

Basi di Dati

90107 - 09CFU

Disclaimer: questi appunti sono un mix,
molto viene dal libro, “Basi di dati VI edizione”
P.Atzeni, S.Ceri, P.Fraternali
in parte vengono dalle slide fornite
dal Prof Montesi a lezione.

(agg. a Sett 2025) Sono gratuitamente pubblicati su
<https://dynamik.vercel.app/basi-di-dati/appunti?from=informatica>

Se trovate errori fatemelo pure sapere scrivendo a

 alberto.zuccari@studio.unibo.it oppure su

 [telegram @b3rt0nes](#)

SOMMARIO

1 Introduzione.....	4
2 Il modello relazionale.....	8
2.1 Strutture	8
2.2 Vincoli di integrità.....	12
3 Algebra e calcolo relazionale	16
3.2 Calcolo relazionale.....	26
3.3 Datalog.....	30
4 SQL: concetti base.....	32
4.1 Definizione dei dati.....	32
4.2 Tipi di dati (attributi)	33
4.3 Vincoli di tabella	33
4.4 Operazioni in SQL: Select.....	37
4.5 Operatori di aggiornamento.....	51
5 SQL Avanzato	52
5.1 Caratteristiche evolute di definizione dei dati	52
5.2 Funzioni scalari	55
5.3 Controllo dell'accesso.....	56
5.4 Transazioni.....	58
6 Modellazione concettuale dei dati	60
6.1 Introduzione alla progettazione	60
6.2 Il modello Entità-Relazione.....	62
7 Progettazione concettuale.....	70
7.1 Raccolta e analisi dei requisiti	70
7.2 Rappresentazione concettuale di dati.....	73
7.3 Strategie di progetto	79
7.4 Qualità di uno schema concettuale.....	81
7.5 Una metodologia generale	81
8 Progettazione logica	82
8.1 Fasi della progettazione logica	82
8.2 Analisi delle prestazioni su schemi E-R.....	82
8.3 Ristrutturazione di schemi E-R	85
9 Normalizzazione.....	89
9.1 Ridondanze e anomalie	89
9.2 Dipendenze funzionali	90
9.3 Forma normale di Boyce e Codd	91

1 INTRODUZIONE

Database: un set di dati organizzato che supporta lo svolgimento di attività (di un ente, un’azienda, un ufficio, una persona)

L’informazione viene espressa come **dato** e ogni dato è di per sé inutile se manca la sua **interpretazione**

DBMS (DataBase Management Systems): sono meccanismi di gestione di collezioni di dati, come mySQL, Oracle, Access, ...

I database sono:

- Grandi
 - o Memorie che vanno dagli 1/5 TB (dati transazionali) ai 500/800 TB (dati scientifici)
- Persistenti
 - o La loro longevità non dipende dai processi dei computer che ne utilizzano i dati
- Condivisi
 - o Le grandi aziende sono suddivise in aree diverse (i dipartimenti dell’unibo)
 - E ogni area ha un (sotto)sistema (non necessariamente isolato dal principale)
 - o Conseguenze: sono necessari
 - **meccanismi di autorizzazione** (attività differenti lavorano su dati condivisi)
 - **controlli di autorizzazione** (più utente accedono a dati condivisi)

Problemi:

1. **Ridondanza**...
 - Gli stessi dati che appaiono più volte
2. ... che può causare **inconsistenza**
 - ovvero descrizioni che non corrispondono

I Database garantiscono:

- **Privacy**
 - o Si possono definire meccanismi di autorizzazione
 - A può leggere tutto e modificare solo i dati sul ricevimento
 - B può leggere di dati X e modificare Y
- **Affidabilità**
 - o Resistenza a malfunzionamenti
 - o Deve essere conservato a lungo termine
 - o Fondamentale la gestione delle **transazioni**

Transazione: insieme di operazioni da considerare indivisibile (“atomico”), che sono corrette anche su di un sistema concorrente con effetti definitivi

Le transazioni sono:

- Atomiche
 - o La sequenza di operazioni sulla base di dati viene eseguita per intero o per niente
 - (trasferimento di fondi da un conto A a un conto B, o si prelevano da A o si versano su B.)
- Concorrenti
 - o L’effetto delle transazioni deve essere coerente
 - (se due assegni emessi sullo stesso conto corrente vengono incassati contemporaneamente si deve evitare di trascurarne uno)

Gli effetti delle transazioni sono permanenti, ovvero la conclusione di una transazione corrisponde ad un impegno (**commit**) a mantenere traccia del risultato in modo definitivo.

I DBMS devono essere:

- # Efficienti
 - Cercano di utilizzare al meglio le risorse e lo spazio di memoria del DB
- # Efficaci
 - Cercano di rendere produttive le attività dei loro utilizzatori, offrendo funzionalità articolate, potenti e flessibili

1.1.1 Schemi e istanze

Ogni software contiene una descrizione interna della struttura dei file che andranno processati. Ne consegue che più software possono avere più descrizioni dei dati, e ciò genera incoerenza con la rappresentazione dei dati.

I dati vengono descritti su più livelli di astrazione

- I dati non dipendono dalla loro rappresentazione fisica
 - I programmi utilizzano una rappresentazione di alto livello, indipendentemente dalle rappresentazioni dei dati di basso livello

Quindi un cambiamento a basso livello non richiede un cambiamento nei programmi

Introduciamo il concetto di **data model**:

3. Un set di costrutti che vengono usati per organizzare e descrivere il comportamento dei dati
4. È un componente cruciale, provvede alle strutture per i dati
5. Il data model fornisce alcuni costruttori di data type di default
6. Il modello relazionale fornisce la relazione costruttore, consentendo di definire un insieme di record omogenei

Questo è uno **schema** {

PROGRAMMA			
Corso	Docente	Aula	Ora
Analisi 1	Luigi Neri	N1	08:00
Basi di Dati	Pier Rossi	N2	09:45
Chimica	Nicola Mori	N1	09:45
Fisica 1	Mario Bruni	N1	11:45
Fisica 2	Mario Bruni	N3	09:45
Informatica	Pier Rossi	N3	08:00

} Queste sono **istanze**

In ogni database, ci sono

- Lo schema: non cambia nel tempo, descrive la struttura della tabella
es. le intestazioni delle tabelle
- L'istanza: i valori attuali che possono cambiare
es. il "corpo" di ciascuna tabella

Ci sono due tipi principali di modelli

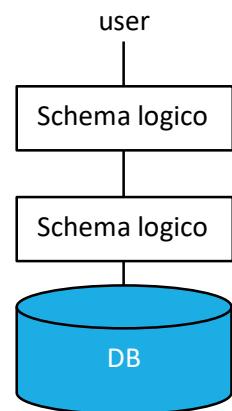
7. Modelli logici
Sono usati nei DBMS esistenti per l'organizzazione dei dati, vengono utilizzati dai programmi e sono indipendenti dalle strutture fisiche;
Per esempio, il modello relazionale, quello reticolare, gerarchico, a oggetti.
8. Modelli concettuali
Permettono di rappresentare i dati in modo indipendente da ogni sistema, cercano di descrivere i concetti del mondo reale e sono utilizzati nelle fasi preliminari della progettazione;
Il più diffuso è il modello Entity-Relationship.

1.1.2 Livelli di astrazione nei DBMS

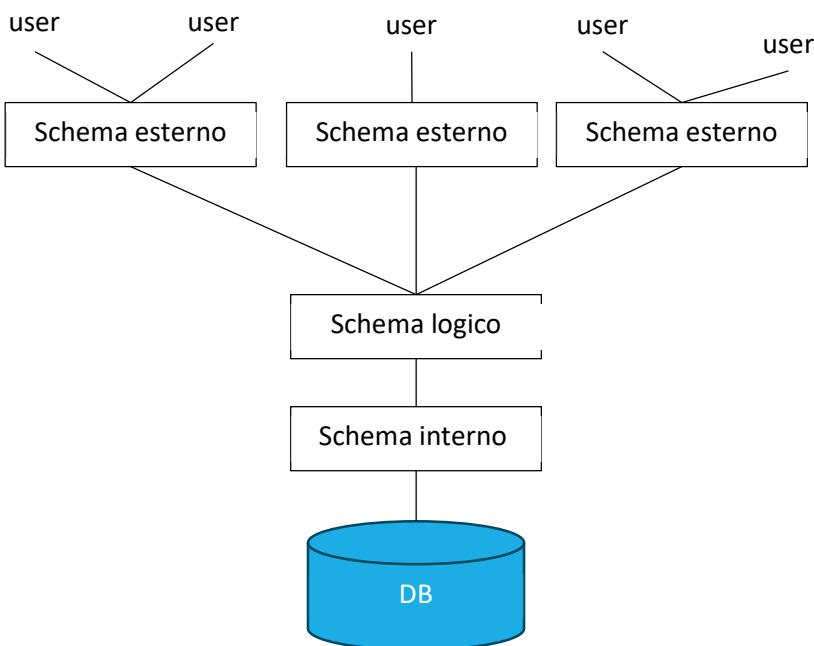
Schema Logico: descrive la struttura del database tramite un modello di dati logico (struttura della tabella)

Schema interno (o fisico): rappresenta lo schema logico al livello di contenuto tramite le specifiche strutture dati effettive (file, record, puntatori)

I due livelli schematici sono indipendenti l'uno dall'altro, noi lavoreremo sullo schema logico.



Architettura standard (ANSI/SPARC) a tre livelli per i DBMS



Schema logico:

Costituisce una descrizione dell'intera base di dati per mezzo del modello logico adottato dal DBMS

Schema interno:

Costituisce la rappresentazione dello schema logico per mezzo di strutture fisiche di memorizzazione.

Schema esterno:

Costituisce la descrizione di una porzione della base di dati di interesse, per mezzo del modello logico.

Uno schema esterno può prevedere organizzazioni dei dati diverse rispetto a quelle utilizzate nello schema logico, che riflettono il punto di vista di un particolare utente (o insieme di utenti).

Pertanto, è possibile associare a uno schema logico vari schemi esterni.

1.1.3 Indipendenza dei dati

È la conseguenza di avere diversi livelli

L'accesso ai dati è fornito dal livello esterno (che può coincidere con quello logico)

9. L'**indipendenza fisica** consente di interagire con il DBMS in modo indipendente dalla struttura fisica dei dati. In base a questa proprietà è possibile modificare le strutture fisiche senza influire sulle descrizioni dei dati ad alto livello e quindi sui programmi che utilizzano i dati stessi.
10. L'**indipendenza logica** consente di interagire con il livello esterno della base di dati in modo indipendente dal livello logico. Per esempio, è possibile aggiungere uno schema esterno in base alle esigenze di un nuovo utente oppure modificare uno schema esterno senza dover modificare lo schema logico, mantenendo inalterate le strutture esterne di interesse per l'utente.

2 IL MODELLO RELAZIONALE

2.1 STRUTTURE

Il *modello relazionale* si basa su due concetti, **relazione** e **tabella**, di natura diversa ma facilmente riconducibili l'uno all'altro.

11. La nozione di *relazione* proviene dalla matematica, in particolare dalla teoria degli insiemi.
12. Il concetto di *tabella* è semplice e intuitivo.

Il modello relazionale risponde al requisito dell'indipendenza dei dati, che prevede una distinzione nella descrizione dei dati fra il livello fisico e il livello logico.

Serve precisare però che il termine *relazione* viene utilizzato in 3 accezioni che nei dettagli sono diverse:

1. **Relazione matematica**, secondo la definizione della teoria degli insiemi. Da questa nozione derivano le altre due;
2. **Relazione** secondo la definizione del modello relazionale che presenta alcune differenze rispetto a quella della teoria degli insiemi;
3. **Relazione** come traduzione di *Relationship*, costrutto del modello concettuale *Entità-relazione* utilizzato per descrivere legami tra entità del mondo reale.

2.1.1 Relazioni e tabelle

In matematica, dati due insiemi D_1 e D_2 , si chiama **prodotto cartesiano** di D_1 e D_2 , in simboli $D_1 \times D_2$, l'insieme delle coppie ordinate (v_1, v_2) tali che $v_1 \in D_1$ e $v_2 \in D_2$.

Per esempio, dati gli insiemi $A = \{1, 2, 4\}$ e $B = \{a, b\}$, il prodotto cartesiano $A \times B$ è costituito dall'insieme di tutte le possibili coppie in cui il primo elemento appartiene ad A e il secondo a B , quindi avremo 6 coppie

$$\{(1, a), (1, b), (2, a), (2, b), (4, a), (4, b)\}$$

1	a
1	b
2	a
2	b
4	a
4	b

Una **relazione logica (o matematica)** sugli insiemi D_1 e D_2 (**domini** della relazione) è un sottoinsieme di $D_1 \times D_2$.

$$r \subseteq D_1 \times D_2$$

1	a
1	b
4	b

Le relazioni possono essere rappresentate graficamente sotto forma tabellare, le due tabelle qui sopra rappresentano una il prodotto cartesiano e la relazione matematica.

D_1, \dots, D_n non sono per forza insiemi differenti

Il *prodotto cartesiano* di più insiemi $D_1 \times \dots \times D_n$ è definito come l'insieme di tutte le tuple (d_1, \dots, d_n) tali che $d_1 \in D_1, \dots, d_n \in D_n$

La *relazione logica* su D_1, \dots, D_n è un sottoinsieme di $D_1 \times \dots \times D_n$, dove D_1, \dots, D_n sono i **domini** della relazione. Questo sottoinsieme è un insieme di tuple ordinate (d_1, \dots, d_n) tali che $d_1 \in D_1, \dots, d_n \in D_n$

Una *relazione* è un insieme:

- Non c'è un ordine tra le tuple
- Le tuple sono tutte distinte tra loro
- Ogni n-upla è ordinata, l' i -esimo valore proviene dall' i -esimo dominio.

Vediamo un esempio:

Partite \subseteq string \times string \times int \times int			
Barca	Bayern	3	1
Bayern	Real	2	0
Barca	Psg	0	2
Psg	Real	0	1

Ogni dominio appare con due **ruoli** distinti che potrebbero essere distinti dalle sue (del dominio) posizioni
Ogni nome univoco nella tabella (**attributo**) è associato a un dominio. L'attributo fornisce il “ruolo” del dominio

Squadra di casa	Squadra ospiti	Reti casa	Reti ospiti
Barca	Bayern	3	1
Bayern	Real	2	0
Barca	Psg	0	2
Psg	Real	0	1

Indichiamo con D l'insieme dei domini e specifichiamo la corrispondenza fra attributi e domini, nell'ambito di una relazione, per mezzo di una funzione $\text{dom} : X \rightarrow D$, che associa a ciascun attributo $a \in X$ un dominio $\text{dom}(a) \in D$.

