valore esiste ma non si conosce il suo valore.

- indicare un valore inesistente, in cui il valore dell'attributo non esiste.
- indicare l'assenza di informazione, in cui non si sa se il valore esiste e soprattutto che valore può rappresentare.

Nei sistemi di basi di dati relazionali non si accentrano sulla differenza tra le diverse tipologie ma suppone in maniera semplicista che si rappresenta sempre il caso di assenza di informazione.

Non sempre il valore nullo può essere sensato, infatti come si nota nella figura 2.13 l'assenza di valore di matricola e codice di corso crea notevoli problemi in quanto non permette di stabilire correlazioni fra tuple di relazioni diverse ed inoltre la presenza di molti valori nulli può generare dubbi sull'effettivo significato delle tuple, in quanto è impossibile riconoscere in maniera univoca le istanze della relazione.

Come vedremo nel prossimo paragrafo, sui vincoli d'integrità, si può indicare in quali attributi sono ammessi valori nulli e in quali no.

4.1 Vincoli d'integrità

Le strutture del modello relazione ci permettono di organizzare le informazioni d'interesse ma non è vero che tutti gli insiemi di tuple rappresentano dati corretti per l'applicazione.

Come ad esempio non è ammissibile che il voto assuma il valore 36, in quanto nel sistema universitario italiano i voti vanno da 0 a 30, poi inoltre compaiono nella relazione esami delle matricole non presenti nella relazione studente.

In un database è opportuno evitare le situazioni appena descritte, per cui a questo scopo sono stati introdotti i *vincoli di integrità*, ossia delle proprietà da soddisfare da tutte le istanze delle relazioni altrimenti lo schema di database risulta non ammissibile.

È possibile classificare i vincoli a seconda degli elementi coinvolti:

- **vincolo intrarelazionale:** sono i vincoli definiti sulle singole relazioni e su cui a sua volta si divide in 2 sottocategorie:
 1. *vincoli di tupla:* sono delle limitazioni effettuate su ciascuna tupla indipendentemente dalle altre e vengono definite tramite delle espressioni booleane, i cui atomi sono valutati attraverso dei confronti, come ad esempio un vincolo sui voti degli esami può essere $(Voto \geq 0) \text{ and } (Voto \leq 30)$.

2. *vincoli di chiavi*: sono i più importanti vincoli intra-relazionali, che permettono di avere la certezza dell'unicità di uno schema di relazione, come ad esempio si garantisce che non possono esistere due studenti con la stessa matricola.

La definizione formale di una chiave è la seguente:

Definizione 4. *Un insieme K di attributi è superchiave di una relazione r se r non contiene 2 tuple distinte t_1 e t_2 , con $t_1[K] = t_2[K]$.*

Si dice che K è chiave di r se è una superchiave minimale di r .

Per la relazione, presente nella figura 2.16, l'insieme $\{\text{Cognome}, \text{Corso}\}$ è una superchiave e poiché vi sono delle tuple uguali su cognomi e su corsi, tale insieme è pure una chiave ma questo vale sulle istanze presenti nella figura 2.16 e non in maniera generale, cosa che quanto sviluppiamo uno schema di relazione cerchiamo di ottenere l'unicità delle tuple su qualsiasi elementi ammissibili, per esempio nella relazione *Studente* la chiave indicata sarebbe la matricola, indice univoco usato da tutte le università per identificare gli studenti. Si può notare che in ogni relazione e relativo schema si ha la presenza di una chiave, infatti una relazione è un insieme, costituita come si sa bene da elementi diversi, da cui di conseguenza per ogni relazione $r(X)$ l'insieme X di tutti gli attributi su cui è definita è senz'altro una superchiave che può essere di due diversi tipi:

- tale insieme è anche una chiave, per cui risulta dimostrato l'esistenza della chiave
- tale insieme non è una chiave, ossia esiste un'altra superchiave, sottoinsieme di quella considerata, da cui si può applicare ricorsivamente le stesse considerazioni fatte, usando un sottoinsieme di relazione fino ad arrivare alla superchiave minimale in una sequenza finita di passi, dato che il numero di attributi è una quantità finita.

Lo stesso ragionamento può essere fatto anche a livello di schema di relazione, in cui l'insieme di tutti gli

attributi è una superchiave per ciascuna relazione e la ricerca della chiave si effettua alla stessa maniera di quella vista per l'istanza di una relazione.

Il fatto appena dimostrato, ossia che ogni relazione possieda una chiave, ci garantisce l'accessibilità e l'unicità di tutti i valori della base di dati ed inoltre permette di stabilire efficacemente le corrispondenze fra dati contenuti in relazioni diverse.

Come già notato nel paragrafo sui valori nulli, si deve avere dei valori senza alcuna informazione e per risolvere codesto problema si adotta una soluzione semplice ma efficace ossia su una delle chiavi, detta *chiave primaria*, si vieta la presenza di valori nulli mentre sugli altri attributi non si ha solitamente nessuna problematica riguardo ai valori null.

Gli attributi che costituiscono la chiave primaria vengono evidenziati attraverso una sottolineatura ed inoltre la maggior parte dei riferimenti avviene tramite chiave primaria.

Solitamente in quasi tutti i casi reali è possibile trovare degli attributi, i cui attributi sono identificativi e sempre disponibili ma in caso ciò non è possibile si introduce un codice non significativo per l'applicazione.

- *vincoli di integrità referenziale* (Foreign Key) fra un insieme di attributi X di una relazione R_1 e una relazione R_2 risulta soddisfatto se i valori di X su ciascuna tupla di R_1 compaiono come valori della chiave dell'istanza di R_2 . Vediamo un esempio:

Infrazioni

<u>Codice</u>	Data	Vigile	Prov	Numero
34321	1/2/95	3987	MI	39548K
53524	4/3/95	3295	TO	E39548
64521	5/4/96	3295	PR	839548
73321	5/2/98	9345	PR	839548

Vigili

<u>Matricola</u>	Cognome	Nome
3987	Rossi	Luca
3295	Neri	Piero
9345	Neri	Mario
7543	Mori	Gino

La definizione precisa e formale richiede un pochino di attenzione in più, in particolare nel caso in cui la chiave della relazione riferita sia composta e nel caso in cui vi siano più chiavi, per cui procediamo per gradi, incominciando prima con il caso di chiave unica:

Definizione 5. Sia R_2 una relazione con una sola chiave B e sia a sua volta $X = \{A\}$ l'insieme di attributi, allora il vincolo di integrità referenziale fra l'attributo A di R_1 e la relazione R_2 risulta soddisfatto se per ogni tupla t_1 in R_1 , in cui $t_1[A]$ non è nullo, esiste una tupla t_2 in R_2 tale che $t_1[A] = t_2[B]$.

Nel caso più generale bisogna stare attenti al fatto che ogni attributo di X deve corrispondere ad uno specifico attributo della chiave primaria di R_2 per questo la definizione è la seguente:

Definizione 6. Siano $X = \{A_1, A_2, \dots, A_n\}$ e $K = \{B_1, B_2, \dots, B_n\}$ due insiemi ordinati di attributi per le relazioni R_1 e R_2 , il vincolo risulta soddisfatto se per ogni tupla t_1 in R_1 , senza attributi nulli

su X , esiste una tupla t_2 in R_2 tale per cui $t_1[A_i] = t_2[B_i]$, per ogni i compreso tra 1 e p .

Per analizzare i vincoli di integrità referenziale consideriamo la figura in cui le informazioni della relazione Infrazioni sono rese significative attraverso il collegamento con la relazione Agenti, con l'attributo agente, e alla relazione Auto, per mezzo degli attributi stato e numero di targa:

Infrazioni

<u>Codice</u>	Data	Vigile	Prov	Numero
34321	1/2/95	3987	MI	39548K
53524	4/3/95	3295	TO	E39548
64521	5/4/96	3295	PR	839548
73321	5/2/98	9345	PR	839548

Vigili

<u>Matricola</u>	Cognome	Nome
3987	Rossi	Luca
3295	Neri	Piero
9345	Neri	Mario
7543	Mori	Gino

In questa situazione analizzata abbiamo deciso di inserire i seguenti vincoli referenziali:

- fra l'attributo *Agente* della relazione Infrarossi e la relazione Agenti.
- fra gli attributi Stato e Numero di Infrarossi e la relazione Auto, in cui l'ordine degli attributi nella chiave preveda Stato e poi Numero.

Relativamente al secondo vincolo, notiamo come il ragionamento sull'ordine degli attributi può risultare pesante, visto che la corrispondenza può, almeno in questo caso, essere realizzata per mezzo dei nomi degli attributi ma in generale ciò non può accedere, quindi l'ordinamento è essenziale.

4.1.1 In pratica

Una relazione è spesso rappresentata da una tabella ove le righe rappresentano specifici record e le colonne corrispondono ai campi dei record, l'ordine di righe e colonne è sostanzialmente irrilevante:

nome	cognome	Data di nascita	professione	tel
mario	rossi	21/10/80	impiegato	02 345678
sara	bianchi	17/03/77	avvocato	031 45678
marco	verdi	11/11/67	medico	06 789052

In una base di dati relazionale ci sono più relazioni:

studenti	Matricola	Cognome	Nome	Data di nascita
	6554	Rossi	Mario	05/12/1978
	8765	Neri	Paolo	03/11/1976
	9283	Verdi	Luisa	12/11/1979
	3456	Rossi	Maria	01/02/1978

esami	Studente	Voto	Corso
	3456	30	04
	3456	24	02
	9283	28	01
	6554	26	01

corsi	Codice	Titolo	Docente
	01	Analisi	Mario
	02	Chimica	Bruni
	04	Chimica	Verdi

Si hanno 2 componenti:

1. lo **schema**, invariante nel tempo, che ne descrive la struttura (aspetto intensionale). Si rappresenta con le intestazioni delle tabelle
2. l'**istanza**, i valori attuali, che possono cambiare anche molto rapidamente (aspetto estensionale). Si rappresenta col corpo di ciascuna tabella

Si hanno 3 accezioni del concetto di relazione:

1. **relazione matematica**: come nella teoria degli insiemi
2. **associazione/correlazione** che rappresenta una classe di fatti, nel modello *Entity-Relationship*
3. **relazione** secondo il modello relazionale dei dati

quindi ad esempio:

Juve	Lazio	3	1
Lazio	Milan	2	0
Juve	Roma	0	2
Roma	Milan	0	1

4.1.2 Relazione Matematica

Una relazione matematica sugli insiemi D_1 e D_2 , detti **domini**, è un sottoinsieme del prodotto cartesiano $D_1 \times D_2$. Le relazioni si possono visualizzare efficacemente con una tabella in cui ogni colonna corrisponde ad un dominio e ogni riga a un elemento della relazione. Quindi una relazione matematica è un insieme di n-uple ordinate (d_1, \dots, d_n) tali che $d_i \in D_i$.

Si hanno delle proprietà:

- non c'è ordinamento fra le n-uple
- le n-uple sono distinte
- ciascuna n-upla è ordinata: l'i-esimo valore proviene dall'i-esimo dominio

Dato che ciascuno dei domini ha un ruolo definito dalla posizione nella tabella si ha una **struttura posizionale**

4.1.3 Relazioni nel Modello Relazionale

Si ha che:

- a ciascun dominio si associa un nome (attributo), che ne descrive il "ruolo"
- gli attributi possono essere usati come intestazione
- struttura non posizionale

Ovvero:

Casa	Fuori	RetiCasa	RetiFuori
Juve	Lazio	3	1
Lazio	Milan	2	0
Juve	Roma	0	2
Roma	Milan	0	1

Si hanno delle regole per poter identificare una relazione con una tabella:

- i valori di ogni colonna sono fra loro omogenei
- le righe sono diverse fra loro
- le intestazioni delle colonne sono diverse tra loro

Inoltre l'ordinamento di righe e colonne è irrilevante.