Diciamo che una **tupla** su un insieme di attributi X è una funzione t che associa a ciascun attributo $a \in X$ un valore del dominio $\text{dom}(a)$.

Una **relazione** su X è un insieme di tuple su X . La differenza fra la def classica e questa sta nella def di tuple; nella relazione matematica abbiamo n -uple i cui elementi sono individuati per posizione, mentre nelle tuple gli elementi sono individuati per mezzo degli attributi, cioè con una tecnica non posizionale.

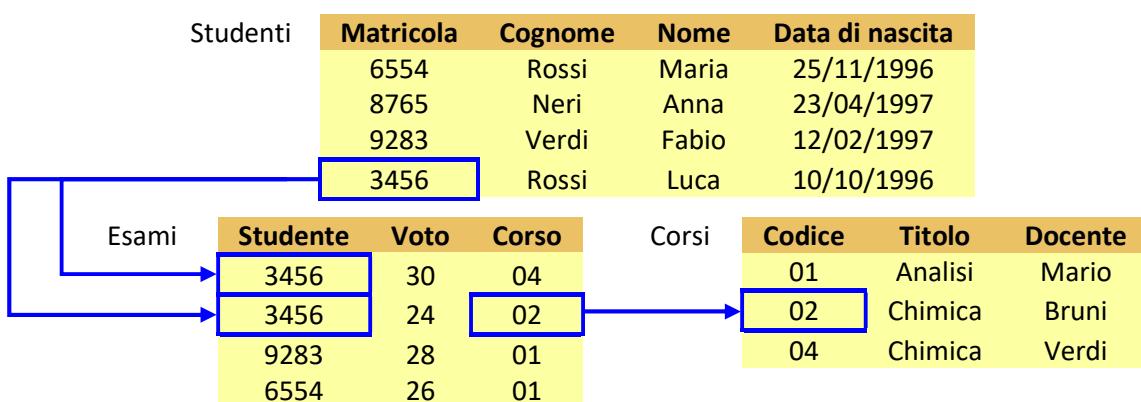
Ora introduciamo una notazione molto utilizzata.

Se t è una tupla su X e $a \in X$, allora $t[a]$ (o $t.a$) indica il valore di t su a . Per esempio, se t è la prima tupla della relazione della tabella qua sopra, allora possiamo dire che:

$$t[\text{Squadra di casa}] = \text{Barca}$$

2.1.2 Relazioni e basi di dati

Una relazione può essere utilizzata per organizzare dati rilevanti nell'ambito di un'applicazione di interesse. Di solito non è sufficiente una singola relazione: una base di dati in generale è costituita da più relazioni, le cui tuple contengono valori comuni per stabilire corrispondenze



Una delle caratteristiche fondamentali del modello relazionale viene spesso indicata dicendo che esso è “basato su valori”: i riferimenti fra relazioni diverse sono rappresentati per mezzo di valori che compaiono nelle tuple.

Riassumiamo le definizioni relative al modello relazionale:

- › **Schema di relazione:** si indica con $R(X)$ ed è costituito da un simbolo R detto *nome della relazione*, e da un insieme di attributi $X = \{A_1, \dots, A_n\}$. A ciascun attributo è associato un dominio;
- › **Schema di base di dati:** è un insieme di diversi schemi di relazione

$$R = \{R_1(X_1), R_2(X_2), \dots, R_n(X_n)\}$$

I nomi di relazione servono per distinguere le varie relazioni nella base di dati;

- › **(istanza di) Relazione:** su uno schema $R(X)$ è un insieme r di tuple su X , si scrive $r(X)$
- › **(istanza di) Base di Dati:** su uno schema $R = \{R_1(X_1), R_2(X_2), \dots, R_n(X_n)\}$

Per semplificare, possiamo dire che lo schema della base di dati delle tabelle di prima è definito così:

$$R = \{ \text{Studenti}(\text{Matricola}, \text{Cognome}, \text{Nome}, \text{Data di nascita}), \\ \text{Esami}(\text{Studente}, \text{Voto}, \text{Corso}), \\ \text{Corsi}(\text{Codice}, \text{Titolo}, \text{Docente}) \}$$

Notiamo come è possibile, secondo la definizione di relazione, l'esistenza di relazioni con un singolo attributo. Una relazione con un singolo attributo può avere senso in basi di dati su più relazioni, in cui la relazione con un singolo attributo contiene valori che appaiono come attributi di un'altra relazione.

Per esempio, nella relazione Studenti si può utilizzare un'altra relazione sull'attributo **matricola** per indicare gli studenti lavoratori

Matricola	Cognome	Nome	Data di nascita	Matricola
6554	Rossi	Maria	25/11/1996	6554
8765	Neri	Anna	23/04/1997	3456
9283	Verdi	Fabio	12/02/1997	
3456	Rossi	Luca	10/10/1996	

Vediamo ora un esempio più articolato che ci permette di capire come il modello relazionale permetta di rappresentare l'informazione strutturata in modo articolato.

“DA MARIO”			“DA MARIO”			“DA MARIO”		
Ricevuta n. 1357			Ricevuta n. 2334			Ricevuta n. 3002		
Del 5/5/2021			Del 4/7/2021			Del 4/8/2021		
3	coperti	6,00	2	coperti	4,00	3	coperti	6,00
2	antipasti	12,00	1	antipasti	6,00	2	antipasti	14,00
3	primi	27,00	2	primi	15,00	3	primi	20,00
2	bisteccche	36,00	2	orate	50,00	1	orate	25,00
			2	caffè	3,00	1	caprese	8,00
Totale		81,00	Totale			Totale		

Queste tre ricevute hanno una struttura che prevede (a parte ciò che è scritto in grassetto) alcune informazioni fisse (numero, data, totale) e un numero di righe variabile, ognuna riservata a un insieme di portate omogenee (con quantità, descrizione, importo complessivo). Non è possibile rappresentare l'insieme delle ricevute in un'unica relazione perché le relazioni non hanno una struttura fissa e le ricevute non hanno un numero di righe prefissato.

Possiamo però rappresentare le relative informazioni per mezzo di due relazioni:

Ricevute

Numero	Data	Totale
1357	5/5/2021	81,00
2334	4/7/2021	78,00
3002	4/8/2021	76,00

La relazione *Ricevute* contiene i dati presenti una sola volta in ciascuna ricevuta (numero, data e totale). La relazione *Dettaglio* contiene le varie righe di ciascuna ricevuta (con quantità, descrizione e importo complessivo), associate alla ricevuta stessa tramite il relativo numero.

Notiamo che questa base di dati (Ricevute e Dettaglio) rappresenta correttamente le ricevute solo se sono vere le due condizioni:

13. Non interessa tenere traccia dell'ordine in cui le righe compaiono in ciascuna ricevuta
14. In una ricevuta non compaiono due righe uguali

Dettaglio

Numero	Quantità	Descrizione	Costo
1357	3	Coperti	6,00
1357	3	coperti	6,00
1357	2	antipasti	12,00
1357	3	primi	27,00
1357	2	bistecche	36,00
2334	2	coperti	4,00
2334	1	antipasti	6,00
2334	2	primi	15,00
2334	2	orate	50,00
2334	2	caffè	3,00
3002	3	coperti	6,00
3002	2	antipasti	14,00
3002	3	primi	20,00
3002	1	orate	25,00
3002	1	caprese	8,00
3002	2	caffè	3,00

Qualora queste due condizioni non fossero soddisfatte si potrebbe risolvere il problema aggiungendo un attributo, che indica la posizione della riga sulla ricevuta, così facendo si potrebbe ricostruire esattamente le ricevute originali.

2.1.3 Informazione incompleta e valori nulli

La struttura del modello relazionale impone un certo grado di rigidità, poiché le informazioni devono essere rappresentate come tuple di dati omogenee; quindi, non tutte le tuple vengono accettate, le tuple valide devono rispettare lo schema. Per esempio, in una relazione sullo schema

Persone(Cognome, Nome, Indirizzo, Telefono)

Il valore dell'attributo **Telefono** potrebbe non essere disponibile per tutte le tuple.

Potremmo utilizzare un valore prefissato per indicare l'assenza. Parlando di numeri di telefono potremmo usare lo 0, ma in generale questa scelta non è soddisfacente per due motivi:

1. Richiede l'esistenza di un valore del dominio mai utilizzato per valori significativi
2. L'utilizzo di valori del dominio può generare confusione, la distinzione tra valori *veri* e valori fittizi è nascosta, e quindi i programmi che accedono alla base di dati devono tenerne conto.

Per rappresentare in modo semplice la non disponibilità di valori viene esteso il concetto di relazione, prevedendo che una tupla possa assumere su ciascun attributo o un valore del dominio o un valore speciale, detto *valore nullo*, che denota l'assenza di informazione. Esso è un valore aggiuntivo rispetto a quelli del dominio e ben distinto da essi. Nelle tabelle scriveremo “NULL”.

Vediamo tre differenti di motivazioni per cui possiamo avere dei valori nulli:

Città	Indirizzo Prefettura
Roma	IV Novembre
Firenze	NULL
Tivoli	NULL
Olbia-Tempio	NULL

- Valore *sconosciuto*; Firenze ha certamente una prefettura, ma per ora non ne sappiamo l'indirizzo
- Valore *inesistente*; Tivoli non è mai stato capoluogo di provincia; quindi, non ha mai avuto una prefettura (e quindi un indirizzo)
- Valore *senza informazione*; La provincia di Olbia-Tempio è stata istituita nel 2001 e abolita nel 2016, non sappiamo se la prefettura è mai stata costruita

Riprendiamo un esempio di una base di dati vista in precedenza ma con dei valori nulli

Studenti	Matricola	Cognome	Nome	Data di nascita			
	6554	Rossi	Maria	NULL			
	NULL	Neri	Anna	23/04/1997			
	NULL	Verdi	Fabio	12/02/1997			
Esami	Studente	Voto	Corso	Corsi	Codice	Titolo	Docente
	3456	30	04		01	Analisi	Mario
	NULL	24	NULL		02	Chimica	NULL
	9283	28	NULL		NULL	Chimica	Verdi

Il valore NULL sulla data di nascita della prima tupla della relazione “Studenti” è tutto sommato ammissibile, perché si può pensare che l’informazione non sia *essenziale*. Mentre un valore nullo sul numero di matricola o sul codice di un corso genera problemi maggiori perché questi valori sono utilizzati per stabilire correlazioni fra tuple di relazioni diverse.

È evidente quindi come sia necessario moderare la presenza di valori nulli nelle relazioni.

In genere, quando si definisce una relazione, è possibile specificare che i nulli siano ammessi solo su alcuni attributi e non su altri.

2.2 VINCOLI DI INTEGRITÀ

Molto spesso non è vero che qualsiasi insieme di tuple sullo schema rappresenti informazioni corrette per l’applicazione. Consideriamo la seguente base di dati e notiamo in essa le varie situazioni che non si dovrebbero presentare.

Studenti	Matricola	Cognome	Nome	Data di nascita				
	200768	Verdi	Fabio	12/02/2001				
	937653	Rossi	Luca	10/10/2000				
	937653	Bruni	Mario	01/12/2000				
Esami	Studente	Voto	Lode	Corso	Corsi	Codice	Titolo	Docente
	200768	36		05		01	Analisi	Giani
	937653	30	Lode	01		03	Chimica	Melli
	937653	28	Lode	04		04	Chimica	Belli
	276545	25		01				

- # Nella prima tupla della relazione “Esami” abbiamo un voto pari a 36 che, nel sistema italiano, non è ammissibile, in quanto i voti devono essere compresi tra 0 e 30
- # Nella terza tupla della relazione “Esami” viene indicato che è stata attribuita la lode in un esame il cui voto è 28, il che è impossibile.
- # Le ultime due tuple della relazione “Studenti” contengono informazioni su due studenti diversi ma con lo stesso numero di matricola, anche qua ciò è impossibile poiché il numero di matricola serve per identificare univocamente gli studenti.
- # La quarta tupla della relazione esami presenta per l’attributo *studente* un valore che non compare fra i numeri di matricola della relazione “Studenti”, anche questa è una situazione indesiderabile. Analogamente, la prima tupla presenta un codice corso non presente nella relazione “Corsi”.

Quindi, per evitare questo tipo di situazioni, è stato introdotto il concetto di “*vincolo di integrità*” ovvero una proprietà che deve essere soddisfatta dalle istanze che rappresentano informazioni corrette per l’applicazione.

Ogni vincolo può essere visto come un **predicato** che associa a ogni istanza il valore *vero* o *falso*. Se il predicato assume il valore **vero**, allora l'istanza soddisfa il vincolo.

Possiamo classificare i vincoli a seconda degli elementi di una base di dati, abbiamo due categorie:

- › Un vincolo si dice *intrarelazionale* se il suo soddisfacimento è definito rispetto a singole relazioni della base di dati
 - un *vincolo di tupla* è un vincolo che può essere valutato su ciascuna tupla, indipendentemente dalle altre
 - un *vincolo su valori* (o *vincolo di dominio*) è un vincolo definito con riferimento a singoli valori
- › Un vincolo si dice *interrelazionale* se coinvolge più relazioni

Esami	Studente	Voto	Lode	Corso
	200768	36	Lode	05
	937653	30	Lode	01
	937653	28	Lode	04
	276545	25	Lode	01

2.2.1 Vincoli di tupla

I vincoli di tupla esprimono condizioni sui valori di ciascuna tupla, indipendentemente dalle altre tuple.

Una sintassi possibile è quella che permette di definire espressioni booleane (*and*, *or* e *not*), riprendendo l'esempio dei voti:

$$(\text{not} (\text{Lode} = 'Lode')) \text{ or } (\text{Voto} = 30)$$

Il secondo vincolo indica che la lode è ammessa solo se il voto è pari a 30.

La definizione data ammette anche espressioni più complesse, basta che siano definite sui valori delle singole tuple. Consideriamo una relazione sul seguente schema:

Pagamenti (Data, Importo, Ritenute, Netto)

Possiamo definire il vincolo che impone (come da definizione di *Netto*) che il netto sia pari alla differenza tra l'importo originario e le ritenute:

$$\text{Netto} = \text{Importo} - \text{Ritenute}$$

2.2.2 Chiavi

Parlamo ora invece dei vincoli di chiave, vediamo un esempio per aiutarci

Matricola	Cognome	Nome	Nascita	Corso
4328	Rossi	Luigi	29/04/2000	Ing. Informatica
6328	Rossi	Dario	29/04/2000	Ing. Informatica
4766	Rossi	Luca	01/05/2001	Ing. Civile
4856	Neri	Luca	01/05/2001	Ing. Meccanica
5536	Neri	Luca	05/03/1999	Ing. Meccanica

Possiamo vedere che i valori delle varie tuple sull'attributo “**Matricola**” sono tutti diversi tra loro, questo perché, come abbiamo detto, il valore della matricola *identifica univocamente* gli studenti.

Anche se non sempre vero, anche i dati anagrafici identificano univocamente le persone sugli attributi “**Cognome, Nome, Nascita**”

Intuitivamente, una chiave è un insieme di attributi utilizzato per identificare univocamente le tuple di una relazione.

15. Un insieme K di attributi è **superchiave** di una relazione r se r non contiene tuple distinte t_1 e t_2 con $t_1[K] = t_2[K]$
16. K è **chiave** di r se è una superchiave minimale di r (cioè, non esiste un'altra superchiave K' di r che sia contenuta in K come sottoinsieme proprio)

Nell'esempio:

- L'insieme **{Matricola}** è superchiave; è anche una superchiave minimale, dato che contiene un solo attributo (l'insieme vuoto non è in grado di identificare tuple),
Quindi **{Matricola}** è una chiave
- L'insieme **{Cognome, Nome, Nascita}** è superchiave; nessuno dei suoi sottoinsiemi è superchiave, infatti esistono due tuple (la prima e la seconda) uguali su **Cognome** e **Nascita**, e tante altre...
Quindi **{Cognome, Nome, Nascita}** è un'altra chiave
- L'insieme **{Matricola, Corso}** è superchiave, ma non è una superchiave minimale, perché esiste un sottoinsieme proprio **{Matricola}**, esso stesso superchiave minimale
Quindi **{Matricola, Corso}** non è una chiave.
- L'insieme **{Nome, Corso}** non è superchiave, perché nella relazione compaiono due tuple (le ultime due) fra loro uguali sia su **Nome** che su **Corso**.

Esaminiamo un'altra relazione; questa non contiene tuple fra loro uguali sia su **Cognome** che su **Corso**.

Matricola	Cognome	Nome	Nascita	Corso
6328	Rossi	Dario	29/04/2000	Ing. Informatica
4766	Rossi	Luca	01/05/2001	Ing. Civile
4856	Neri	Luca	01/05/2001	Ing. Meccanica
5536	Neri	Luca	05/03/1999	Ing. Civile

Quindi per questa relazione l'insieme **{Cognome, Corso}** è superchiave ed essendoci tuple uguali su **Cognome** (le prime due) e su **Corso** (la 2° e la 4°) questo insieme è superchiave minimale, cioè chiave.

In questa relazione **Cognome** e **Corso** identificano univocamente le tuple ma ciò non è vero in generale. Poiché possono esistere studenti con lo stesso cognome iscritti allo stesso corso.

Possiamo quindi dire che **{Cognome, Corso}** è “casualmente” una chiave, mentre a noi interessano le chiavi corrispondenti ai vincoli di integrità soddisfatti da tutte le relazioni lecite su un certo schema.

Definendo uno schema associamo a esso i vincoli di interesse corrispondenti a proprietà del mondo reale che rappresentiamo con la base di dati.

I vincoli sono quindi definiti a livello di schema, con riferimento a tutte le istanze, consideriamo corrette solo le istanze che soddisfano tutti i vincoli.

2.2.3 Chiavi e valori nulli

In presenza di valori nulli non è più vero che i valori delle chiavi permettono di identificare univocamente le tuple delle relazioni e di stabilire riferimenti fra tuple di relazioni diverse.

Dobbiamo quindi limitare la presenza dei valori nulli nelle chiavi delle relazioni, per farlo selezioniamo una chiave (detta **chiave primaria**) e su di essa si vieta la presenza di valori nulli; sulle altre chiavi sono in genere ammessi. Gli attributi che costituiscono una chiave primaria vengono evidenziati con una sottolineatura.

Matricola	Cognome	Nome	Nascita	Corso
3976	Rossi	Mario	NULL	Ing. Informatica
4856	Neri	Luca	NULL	NULL

2.2.4 Vincoli di integrità referenziale

Consideriamo la seguente base di dati

<i>Infrazioni</i>	<u>Codice</u>	<u>Data</u>	<u>Agente</u>	<u>Articolo</u>	<u>Stato</u>	<u>Numero</u>
	143256	25/10/2021	567	44	I	AB234ZK
	987554	26/10/2021	456	34	I	AB234ZK
	987557	26/10/2021	456	34	I	CB123AA
	630876	15/10/2021	456	53	F	CB123AA
	539856	12/10/2021	567	44	F	CB123AA

Un **vincolo di integrità referenziale** (*foreign key*) fra un insieme di attributi X di una relazione R_1 e un'altra relazione R_2 è soddisfatto se i valori su X di ciascuna tupla dell'istanza di R_1 compaiono come valori della chiave primaria dell'istanza di R_2 .

Vediamo prima il caso in cui la chiave di R_2 è unica e composta da un solo attributo B:

il vincolo di integrità referenziale fra l'attributo A di R_1 e la relazione R_2 è soddisfatto se

<i>Agenti</i>	<u>Matricola</u>	<u>CF</u>	<u>Cognome</u>	<u>Nome</u>
	567	RSSM...	Rossi	Mario
	456	NREL...	Neri	Luigi
	638	NREP...	Neri	Piero

<i>Auto</i>	<u>Stato</u>	<u>Numero</u>	<u>Proprietario</u>	<u>Indirizzo</u>
	I	CB123AA	Verdi Piero	Via Tigli...
	I	DE834ZZ	Verdi Piero	Via Tigli...
	I	AB234ZK	Bini Luca	Via Aceri...
	F	CB123AA	Beau Marcel	Rue Luis XIV

$$\forall t_1 \in R_1 : t_1 \neq \text{NULL} , \quad \exists t_2 \in R_2 : t_1[A] = t_2[B]$$

Nel caso generale, bisogna fare attenzione alla chiave primaria K di R_2 , ed è necessario specificare un ordinamento nell'insieme X e in K . Indicando gli attributi in ordine $X = A_1, A_2, \dots, A_p$ e $K = B_1, B_2, \dots, B_p$ il vincolo è soddisfatto se

$$\forall t_1 \in R_1 : t_1 \neq \text{NULL} , \quad \text{su } X \exists t_2 \in R_2 : t_1[A_i] = t_2[B_i], \quad \forall i = 1, \dots, p$$

Sulla base di dati delle multe ha senso definire i vincoli di integrità referenziale:

- fra **Agente** della relazione *Infrazioni* e la relazione *Agenti*
- fra **Stato** e **Numero** di *infrazioni* e la relazione *Auto*, in cui l'ordine sia **Stato, Numero**

Possiamo notare come il ragionamento sull'ordine degli attributi possa sembrare inutile, vediamo quindi un esempio che ci aiuti a capire come mai sia indispensabile.

<i>Incidenti</i>	<u>Codice</u>	<u>Stato 1</u>	<u>Numero 1</u>	<u>Stato 2</u>	<u>Numero 2</u>	...
	6207	I	CB123AA	I	DE834ZZ	...
	6974	F	AB234ZK	I	CB123AA	...

<i>Auto</i>	<u>Stato</u>	<u>Numero</u>	<u>Proprietario</u>	<u>Indirizzo</u>
	I	DE834ZZ	Verdi Piero	Via Tigli...
	F	AB234ZK	Beau Marcel	Rue Louis XIV...
	F	BC964ZK	Charles Armand	Rue Mont Blanc...
	I	CB123AA	Luci Gino	Via Noci...

In questo caso, non è possibile stabilire la corrispondenza nel vincolo di integrità referenziale verso la relazione *Auto* con i nomi degli attributi, perché sono diversi da quelli della chiave primaria di *Auto*. Solo con l'ordinamento si può specificare che il riferimento associa **Stato1** a **Stato** e **Numero1** a **Numero**.

3 ALGEBRA E CALCOLO RELAZIONALE

L'algebra relazionale è un linguaggio procedurale basato su concetti di tipo algebrico. Questo linguaggio è costituito da un insieme di operatori definiti su relazioni che producono ancora relazioni come risultati. Così possiamo costruire espressioni che coinvolgono più operatori per formulare interrogazioni.

I vari operatori sono:

- Quelli insiemistici tradizionali: *unione*, *intersezione*, *differenza*
- Quelli più specifici: *ridenominazione*, *selezione*, *proiezione*
- Il più importante: *join*
- L'operatore di *divisione* (ridondante poiché si può formulare per mezzo di altri operatori)

3.1.1 Unione, intersezione, differenza

Le relazioni sono insiemi, quindi ha senso definire gli operatori insiemistici.

Una relazione è un insieme di tuple *omogenee*, cioè, definite sugli stessi attributi. Quindi non ha senso definirli con riferimenti a relazioni su attributi diversi.

Per esempio, l'unione di due relazioni su schemi diversi darebbe un insieme di tuple disomogenee, alcune definite su attributi della prima e altre su quelli della seconda.

Un insieme di tuple disomogenee non è una relazione e noi vogliamo che i risultati siano relazioni.

Quindi definiamo ammissibili, solo l'utilizzo degli operatori sugli stessi attributi

<i>Laureati</i>	Matricola	Cognome	Età	<i>Dirigenti</i>	Matricola	Cognome	Età
	7274	Rossi	37		9297	Neri	56
	7432	Neri	39		7432	Neri	39
	9824	Verdi	38		9824	Verdi	38

Laureati U Dirigenti

	Matricola	Cognome	Età
	7274	Rossi	37
	7432	Neri	39
	9824	Verdi	38
	9297	Neri	56

Laureati ∩ Dirigenti

	Matricola	Cognome	Età
	7432	Neri	39
	9824	Verdi	38

Laureati – Dirigenti

	Matricola	Cognome	Età
	7274	Rossi	37

- L'*unione* di due relazioni r_1 e r_2 è indicata con $r_1 \cup r_2$ ed è una relazione contenente le tuple che appartengono a r_1 oppure a r_2
- L'*intersezione* di $r_1(X)$ e $r_2(X)$ è indicata con $r_1 \cap r_2$ ed è una relazione contenente le tuple che appartengono sia a r_1 che a r_2
- La *differenza* di r_1 e r_2 è indicata con $r_1 - r_2$ ed è una relazione contenente le tuple che appartengono a r_1 e non a r_2

3.1.2 Ridenominazione

La limitazione dell'eseguire operazioni solo su tuple definite sugli stessi attributi, seppur giustificata risulta pesante, considerando queste due relazioni:

<i>Paternità</i>	Padre	Figlio	<i>Maternità</i>	Madre	Figlio
	Adamò	Caino		Eva	Caino
	Adamò	Abele		Eva	Set
	Abramo	Isacco		Sara	Isacco
	Abramo	Ismaele		Agar	Ismaele

Paternità U Maternità ?

Sarebbe sensato eseguire un'unione così da ottenere tutte le coppie "genitore-figlio", ma non è possibile perché quello che chiameremmo *Genitore* si chiama *Padre* in una relazione e *Madre* nell'altra.

Per risolvere questo problema introduciamo un operatore specifico il cui obiettivo è aeguare i nomi degli attributi, per poter facilitare le operazioni insiemistiche.

L'operatore è detto **ridenominazione** perché "cambia il nome degli attributi" lasciando inalterato il contenuto delle relazioni.

Consideriamo un esempio, riprendendo la relazione *Paternità*

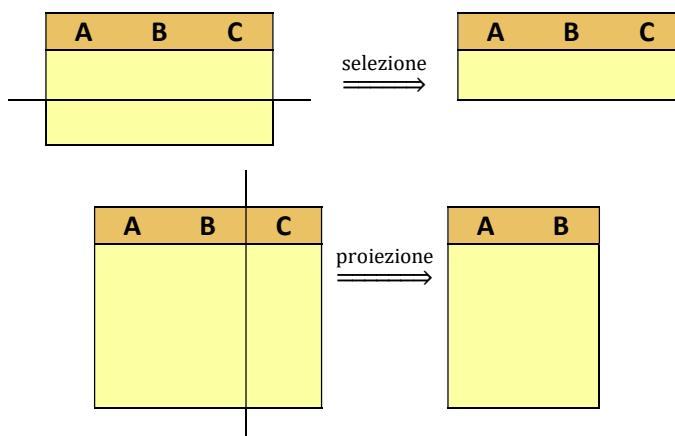
<i>Paternità</i>	Padre	Figlio	$\rho_{\text{Genitore} \leftarrow \text{Padre}}$	Genitore	Figlio
	Adamò	Caino		Adamò	Caino
	Adamò	Abele	⇒	Adamò	Abele
	Abramo	Isacco		Abramo	Isacco
	Abramo	Ismaele		Abramo	Ismaele

La ridenominazione agisce sullo schema agisce solo sullo schema, cambiando il nome dell'attributo **Padre** in **Genitore**, come indicato dalla notazione **Genitore** \leftarrow **Padre** al pedice di ρ , che è il simbolo che denota l'operatore ridenominazione.

3.1.3 Selezione

Esaminiamo gli operatori che ci permettono di manipolare le relazioni, ovvero *selezione*, *proiezione* e *join*.

Facciamo prima una considerazione sui primi due: selezione e proiezione svolgono funzioni che potremmo definire complementari, entrambe sono definite su di un solo operando e producono come risultato una porzione dell'operando



- La selezione produce un sottoinsieme delle tuple su tutti gli attributi
- La proiezione da un risultato a cui contribuiscono tutte le tuple, ma su un sottoinsieme degli attributi

<i>Impiegati</i>	Cognome	Nome	Età	Stipendio
	Rossi	Mario	25	2000,00
	Neri	Luca	40	3000,00
	Verdi	Nico	36	4500,00
	Rossi	Marco	40	3900,00

$\sigma_{\text{Età} > 30 \wedge \text{Stipendio} > 4000,00}(\text{Impiegati})$	Cognome	Nome	Età	Stipendio
	Verdi	Nico	36	4500,00

Questo esempio illustra le caratteristiche dell'operatore, denotato dal simbolo σ al cui pedice viene indicata la “condizione di selezione” opportuna. Il risultato contiene le tuple dell'operando che soddisfano la condizione di selezione. Le condizioni di selezione possono prevedere confronti tra attributi e confronti fra attributi e costanti, e possono essere anche ottenute combinando condizioni semplici con i connettivi logici \vee (or) \wedge (and) \neg (not).

3.1.4 Proiezione

La proiezione permette di decomporre verticalmente le relazioni

<i>Impiegati</i>	Cognome	Nome	Reparto	Capo	$\pi_{\text{Cognome}, \text{Nome}}(\text{Impiegati})$	Cognome	Nome
	Rossi	Mario	Vendite	Gatti		Rossi	Mario
	Neri	Luca	Vendite	Gatti		Neri	Luca
	Verdi	Mario	Personale	Lupi		Verdi	Mario
	Rossi	Marco	Personale	Lupi		Rossi	Marco

Mentre in quest'altro esempio si nota una situazione diversa, ovvero il risultato della proiezione contiene un numero di tuple inferiore.