Si possono creare corrispondenze fra le tuple di relazioni distinte per mezzo di valori degli attributi che compaiono nelle ennuple:

studenti	Matricola	Cognome	Nome	Data di nascita
	6554	Rossi	Mario	05/12/1978
	8765	Neri	Paolo	03/11/1976
	9283	Verdi	Luisa	12/11/1979
	3456	Rossi	Maria	01/02/1978

esami	Studente	Voto	Corso
	3456	30	04
	3456	24	02
	9283	28	01
	6554	26	01

corsi	Codice	Titolo	Docente
	01	Analisi	Mario
	02	Chimica	Bruni
	04	Chimica	Verdi

dove matricola e studente rappresentano la stessa informazione, così come corso e codice.

Si hanno diversi vantaggi nell'uso della struttura basata su valori:

- indipendenza dalle strutture fisiche (si potrebbe avere anche con puntatori di alto livello) che possono cambiare dinamicamente
- si rappresenta solo ciò che è rilevante dal punto di vista dell'applicazione
- i dati sono portabili più facilmente da un sistema ad un altro
- per accedere ai dati non serve sapere come sono memorizzati fisicamente

vediamo quindi un paio di esempi:

STUDENTI(Matricola,Cognome,Nome,Data di Nascita)**Istanza della relazione studenti**

studenti	Matricola	Cognome	Nome	Data di nascita
	6554	Rossi	Mario	05/12/1978
	8765	Neri	Paolo	03/11/1976
	9283	Verdi	Luisa	12/11/1979
tupla i →	3456	Rossi	Maria	01/02/1978

Da una tupla possiamo estrarre

il valore degli attributi t_i [matricola]=3456

PARTITE (Casa, Fuori, RetiCasa, RetiFuori)**Istanza della relazione partite**

Partita	Casa	Fuori	RetiCasa	RetiFuori
t1	Juve	Lazio	3	1
t2	Lazio	Milan	2	0
t3	Juve	Roma	0	2
t4	Roma	Milan	0	1

t1[casa]= Juve

t2[Fuori]= Milan

t4[casa]= Roma

t4[Fuori]= Milan

t1[RetiCasa]= 3

t3[RetiFuori]= 2

e possono esistere relazioni su un solo attributo:

studenti

Matricola	Cognome	Nome	Data di nascita
6554	Rossi	Mario	05/12/1978
8765	Neri	Paolo	03/11/1976
9283	Verdi	Luisa	12/11/1979
3456	Rossi	Maria	01/02/1978

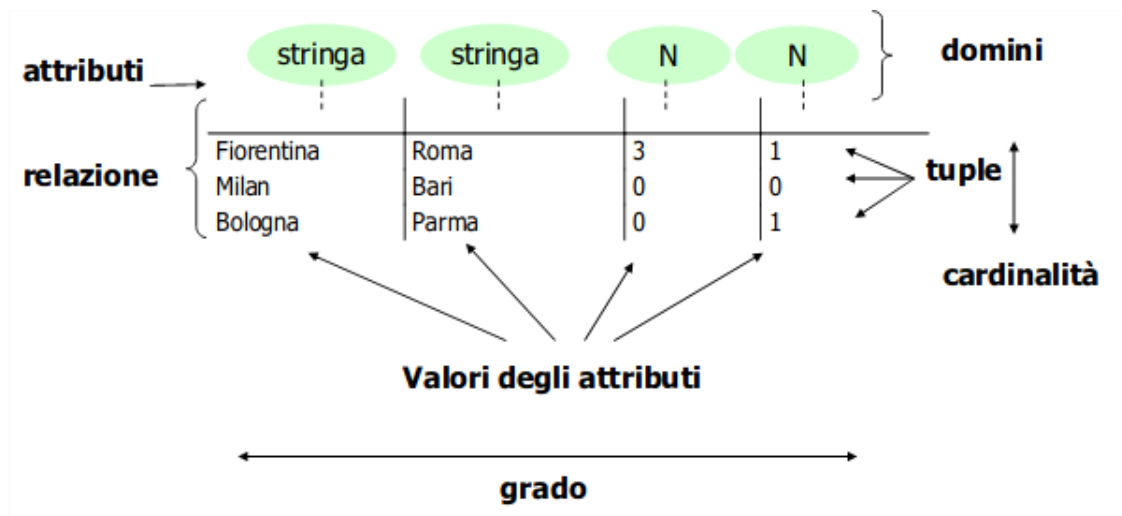
studenti lavoratori

Matricola
6554
3456

Si hanno delle regole di notazione:

- **attributi:** lettere iniziali dell'alfabeto, maiuscole: $A, B, C, A', A1, \dots$
- **insieme di attributi:** lettere finali dell'alfabeto, maiuscole: $X, Y, Z, X', X1, \dots$
- **unioni di insiemi:** XY anziché $X \cup Y$
- **nomi di relazioni:** R e lettere circostanti, maiuscole, anche con indici e pedici: $R1, S, S', \dots$
- **relazione:** come il nome della relazione, ma in minuscolo
- **schema di base di dati:** lettera maiuscola in grassetto $\mathbf{R}, \mathbf{S}, \dots$
- **base di dati:** stesso simbolo dello schema, ma in minuscolo

Vediamo un'immagine riassuntiva:



Si possono rappresentare strutture nidificate, diverse a seconda della necessità:

Ricevute	Numero	Data	Totale
	1235	12/10/2000	39,20
	1240	13/10/2000	39,00

Dettaglio	Numero	Qtà	piatto	costo
	1235	3	Coperti	3,00
	1235	2	Antipasti	6,20
	1235	3	Primi	12,00
	1235	2	Bistecche	18,00
	1240	2	Coperti	2,00

Ricevute	Numero	Data	Totale
	1235	12/10/2000	39,20
	1240	13/10/2000	39,00

Dettaglio	Numero	Riga	Qtà	Descrizione	Importo
	1235	1	3	Coperti	3,00
	1235	2	2	Antipasti	6,20
	1235	3	3	Primi	12,00
	1235	4	2	Bistecche	18,00
	1240	1	2	Coperti	2,00

Si definisce la **chiave** come *l'insieme di attributi che identificano le tuple di una relazione*.

Più formalmente un insieme di K attributi è una **superchiave** per r se r non contiene due tuple distinte r_1 e t_2 con $t_1[K] = t_2[K]$. Inoltre è chiave

per r se è una **superchiave minimale** per r (cioè non contiene un'altra superchiave). Vediamo un esempio:

Matricola	Cognome	Nome	Corso	Nascita
27655	Rossi	Mario	Ing Inf	5/12/78
78763	Rossi	Mario	Ing Inf	3/11/76
65432	Neri	Piero	Ing Mecc	10/7/79
87654	Neri	Mario	Ing Inf	3/11/76
67653	Rossi	Piero	Ing Mecc	5/12/78

Nella tabella Matricola è una superchiave ed è minimale (contiene un solo attributo). L'insieme dato da Cognome, Nome e Nascita è una superchiave minimale. Volendo sarebbe anche una chiave la coppia Cognome-Corso ma noi però interessano le chiavi corrispondenti a vincoli di integrità soddisfatti da tutte le relazioni lecite dello schema (sarebbe una chiave senza senso).

Il modello relazionale ha una **struttura rigida**, dove le informazioni sono rappresentate da tuple e solo alcuni formati di tuple sono ammessi (quelli che corrispondono agli schemi di relazione). **I dati disponibili possono non corrispondere al formato previsto.**

Può capitare di avere un'informazione incompleta. In tal caso non conviene usare valori del dominio (0, stringa nulla) perché potrebbero diventare significativi. Si usa quindi il valore nullo **NULL**, che denota l'assenza di un valore del dominio (e non è un valore del dominio). Si hanno almeno tre casi di valore nullo anche se i DBMS non li distinguono:

1. valore sconosciuto
2. valore inesistente
3. valore senza informazione

Ovviamente l'uso del valore NULL deve essere sensato (per esempio di uno studente non si può avere data di nascita o matricola NULL). Esistono istanze di basi di dati che, pur sintatticamente corrette, non rappresentano informazioni possibili per l'applicazione di interesse. Una chiave in cui non sono ammessi NULL è detta **chiave primaria** e come notazione si usa la sottolineatura

Capitolo 5

SQL

Il linguaggio SQL è un linguaggio per la definizione e la manipolazione dei dati in database relazionali, sviluppato originariamente presso il laboratorio IBM a San Jose' (California) e adottato nel sistema System R. L'SQL è stato poi adottato da molti altri DBMS e quindi è stato soggetto ad un'intensa attività di standardizzazione. È un linguaggio con varie funzionalità che contiene:

- **DDL:** definizione di domini, tabelle, autorizzazioni, vincoli, procedure, ecc.
- **DML:** linguaggio di query, modifica, ...

Si studierà una versione vecchia di SQL in quanto è la più implementata. Nello specifico studieremo SQL-2. SQL-3 è comunque compatibile con SQL-2 e introduce il concetto di oggetto. Noi useremo SQL per definire, aggiornare e interrogare la base di dati. Sono sempre più frequenti in realtà sistemi dotati di interfacce più facili da usare. Questi programmi generano le istruzioni SQL corrispondenti.

Esistono diverse implementazioni di SQL, per esempio il DBMS di Oracle, Access, Mysql...

SQL è relazionale completo e ogni espressione dell'algebra relazionale può essere tradotta in SQL. Il modello dei dati di SQL è basato su tabelle anziché relazioni e possono essere presenti righe duplicate (le tuple) e adotta la logica a 3 valori dell'algebra relazionale (gestendo il null e aggiungendo il valore di verità U, *unknown*).

Le tabelle di verità diventano quindi:

Tabelle di verità

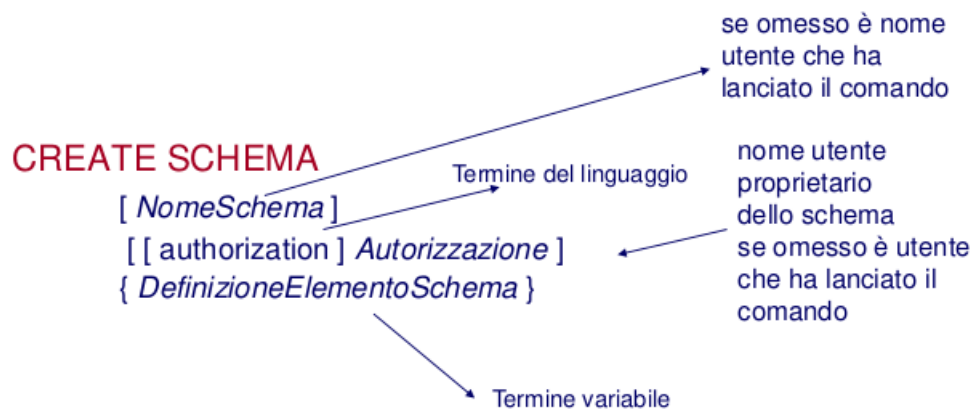
<i>not</i>		<i>and</i>	F T U	<i>or</i>	F T U
T	F	T	F T U	T	T T T
F	T	F	F F F	F	F T U
U	U	U	F U U	U	U T U

Vediamo la notazione per definire la sintassi di SQL:

- $\langle x \rangle$ per isolare un termine x
- $[x]$ indicano che un termine x è opzionale
- $\{x\}$ indicano che un termine x può essere ripetuto 0 volte o un numero arbitrario di volte
- $|$ separa opzioni alternative

Le parentesi tonde () appartengono al linguaggio SQL e non alla notazione sopra descritta.

Uno **schema di base di dati** è una collezione di oggetti e ogni schema ha un nome e un proprietario. SI ha la seguente sintassi:



Dopo il comando CREATE SCHEMA compaiono le definizioni dei vari elementi. Non è necessario che la definizione di tutti gli elementi avvenga contemporaneamente alla creazione dello schema. Può avvenire in più fasi successive.

Una **tabella** è costituita da una collezione ordinata di attributi e da un insieme (eventualmente vuoto) di vincoli:

```
CREATE TABLE NomeTabella (  
    NomeAttributo Dominio [ValoreDiDefault] [Vincoli]  
    {,NomeAttributo Dominio [ValoreDiDefault] [Vincoli] }  
    [AltriVincoli])
```

questa istruzione quindi definisce uno schema di relazione e ne crea un'istanza vuota. Per ogni attributo va specificato il dominio, un eventuale valore di default (per questo è comodo il null, che è il valore di default di base) ed eventuali vincoli. Possono essere espressi altri vincoli a livello di tabella (tra più attributi).