Questo perché alcune tuple, avendo uguali valori su tutti gli attributi della proiezione danno lo stesso contributo alla proiezione, ed essendo le relazioni definite come insiemi non possono comparire più tuple uguali fra loro; i contributi uguali “collassano” in una sola tupla.

<i>Impiegati</i>	Cognome	Nome	Reparto	Capo	$\pi_{\text{Reparto}, \text{Capo}}(\text{Impiegati})$	Reparto	Capo
	Rossi	Mario	Vendite	Gatti		Vendite	Gatti
	Neri	Luca	Vendite	Gatti		Personale	Lupi
	Verdi	Mario	Personale	Lupi			
	Rossi	Marco	Personale	Lupi			

Esiste un legame fra i vincoli di chiave e le proiezioni,
 $\pi_Y(r)$ contiene lo stesso numero di tuple di $r \Leftrightarrow Y$ è superchiave per r , infatti:

- Se Y è superchiave $\Rightarrow r$ non contiene tuple uguali su Y
- Se la proiezione ha tante tuple quante l'operando $\Rightarrow r$ non contiene coppie di tuple uguali (def di superchiave)

Per la relazione *Impiegati* gli attributi **Cognome** e **Nome** formano una chiave (e quindi una superchiave), mentre gli attributi **Reparto** e **Capo** non formano una superchiave.

3.1.5 Join

È l'operatore che permette di correlare dati contenuti in relazioni diverse. Ne esistono diverse varianti:

Join naturale È un operatore che correla dati in relazioni diverse sulla base di valori uguali su attributi con lo stesso nome. L'operatore è denotato con il simbolo \bowtie . Il risultato è una relazione sull'unione degli insiemi di attributi delle relazioni in input; le sue tuple sono ottenute combinando le tuple degli operandi con valori uguali sugli attributi comuni (**Reparto**)

r_1	Impiegato	Reparto	r_2	Reparto	Capo
	Rossi	Vendite		Produzione	Mori
	Neri	Produzione		Vendite	Bruni
	Bianchi	Produzione			

$r_1 \bowtie r_2$	Impiegato	Reparto	Capo
	Rossi	Vendite	Bruni
	Neri	Produzione	Mori
	Bianchi	Produzione	Mori

È molto frequente eseguire join sulla base di valori della chiave di una delle relazioni coinvolte, esplicitando i riferimenti fra tuple che sono realizzati per mezzo di valori.

Riconsideriamo, per esempio, le relazioni *Infrazioni* e *Auto* assieme al loro join.

<i>Infrazioni</i>	Codice	Data	Agente	Articolo	Stato	Numero
	143256	25/10/2021	567	44	I	AB234ZK
	987554	26/10/2021	456	34	I	AB234ZK
	987557	26/10/2021	456	34	I	CB123AA
	630876	15/10/2021	456	53	F	CB123AA
	539856	12/10/2021	567	44	F	CB123AA
<i>Auto</i>	Stato	Numero	Proprietario		Indirizzo	
	I	CB123AA	Verdi Piero		Via Tigli...	
	I	DE834ZZ	Verdi Piero		Via Tigli...	
	I	AB234ZK	Bini Luca		Via Aceri...	
	F	CB123AA	Beau Marcel		Rue Luis XIV	

Infrazioni \bowtie *Auto*

Codice	Data	Agente	Articolo	Stato	Numero	Proprietario	Indirizzo
143256	25/10/2021	567	44	I	AB234ZK	Bini Luca	Via Aceri...
987554	26/10/2021	456	34	I	AB234ZK	Bini Luca	Via Aceri...
987557	26/10/2021	456	34	I	CB123AA	Verdi Piero	Via Tigli...
630876	15/10/2021	456	53	F	CB123AA	Beau Marcel	Rue Luis XIV
539856	12/10/2021	567	44	F	CB123AA	Beau Marcel	Rue Luis XIV

Notiamo che ciascuna delle tuple di *Infrazioni* è stata combinata con una e una sola di *Auto*

- Una sola → perché **Stato** e **Numero** formano una chiave di *Auto*
- Almeno una → perché è definito il vincolo di integrità referenziale fra **Stato** e **Numero** in *Infrazioni* e la chiave primaria di *Auto*

Paternità	Padre	Figlio	Maternità	Madre	Figlio	Pat \bowtie Mat	Padre	Figlio	Madre
	Adamo	Caino		Eva	Caino		Adamo	Caino	Eva
	Adamo	Abele		Eva	Set		Adamo	Abele	Eva
	Abramo	Isacco		Sara	Isacco		Abramo	Isacco	Sara
	Abramo	Ismaele		Agar	Ismaele		Abramo	Ismaele	Agar

Join completi e incompleti Vediamo diversi tipi di join; Nelle relazioni con **Reparto**, **Impiegato** e **Capo** abbiamo che ogni tupla di ogni operando contribuisce ad almeno una tupla del risultato, join completo.

Invece qua sotto abbiamo un join in cui alcune tuple degli operandi non contribuiscono al risultato, Questo perché l'altra relazione non contiene tuple con gli stessi valori sull'attributo in comune (**Reparto**).

r_1	Impiegato	Reparto	r_2	Reparto	Capo
	Rossi	Vendite		Produzione	Mori
	Neri	Produzione		Acquisti	Bruni
	Bianchi	Produzione			

$r_1 \bowtie r_2$	Impiegato	Reparto	Capo
	Neri	Produzione	Mori
	Bianchi	Produzione	Mori

Queste tuple vengono chiamate *dangling* ("dondolanti")

Come caso limite, è possibile che nessun tupla contenga gli stessi valori sugli attributi in comune e che quindi il risultato del join sia una relazione vuota.

r_1	Impiegato	Reparto	r_2	Reparto	Capo
	Rossi	Vendite		Concorsi	Mori
	Neri	Produzione		Acquisti	Bruni
	Bianchi	Produzione			

$r_1 \bowtie r_2$	Impiegato	Reparto	Capo

All'estremo opposto, è possibile che ciascuna delle tuple di ciascuno degli operandi sia compatibile con tutte le tuple dell'altro, il risultato contiene un numero di tuple pari al prodotto della cardinalità degli operandi cioè $|r_1| \times |r_2|$ ($|r|$ indica la cardinalità della relazione r)

r_1	Impiegato	Progetto	r_2	Progetto	Capo
	Rossi	A		A	Mori
	Neri	A		A	Bruni
	Bianchi	A			

$r_1 \bowtie r_2$	Impiegato	Progetto	Capo
	Rossi	A	Mori
	Neri	A	Mori
	Bianchi	A	Mori
	Rossi	A	Bruni
	Neri	A	Bruni
	Bianchi	A	Bruni

Ricapitolando, possiamo dire che il join di r_1 e r_2 contiene un numero di tuple compreso tra 0 e $|r_1| \times |r_2|$. Inoltre:

- › se il join di r_1 e r_2 è completo, allora contiene almeno un numero di tuple pari al massimo fra $|r_1|$ e $|r_2|$
- › se $X_1 \cap X_2$ contiene una chiave per $r_2 \Rightarrow$ il join di $r_1(X_1)$ e $r_2(X_2)$ contiene al più $|r_1|$ tuple
- › se $X_1 \cap X_2$ coincide con una chiave per r_2 e sussiste il vincolo di riferimento fra $X_1 \cap X_2$ in r_1 e la chiave in $r_2 \Rightarrow$ il join di $r_1(X_1)$ e $r_2(X_2)$ contiene esattamente $|r_1|$ tuple.

Join esterni La caratteristica del join di “tralasciare” le tuple di una relazione che non abbia una controparte nell’altra è utile ma potenzialmente pericolosa (omissione di informazioni rilevanti).

Supponiamo di avere le relazioni, e di essere interessati a tutti gli impiegati indicando il capo se noto

r_1	Impiegato	Reparto
Rossi	Vendite	
Neri	Produzione	
Bianchi	Produzione	

r_2	Reparto	Capo
Produzione	Mori	
Acquisti	Bruni	

Per questo scopo (scavalcare questo “tralasciare” le tuple) esiste una variante del join, detta **join esterno** (o *outer join*) che impone che tutte le tuple diano un contributo al risultato estendendole con valori nulli dove non ci siano controparti per le tuple in input.

Del join esterno abbiamo tre varianti:

- *sinistro*, estende solo le tuple del primo operando

$r_1 \bowtie r_2$	Impiegato	Reparto	Capo
Rossi	Vendite	NULL	
Neri	Produzione	Mori	
Bianchi	Produzione	Mori	

- *destro*, estende solo le tuple del secondo operando

$r_1 \bowtie r_2$	Impiegato	Reparto	Capo
Neri	Produzione	Mori	
Bianchi	Produzione	Mori	
NULL	Acquisti	Bruni	

- *completo*, estende tutte le tuple

$r_1 \bowtie r_2$	Impiegato	Reparto	Capo
Rossi	Vendite	NULL	
Neri	Produzione	Mori	
Bianchi	Produzione	NULL	
NULL	Acquisti	Bruni	

Semijoin Il semijoin è un operatore derivato dal join che restituisce le tuple di una relazione che partecipano al join naturale della stessa relazione con un’altra.

Si indica con il simbolo \bowtie

r_1	Impiegato	Reparto
Rossi	Vendite	
Neri	Produzione	
Bianchi	Produzione	

r_2	Reparto	Capo
Produzione	Mori	
Acquisti	Bruni	

$r_1 \bowtie r_2$	Impiegato	Reparto
Neri	Produzione	
Bianchi	Produzione	

Questo operatore può essere espresso anche come

$$r_1 \bowtie r_2 = \pi_{X_1}(r_1 \bowtie r_2)$$

Dove X_1 è l’attributo che le due relazioni hanno in comune.

Prodotto cartesiano Il join diventa *prodotto cartesiano* quando gli insiemi degli attributi degli operandi sono disgiunti

<i>Impiegati</i>	Impiegato	Progetto	<i>Progetti</i>	Codice	Capo
	Rossi	A		A	Venere
	Neri	A		B	Marte
	Neri	B			

<i>Impiegati</i> \bowtie <i>Progetti</i>	Impiegato	Progetto	Codice	Capo
	Rossi	A	A	Venere
	Neri	A	A	Venere
	Neri	B	A	Venere
	Rossi	A	B	Marte
	Neri	A	B	Marte
	Neri	B	B	Marte

Theta-join ed equi-join Il prodotto cartesiano di solito ha poca utilità perché concatena tuple non necessariamente correlate dal punto di vista semantico. Spesso è seguito da una selezione che centra l'attenzione su tuple correlate secondo le esigenze (eg. quando Progetto e Codice coincidono)

$$\sigma_{\text{Progetto} = \text{Codice}}(\text{Impiegati} \bowtie \text{Progetti})$$

Impiegato	Progetto	Codice	Capo
Rossi	A	A	Venere
Neri	A	A	Venere
Neri	B	B	Marte

Viene spesso definito come un operatore derivato perché corrisponde ad un prodotto cartesiano seguito da una selezione

$$r_1 \bowtie r_2 = \sigma_F(r_1 \bowtie r_2)$$

F

Un theta-join che ha come condizione *F* un'uguaglianza tra due attributi delle due relazioni viene chiamato **equi-join**. Quindi in realtà quello che abbiamo visto prima era un equi-join.

3.1.6 Interrogazioni in algebra relazionale

Un'interrogazione è definita come una funzione che, applicata a istanze di basi di dati, produce relazioni

In algebra relazionale le interrogazioni si uno schema di base di dati **R** vengono formulate con espressioni i cui atomi sono nomi di relazioni in **R**.

Mostriamo la formulazione di alcune interrogazioni che fanno riferimento allo schema con queste due relazioni:

Impiegati(Matr, Nome, Età, Stipendio)

Supervisione(Capo, Impiegato)

<i>Impiegati</i>	Matr	Nome	Età	Stipendio	<i>Supervisione</i>	Capo	Impiegato
	101	Mario Rossi	34	40		210	101
	103	Mario Bianchi	23	35		210	103
	104	Luigi Neri	38	61		210	104
	105	Nico Bini	44	38		231	105
	210	Marco Celli	49	60		301	210
	231	Siro Bisi	50	60		301	231
	252	Nico Bini	44	70		375	252
	301	Sergio Rossi	34	70			
	375	Mario Rossi	50	65			

1. "Trovare matricola, nome ed età degli impiegati che guadagnano più di 40"

$\pi_{\text{Matr}, \text{Nome}, \text{Età}}(\sigma_{\text{Stipendio} > 40}(\text{Impiegati}))$

Matr	Nome	Età
104	Luigi Neri	38
210	Marco Celli	49
231	Siro Bisi	50
252	Nico Bini	44
301	Sergio Rossi	34
375	Mario Rossi	50

2. "Trovare le matricole dei capi che guadagnano più di 40"

$$\pi_{\text{Capo}} \left(\begin{array}{ccc} \text{Supervisione} & \bowtie & \sigma_{\text{Stipendio} > 40}(\text{Impiegati}) \\ \text{Impiegato} = \text{Matr} & & \end{array} \right)$$

Capo
210
301
375

3. "Trovare nome e stipendio dei capi degli impiegati che guadagnano più di 40"

$$\begin{aligned} \pi_{\text{NomeC}, \text{StipC}} & \left(\rho_{\text{MatC}, \text{NomeC}, \text{StipC}, \text{EtàC} \leftarrow \text{Matr}, \text{Nome}, \text{Stip}, \text{Età}}(\text{Impiegati}) \right) \\ & \bowtie_{\text{MatrC} = \text{Capo}} \\ \text{Supervisione} & \bowtie_{\text{Imp} = \text{Matr}} \sigma_{\text{Stip} > 40}(\text{Impiegati}) \end{aligned}$$

NomeC	StipC
Marco Celli	60
Sergio Rossi	70
Mario Rossi	56

3.1.7 Espressioni equivalenti

L'algebra relazionale permette di formulare espressioni fra loro *equivalenti*, cioè che producono lo stesso risultato, per esempio con gli operatori di addizione e moltiplicazione vale:

$$x \cdot (y + z) = x \cdot y + x \cdot z$$

Nell'algebra relazionale possiamo dare una definizione analoga, facendo attenzione al fatto che l'equivalenza può dipendere dallo schema, oppure essere assoluta:

- 17. $E_1 \equiv_R E_2$ se $E_1(r) = E_2(r)$, per ogni istanza r di R
- 18. $E_1 \equiv E_2$ se $E_1 \equiv_R E_2$ per ogni schema di R

La distinzione fra i due casi è dovuta al fatto che gli schemi degli operandi non vengono esplicitati nelle espressioni e il comportamento può variare a seconda degli attributi nei vari schemi di relazione.