I **domini** specificano i valori ammissibili per gli attributi di una relazione. SQL ha 6 domini predefiniti:

1. bit SQL-2 poi eliminato e sostituito parzialmente da BOOLEAN in SQL-3
2. carattere
3. numerico Esatto
4. numerico Approssimato
5. data/Ora
6. intervallo Temporale

più ovviamente i domini definiti dall'utente.

Vediamo il **dominio di tipo Bit**. Può essere di lunghezza fissa o variabile. Se la lunghezza non è specificata corrisponde ad 1 singolo valore. Corrisponde ad attributi che possono assumere solo due valori (0,1). Attributi di questo tipo (*flag*) indicano se l'oggetto rappresentato possiede o meno una certa proprietà:

```
bit [varying] [(lunghezza)]  
bit  
bit(lunghezza)  
bit varying (lunghezza)  
varbit (lunghezza)
```

Esempio 4. *Si definisce l'attributo `Lavoratore` nella relazione `STUDENTI` per indicare se lo studente è o meno lavoratore:*

Lavoratore **bit**

Vediamo il **tipo carattere**. Rappresenta singoli caratteri alfanumerici oppure stringhe di lunghezza fissa o variabile:

```
character [varying] [(lunghezza)]
character o char
character(lunghezza)
varchar(lunghezza)
character varying (lunghezza)
```

Esempio 5. *Si definisce l'attributo `Nome` della relazione `IMPIEGATI` come sequenza di caratteri di lunghezza massima 20:*

Nome **char**(20)
Nome **varchar**(20)

con il primo che genera `Paolo_Bianchi_ _ _ _ _` e il secondo `Paolo_Bianchi`

Vediamo i **tipi numerici esatti**. Rappresentano numeri interi o numeri decimali in virgola fissa (con un numero prefissato di decimali, ad esempio i valori monetari). Precision è numero di cifre significative, scala il numero di cifre dopo la virgola:

```
integer
smallint
numeric(precisione) numeric(precisione,scala)
decimal(precisione) decimal(precisione,scala)
```

Esempio 6. *Si definisce l'attributo `Eta` nella relazione `IMPIEGATI`:*

Eta **decimal**(2) // Rappresenta tutti i numeri fra -99 e +99

Si definisce l'attributo `Cambio` nella relazione `PAGAMENTO` per il valore del cambio di una certa moneta preciso al centesimo

Cambio **numeric**(6,2) // Rappresenta tutti i numeri fra -9999,99 e +9999,99

quindi `INTEGER` / `SMALLINT` rappresentano valori interi. La precisione (numero totale di cifre) varia a seconda della specifica implementazione di SQL, non è specificata nello standard. `SMALLINT` richiede minore spazio di

memorizzazione.

NUMERIC / *DECIMAL* rappresentano i valori decimali. La differenza tra *NUMERIC* e *DECIMAL* è che il primo deve essere implementato esattamente con la precisione richiesta, mentre il secondo può avere una precisione maggiore, la precisione segnalata è requisito minimo. Se la precisione non è specificata si usa il valore caratteristico dell'implementazione. Per la scala si usa valore 0. Si hanno ovviamente:

```
float float(precisione)
double precision
real
```

che sono utili per rappresentare valori reali approssimati, ad esempio grandezze fisiche (rappresentazione in virgola mobile, in cui a ciascun numero corrisponde una coppia di valori: mantissa e esponente).

Per la data si ha:

```
date
time time(precisione)
timestamp timestamp(precisione)
```

Ciascuno di questi domini è strutturato e decomponibile in un insieme di campi (anno, mese, giorno, ora, minuti, secondi):

```
DataDiNascita date // 1999-09-18
OraDiConsegna time // 19.24.16
Arrivo timestamp // 2000-09-18 21.15.20
```

SI possono avere intervalli temporali:

```
interval PrimaUnitàDiTempo
interval PrimaUnitàDiTempo to UltimaUnitàDiTempo
```

Infine si hanno i tipi **blob**. Permettono di includere direttamente nel database oggetti molto grandi. *Binary Large Object (BLOB)* e *Character Large Object (CLOB)*:

```
fotografia BLOB(10M)
descrizione CLOB(100k)
```

Definiti solo in SQL-3, ma implementati in diversi DBMS commerciali. Il sistema garantisce solo di memorizzarne il valore. Non possono essere usati come criterio di selezione per le interrogazioni.

Per creare un dominio uso la primitiva *create domain*:


```
CREATE DOMAIN NomeDominio AS DominioElementare
    [ValoreDefault] [Constraints]
```

Un nuovo dominio è caratterizzato dalle seguenti informazioni: nome, dominio elementare, valore di default, insieme di vincoli (constraints). Vediamo un esempio:

```
CREATE DOMAIN Voto AS SMALLINT
    DEFAULT 0
    NOT NULL
```

Il nuovo dominio Voto è definito come uno SMALLINT con valore di default e che non deve essere null.

La definizione di "nuovi domini" è utile perché permette di associare dei vincoli a un nome di dominio: questo è importante quando si deve ripetere la stessa definizione di attributo su diverse tabelle: ad esempio, modifiche alla definizione di Voto si ripercuotono in tutte le occorrenze di questo dominio nello schema del Database.

Esempio 7. *Vediamo un esempio concreto:*

```
CREATE TABLE NomeTabella (
    NomeAttributo Dominio [ValoreDiDefault] [Vincoli]
    {, NomeAttributo Dominio [ValoreDiDefault] [Vincoli] }
    [AltriVincoli] )
```

```
CREATE TABLE Impiegato (
    Matricola CHAR(6) PRIMARY KEY,
    Nome CHAR(20) NOT NULL,
    Cognome CHAR(20) NOT NULL,
    Dipart CHAR(15)
    Stipendio NUMERIC(9) DEFAULT 0,
    UNIQUE (Cognome, Nome)
)
```

UNIQUE richiede che non ci siano coppie cognome-nome ripetute

Un vincolo (constraint) è una regola che specifica delle condizioni sui valori di un elemento dello schema del database. Un vincolo può essere associato ad una tabella, ad un attributo, ad un dominio. Sono di due tipi:

1. **vincoli intrarelazionali** che si applicano all'interno di una relazione. Possono essere:

```
NOT NULL // (Il valore deve essere non null)
UNIQUE // (I valori devono essere non ripetuti)
PRIMARY KEY // (Chiave primaria)
CHECK // (Condizioni complesse)
```

Il vincolo PRIMARY KEY può essere definito una sola volta all'interno della relazione. In alcune implementazioni di SQL potrebbe essere necessario specificare comunque anche il vincolo NOT NULL per tutti gli attributi coinvolti.

2. **vincoli interrelazionali** che si applicano tra relazioni diverse. Possono essere definiti attraverso i costrutti sintattici:

```
REFERENCES // Permettono di definire vincoli di integrità
FOREIGN KEY // referenziale
CHECK // (Vincoli complessi)
```

e si hanno sintassi per singoli o più attributi. È possibile definire politiche di reazione alle violazioni. Si ha l'**integrità referenziale** che esprime un legame gerarchico (padre / figlio) fra tabelle. Alcuni attributi della tabella figlio sono definiti FOREIGN KEY e si devono riferire (REFERENCES) ad alcuni attributi della tabella padre che costituiscono una chiave (devono essere UNIQUE e NOT NULL oppure PRIMARY KEY). I valori contenuti nella FOREIGN KEY devono essere sempre presenti nella tabella padre. Si hanno due sintassi:

- (a) nella parte di definizione degli attributi con il costrutto sintattico REFERENCES:

```
AttrFiglio CHAR(3) REFERENCES TabellaPadre(AttrPadre)
```

- (b) oppure dopo le definizioni degli attributi con i costrutti FOREIGN KEY e REFERENCES:

```
FOREIGN KEY (AttrFiglio)
REFERENCES TabellaPadre(AttrPadre)
```

quindi:

```
CREATE TABLE Impiegato (
  Matricola CHAR(6) PRIMARY KEY,
  Nome CHAR(20) NOT NULL,
  Cognome CHAR(20) NOT NULL,
  Dipart CHAR(15) REFERENCES Dipartimento(NomeDip) ,
```

```

        Stipendio NUMERIC(9) DEFAULT 0,
        UNIQUE (Cognome, Nome)
    )

```

Se si omettono gli attributi destinazione, vengono assunti quelli della chiave primaria.

```

AttrFiglio CHAR(3) REFERENCES TabellaPadre

```

Quando si hanno più attributi da riferire, si utilizza sempre FOREIGN KEY:

```

FOREIGN KEY (AttrFiglio1 { }, AttrFiglio2 { })
    REFERENCES TabellaPadre (AttrPadre1 { }, AttrPadre2 { })

```

Esempio 8. Definiamo tre tabelle con le informazioni degli esami sostenuti dagli studenti: *Studente*, *Esame*, *Corso*:

```

CREATE TABLE Studente (
    Matr CHAR(6)
    Nome VARCHAR(30)
    Città VARCHAR(20),
    CDip CHAR(3)
)

```

<u>Matr</u>	Nome	Città	CDip
34321	Luca	Mi	Inf
53524	Giovanni	To	Mat
64521	Emilio	Ge	Ing
73321	Francesca	Vr	Mat

```

CREATE TABLE Esame (
    Matr CHAR(6),
    CodCorso CHAR(6),
    Data DATE NOT NULL,
    Voto Voto,
    PRIMARY KEY (Matr, CodCorso)
)

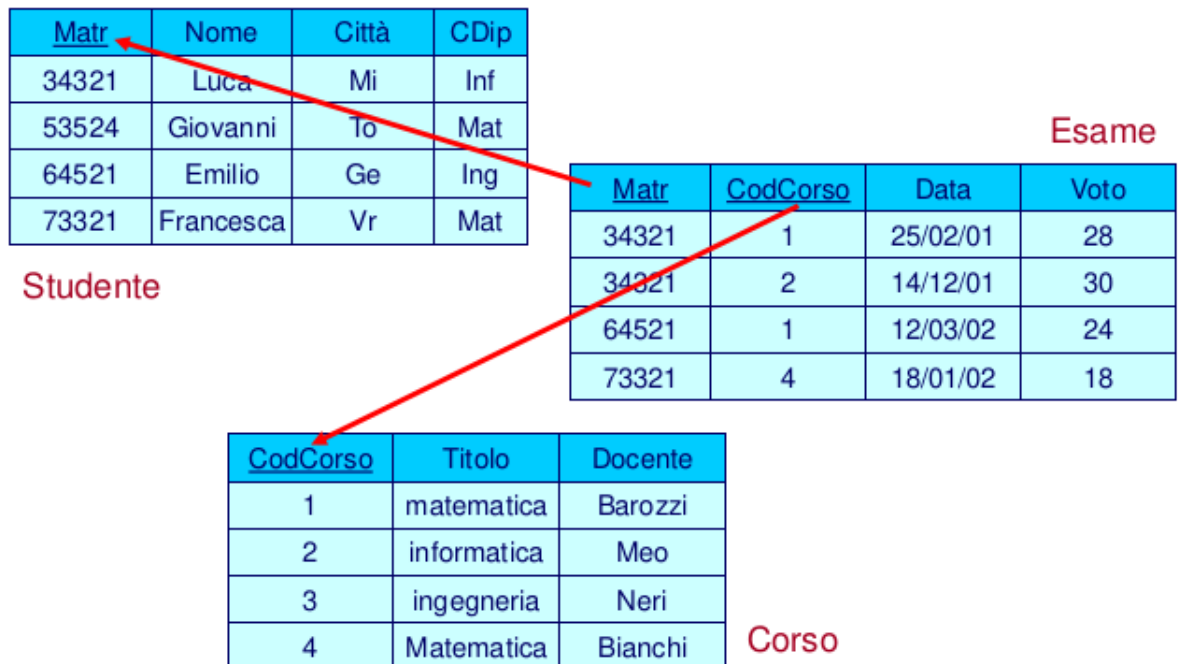
```

<u>Matr</u>	<u>CodCorso</u>	Data	Voto
34321	1	25/02/01	28
34321	2	14/12/01	30
64521	1	12/03/02	24
73321	4	18/01/02	18

```
CREATE TABLE Corso (
  CodCorso CHAR(6) PRIMARY KEY,
  Titolo VARCHAR(30) NOT NULL,
  Docente VARCHAR(20)
)
```

<u>CodCorso</u>	Titolo	Docente
1	matematica	Barozzi
2	informatica	Meo
3	ingegneria	Neri
4	Matematica	Bianchi

con le relazioni:



aggiungiamo un vincolo a esame:

```
CREATE TABLE Esame (
  Matr CHAR(6),
  CodCorso CHAR(6),
  Data DATE NOT NULL,
  Voto Voto,
  PRIMARY KEY (Matr, CodCorso)
  FOREIGN KEY (Matr) REFERENCES Studente
  FOREIGN KEY (CodCorso) REFERENCES Corso
```

oppure:

```
CREATE TABLE Esame (
  Matr CHAR(6), REFERENCES Studente,
  CodCorso CHAR(6), REFERENCES Corso,
  Data DATE NOT NULL,
  Voto Voto,
  PRIMARY KEY (Matr, CodCorso)
```

Si introduce il **problema delle violazioni**.

Per tutti gli altri vincoli visti fino ad ora, a seguito di una violazione, il comando di aggiornamento viene rifiutato segnalando l'errore all'utente.

Per i vincoli di integrità referenziale invece SQL permette di scegliere delle reazioni da adottare in caso di violazioni.

Si possono introdurre violazioni operando sulle righe della tabella padre (tabella esterna) o sulle righe della tabella figlio (tabella interna). Si hanno le seguenti modifiche sulla tabella interna (figlio):

- inserimento di una nuova riga
- modifiche della *foreign key*

Inoltre non vengono proposte reazioni, solo il rifiuto dell'operazione. Per la tabella esterna (padre) si hanno le seguenti modifiche:

- cancellazione di una riga
- modifica dell'attributo riferito

inoltre vengono proposte diverse reazioni.

Si ha anche il problema dell'aggiornamento. Se rimuovi una matricola alla tabella Esame si aggiunge un problema, il **problema degli orfani**.

Si hanno le reazioni alle violazioni, che si introducono con una modifica (update) dell'attributo cui si fa riferimento o con la cancellazione di tuple. Le reazioni operano sulla tabella figlio (es Esami), in seguito a modifiche alla tabella padre (es Studente):

CASCADE: propaga la modifica

SET NULL: annulla l'attributo che fa riferimento

SET DEFAULT: assegna il valore di **default** all'attributo

NO ACTION: impedisce che la modifica possa avvenire

A fini diagnostici (e di documentazione) è spesso utile sapere quale vincolo è stato violato a seguito di un'azione sul DB. A tale scopo è possibile associare dei nomi ai vincoli, ad esempio:

```
tipendio INTEGER CONSTRAINT StipendioPositivo  
CHECK (Stipendio > 0),
```

```
CONSTRAINT ForeignKeySedi  
FOREIGN KEY (Sede) REFERENCES Sedi
```

Per le modifiche degli schemi si hanno:

- **ALTER** per la modifica
- **DROP** che cancella oggetti dallo schema

```
ALTER DOMAIN NomeDominio <
  SET DEFAULT ValoreDefault |
  DROP DEFAULT |
  ADD CONSTRAINT DefVincolo |
  DROP CONSTRAINT NomeVincolo >
```

```
ALTER TABLE NomeTabella <
  ALTER COLUMN NomeAttributo <
    SET DEFAULT NuovoDefault |
    DROP DEFAULT > |
  DROP COLUMN NomeAttributo |
  ADD COLUMN DefAttributo |
  DROP CONSTRAINT NomeVincolo
  ADD CONSTRAINT DefVincolo >
```

DROP cancella oggetti DDL, si applica su domini, tabelle, indici, view, asserzioni, procedure,...

```
DROP < schema, domain, table, view, ... > NomeElemento
[ RESTRICT | CASCADE ]
```

- RESTRICT: impedisce drop se gli oggetti comprendono istanze non vuote
- CASCADE applica drop agli oggetti collegati. Potenziale pericolosa reazione a catena

5.0.1 Interrogazioni in SQL

Le interrogazioni si fanno con la SELECT:

```
SELECT ListaAttributi
FROM ListaTabelle
[WHERE Vondizioni]
```

```
SELECT AttrEspr [ [ AS ] Alias ] {, AttrEspr [ [ AS ] Alias ] }
FROM Tabella [ [ AS ] Alias ] {, Tabella [ [ AS ] Alias ] }
[ WHERE Condizione ]
```

con gli AS setto gli alias. Vediamo qualche esempio:

```
-- Individuare lo stipendio degli impiegati di cognome
"Rossi":
```

```
SELECT Stipendio
FROM Impiegato
WHERE Cognome="Rossi"
```

```
-- Selezionare nome, cognome e mansione degli
impiegati dell'ufficio 10
```

```
SELECT Nome, Cognome, Mansione
FROM Impiegato
WHERE ufficio='10'
```

```
-- Individuare le informazioni degli impiegati con
stipendio > 40
Individuare le informazioni degli impiegati con
stipendio > 40
```

```
-- Selezionare tutte le informazioni sui dipendenti:
```

```
SELECT id_impiegato, Nome, Cognome, Dipart, Ufficio,
stipendio, premioprod, Mansione, Città, idcapo
FROM Impiegato
```

```
SELECT *
FROM Impiegato
```

```
-- Selezionare cognome, stipendio e premio di
produzione per gli impiegati che hanno uno
stipendio compreso tra 38 e 60:
```

```
ELECT cognome, stipendio, premioprod AS premio_di_produzione
FROM Impiegato
WHERE ( stipendio>=38 AND stipendio <=60 )
```

```
-- Selezionare i nomi, i cognomi e le città di provenienza
degli impiegati
```

```
SELECT Nome, Cognome, città AS città_provenienza
FROM Impiegato
```

```
-- Selezionare i nomi e i cognomi degli impiegati, le città
di provenienza e le città in cui lavorano:
```

```
SELECT Nome, Cognome, CittàDip AS città_lavoro,
Città AS città_provenienza
FROM Impiegato, Dip
WHERE Dipart = NomeDip
```


/ Specificare più tabelle su cui operare per determinare il risultato significa: eseguire il prodotto cartesiano delle tabelle e poi selezionare le righe che soddisfano clausola WHERE.
Join di algebra relazionale*/*

-- Selezionare i nomi e i cognomi degli impiegati, le città di provenienza e le città in cui lavorano:

```
SELECT Nome, Cognome, CittàDip AS città_lavoro,  
Città AS città_provenienza  
FROM Impiegato, Dip  
WHERE Dipart = NomeDip
```

-- Selezionare i nomi e i cognomi degli impiegati, le città di provenienza e le città in cui lavorano:

```
SELECT Nome, Cognome, Città AS città_lavoro,  
Città AS città_provenienza  
FROM Impiegato, Dip  
WHERE Dipart = Nome
```

/ Notazione punto: si utilizza il nome della tabella cui l'attributo fa riferimento */*

```
SELECT Impiegato.Nome, Cognome, Dip.Città AS città_lavoro,  
Impiegato.Città AS città_provenienza  
FROM Impiegato, Dip  
WHERE Dipart = Dip.Nome
```

/ Si usano Alias per ridenominare le tabelle:*

1) Abbreviano riferimento a tabelle

2) Risolvono ambiguità di riferimento/*

```
SELECT I.Nome, Cognome, D.Città AS città_lavoro,  
I.Città AS città_provenienza  
FROM Impiegato I, Dip D  
WHERE Dipart = D.Nome
```

pagina 18

5.1 Progettazione Concettuale, approfondimento

La **progettazione concettuale** di una base di dati consiste nella costruzione di uno schema **Entità-Relazione** in grado di descrivere al meglio le specifiche sui dati di una applicazione. Anche nel caso di applicazioni non particolarmente complesse, lo schema che si ottiene può contenere molti concetti correlati in una maniera piuttosto complicata. Ne consegue che la costruzione dello schema finale è, necessariamente, un processo graduale: lo schema concettuale viene progressivamente raffinato e arricchito attraverso una serie di trasformazioni ed eventuali correzioni. In questo capitolo verranno descritte le strategie che è possibile seguire in questo processo di sviluppo di uno schema concettuale.

Per **raccolta dei requisiti** si intende la completa individuazione dei problemi che l'applicazione da realizzare deve risolvere, con le proprie caratteristiche. Per **caratteristiche del sistema** si intendono sia gli aspetti *statici*, ovvero di **dati**, che quelli *dinamici*, ovvero le **operazioni sui dati**. I requisiti vengono innanzitutto raccolti in specifiche espresse in linguaggio naturale per questo motivo, spesso ambigue e disorganizzate.

L'analisi dei requisiti consiste nel chiarimento e nell'organizzazione delle specifiche dei requisiti. Si tratta ovviamente di attività fortemente interconnesse: l'attività di analisi inizia con i primi requisiti ottenuti per poi procedere di pari passo con l'attività di raccolta. In molti casi è l'attività stessa di analisi dei requisiti che suggerisce successive attività di raccolta.

I requisiti provengono da vari tipi di fonte:

- **utenti dell'applicazione**
- **documentazione pre-esistente**
- **implementazione pre-esistente**

Risulta chiaro che, nella fase di acquisizione delle specifiche, gioca un importante ruolo l'interazione con gli utenti del sistema informativo.

Come criterio generale da seguire possiamo dire che, nel corso delle interviste, è opportuno effettuare con l'utente verifiche di comprensione e consistenza sulle informazioni che si stanno raccogliendo. Questo può essere fatto attraverso esempi (generali e relativi a casi limite) oppure richiedendo definizioni e classificazioni precise. E inoltre molto importante in questa fase cercare di individuare gli aspetti essenziali rispetto a quelli marginali e procedere per raffinamenti successivi. Partendo quindi dai principali aspetti del

problema allo studio, dei quali si ha inizialmente una conoscenza solo parziale, si procede cercando di acquisire via via maggiori dettagli.

Sappiamo bene però che il linguaggio naturale è fonte di ambiguità e fraintendimenti. E molto importante quindi effettuare una profonda analisi del testo che descrive le specifiche per filtrare le eventuali inesattezze e i termini ambigui presenti.

Proviamo a fissare alcune regole generali per ottenere una specifica dei requisiti più precisa e senza ambiguità:

- **scegliere il corretto livello di astrazione**, evitando di utilizzare termini troppo generici o troppo specifici che rendono poco chiaro un concetto
- **standardizzare la struttura delle frasi**, utilizzando sempre lo stesso stile sintattico
- **evitare frasi contorte**, preferendo semplicità e chiarezza
- **individuare sinonimi/omonimi e unificare i termini**, per evitare ambiguità
- **rendere esplicito il riferimento tra termini**, infatti può succedere che l'assenza di un contesto di riferimento renda alcuni concetti ambigui: in questi casi bisogna esplicitare il riferimento tra termini. In questo caso si deve esplicitare chiaramente a chi ci si sta riferendo, per evitare confusione

- **costruire un glossario dei termini**, infatti è molto utile, per la comprensione e la precisazione dei termini usati, definire un glossario che, per ogni termine, contenga: una breve descrizione, possibili sinonimi e altri termini contenuti nel glossario con i quali esiste un legame logico. Vediamo un esempio pratico:

Termine	Descrizione	Sinonimi	Collegamenti
Partecipante	Partecipante ai corsi. Può essere un dipendente o un professionista.	Studente	Corso, Datore
Docente	Docente dei corsi. Possono essere collaboratori esterni.	Insegnante	Corso
Corso	Corsi offerti. Possono avere varie edizioni.	Seminario	Docente, Partecipante
Datore	Datori di lavoro attuali e passati dei partecipanti ai corsi.	Posto	Partecipante

Dopo aver individuato le varie ambiguità e le imprecisioni, esse vanno eliminate sostituendo i termini non corretti con termini più adeguati. In caso di dubbio, è necessario intervistare nuovamente colui che ha fornito il dato o consultare la documentazione relativa.