Esempio:

$$\pi_{AB}(\sigma_{A>0}(R)) \equiv \sigma_{A>0}(\pi_{AB}(R))$$

Mentre questa equivalenza:

$$\pi_{AB}(R_1) \bowtie \pi_{AC}(R_2) \equiv_R \pi_{ABC}(R_1 \bowtie R_2)$$

sussiste se e solo se nello stesso schema R l'intersezione fra gli insiemi di attributi R_1 e R_2 è pari ad A . Infatti se ci fossero anche altri attributi, il join opererebbe solo su A nella prima espressione e su A e tali altri attributi nella seconda, con risultati in generale diversi.

Vediamo un primo insieme di trasformazioni interessanti:

1. Combinare una cascata di selezioni atomiche

$$\sigma_{F_1 \wedge F_2}(E) \equiv \sigma_{F_1}(\sigma_{F_2}(E))$$

2. Idempotenza delle proiezioni (una proiezione può essere trasformata in una cascata di proiezioni che "eliminano" i vari attributi in fasi successive)

$$\pi_X(E) \equiv \pi_X(\pi_{XY}(E))$$

3. Commutatività di proiezione e selezione

$$\pi_{XY}(\sigma_X(R)) = \sigma_X(\pi_{XY}(R))$$

4. *Pushing selections down* – Anticipazione della selezione rispetto al join

$$\sigma_F(E_1 \bowtie E_2) \equiv E_1 \bowtie \sigma_F(E_2)$$

5. *Pushing projections down* – Anticipazione della proiezione rispetto al join

$$\pi_{X_1 Y_2}(E_1 \bowtie E_2) \equiv E_1 \bowtie \pi_{Y_1}(E_2)$$

- E_1 ha come schema X_1 , E_2 ha come schema Y_2

- Se $Y_2 \subseteq (X_1 \cap X_2)$ gli attributi in $X_2 - Y_2$ non sono coinvolti nel join e l'equivalenza vale

6. Utilizzando la definizione del theta-join

$$\sigma_F(E_1 \bowtie E_2) \equiv E_1 \bowtie_F E_2$$

Con $X_1 \cap X_2 = \emptyset$, indichiamo ancora con X_1 e X_2 rispettivamente gli attributi di E_1 e quelli di E_2 .

Vediamo un esempio che chiarisce l'uso delle trasformazioni.

Supponiamo di voler trovare i numeri di matricola dei capi di impiegati con meno di trent'anni.

$$\begin{aligned} & \pi_{\text{Capo}} \left(\sigma_{\text{Matr} = \text{Impiegato} \wedge \text{Età} < 30} (\text{Impiegati} \bowtie \text{Supervisione}) \right) \\ & \pi_{\text{Capo}} \left(\sigma_{\text{Matr} = \text{Impiegato}} \left(\sigma_{\text{Età} < 30} (\text{Impiegati} \bowtie \text{Supervisione}) \right) \right) \\ & \pi_{\text{Capo}} \left(\sigma_{\text{Età} > 30} (\text{Impiegati}) \bowtie_{\text{Matr} = \text{Impiegato}} \text{Supervisione} \right) \\ & \pi_{\text{Capo}} \left(\pi_{\text{Matr}} \left(\sigma_{\text{Età} < 30} (\text{Impiegati}) \bowtie_{\text{Matr} = \text{Impiegato}} \text{Supervisione} \right) \right) \end{aligned}$$

7. $\sigma_F(E_1 \cup E_2) \equiv \sigma_F(E_1) \cup \sigma_F(E_2)$
8. $\sigma_F(E_1 - E_2) \equiv \sigma_F(E_1) - \sigma_F(E_2)$
9. $\pi_X(E_1 \cup E_2) \equiv \pi_X(E_1) \cup \pi_X(E_2)$
10. $\sigma_{F_1 \vee F_2}(R) \equiv \sigma_{F_1}(R) \cup \sigma_{F_2}(R)$
11. $\sigma_{F_1 \wedge F_2}(R) \equiv \sigma_{F_1}(R) \cap \sigma_{F_2}(R) \equiv \sigma_{F_1}(R) \bowtie \sigma_{F_2}(R)$
12. $\sigma_{F_1 \wedge \neg(F_2)}(R) \equiv \sigma_{F_1}(R) - \sigma_{F_2}(R)$
13. $E \bowtie (E_1 \cup E_2) \equiv (E \bowtie E_1) \cup (E \bowtie E_2)$

3.1.8 Algebra con valori nulli

Consideriamo la relazione *Persone* e la selezione:

<i>Persone</i>	Nome	Età	Reddito
	Aldo	35	15
	Andrea	27	21
	Maria	NULL	42

$$\sigma_{\text{Età} > 30}(\text{Persone})$$

La prima tupla sicuramente farà parte del risultato e la seconda no, ma la terza?

Diciamo che il valore è *sconosciuto (Unknown)*

$$\sigma_{\text{Età} > 30}(\text{Persone}) \cup \sigma_{\text{Età} \leq 30}(\text{Persone})$$

Logica vorrebbe che il risultato di questa espressione sia esattamente *Persone*, per la prima parte tutti i maggiori di 30 e per la seconda i non maggiori di 30. Ma se le due sottoespressioni vengono valutate separatamente la terza tupla dell'esempio ha un'appartenenza sconosciuta a entrambe le sottoespressioni.

Il metodo migliore per superare gli inconvenienti di cui abbiamo parlato consiste nel trattare i valori nulli da un punto di vista sintattico. Allo scopo si introducono due nuove forme di condizioni atomiche di selezione, che valutano se un valore è specificato oppure nullo:

- › A is NULL → assume valore vero sulla tupla t se il valore su A è nullo, falso se specificato
- › A is not NULL → assume valore vero sulla tupla t se il valore su A è specificato, falso altrimenti

Quindi

$$\sigma_{\text{Età} > 30}(\text{Persone}) \cup \sigma_{\text{Età} \leq 30}(\text{Persone}) \cup \sigma_{\text{Età} \text{ is NULL}}(\text{Persone}) = \text{Persone}$$

3.1.9 Viste

Può risultar utile mettere a disposizione degli utenti rappresentazioni diverse per gli stessi dati.

Nel modello relazionale le chiamiamo *relazioni derivate* e ne abbiamo due tipologie:

- › *Viste materializzate*: relazioni derivate effettivamente memorizzate nella base di dati
- › *Relazioni virtuali*: relazioni definite per mezzo di funzioni, non memorizzate nella base di dati ma utilizzabili nelle interrogazioni come se lo fossero

Le viste vengono definite nei sistemi relazionali per mezzo di espressioni del linguaggio di interrogazione. Eventuali interrogazioni che si riferiscono alle viste vengono risolte dopo aver sostituito alla vista la sua definizione, per esempio:

<i>Afferenza</i>	Impiegato	Dipartimento	<i>Direzione</i>	Dipartimento	Direttore
	Rossi	A		A	Mori
	Neri	B		B	Bruni
	Bianchi	B			

Una vista può essere definita:

$$\text{Supervisore} := \pi_{\text{Impiegato}, \text{Direttore}}(\text{Afferenza} \bowtie \text{Direzione})$$

3.2 CALCOLO RELAZIONALE

Il **Calcolo relazionale** è una famiglia di linguaggi di interrogazione, basati sul calcolo dei predicati del primo ordine, sono tutti linguaggi dichiarativi, ovvero specificano le proprietà del risultato delle interrogazioni.

Esistono diverse versioni del calcolo relazionale, noi ne vedremo due:

- Il *calcolo relazionale su domini*: presenta naturalmente le caratteristiche di questi linguaggi
- Il *calcolo su tuple con dichiarazioni in range*: è la base dei costrutti di SQL

3.2.1 Calcolo relazionale su domini

Sintassi: le espressioni del calcolo relazionale sui domini hanno la forma:

$$\{A_1:x_1, \dots, A_k:x_k | f\}$$

Dove:

- $A_1:x_1, \dots, A_k:x_k$ target list (lista degli obiettivi) perché definisce la struttura del risultato, che è costituito dalla relazione su A_1, \dots, A_k che contiene le tuple i cui valori sostituiti a x_1, \dots, x_k rendono vera la formula, rispetto a un database
- A_1, \dots, A_k sono attributi distinti (possono appartenere a database differenti)
- x_1, \dots, x_k sono variabili
- f è una formula (con operatori booleani e quantificatori)

Semantica: il risultato è una relazione su A_1, \dots, A_k che contiene le tuple di valori per x_1, \dots, x_k soddisfacendo la formula f .

Mostriamo le espressioni del calcolo che realizzano le stesse interrogazioni che abbiamo formulato a 3.1.6

$$\begin{aligned} &\text{Impiegati}(\underline{\text{Matr}}, \underline{\text{Nome}}, \underline{\text{Età}}, \underline{\text{Stipendio}}) \\ &\text{Supervisione}(\underline{\text{Capo}}, \underline{\text{Impiegato}}) \end{aligned}$$

1. *Trovare matricola, nome età e stipendio degli impiegati che guadagnano più di 40 mila euro*
Che formuleremo in algebra con una selezione

$$\sigma_{\text{Stipendio} > 40}(\text{Impiegati})$$

Nel calcolo relazionale su domini abbiamo una formulazione altrettanto semplice con l'espressione

$$\{\text{Matr}: m, \text{Nome}: n, \text{Età}: e, \text{Stipendio}: s | \text{Impiegati}(\text{Matr}: m, \text{Nome}: n, \text{Età}: e, \text{Stipendio}: s) \wedge s > 40\}$$

Notiamo la presenza di due condizioni nella formula (dopo |) connesse dall'operatore logico *and* (\wedge)

19. la prima, *Impiegati*($\text{Matr}: m, \text{Nome}: n, \text{Età}: e, \text{Stipendio}: s$) \rightarrow Richiede che i valori rispettivamente sostituiti alle variabili m, n, e, s costituiscano una tupla della relazione *Impiegati*
20. La seconda richiede che il valore della variabile s sia maggiore di 40

2. *Trovare matricola, nome ed età degli impiegati che guadagnano più di 40 mila euro*

$$\pi_{\text{Matr}, \text{Nome}, \text{Età}} \left(\sigma_{\text{Stipendio} > 40}(\text{Impiegati}) \right)$$

Può essere formulata in vari modi, il più diretto è basato sul notare che ciò ci interessa sono i valori: matricola, nome ed età che partecipano a tuple per le quali lo stipendio è maggiore di 40

$$\{\text{Matr}: m, \text{Nome}: n, \text{Età}: e | \text{Impiegati}(\text{Matr}: m, \text{Nome}: n, \text{Età}: e, \text{Stipendio}: s) \wedge s > 40\}$$

Questa struttura si estende a interrogazioni più complesse formulate in algebra con l'operatore di join: avremo bisogno di più condizioni atomiche, una per ciascuna relazione coinvolta, e possiamo utilizzare variabili ripetute per indicare le condizioni di join.

3. *Trovare le matricole dei capi degli impiegati che guadagnano più di 40 mila euro*

$$\pi_{\text{Capo}} \left(\text{Supervisione} \bowtie_{\text{Impiegato} = \text{Matr}} \sigma_{\text{Stipendio} > 40} (\text{Impiegati}) \right)$$

Può essere formulata con:

$$\{\text{Capo: } c \mid \text{Impiegati}(\text{Matr: } m, \text{Nome: } n, \text{Età: } e, \text{Stipendio: } s) \wedge \text{Supervisione}(\text{Impiegato: } m, \text{Capo: } c) \wedge s > 40\}$$

Dove la variabile m , comune alle due condizioni, realizza la stessa correlazione fra le due tuple specificata nel join.

Se in un'espressione è richiesto il coinvolgimento di tuple diverse di una stessa relazione (in algebra, il join di una relazione con se stessa) è sufficiente includere nella formula più condizioni sullo stesso predicato con variabili diverse.

4. *Trovare nome e stipendio dei capi degli impiegati che guadagnano più di 40 mila euro*

$$\begin{aligned} \pi_{\text{NomeC}, \text{StipC}} \left(\rho_{\text{MatrC}, \text{NomeC}, \text{StipC}, \text{EtàC} \leftarrow \text{Matr}, \text{Nome}, \text{Stip}, \text{Età}} (\text{Impiegati}) \right. \\ \left. \bowtie_{\text{MatrC} = \text{Capo}} \right. \\ \left. \text{Supervisione} \bowtie_{\text{Impiegato} = \text{Matr}} \sigma_{\text{Stipendio} > 40} (\text{Impiegati}) \right) \end{aligned}$$

Viene formulato richiedendo, per ciascuna tupla del risultato, l'esistenza di tre tuple:
una relativa a un impiegato che guadagna più di 40 mila euro,
una seconda che indica chi è il suo capo
un'ultima che fornisce informazioni di dettaglio sul capo:

$$\begin{aligned} \{\text{NomeC: } nc, \text{StipC: } sc \mid \text{Impiegati}(\text{Matr: } m, \text{Nome: } n, \text{Età: } e, \text{Stipendio: } s) \wedge s > 40 \\ \text{Supervisione}(\text{Impiegato: } m, \text{Capo: } c) \wedge \text{Impiegati}(\text{Matr: } c, \text{Nome: } nc, \text{Età: } ec, \text{Stipendio: } sc)\} \end{aligned}$$

5. *Trovare gli impiegati che guadagnano più del rispettivo capo, mostrando matricola, nome e stipendio di ciascuno di essi e del capo*

$$\begin{aligned} \{\text{Matr: } m, \text{Nome: } n, \text{Stip: } s, \text{MatrC: } c, \text{NomeC: } nc, \text{StipC: } sc \mid \\ \text{Impiegati}(\text{Matr: } m, \text{Nome: } n, \text{Età: } e, \text{Stip: } s) \wedge \text{Supervisione}(\text{Impiegato: } m, \text{Capo: } c) \wedge \\ \text{Impiegati}(\text{Matr: } c, \text{Nome: } nc, \text{Età: } ec, \text{Stip: } sc) \wedge s > sc\} \end{aligned}$$

6. *Trovare matricola e nome dei capi i cui impiegati guadagnano tutti più di 40 mila euro*

$$\begin{aligned} \pi_{\text{Matr}, \text{Nome}} \left(\text{Impiegati} \bowtie_{\text{Matr} = \text{Capo}} \left(\pi_{\text{Capo}} (\text{Supervisione}) \right. \right. \\ \left. \left. - \pi_{\text{Capo}} \left(\text{Supervisione} \bowtie_{\text{Imp} = \text{Matr}} \sigma_{\text{Stip} \leq 40} (\text{Impiegati}) \right) \right) \right) \end{aligned}$$

Nel calcolo dobbiamo utilizzare un quantificatore. Possiamo utilizzare un quantificatore esistenziale negato, trovando i capi per i quali non esiste un impiegato che guadagna non più di 40mila euro

$$\begin{aligned} \{\text{Matr: } c, \text{Nome: } n \mid \text{Impiegati}(\text{Matr: } c, \text{Nome: } n, \text{Età: } e, \text{Stip: } s) \wedge \text{Supervisione}(\text{Impiegato: } m, \text{Capo: } c) \\ \wedge \neg \exists m' \left(\exists n' \left(\exists e' \left(\exists s' (\text{Impiegati}(\text{Matr: } m', \text{Nome: } n', \text{Età: } e', \text{Stip: } s') \right. \right. \right. \right. \\ \left. \left. \left. \left. \wedge \text{Supervisione}(\text{Impiegato: } m', \text{Capo: } c) \wedge s' \leq 40 \right) \right) \right) \} \end{aligned}$$

3.2.2 Pregi e difetti del calcolo su domini

Pros:

21. Dichiаративо

Cons:

22. "Verboso", ci sono molte variabili
23. Esistono espressioni senza senso

Calcolo su domini VS Algebra relazionale

Il Calcolo Relazionale su Domini (DRC) e l'Algebra Relazionale (RA) sono "equivalenti"

- Per ogni espressione del DRC che sia indipendente dal dominio esiste un'espressione equivalente dell'RA
- Per ogni espressione dell'RA esiste un'espressione del DRC equivalente (e indipendente dal dominio)

3.2.3 Calcolo su tuple con dichiarazioni in range

Cosa dobbiamo fare per superare le limitazioni del calcolo su domini?