A questo punto possiamo riscrivere le nostre specifiche apportando le modifiche proposte. È molto utile, in questa fase, decomporre il testo in gruppi di frasi omogenee, relative cioè agli stessi concetti. Naturalmente, accanto alle specifiche sui dati, vanno raccolte le specifiche sulle operazioni da effettuare su questi dati. Bisogna cercare di utilizzare la medesima terminologia usata per i dati (possiamo per questo far riferimento al glossario dei termini) e informarci anche sulla frequenza con la quale le varie operazioni vengono eseguite. Come vedremo, la conoscenza di questa informazione sarà determinante nella fase di progettazione logica.

Capitolo 6

Progettazione Logica

L'obiettivo della progettazione **logica** è quello di costruire uno schema logico in grado di descrivere, in maniera corretta ed efficiente, tutte le informazioni contenute nello schema Entità-Relazione prodotto nella fase di progettazione concettuale. Si hanno due necessità:

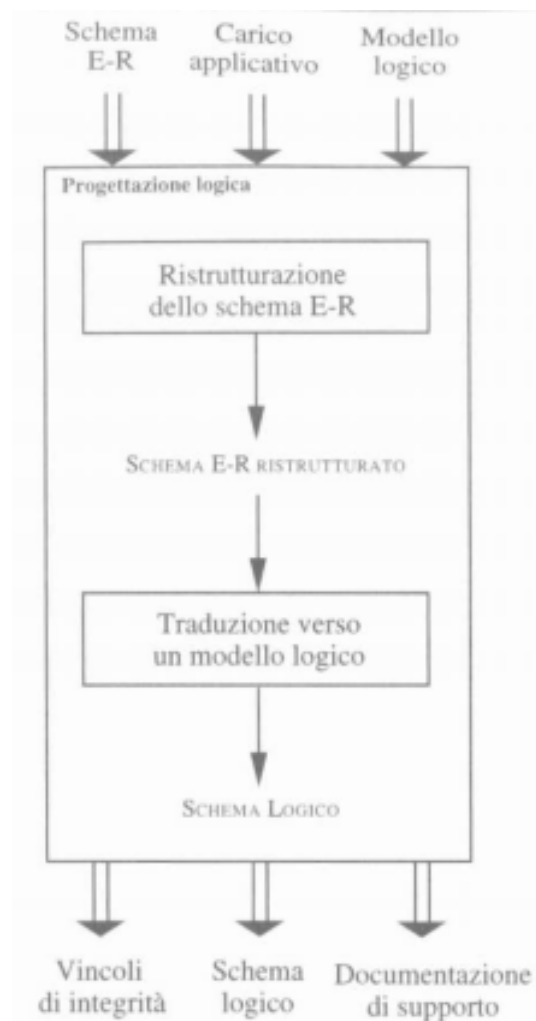
1. "semplificare" la traduzione
2. "ottimizzare" il progetto

La semplificazione dello schema si rende necessaria perché non tutti i costrutti del modello Entità-Relazione hanno una traduzione naturale nei modelli logici, vedisi la generalizzazione.

Inoltre, mentre la progettazione concettuale ha come obiettivo la rappresentazione accurata e naturale dei dati d'interesse dal punto di vista del significato che hanno nell'applicazione, la progettazione logica costituisce la base per l'effettiva realizzazione dell'applicazione e deve tenere conto, per quanto possibile, delle sue prestazioni: questa necessità può portare a una ristrutturazione dello schema concettuale che renda più efficiente l'esecuzione delle operazioni previste. Pertanto, è necessario prevedere sia un'attività di **riorganizzazione**, sia un'attività di **traduzione** (dal modello concettuale a quello logico). Nel resto di questo capitolo, dopo un breve inquadramento metodologico, presenteremo separatamente queste due attività.

Si hanno due fasi principali nella programmazione logica:

1. **ristrutturazione dello schema E-R**, che è una fase indipendente dal modello logico scelto e si basa su criteri di ottimizzazione dello schema e di semplificazione della fase successiva
2. **traduzione verso il modello logico**, che fa riferimento a uno specifico modello logico (come il modello relazionale) e può includere una ulteriore ottimizzazione che si basa sulle caratteristiche del modello logico stesso



I dati di ingresso della prima fase sono lo schema concettuale prodotto nella fase precedente e il carico applicativo previsto, in termini di dimensione dei dati e caratteristiche delle operazioni. Il risultato che si ottiene è uno

schema E-R ristrutturato, che non è più uno schema concettuale nel senso stretto del termine, in quanto costituisce una rappresentazione dei dati che tiene conto degli aspetti realizzativi. Questo schema e il modello logico scelto costituiscono i dati di ingresso della seconda fase, che produce lo schema logico della nostra base di dati. In questa seconda fase è possibile effettuare verifiche della qualità dello schema ed eventuali ulteriori ottimizzazioni mediante tecniche basate sulle caratteristiche del modello logico. Si parlerà in seguito della normalizzazione. Lo schema logico finale, i vincoli di integrità definiti su di esso e la relativa documentazione, costituiscono i prodotti finali della progettazione logica.

6.1 Analisi dell'E-R

Abbiamo detto che uno schema E-R può essere modificato per ottimizzare alcuni indici di prestazione del progetto. Parliamo di indici di prestazione e non di prestazioni perché, in realtà, le prestazioni di una base di dati non sono valutabili in maniera precisa in sede di progettazione logica, in quanto dipendenti anche da parametri fisici: dal sistema di gestione di basi di dati che verrà utilizzato e da altri fattori difficilmente prevedibili in questa fase. Si possono comunque studiare due fattori che regolano anche le prestazioni dei sistemi software:

1. **costo di un'operazione**, ovvero viene valutato in termini di numero di occorrenze di entità e associazioni che mediamente vanno visitate per rispondere a una operazione sulla base di dati; questa schematizzazione è molto forte e sarà talvolta necessario riferirsi a un criterio più fine
2. **occupazione della memoria**, ovvero viene valutato in termini dello spazio di memoria (misurato per esempio in numero di byte) necessario per memorizzare i dati descritti dallo schema

Oltre allo schema si necessiterà anche di:

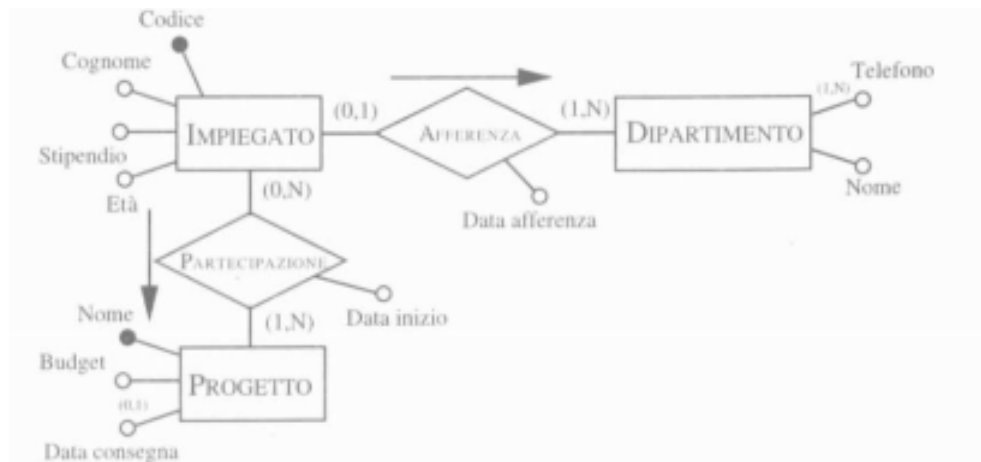
- **volume dei dati**, ovvero il numero di occorrenze di ogni entità e associazione dello schema e la dimensione di qualsiasi attributo, sia di relazione che di associazione
- **caratteristiche delle operazioni**, ovvero il tipo dell'operazione, la frequenza (ovvero il numero medio di esecuzioni in un intervallo di tempo) della stessa e i dati coinvolti, che siano associazioni e/o entità

Si ha la cosiddetta **ottanta-venti** in base alla quale l'80% del carico è generato dal 20% delle operazioni.

Il volume dei dati e le caratteristiche generali delle operazioni possono essere descritti facendo uso di tabelle. Nella tavola dei volumi vengono riportati tutti i concetti dello schema (entità e associazioni) con il volume previsto a regime. Nella tavola delle operazioni riportiamo, per ogni operazione, la frequenza prevista e un simbolo che indica se l'operazione è interattiva (I) o batch (B). Nella tavola dei volumi, il numero delle occorrenze delle associazioni dipende da due parametri: il numero di occorrenze delle entità coinvolte nelle associazioni e il numero (medio) di partecipazioni di una occorrenza di entità alle occorrenze di associazioni. Il secondo parametro dipende a sua volta dalle cardinalità delle associazioni. Vediamo un esempio:

Tavola dei volumi			Tavola delle operazioni		
Concetto	Tipo	Volume	Operazione	Tipo	Frequenza
Sede	E	10	Op. 1	I	50 al giorno
Dipartimento	E	80	Op. 2	I	100 al giorno
Impiegato	E	2000	Op. 3	I	10 al giorno
Progetto	E	500	Op. 4	B	2 a settimana
Composizione	R	80			
Afferenza	R	1900			
Direzione	R	80			
Partecipazione	R	6000			

Per ogni operazione, possiamo inoltre descrivere graficamente i dati coinvolti con uno schema di operazione che consiste nel frammento dello schema E-R interessato dall'operazione, sul quale viene disegnato il "cammino logico" da percorrere per accedere alle informazioni d'interesse. Si ha quindi un esempio di **schema di operazione**:



Si ha poi la **tavola degli accessi**, che presenta nell'ultima colonna di questa tabella viene riportato il tipo di accesso: L per accesso in lettura e S per accesso in scrittura. Questa distinzione va fatta perché, generalmente, le operazioni di scrittura sono più onerose di quelle in lettura (in quanto devono essere eseguite in modo esclusivo e possono richiedere l'aggiornamento di indici, che sono strutture ausiliarie per l'accesso efficiente ai dati): Vediamo un esempio:

Tavola degli accessi			
Concetto	Costrutto	Accessi	Tipo
Impiegato	Entità	1	L
Afferenza	Relazione	1	L
Dipartimento	Entità	1	L
Partecipazione	Relazione	3	L
Progetto	Entità	3	L

6.1.1 Ristrutturazione dell'E-R

La fase di ristrutturazione di uno schema Entità-Relazione si può suddividere in una serie di passi da effettuare in sequenza:

1. **analisi delle ridondanze**, dove si decide se eliminare o mantenere eventuali ridondanze presenti nello schema
2. **eliminazione delle generalizzazioni**, dove tutte le generalizzazioni presenti nello schema vengono analizzate e sostituite da altri costrutti
3. **partizionamento/accorpamento di entità e associazioni**, dove si decide se è opportuno partizionare concetti dello schema (entità e/o associazioni) in più concetti o, viceversa, accorpare concetti separati in un unico concetto
4. **scelta degli identificatori principali**, dove si seleziona un identificatore per quelle entità che ne hanno più di uno

Si ha quindi:

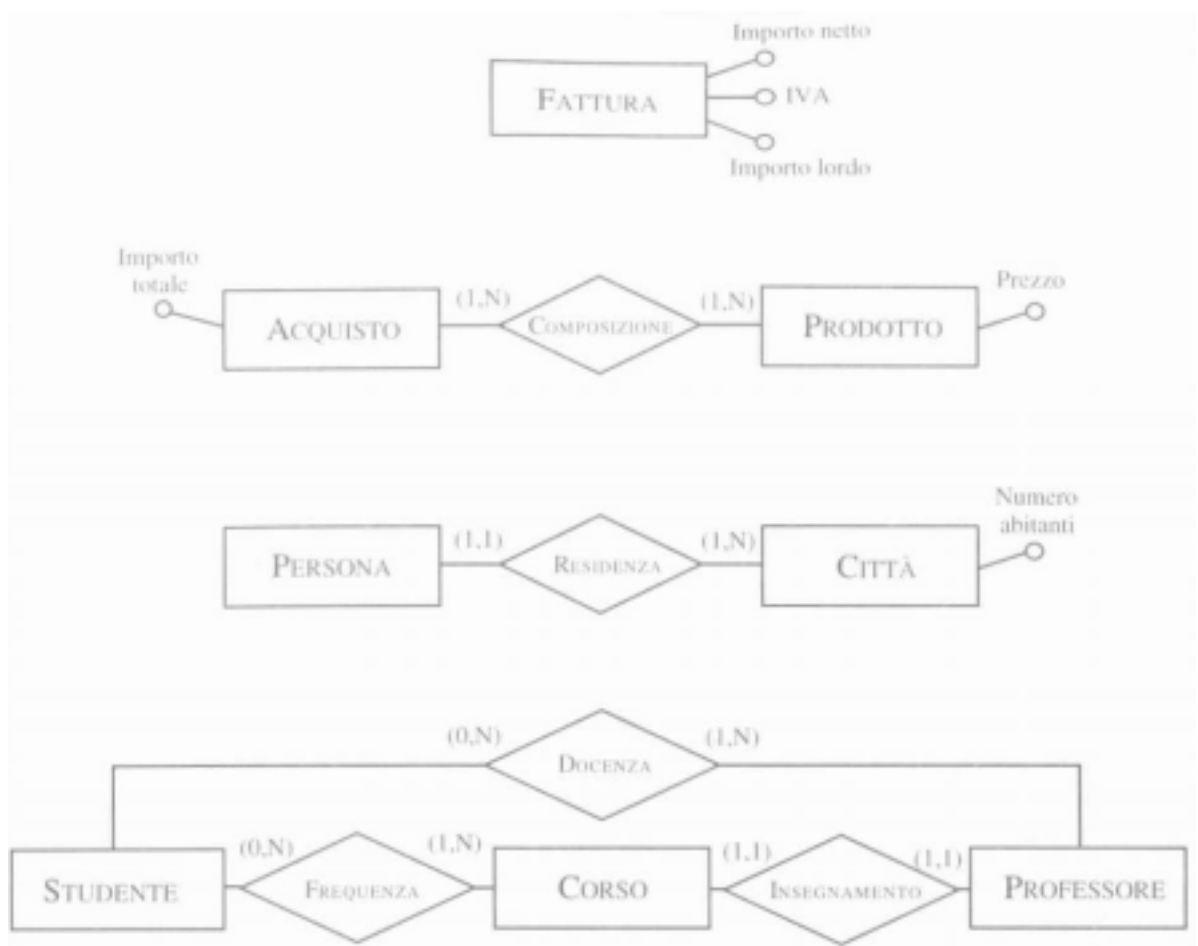


Analisi delle Ridondanze

Ricordiamo che una ridondanza in uno schema concettuale corrisponde alla presenza di un dato che può essere derivato (cioè ottenuto attraverso una serie di operazioni) da altri dati. In particolare, in uno schema Entità-Relazione si possono presentare varie forme di ridondanza. Si hanno dei casi frequenti:

- attributi derivabili, occorrenza per occorrenza, da altri attributi della stessa entità o associazione
- attributi derivabili da attributi di altre entità (o associazioni), di solito attraverso funzioni aggregative

Vediamo qualche esempio di schemi con ridondanze:



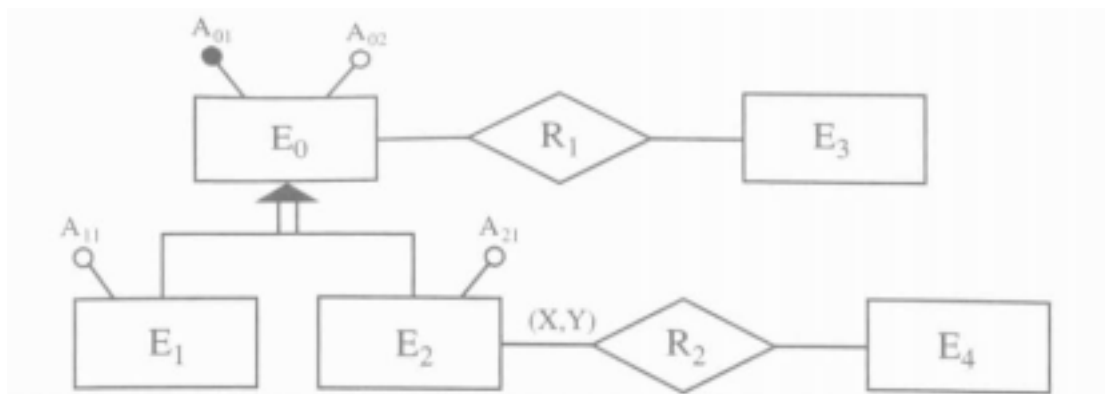
La presenza di un dato derivato presenta un vantaggio e alcuni svantaggi. Il vantaggio è una riduzione degli accessi necessari per calcolare il dato derivato, gli svantaggi sono una maggiore occupazione di memoria (che è comunque spesso un costo trascurabile) e la necessità di effettuare operazioni aggiuntive per mantenere il dato derivato aggiornato. La decisione di mantenere o eliminare una ridondanza va quindi presa confrontando costo di esecuzione delle operazioni che coinvolgono il dato ridondante e relativa occupazione di memoria, nei casi di presenza e assenza della ridondanza. Vediamo come cambiano le tavole degli accessi:

Tavole degli accessi in presenza di ridondanza			
Operazione 1			
Concetto	Costr.	Acc.	Tipo
Persona	E	1	S
Residenza	R	1	S
Città	E	1	L
Città	E	1	S
Operazione 2			
Concetto	Costr.	Acc.	Tipo
Città	E	1	L

Tavole degli accessi in assenza di ridondanza			
Operazione 1			
Concetto	Costr.	Acc.	Tipo
Persona	E	1	S
Residenza	R	1	S
Operazione 2			
Concetto	Costr.	Acc.	Tipo
Città	E	1	L
Residenza	R	5000	L

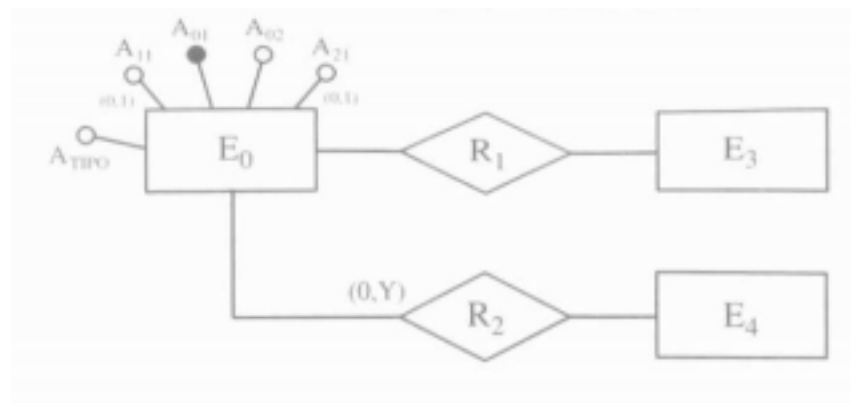
Eliminazione delle Generalizzazioni

Dato che i sistemi tradizionali per la gestione delle basi di dati non consentono di rappresentare direttamente una **generalizzazione**, risulta spesso necessario trasformare questo costrutto in altri costrutti del modello E-R per i quali esiste invece una implementazione naturale: le entità e le associazioni. Per rappresentare una generalizzazione mediante entità e associazioni abbiamo essenzialmente tre alternative possibili. Vediamo un E-R a cui togliere la generalizzazione:



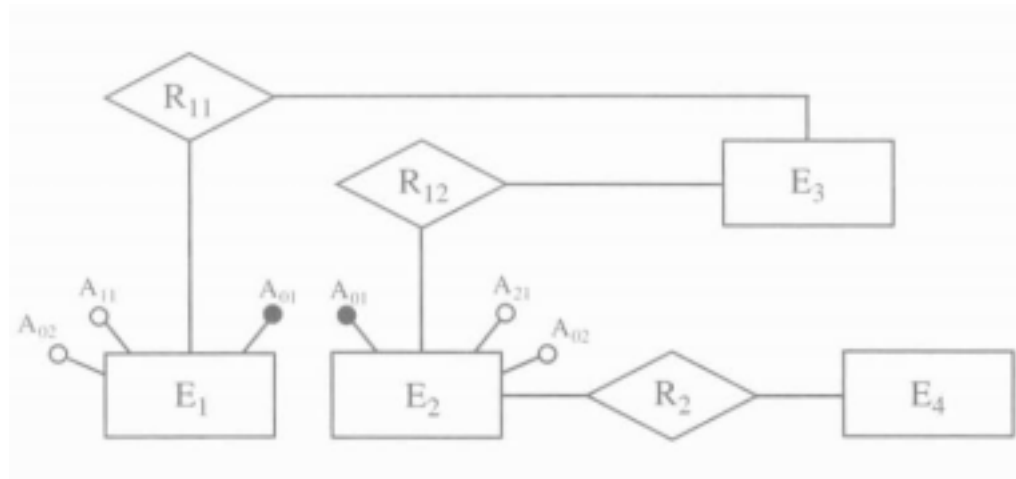
Vediamo i 3 metodi:

1. **accorpamento delle figlie della generalizzazione nel genitore**, ovvero le entità E_1, E_2 vengono eliminate e le loro proprietà (attributi e partecipazioni ad associazioni e generalizzazioni) vengono aggiunte all'entità genitore E_0 . A tale entità viene aggiunto un ulteriore attributo che serve a distinguere il "tipo" di una occorrenza di E_0 , cioè se tale occorrenza apparteneva a E_1 , a E_2 o, nel caso di generalizzazione non totale, a nessuna di esse:



è conveniente quando le operazioni non fanno molta distinzione tra le occorrenze e tra gli attributi di E_0 , E_1 ed E_2 . In questo caso infatti, anche se abbiamo uno spreco di memoria per la presenza di valori nulli, la scelta ci assicura un numero minore di accessi rispetto alle altre nelle quali le occorrenze e gli attributi sono distribuiti tra le varie entità

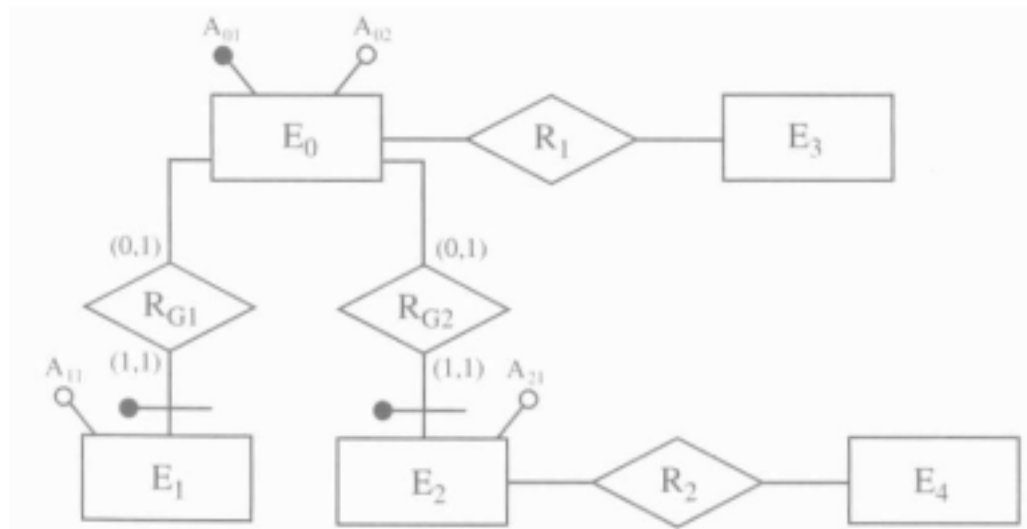
2. **accorpamento del genitore della generalizzazione nelle figlie**, ovvero l'entità genitore E_0 viene eliminata e, per la proprietà dell'ereditarietà, i suoi attributi, il suo identificatore e le relazioni a cui tale entità partecipava, vengono aggiunti alle entità figlie E_1 ed E_2 . Le relazioni R_n e R_n rappresentano rispettivamente la restrizione della relazione R_1 sulle occorrenze delle entità E_1 ed E_2 :



possibile solo se la generalizzazione è totale, altrimenti le occorrenze di E_0 che non sono occorrenze né di E_1 né di E_2 non sarebbero rappresentate. È conveniente quando ci sono operazioni che si riferiscono solo a occorrenze di E_1 , oppure di E_2 , e dunque fanno delle distinzioni tra tali entità. In questo caso abbiamo un risparmio di memoria rispetto alla prima scelta, perché, in linea di principio, gli attributi non assumono mai valori nulli. Inoltre, c'è una riduzione degli accessi rispetto alla terza scelta perché non si deve visitare E_0 per accedere ad alcuni attributi di E_1 ed E_2 .