- Ridurre il numero di variabili.
Un buon modo per farlo è restringere le variabili alle tuple, una variabile per ogni tupla
- Tutti i valori dei dati devono venire dal database

Il *calcolo su tuple con dichiarazioni in range* soddisfa entrambi i requisiti

Le espressioni hanno la forma:

$$\{TargetList|RangeList|Formula\}$$

- *TargetList* = T (lista degli obiettivi dell'interrogazione)
Contiene elementi come $Y: x. Z$ (oppure $x. Z$ o addirittura $x.^*$)
Dove x variabile e Y e Z sequenze di attributi
 $x.^*$ si intende $X: x. X$ dove il range della variabile x è una relazione sull'insieme di attributi X
- *RangeList* = L : elenca le variabili libere della *Formula*, con i relativi range
Infatti, L è una lista di elementi del tipo $x(R)$ con x variabile e R nome di relazione
- *Formula* = f :
 - Atomi del tipo $x. A \theta c$ o $x_1. A_1 \theta x_2. A_2$
che confrontano, rispettivamente:
 - Il valore di x sull'attributo A con la costante c
 - Il valore di x_1 su A_1 con quello di x_2 su A_2
 - Connettivi come nel calcolo su domini
 - Quantificatori che associano i range alle relative variabili
 $\exists x(R)(f), \quad \forall x(R)(f)$
 $\exists x(R)(f)$ significa "esiste nella relazione R una tupla x che soddisfa la formula f "

Vediamo qualche esempio:

1. *Matricola, nome età e stipendio degli impiegati che guadagnano più di 40 mila euro*

$$\sigma_{\text{Stipendio} > 40}(\text{Impiegati})$$

$$\{\text{Matr: } m, \text{Nome: } n, \text{Età: } e, \text{Stipendio: } s | \text{Impiegati}(\text{Matr: } m, \text{Nome: } n, \text{Età: } e, \text{Stipendio: } s) \wedge s > 40\}$$

$$\{i.* | i(\text{Impiegati}) | i.\text{Impiegati} > 40\}$$

2. *Matricola, nome ed età degli impiegati che guadagnano più di 40 mila euro*

$$\pi_{\text{Matr}, \text{Nome}, \text{Età}} \left(\sigma_{\text{Stip} > 40} (\text{Impiegati}) \right)$$

$$\{ i. (\text{Matr}, \text{Nome}, \text{Età}) | i(\text{Impiegati}) | i.\text{Stip} > 40 \}$$

3. *Matricole dei capi degli impiegati che guadagnano più di 40 mila euro*

$$\{ x. \text{Capo} | i(\text{Impiegati}), s. (\text{Supervisione}) | i.\text{Matr} = s.\text{Impiegato} \wedge i.\text{Stipendio} > 40 \}$$

La formula prevede la congiunzione di due condizioni atomiche:

- condizione di join $(i.\text{Matr} = s.\text{Impiegato})$
- condizione di selezione $(i.\text{Stipendio} > 40)$

Nel caso di espressioni corrispondenti al join di una relazione con se stessa, abbiamo più variabili aventi la stessa relazione come range.

4. *Trovare nome e stipendio dei capi degli impiegati che guadagnano più di 40 mila euro*

$$\begin{aligned} & \{ \text{NomeC}, \text{StipC}, : i'. (\text{Nome}, \text{Stip}) | \\ & \quad i'(\text{Impiegati}), s(\text{Supervisione}), i(\text{Impiegati}) | \\ & \quad i'. \text{Matr} = s.\text{Capo} \wedge s.\text{Impiegato} = i.\text{Matr} \wedge i.\text{Stipendio} > 40 \} \end{aligned}$$

5. *Trovare gli impiegati che guadagnano più del rispettivo capo, mostrando matricola, nome e stipendio di ciascuno di essi e del capo*

$$\begin{aligned} & \{ i. (\text{Nome}, \text{Matr}, \text{Stip}), \text{NomeC}, \text{MatrC}, \text{StipC} : i'. (\text{Nome}, \text{Matr}, \text{Stip}) | \\ & \quad i(\text{Impiegati}), s(\text{Supervisione}), i'(\text{Impiegati}) | \\ & \quad i.\text{Matr} = s.\text{Impiegato} \wedge s.\text{Capo} = i'. \text{Matr} \wedge i.\text{Stipendio} > i'. \text{Stipendio} \} \end{aligned}$$

6. *Trovare matricola e nome dei capi i cui impiegati guadagnano tutti più di 40 mila euro*

$$\begin{aligned} & \{ i. (\text{Matr}, \text{Nome}) | i(\text{Impiegati}), s(\text{Supervisione}) \\ & \quad i.\text{Matr} = s.\text{Capo} \wedge \forall i'(\text{Impiegati})(\forall s'(\text{Supervisione}) \\ & \quad (\neg(s.\text{Capo} = s'.\text{Capo} \wedge s'.\text{Impiegato} = i'.\text{Matr}) \vee i'.\text{Stipendio} > 40)) \} \end{aligned}$$

Con quantificatori esistenziali negati:

$$\begin{aligned} & \{ i. (\text{Matr}, \text{Nome}) | i(\text{Impiegati}), s. (\text{Supervisione}) | \\ & \quad i.\text{Matr} = s.\text{Capo} \wedge \neg(\exists i'(\text{Impiegati})(\exists s'(\text{Supervisione}) \\ & \quad (s.\text{Capo} = s'.\text{Capo} \wedge s'.\text{Impiegato} = i'.\text{Matr} \wedge i'.\text{Stipendio} \leq 40))) \} \end{aligned}$$

Il calcolo su tuple con dichiarazioni in range non permette di esprimere tutte le interrogazioni che possono essere formulate in algebra relazionale (o, equivalentemente, nel calcolo su domini).

In particolare, le interrogazioni i cui risultati possono provenire indifferentemente da due o più relazioni (che in algebra realizziamo con l'operatore di unione) non possono essere espresse in questa versione del calcolo.

Per questo motivo, SQL, il linguaggio pratico effettivamente utilizzato per le interrogazioni, che è basato sul calcolo su tuple con dichiarazioni di range, prevede un costrutto esplicito di unione, per esprimere interrogazioni che altrimenti risulterebbero non esprimibili.

3.3 DATALOG

Concludiamo con un altro linguaggio di interrogazione per basi di dati.

Datalog si basa sull'idea di adattare alle basi di dati il linguaggio di programmazione logica *Prolog*.

Sintatticamente è una versione semplificata del Prolog, linguaggio basato sul calcolo dei predicati del primo ordine, ma con un approccio diverso rispetto al calcolo relazionale. In Datalog abbiamo due tipi di predicati:

- I predicati *estensionali*, che corrispondono alle relazioni nella base di dati
- I predicati *intensionali*, che sono specificati per mezzo di **regole** logiche.
Questi predicati definiscono viste sulla base di dati

3.3.1 Sintassi Datalog

Regole:

testa \leftarrow *corpo*

In cui:

- La *testa* è un predicato atomico simile a quelli usati nel calcolo relazionale su domini $R(A_1: a_1, \dots, A_p: a_p)$, dove però ciascuno degli a_i può essere una costante o una variabile
- Il *corpo* è una lista di condizioni atomiche dello stesso tipo e/o di condizioni di confronto fra variabili o fra variabili e costanti

Sono imposte le seguenti condizioni

- I predicati estensionali possono comparire solo nel corpo delle regole
- Se una variabile compare nella testa di una regola allora deve comparire anche nel corpo della stessa regola
- Se una variabile compare in un atomo di confronto allora deve comparir anche in un atomo del copro della stessa regola

Una caratteristica fondamentale del Datalog è la *ricorsività*: è possibile che un predicato intensionale sia definito in termini di se stesso (direttamente o indirettamente).

Le interrogazioni Datalog sono specificate per mezzo di atomi $R(A_1: a_1, \dots, A_p: a_p)$ preceduti da un punto interrogativo "?" per sottolineare che si tratta di interrogazioni; queste interrogazioni producono come risultato le tuple della relazione R che possono essere ottenute sostituendo correttamente le variabili.

Trovare matricola, nome età e salario degli impiegati di 30 anni

? *Impiegati*(**Matr**: *m*, **Nome**: *n*, **Età**: 30, **Stipendio**: *s*)

Trovare le matricole dei capi degli impiegati che guadagnano più di 40 mila euro

Definiamo un predicato intensionale *CapiDeiRicchi*, con la regola:

```
CapiDeiRicchi(Capo: c)  $\leftarrow$ 
    Impiegati(Matr: m, Nome: n, Età: e, Stipendio: s),
    Supervisione(Impiegato: m, Capo: c), s > 40
```

Per valutare un'interrogazione come questa è necessario definire la semantica delle regole.

L'idea di base è che il corpo di una regola va considerato come la congiunzione degli atomi che in esso compaiono, quindi la regola può essere valutata come un'espressione del calcolo su domini.

La regola definisce la relazione intensionale *CapiDeiRicchi* come costituita dalle stesse tuple che compaiono nel risultato dell'espressione qua sotto del calcolo

{Capo: *c* || *Impiegati*(**Matr**: *m*, **Nome**: *n*, **Età**: *e*, **Stipendio**: *s*) \wedge *Supervisione*(**Impiegato**: *m*, **Capo**: *c*) \wedge *s* > 40}

In modo analogo possiamo scrivere regole per molte delle interrogazioni viste in precedenza.

In assenza di definizioni ricorsive, la semantica del Datalog è quindi molto semplice, perché i vari predicati possono essere calcolati per mezzo di espressioni simili a quelle nel calcolo (e nell'algebra), perché non è disponibile un costrutto che corrisponda al quantificatore universale. In effetti si può dimostrare che:

24. Il Datalog non ricorsivo è equivalente al calcolo su domini senza negazione né quantificazione universale.

Per far acquistare al Datalog lo stesso potere espressivo del calcolo è necessario aggiungere alla struttura base la possibilità di includere nel corpo non solo condizioni atomiche, ma anche negazioni di condizioni atomiche.

Trovare matricola e nome dei capi i cui impiegati guadagnano tutti più di 40 mila euro

$$\left\{ \begin{array}{l} \text{Matr: } c, \text{Nome: } n \mid \text{Impiegati}(\text{Matr: } c, \text{Nome: } n, \text{Età: } e, \text{Stip: } s) \wedge \text{Supervisione}(\text{Impiegato: } m, \text{Capo: } c) \\ \wedge \neg \exists m' \left(\exists n' \left(\exists e' \left(\exists s' \left(\text{Impiegati}(\text{Matr: } m', \text{Nome: } n', \text{Età: } e', \text{Stip: } s') \right. \right. \right. \right. \right. \\ \left. \left. \left. \left. \left. \left. \wedge \text{Supervisione}(\text{Impiegato: } m', \text{Capo: } c) \wedge s' \leq 40 \right) \right) \right) \right) \end{array} \right\}$$

Definiamo un predicato per i capi che non soddisfano la condizione:

$$\begin{aligned} \text{CapiDiNonRicchi}(\text{Capo: } c) \leftarrow \\ \text{Supervisione}(\text{Impiegato: } m, \text{Capo: } c), \\ \text{Impiegati}(\text{Matr: } c, \text{Nome: } n, \text{Età: } e, \text{Stip: } s), s \leq 40 \end{aligned}$$

Quindi utilizziamo questo predicato in forma negata:

$$\begin{aligned} \text{CapiSoloDiRicchi}(\text{Matr: } c, \text{Nome: } n) \leftarrow \\ \text{Impiegati}(\text{Matr: } c, \text{Nome: } n, \text{Età: } e, \text{Stip: } s) \\ \text{Supervisione}(\text{Impiegato: } m, \text{Capo: } c), \\ \text{not CapiDiNonRicchi}(\text{Capo: } c) \end{aligned}$$

Si può dimostrare che:

25. Il Datalog non ricorsivo con negazione è equivalente al calcolo su domini

Utilizzando regole ricorsive si può ottenere un'espressività maggiore.

Per esempio, sempre sulla base di dati con le relazioni *Impiegati* e *Supervisione*, è possibile definire il predicato intensionale *Superiore*, che descrive per ogni impiegato il capo, il capo del capo e così via, senza limiti

$$\begin{aligned} \text{Superiore}(\text{Impiegato: } i, \text{SuperCapo: } c) \leftarrow \\ \text{Supervisione}(\text{Impiegato: } i, \text{Capo: } c) \\ \text{Superiore}(\text{Impiegato: } i, \text{SuperCapo: } c) \leftarrow \\ \text{Supervisione}(\text{Impiegato: } i, \text{Capo: } c'), \\ \text{Superiore}(\text{Impiegato: } c', \text{SuperCapo: } c) \end{aligned}$$

La seconda regola è ricorsiva perché definisce la relazione *Superiore* su di sé stessa. Per valutare questa regola, non possiamo procedere come visto finora, perché una singola valutazione del corpo non sarebbe sufficiente per calcolare completamente il predicato ricorsivo. Ci sono varie tecniche ma non le vedremo.