3. **sostituzione della generalizzazione con associazioni**, ovvero la generalizzazione si trasforma in due associazioni uno a uno che

legano rispettivamente l'entità genitore con le entità figlie E_1 ed E_2 . Non ci sono trasferimenti di attributi o associazioni e le entità E_1 ed E_2 sono identificate esternamente dall'entità E_0



è conveniente quando la generalizzazione non è totale (sebbene ciò non sia necessario) e ci sono operazioni che si riferiscono solo a occorrenze di e_1, E_2E oppure di E_0 , e dunque fanno delle distinzioni tra entità figlia ed entità genitore. In questo caso abbiamo un risparmio di memoria rispetto alla prima scelta, per l'assenza di valori nulli, ma c'è un incremento degli accessi per mantenere la consistenza delle occorrenze rispetto ai vincoli introdotti.

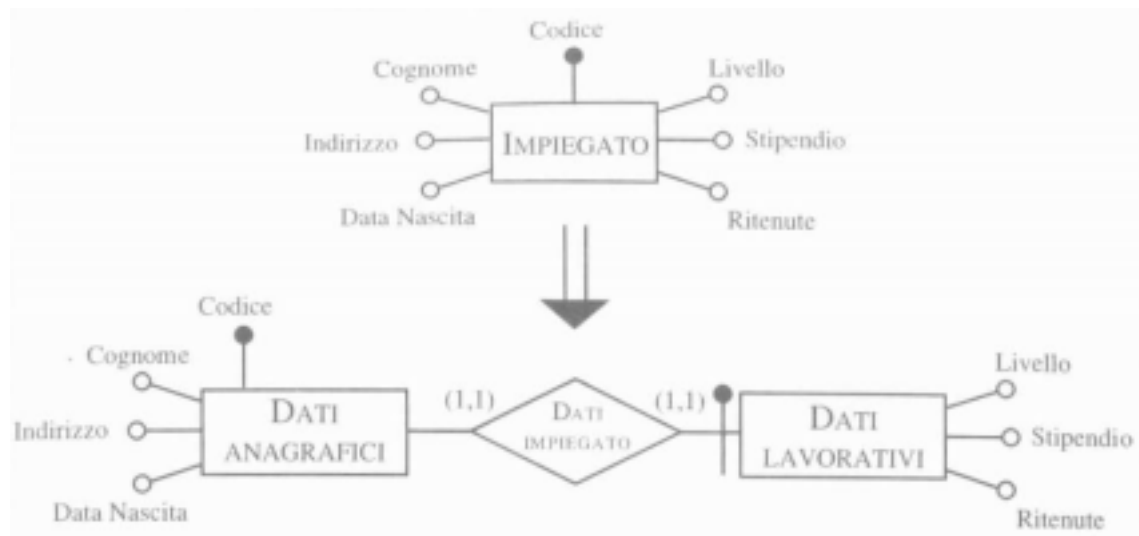
6.1.2 Elaborazione dei Concetti

Entità e associazioni in uno schema E-R possono essere partizionati o accorpati per garantire una maggior efficienza delle operazioni in base al seguente principio: gli accessi si riducono separando attributi di uno stesso concetto che vengono acceduti da operazioni diverse e raggruppando attributi di concetti diversi che vengono acceduti dalle medesime operazioni. Le stesse tecniche discusse per l'analisi delle generalizzazioni possono essere usate per prendere decisioni di questo tipo.

Partizionamento di Entità

Questa ristrutturazione è conveniente se le operazioni che coinvolgono frequentemente l'entità originaria richiedono, per un impiegato, o solo informa-

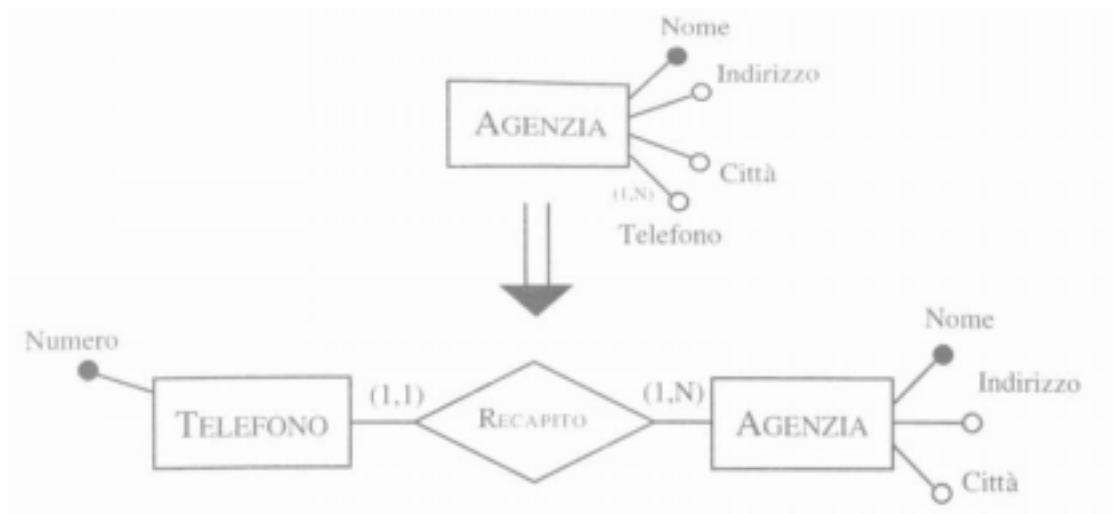
zioni di carattere anagrafico o solo informazioni relative alla sua retribuzione. Un partizionamento di questo tipo è un esempio di *decomposizione verticale* di una entità, nel senso che si suddivide il concetto operando sui suoi attributi. E comunque possibile effettuare anche delle *decomposizioni orizzontali* nelle quali la suddivisione avviene sulle occorrenze dell'entità. Vediamo un esempio di decomposizione verticale:



I partizionamenti orizzontali hanno un effetto collaterale: quello di dover duplicare tutte le associazioni a cui l'entità originaria partecipa. Questo fenomeno può avere delle ripercussioni negative sulle prestazioni del sistema. D'altra parte, i partizionamenti verticali generano entità con pochi attributi che possono essere tradotte in strutture logiche sulle quali, con un solo accesso, è possibile recuperare molti dati. Come per le generalizzazioni, anche in questo caso il semplice conteggio delle occorrenze e degli accessi, non è sempre sufficiente per scegliere la migliore alternativa possibile. Il problema del partizionamento dei dati viene ulteriormente discusso nel secondo volume, nel capitolo dedicato alle basi di dati distribuite.

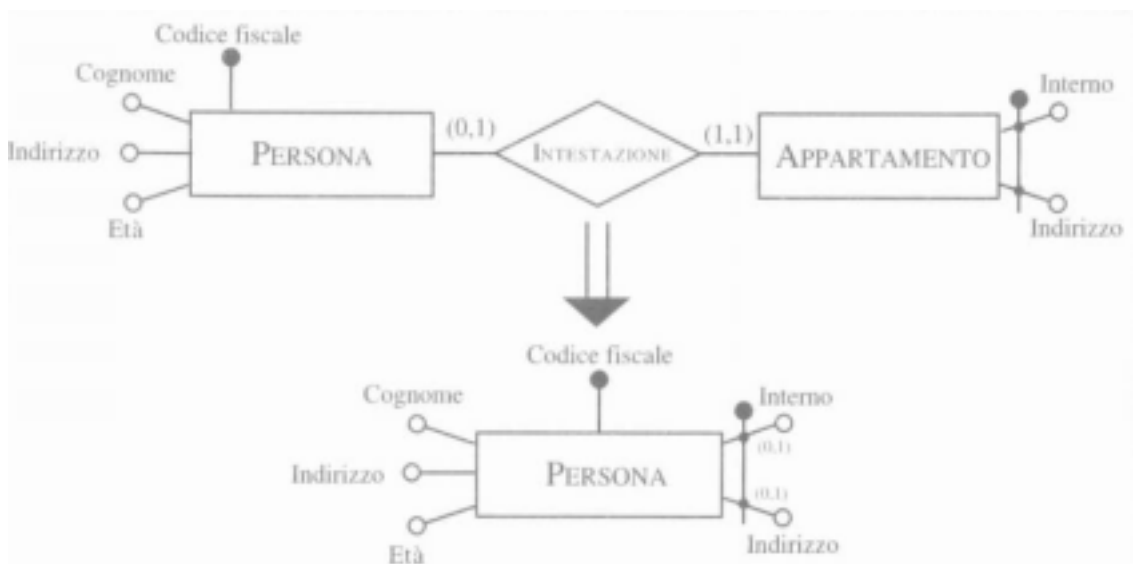
Eliminazione di Attributi Multivalore

Un particolare tipo di partizionamento che è opportuno trattare a parte è quello che riguarda l'eliminazione di attributi multivalore. Questa ristrutturazione si rende necessaria perché, come per le generalizzazioni, il modello relazionale non permette di rappresentare in maniera diretta questo tipo di attributo. La spiegazione del metodo è data dalla seguente immagine:



Accorpamento di Entità

L'accorpamento è l'operazione inversa del partizionamento. Un effetto collaterale di questa ristrutturazione è la possibile presenza di valori nulli. Vediamo un'immagine esplicativa:

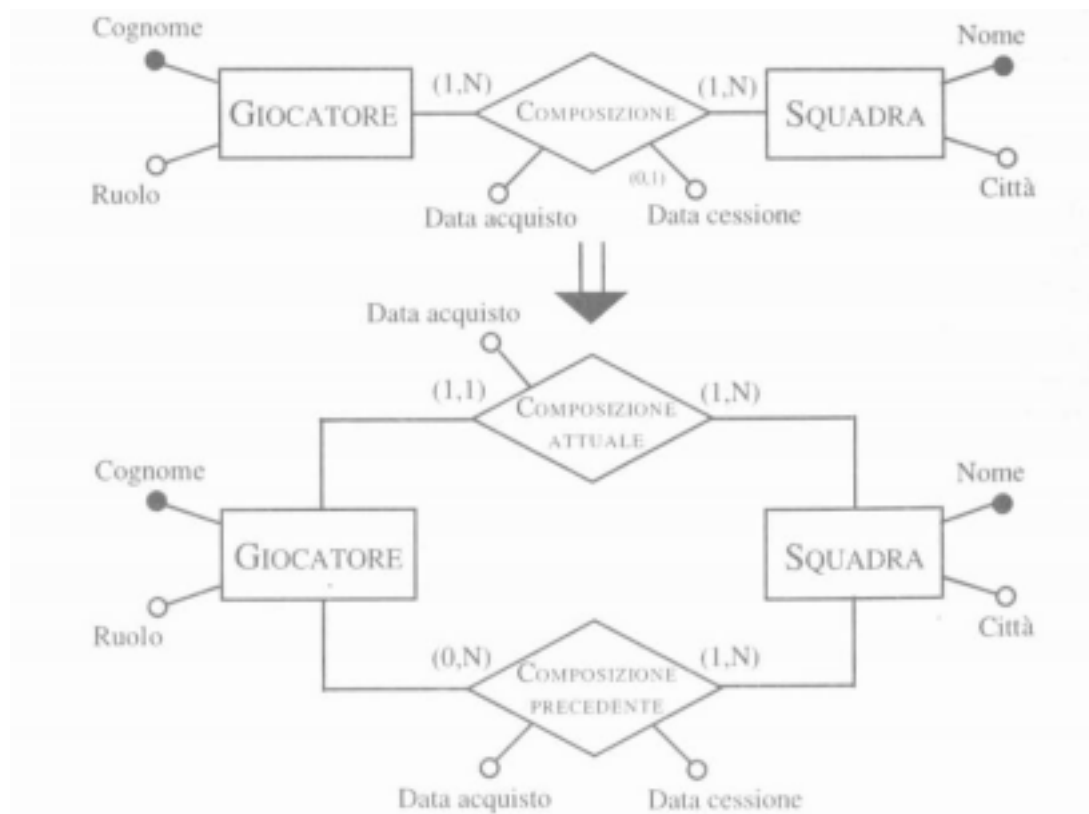


Gli accorpamenti si effettuano in genere su associazioni di tipo uno a uno, raramente su associazione uno a molti e praticamente mai su relazioni molti a molti. Questo perché gli accorpamenti di entità legate da un'associazione uno a molti o molti a molti generano ridondanze. In particolare, è facile verificare

che si possono presentare ridondanze su attributi non chiave dell'entità che partecipava all'associazione originaria con una cardinalità massima pari a N . La presenza di ridondanze può essere comunque analizzata e discussa in maniera efficace con la tecnica della normalizzazione.

Altre Tecniche

Può convenire cioè, in alcuni casi, decomporre una associazione tra due entità in due (o più) associazioni tra le medesime entità, per separare occorrenze dell'associazione originale accedute sempre separatamente e, viceversa, accorpate due (o più) associazioni tra le medesime entità (che si riferiscono però a due aspetti dello stesso concetto) in un'unica associazione, quando le relative occorrenze vengono sempre accedute contemporaneamente. Per esempio:



I problemi di partizione/accorpamento possono essere rinviati, in molti casi, alla fase di progettazione fisica. Diversi sistemi di gestione di basi di dati correnti permettono infatti di specificare cluster di strutture logiche (relazioni

nei sistemi relazionali), ovvero raggruppamenti di dati, fatti a livello fisico, che permettono l'accesso rapido a dati distribuiti su strutture logiche separate.

6.1.3 Scelta degli Identificatori Principali

La scelta degli identificatori principali è essenziale nelle traduzioni verso il modello relazionale perché in questo modello le chiavi vengono usate per stabilire legami tra dati in relazioni diverse. Inoltre, i sistemi di gestione di basi di dati richiedono generalmente di specificare una chiave primaria sulla quale vengono costruite automaticamente delle strutture ausiliarie, dette indici, per il reperimento efficiente di dati. Quindi, nei casi in cui esistono entità per le quali sono stati specificati più identificatori, bisogna decidere quale di questi identificatori verrà utilizzato come chiave primaria. I criteri di decisione per questa scelta sono i seguenti:

- gli attributi con valori nulli non possono costituire identificatori principali. Tali attributi infatti non garantiscono l'accesso a tutte le occorrenze dell'entità corrispondente, come sottolineato quando abbiamo discusso le chiavi nel modello relazionale
- un identificatore composto da uno o da pochi attributi è da preferire a identificatori costituiti da molti attributi. Questo infatti garantisce che le strutture ausiliarie create per accedere ai dati (gli indici) siano di dimensioni ridotte, permette un risparmio di memoria nella realizzazione dei legami logici tra le varie relazioni e facilita le operazioni di join
- un identificatore interno con pochi attributi è da preferire a un identificatore esterno, che magari coinvolge diverse entità. Infatti gli identificatori esterni vengono tradotti in chiavi che includono gli identificatori delle entità coinvolte nell'identificazione esterna: chiaramente in questa maniera si possono generare chiavi con molti attributi
- un identificatore che viene utilizzato da molte operazioni per accedere alle occorrenze di un'entità è da preferire rispetto agli altri. In questa maniera infatti tali operazioni possono essere eseguite efficientemente perché possono trarre vantaggio dagli indici creati automaticamente dal *DBMS*

*A questo punto, se nessuno degli identificatori candidati soddisfa tali requisiti, è possibile pensare di introdurre un ulteriore attributo all'entità: questo attributo conterrà valori speciali, detti **codici**, generati appositamente per*

identificare le occorrenze delle entità. È comunque consigliabile tenere traccia in questa fase anche degli identificatori non selezionati come principali ma che vengono utilizzati da qualche operazione per accedere ai dati. Per questi identificatori è infatti possibile definire, in sede di progettazione fisica, degli indici secondari. Gli indici secondari consentono l'accesso efficiente ai dati e possono essere usati in alternativa agli indici definiti automaticamente sugli identificatori principali.

6.2 Dall'E-R al Modello relazionale

Siamo nella seconda fase della progettazione logica e si hanno varie situazioni affrontabili.

6.2.1 Molti a Molti

Prendiamo come esempio:



La sua traduzione naturale nel modello relazionale prevede:

1. per ogni entità, una relazione con lo stesso nome avente per attributi i medesimi attributi dell'entità e per chiave il suo identificatore
2. per l'associazione, una relazione con lo stesso nome avente per attributi gli attributi dell'associazione e gli identificatori delle entità coinvolte; tali identificatori formano la chiave della relazione

Se gli attributi originali di entità o associazioni sono opzionali, i corrispondenti attributi di relazione possono assumere valori nulli. Si ottiene quindi:

```
IMPIEGATO(Matricola, Cognome, Stipendio)
PROGETTO(Codice, Nome, Budget)
PARTECIPAZIONE(Matricola, Codice, DataInizio)
```

6.2.2 Uno a Molti

Partiamo anche qui dall'E-R:



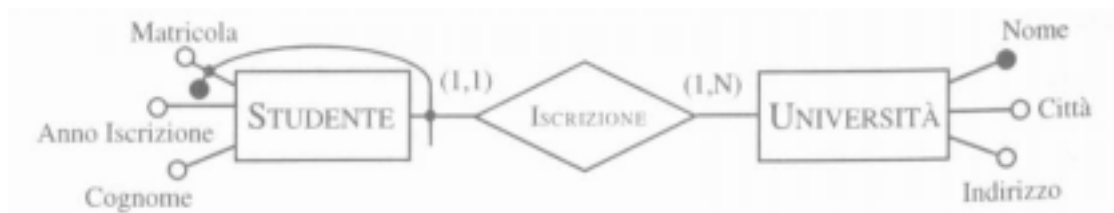
L'entità con cardinalità massima pari a 1 viene tradotta in una relazione che contiene anche gli identificatori delle altre entità coinvolte nell'associazione e non c'è bisogno di rappresentare esplicitamente l'associazione di partenza. Si ottiene quindi:

GiocATQRE(Cognome, Datanascita, Ruolo, NomeSquadra, Ingaggio)
 SQUADRA(Nome, Città, ColoriSociali)

6.2.3 Entità Esterna

Le entità con identificatori esterni danno luogo a relazioni con chiavi che includono gli identificatori delle entità "identificanti".

Partiamo dall'E-R:



Si ottiene:

STUDENTE(Matricola, NomeUniversità, Cognome, AnnoIscrizione)
 UNIVERSITÀ(Nome, Città, indirizzo)

Come si può vedere, rappresentando l'identificatore esterno si rappresenta direttamente anche l'associazione tra le due entità. Ricordiamo infatti che le entità identificate esternamente partecipano all'associazione sempre con una cardinalità minima e massima pari a uno. Questo tipo di traduzione è valido indipendentemente dalla cardinalità con cui l'altra entità partecipa all'associazione.

6.2.4 Uno a Uno

Si hanno varie possibilità. parto dall'E-R:



Se si hanno associazioni uno a uno con partecipazioni obbligatorie per entrambe le entità si hanno due possibilità:

1. DIRETTORE(Codice, Cognome, Stipendio,
DipartimentoDiretto, InizioDirezione)
DIPARTIMENTO(Nome, Telefono, Sede)
2. DIRETTORE(Codice, Cognome, Stipendio)
DIPARTIMENTO(Nome, Telefono, Sede, Direttore,
InizioDirezione)

Consideriamo infine il caso in cui entrambe le entità hanno partecipazione opzionale e si ha:

IMPIEGATO(Codice, Cognome, Stipendio)
DIPARTIMENTO(Nome, Telefono, Sede)
DIREZIONE(Direttore, Dipartimento, DataInizioDirezione)

Diciamo quindi che la soluzione con tre relazioni è da prendere in considerazione solo se il numero di occorrenze dell'associazione è molto basso rispetto alle occorrenze delle entità che partecipano all'associazione. In questo caso, c'è infatti il vantaggio di evitare la presenza di molti valori nulli.

6.2.5 Tabella Riassuntiva

Tipologia	Concetto iniziale	Risultati possibili
Associazione binaria molti a molti		$E_1(\underline{A_{E11}}, A_{E12})$ $E_2(\underline{A_{E21}}, A_{E22})$ $R(\underline{A_{E11}, A_{E21}}, A_R)$
Associazione ternaria molti a molti		$E_1(\underline{A_{E11}}, A_{E12})$ $E_2(\underline{A_{E21}}, A_{E22})$ $E_3(\underline{A_{E31}}, A_{E32})$ $R(\underline{A_{E11}, A_{E21}, A_{E31}}, A_R)$
Associazione uno a molti con partecipazione obbligatoria		$E_1(\underline{A_{E11}}, A_{E12}, A_{E21}, A_R)$ $E_2(\underline{A_{E21}}, A_{E22})$
Associazione uno a molti con partecipazione opzionale		$E_1(\underline{A_{E11}}, A_{E12})$ $E_2(\underline{A_{E21}}, A_{E22})$ $R(\underline{A_{E11}}, A_{E21}, A_R)$ Oppure: $E_1(\underline{A_{E11}}, A_{E12}, A_{E21}^*, A_R^*)$ $E_2(\underline{A_{E21}}, A_{E22})$
Associazione con identificatore esterno		$E_1(\underline{A_{E12}}, A_{E21}, A_{E11}, A_R)$ $E_2(\underline{A_{E21}}, A_{E22})$

Tipologia	Concetto iniziale	Risultati possibili
Associazione uno a uno con partecipazione obbligatoria per entrambe le entità		$E_1(\underline{A_{E11}}, A_{E12}, A_{E21}, A_R)$ $\underline{E_2(A_{E21}, A_{E22})}$ <p>Oppure:</p> $E_2(\underline{A_{E21}}, A_{E22}, A_{E11}, A_R)$ $\underline{E_1(A_{E11}, A_{E12})}$
Associazione uno a uno con partecipazione opzionale per una entità		$E_1(\underline{A_{E11}}, A_{E12}, A_{E21}, A_R)$ $\underline{E_2(A_{E21}, A_{E22})}$
Associazione uno a uno con partecipazione opzionale per entrambe le entità		$E_1(\underline{A_{E11}}, A_{E12})$ $E_2(\underline{A_{E21}}, A_{E22}, A_{E11}^*, A_R^*)$ <p>Oppure:</p> $E_1(\underline{A_{E11}}, A_{E12}, A_{E21}^*, A_R^*)$ $\underline{E_2(A_{E21}, A_{E22})}$ <p>Oppure:</p> $E_1(\underline{A_{E11}}, A_{E12})$ $E_2(\underline{A_{E21}}, A_{E22})$ $R(\underline{A_{E11}}, \underline{A_{E21}}, A_R)$

Capitolo 7

Algebra Relazionale

L'algebra relazionale è un linguaggio procedurale, basato su concetti di tipo algebrico. Sostanzialmente, esso è costituito da un insieme di operatori, definiti su relazioni e che producono ancora relazioni come risultati. In questo modo, è possibile costruire espressioni che coinvolgono più operatori, allo scopo di formulare interrogazioni anche complesse.

7.1 Unione, Intersezione e Differenza