4 SQL: CONCETTI BASE

Acronimo che sta per "Structured Query Language"

È un linguaggio con tante funzionalità, implementa sia un DDL (Data Definition Language) sia quelle di un DML (Data Manipulation Language)

Esiste un linguaggio ISO standard ma diversi DBMS hanno la loro grammatica linguistica

Per il momento analizzeremo le basi di questo linguaggio

Unofficial Name	Official Name	Caratteristiche
SQL-Base	SQL-86	Costrutti base
	SQL-89	Integrità referenziale
SQL-2	SQL-92	Modello relazionale Vari costrutti nuovi 3 livelli: entry, intermediate, full
SQL-3	SQL:1999	Modello relazionale a oggetti Organizzato in diverse parti Trigger, funzioni esterne, ...
	SQL:2003	Estensioni del modello a oggetti Eliminazione di costrutti non usati Nuove parti: SQL/JRT, SQL/XML, ...
	SQL:2006	Estensione della parte XML
	SQL:2008	Lievi aggiunte (per esempio, trigger instead of)
	SQL:2011	Ricco supporto per dati temporali
	SQL:2016	Gestione del formato JSON

4.1 DEFINIZIONE DEI DATI

CREATE DATABASE:

Ogni database appena creato contiene tabelle, viste, trigger e altre cose

Per esempio:

```
CREATE DATABASE db_name
```

CREATE SCHEMA:

Uno schema SQL è identificato da un nome e descrive gli elementi che gli appartengono (tabelle, tipi, vincoli, viste, domini, ...). Lo schema apparirà all'utente che ha digitato il comando

Per esempio:

```
CREATE SCHEMA schema_name
```

Questo comando potrebbe essere seguito dalla parola chiave **AUTHORIZATION** per indicare un utente specifico a cui appartiene lo schema

Per esempio:

```
CREATE SCHEMA schema_name
AUTHORIZATION 'user_name'
```

CREATE TABLE:

Specifica una nuova relazione e crea la sua istanza vuota, specifica i suoi attributi (con i loro tipi) e i vincoli iniziali

```
CREATE TABLE EMPLOYEE (
    Number   CHARACTER(6) PRIMARY KEY,
    Name     CHARACTER(20) NOT NULL,
    Surname  CHARACTER(20) NOT NULL,
    Dept     CHARACTER(15),
    Wage     NUMERIC(9) DEFAULT 0,
    FOREIGN KEY(Dept) REFERENCES
                      DEPARTMENT(Dept),
    UNIQUE (Surname, Name)
)
```

4.2 TIPI DI DATI (ATTRIBUTI)

I tipi di dati (o attributi) in SQL corrispondono ai domini del calcolo relazionale

26. Tipi di dati *basilari* (già disponibili di default)
27. Tipi di dati *personalizzati* (chiamati “domini” semplici e riutilizzabili)

4.2.1 Tipi di dati basilari

- Character-string: tipi di dati che sono di lunghezza sia fissa che variabile
- Numeric: includono numeri interi e diversi floating point
- DATE, TIME, INTERVAL
- Boolean
- BLOB, CLOB (Oggetto binario/carattere grande): rappresentano enormi raccolte di dati (testuali o no)

4.2.2 Tipi di dati personalizzati**CREATE DOMAIN:**

Ogni tipo di dati personalizzato potrebbe essere utilizzato durante la definizione di nuove relazioni, indicando vincoli e valori predefiniti

```
CREATE DOMAIN Grade (
    AS SMALLINT DEFAULT NULL
    CHECK ( value >= 18 AND value <= 30 )
```

4.3 VINCOLI DI TABELLA

- **NOT NULL**
- **UNIQUE** definisce le chiavi
- **PRIMARY KEY**: (solo uno, implica **NOT NULL**; DB2 non ha un comportamento standard)
- **CHECK**, lo vedremo più avanti

4.3.1 UNIQUE e PRIMARY KEY

Possono essere usati quando:

- › Definiamo un attributo che definisce la chiave

```
CREATE TABLE EMPLOYEE (
    Number  CHARACTER(6) PRIMARY KEY,
    Name    CHARACTER(20) NOT NULL,
    Surname CHARACTER(20) NOT NULL,
```

- › Come elemento autonomo

```
Number      CHARACTER(6) PRIMARY KEY,
Number      CHARACTER(6)
...
PRIMARY KEY (Number)
```

```
CREATE TABLE EMPLOYEE (
    Number      CHARACTER(6) PRIMARY KEY,
    Name        CHARACTER(20) NOT NULL,
    Surname     CHARACTER(20) NOT NULL,
    Dept        CHARACTER(15),
    Wage        NUMERIC(9) DEFAULT 0,
    FOREIGN KEY(Dept) REFERENCES
        DEPARTMENT(Dept),
    UNIQUE (Surname, Name)
)
```

4.3.2 Key e vincoli di integrità referenziale

REFERENCES e **FOREIGN KEY** definiscono i vincoli di integrità referenziale

Possono essere definiti:

- › Su di un singolo attributo
- › Su molteplici attributi

Possiamo definire *azioni innescate referenziali* quando questi vincoli vengono violati

Infrazioni	Codice	Data	Agente	Stato	Targa
	34321	95/02/01	3987	IT	AG548UK
	53524	95/03/04	3295	IT	TE395AB
	64521	96/04/05	3295	FR	ZT395AB
	73321	98/02/05	9345	FR	ZT395AB

Agente	Id	Cognome	Nome	Auto	Stato	Targa	Cognome	Nome
	3987	Rossi	Luca		IT	AG548UK	Verdi	Giuseppe
	3295	Neri	Piero		IT	TE395AB	Verdi	Giuseppe
	9345	Neri	Mario		FR	ZT395AB	Quinault	Philippe
	7543	Mori	Gino					

```
CREATE TABLE EMPLOYEE (
    Code CHARACTER(6) PRIMARY KEY,
    Day DATE NOT NULL,
    Officer INTEGER NOT NULL
        REFERENCES OFFICER(Id),
    State CHARACTER(2),
    Number CHARACTER(6),
    FOREIGN KEY(State, Number)
        REFERENCES CAR(State, Number)
)
```

Dopo ogni vincolo referenziale possiamo specificare l'invocazione di una triggered action (delete, update) se l'operazione viene rifiutata

```
ON < DELETE | UPDATE >
  < CASCADE | SET NULL |
    SET DEFAULT | NO ACTION >
```

DELETE:

- **CASCADE**: elimina le tuple di riferimento
- **SET NULL**: i valori dell'attributo di riferimento eliminato viene sostituito con NULL
- **SET DEFAULT**: i valori dell'attributo di riferimento eliminato viene sostituito con il valore specificato di default
- **NO ACTION**: Non è consentita alcuna rimozione

UPDATE:

- **CASCADE**: il valore degli attributi chiave esterni di riferimento viene aggiornato col nuovo valore
- **SET NULL**: il valore dell'attributo di riferimento interessato è sostituito con NULL
- **SET DEFAULT**: il valore dell'attributo di riferimento interessato è sostituito con il valore specificato
- **NO ACTION**: nessun update è consentito

4.3.3 Dichiarazioni di modifica dello schema

- **ALTER DOMAIN**
- **ALTER TABLE**
- **DROP DOMAIN**
- **DROP TABLE**

ALTER DOMAIN:

Consente di cambiare domini definiti in precedenza

Questa dichiarazione dev'essere usata assieme a queste altre:

- **SET DEFAULT**
- **DROP DEFAULT**
- **ADD CONSTRAINT**
- **DROP CONSTRAINT**

Esempi:

```
ALTER DOMAIN Grade SET DEFAULT 30
```

Imposta il voto base **Grade** a **30**. Questo comando viene applicato solo quando viene invocato il comando e non si trova il valore del voto.

```
ALTER DOMAIN Grade DROP DEFAULT
```

Rimuove il valore di default di **Grade**

```
ALTER DOMAIN Grade
ADD CONSTRAINT isValid
CHECK (value >=18 AND value <= 30)
```

Aggiunge il vincolo **isValid** al tipo di dato **Grade**

```
ALTER DOMAIN Grade DROP CONSTRAINT isValid
```

Rimuove il vincolo associato al tipo di dato

ALTER DOMAIN:

Esegue cambiamenti a tabelle definite in precedenza.
Questa dichiarazione dev'essere usata assieme a queste altre:

- › ALTER COLUMN
- › ADD COLUMN
- › DROP COLUMN
- › DROP CONSTRAINT
- › ADD CONSTRAINT

Esempi:

1. ALTER TABLE EMPLOYEE
2. ALTER COLUMN Number SET NOT NULL
Number dalla tabella EMPLOYEE non possono avere valori nulli.
1. ALTER TABLE EMPLOYEE
2. ADD COLUMN Level CHARACTER(10)
Un attributo Level è aggiunto alla tabella EMPLOYEE.
1. ALTER TABLE EMPLOYEE
2. DROP COLUMN Level RESTRICT
Rimuove l'attributo Level da EMPLOYEE <u>solo</u> se non contiene valori.

1. ALTER TABLE EMPLOYEE
2. DROP COLUMN Level CASCADE

Rimuove l'attributo Level da EMPLOYEE assieme ai suoi valori.

1. ALTER TABLE EMPLOYEE
2. ADD CONSTRAINT validNum
3. CHECK (char_length(Number) = 10)

Aggiunge il vincolo validNum all'attributo Number da EMPLOYEE.

1. ALTER TABLE EMPLOYEE
2. DROP CONSTRAINT validNum

Rimuove il vincolo definito prima.

DROP DOMAIN:

Rimuove un tipo di dato definito dall'utente

Esempio:

DROP DOMAIN OFFENCES

DROP TABLE:

Rimuove tutte le istanze di una tabella assieme coi suoi schemi e dati

Esempio:

DROP TABLE OFFENCES

4.3.4 Definire gli indici

Di solito migliorano il tempo di interrogazione, rilevante per l'efficienza di calcolo.

Sono definiti a livello fisico, non logico

Un tempo era l'unico modo per definire le chiavi

CREATE INDEX

Esempio:

CREATE INDEX idx_Surname
ON OFFICER (Surname)

Crea l'indice idx_Surname sull'attributo Surname dalla tabella OFFICER.

4.3.5 DDL nel pratico

In molti sistemi e progetti, per definire uno schema di database, vengono utilizzati diversi strumenti anziché comandi SQL. (Per esempio strumenti con un'interfaccia utente grafica)

4.4 OPERAZIONI IN SQL: SELECT

4.4.1 Come interpretare SELECT

3. **SELECT Number, Name**
 1. **FROM OFFICER**
 2. **WHERE Surname = 'Jones'**

1. Dalla tabella **OFFICER**
2. Porta gli ufficiali che hanno '**Jones**' come attributo **Surname**
3. Mostrando per ogni tupla sia **Number** che **Name**.

4.4.1.1 Scorcioatoie Select

```
SELECT *
FROM PEOPLE
WHERE Age < 30
```

```
SELECT Name, Age, Income
FROM PEOPLE
WHERE Age < 30
```

4.4.1.2 Comando Select Base

```
SELECT <AttributeList>
FROM <TableList>
[ WHERE <Condition> ]
```

4.4.1.3 Esempio su Database

PERSONE

NOME	ETÀ	REDDITO
Jim	27	21
James	25	15
Alice	55	42
Jesse	50	35
Phil	26	30
Louis	50	40
Frank	60	20
Olga	30	41
Steve	85	35
Abby	75	87

MATERNITÀ

MADRE	FIGLIO
Abby	Alice
Abby	Louis
Jesse	Olga
Jesse	Phil
Alice	Jim
Alice	James

PATERNITÀ

PADRE	FIGLIO
Steve	Frank
Louis	Olga
Louis	Phil
Frank	Jim
Frank	James

4.4.2 Selezione e proiezione

Riporta nome e Guadagno delle persone
sotto i 30 anni

 $\pi_{\text{Nome}, \text{Reddito}}(\sigma_{\text{Età} < 30}(\text{PERSONE}))$

```
SELECT Nome, Reddito
FROM PEOPLE
WHERE Età < 30
```

PERSONE

NOME	ETÀ	REDDITO
Jim	27	21
James	25	15
Alice	55	42
Jesse	50	35
Phil	26	30
Louis	50	40
Frank	60	20
Olga	30	41
Steve	85	35
Abby	75	87

PERSONE

NOME	REDDITO
Jim	21
James	15
Phil	30

⇒

4.4.2.1 Renominazione degli attributi

```
SELECT P.Nome AS NomeData
      P.Reddito AS Revenue
  FROM PEOPLE AS P
 WHERE P.Età < 30
```

Nome	Reddito	NomeData	Revenue
Jim	21	Jim	21
James	15	James	15
Phil	30	Phil	30

4.4.3 Selezione pura

Riporta Nome, Età e Guadagno delle persone sotto i 30 anni

 $\sigma_{\text{Età} < 30}(\text{PERSONE})$

```
SELECT *
  FROM PERSONE
 WHERE Età < 30
```

PERSONE

NOME	ETÀ	REDDITO
Jim	27	21
James	25	15
Alice	55	42
Jesse	50	35
Phil	26	30
Louis	50	40
Frank	60	20
Olga	30	41
Steve	85	35
Abby	75	87

NOME	ETÀ	REDDITO
Jim	27	21
James	25	15
Phil	26	30

4.4.4 Proiezione senza selezione

Riporta i nomi e i guadagni di tutte le persone

 $\pi_{\text{Nome}, \text{Reddito}}(\text{PERSONE})$

```
SELECT Nome, Reddito
  FROM PERSONE
```

PERSONE

NOME	ETÀ	REDDITO
Jim	27	21
James	25	15
Alice	55	42
Jesse	50	35
Phil	26	30
Louis	50	40
Frank	60	20
Olga	30	41
Steve	85	35
Abby	75	87

NOME	REDDITO
Jim	21
James	15
Alice	42
Jesse	35
Phil	30
Louis	40
Frank	20
Olga	41
Steve	35
Abby	87

4.4.4.1 Scorcioiole Select pt.2

Data una relazione $R(A, B)$

Corrisponde a

```
SELECT *
  FROM R
```

```
SELECT X.A AS A, X.B AS B
  FROM R AS X
 WHERE true
```

4.4.5 Condizioni composite

```
SELECT *
  FROM PERSONE
 WHERE Reddito > 25 AND
       (Età < 30 OR Età > 60)
```

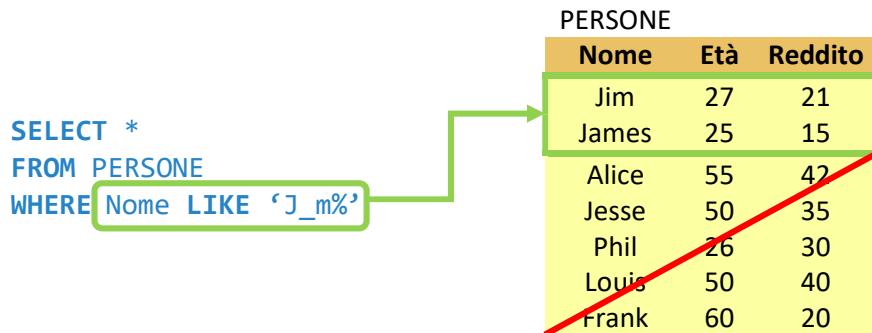
PERSONE

Nome	Età	Reddito
Phil	26	30
Frank	60	20
Olga	30	41
Steve	85	35

Nome	Età	Reddito
Phil	26	30
Steve	85	35

4.4.6 Operatore 'Like'

Restituisce le persone che hanno un nome che inizia con 'J' e hanno 'm' come terza lettera:



4.4.7 Gestione dei valori nulli

IMPIEGATI

Numero	Cognome	Agenzia	Età
5998	Neri	Milan	45
9553	Bruni	Milan	NULL

Restituire gli impiegati che abbiano almeno 40 anni oppure un valore NULL

$\sigma_{(\text{Età}>40) \text{ OR } (\text{Età è NULL})}(\text{IMPIEGATI})$

```

SELECT *
FROM IMPIEGATI
WHERE Età>40 OR Età IS NULL
    
```

4.4.8 Proiezione

IMPIEGATI

Numero	Cognome	Agenzia	Età
5998	Neri	Milano	45
9553	Rossi	Roma	44
7309	Neri	Napoli	55
6598	Rossi	Roma	64

Restituire il cognome e l'Agenzia per tutti gli impiegati

$\pi_{\text{Cognome}, \text{Agenzia}}(\text{IMPIEGATI})$

```

SELECT
    Cognome, Agenzia
FROM IMPIEGATI
    
```

Cognome	Agenzia
Neri	Napoli
Neri	Milano
Rossi	Roma
Rossi	Roma

```

SELECT DISTINCT
    Cognome, Agenzia
FROM IMPIEGATI
    
```

Cognome	Agenzia
Neri	Napoli
Neri	Milano
Rossi	Roma

4.4.9 Select, Project, Join

Con una sola relazione nella clausola **FROM**, una singola query SQL può esprimere: *select*, *project* e *rename*

Utilizzando più relazione nella clausola **FROM** abbiamo *joins* (e prodotti cartesiani)

R1(A1,A2) R2(A3,A4)

```

SELECT DISTINCT R1.A1, R2.A4
FROM R1, R2
WHERE R1.A2 = R2.A3
    
```

28. Prodotto cartesiano (**FROM**)

29. Selezione (**WHERE**)

30. Proiezione (**SELECT**)

$\pi_{\text{A1}, \text{A4}}(\sigma_{\text{A2}=\text{A3}}(\text{R1} \bowtie \text{R2}))$

4.4.9.1 SQL: valutare le interrogazioni

SQL è un linguaggio dichiarativo, forniamo la semantica tramite esempi.

I DBMS hanno *query execution plans* per valutazioni efficienti

31. Le selezioni vengono eseguite il prima possibile
32. Quando possibile vengono eseguiti join piuttosto che prodotti cartesiani

4.4.9.2 SQL: Formulare le interrogazioni

Non dobbiamo per forza scrivere query efficienti poiché i DBMS hanno incorporati dei query optimizer.

Quindi è più importante che le query siano facili da capire (evitando errori nel formulare la query)

4.4.9.2.1 Esempi:

1. Padri delle persone che guadagnano più di 20

$$\pi_{\text{Padre}}(\text{PATERNITÀ} \bowtie_{\text{Figlio} = \text{Nome}} \sigma_{\text{Reddito} > 20}(\text{PERSONE}))$$

```
SELECT DISTINCT Padre
FROM PERSONE, PATERNITÀ
WHERE Nome=Figlio AND Reddito>20
```

2. Restituire il nome delle persone, il reddito e il reddito dei loro padri, se queste persone guadagnano più dei loro padri

$$\begin{aligned} \pi_{\text{Nome}, \text{Reddito}, \text{RP}} \left(\sigma_{\text{Reddito} > \text{RP}} \left(\rho_{\text{NP}, \text{EP}, \text{RP} \leftarrow \text{Nome}, \text{Età}, \text{Reddito}}(\text{PERSONE}) \right) \right) \\ \bowtie_{\text{NP} = \text{Padre}} \\ \left(\text{PATERNITÀ} \bowtie_{\text{Figlio} = \text{Nome}} (\text{PERSONE}) \right) \end{aligned}$$

```
SELECT C.Nome, C.Reddito, F.Reddito
FROM PERSONE F, PATERNITÀ, PERSONE C
WHERE F.Nome=Padre AND Figlio=C.Nome
AND C.Reddito > F.Reddito
```

4.4.9.3 Select con Renaming

```
SELECT C.Nome AS Nome
C.Reddito AS Reddito
F.Reddito AS redditoPadre
FROM PERSONE F, PATERNITÀ, PERSONE C
WHERE F.Nome = Padre AND
Figlio = C.Nome AND
C.Reddito > F.Reddito
```

4.4.9.4 Utilizzare espressioni nella Target list

SELECT Reddito/2 AS redditoDimez
FROM PERSONE
WHERE Nome = 'Louis'

PERSONE		
Nome	Età	Reddito
Jim	27	21
James	25	15
Alice	55	42
Louis	50	40

4.4.10 Comando JOIN

Restituire la madre e il padre di ogni persona

33. JOIN implicito

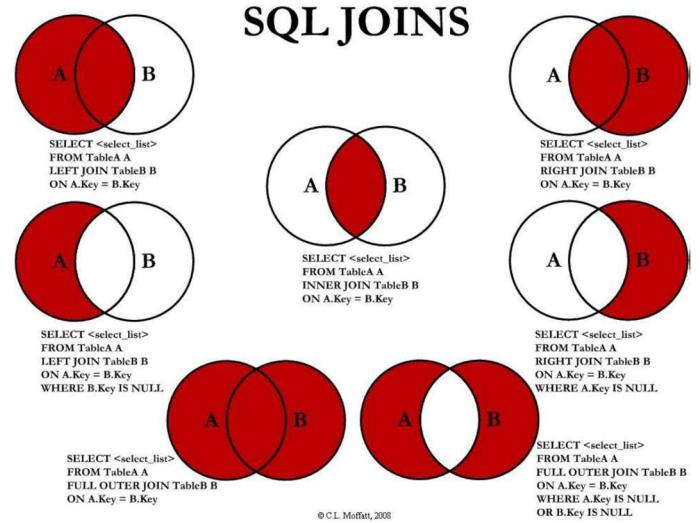
```
SELECT F.Figlio, Padre, Madre
FROM MARERNITÀ M, PATERNITÀ F
WHERE M.Figlio = F.Figlio
```

34. JOIN esplicito

```
SELECT Madre, PATERNITÀ.Figlio, Padre
FROM MARERNITÀ JOIN PATERNITÀ ON
PATERNITÀ.Figlio=MATERNITÀ.Figlio
```

SELECT ...

```
FROM LeftTable { ... JOIN RightTable
ON joincondition }, ...
[ WHERE otherPredicate ]
```



4.4.10.1 Join naturale

$$\pi_{\text{Figlio}, \text{Padre}, \text{Madre}}(\text{PATERNITÀ} \bowtie_{\text{Figlio} = \text{Nome}} \rho_{\text{Nome} \leftarrow \text{Figlio}}(\text{MATERNITÀ})) \\ \text{PATERNITÀ} \bowtie \text{MATERNITÀ}$$

```
SELECT Madre, PATERNITÀ.Figlio, Padre
FROM MATERNITÀ JOIN PATERNITÀ ON
PATERNITÀ.Figlio = MATERNITÀ.Figlio
```

```
SELECT Madre, .Figlio, Padre
FROM MATERNITÀ NATURAL JOIN PATERNITÀ
```

4.4.10.2 Join esterno

Coi Join precedente, anche detti *inner joins*, slacune tuple potrebbero essere scartate dal risultato finale; ciò accade se non hanno una tupla corrispondente nell'altra tabella

Per evitare questa perdita di informazioni possiamo usare:

LEFT/RIGHT/FULL OUTER JOIN

Quando questi join sono sia left o right, la parola chiave "**OUTER**" può essere omessa perché left e right sono "outer" per definizione

4.4.10.3 Join esterno sinistro

Ritornare il padre e la madre se conosciuti

```
SELECT PATERNITÀ.Figlio, Padre, Madre
FROM MATERNITÀ LEFT [OUTER] JOIN
MATERNITÀ ON PATERNITÀ.Figlio
MATERNITÀ.Figlio
```

PATERNITÀ.Figlio	Padre	Madre
Frank	Steve	NULL
Olga	Louis	Jesse
Phil	Louis	Jesse
Jim	Frank	Alice
James	Frank	Alice

4.4.11 Ordinare i risultati

Restituire il nome e il reddito delle persone che hanno meno di 30 anni in ordine alfabetico

```
SELECT Nome, Reddito
FROM PERSONE
WHERE Età < 30
ORDER BY Nome ASC
```

35. **ASC** ordine crescente (default)

36. **DESC** ordine decrescente

4.4.11.1 Esempio:

PERSONE		
NOME	ETÀ	REDDITO
Jim	27	21
James	25	15
Alice	55	42
Jesse	50	35
Phil	26	30
Louis	50	40
Frank	60	20
Olga	30	41
Steve	85	35
Abby	75	87

```
SELECT Nome, Reddito
FROM PERSONE
WHERE Età < 30
```

NOME	REDDITO
Jim	21
James	15
Phil	30

```
SELECT Nome, Reddito
FROM PERSONE
WHERE Età < 30
ORDER BY Nome
```

NOME	REDDITO
James	15
Jim	21
Phil	30

```
SELECT Nome, Reddito
FROM PERSONE
WHERE Età < 30
ORDER BY Nome DESC
```

NOME	REDDITO
Phil	30
Jim	21
James	15

4.4.12 Unione, Intersezione, Differenza

L'operatore **SELECT** richiede un comando specifico per eseguire le unioni:

```
SELECT ...
UNION [ALL]
SELECT ...
```

Nel risultato le righe sono univoche (Tranne quando si usa **ALL**. In questo caso abbiamo un'unione multiset)

4.4.12.1 Set Union

```
SELECT Figlio
FROM MATERNITÀ
UNION
SELECT Figlio
FROM PATERNITÀ
```

MATERNITÀ		PATERNITÀ	
MADRE	FIGLIO	PADRE	FIGLIO
Abby	Alice	Steve	Frank
Abby	Louis	Louis	Olga
Jesse	Olga	Louis	Phil

Figlio

Alice
Louis
Olga
Frank
Phil

4.4.12.2 Multiset Union

```
SELECT Figlio
FROM MATERNITÀ
UNION ALL
SELECT Figlio
FROM PATERNITÀ
```

MATERNITÀ		PATERNITÀ	
MADRE	FIGLIO	PADRE	FIGLIO
Abby	Alice	Steve	Frank
Abby	Louis	Louis	Olga
Jesse	Olga	Louis	Phil

Figlio
Alice
Louis
Olga
Frank
Olga
Phil

Olga compare due volte

4.4.12.3 Notazione posizionale

```
SELECT Padre, Figlio
FROM PATERNITÀ
UNION
SELECT Madre, Figlio
FROM MATERNITÀ
```

Come possiamo risolvere il conflitto con la rinominazione quando due tabelle hanno uno schema diverso?

- 37. Che sia fittizio o meno
- 38. Si assumono sempre i nomi del primo operando
- 39. Unendo gli attributi in conflitto

Padre	Figlio
Steve	Frank
Louis	Olga
Louis	Phil
Frank	Jim
Frank	James
Abby	Alice
Abby	Louis
Jesse	Olga
Jesse	Phil
Alice	Jim
Alice	James

```
SELECT Padre, Figlio
FROM PATERNITÀ
UNION
SELECT Figlio, Madre
FROM MATERNITÀ
```

```
SELECT Padre, Figlio
FROM PATERNITÀ
UNION
SELECT Madre, Figlio
FROM MATERNITÀ
```

Lo schema risultante delle tabelle risultanti in entrambi i casi è (Padre, Figlio)

4.4.12.4 Differenza

```
SELECT Nome
FROM IMPIEGATI
EXCEPT
SELECT Cognome AS Nome
FROM IMPIEGATI
```

Potremmo successivamente esprimere questo operatore attraverso query di selezione nidificate

4.4.12.5 Intersezione

```
SELECT Nome
FROM IMPIEGATI
INTERSECT
SELECT Cognome AS Nome
FROM IMPIEGATI
```

Corrisponde a scrivere

```
SELECT I.Nome
FROM IMPIEGATI I, IMPIEGATI F
WHERE E.Nome = F.Cognome
```

4.4.13 Query nidificate

I predicati ci consentono di:

40. Confrontare uno (o più) attributi con il risultato di una query ("sotto") nidificata
41. Usando il quantificatore esistenziale (*exists*, \exists)

4.4.13.1.1 Esempio:

Resistuire il nome e il reddito del padre di Frank

```
SELECT Nome, Reddito
FROM PERSONE, PATERNITÀ
WHERE Nome=Padre AND
      Figlio='Frank'
```

```
SELECT Nome, Reddito
FROM PERSONE
WHERE Nome=(SELECT Padre
              FROM PATERNITÀ
              WHERE Figlio='Frank'))
```

- Prodotto cartesiano e **WHERE** (equi-join)
- La condizione **WHERE** è vera quando il risultato della sotto-query è uguale a Nome.
- In più, solo una tupla viene processata dalla sotto-query
- Le query nidificate sono "meno dichiarative", ma a volte più leggibili poiché richiedono meno variabili
- Annidate e non-annidate possono essere combinate
- Le "sotto-query" all'interno di quelle nidificate non possono esprimere operazioni di set (l'unione può essere eseguita all'interno della query esterna) questo limite non è significativo
- Gli operatori di confronto richiedono valori singoli come operandi. È necessaria una soluzione per confrontare un valore con il risultato di una query (per esempio, una relazione)

4.4.13.2 ANY & ALL

Le query nidificate possono essere formulate attraverso predicati usando **ANY** oppure **ALL** assieme a un operatore di confronto ($<$, $>$, $=$, \geq , ...), risolvendo il problema di omogeneità

Attribute op ANY(Expr)

Una tupla di query esterna fa match se soddisfa il predicato rispetto a qualsiasi tupla all'interno di Expr

Attribute op ALL(Expr)

Una tupla di query esterna fa match se soddisfa il predicato rispetto a tutte le tuple all'interno di Expr

4.4.13.3 IN

Attribute op IN(Expr)

Una tupla di query esterna fa match se il suo valore in **Attribute** è presente tra gli elementi riportati da *Expr*

ANY, **ALL** e **IN** possono essere negati ponendo la word **NOT** subito prima

- A **IN(Expr)** \equiv A = **ANY(Expr)**

- A **NOT IN(Expr)** \equiv A \neq **ALL(Expr)**

4.4.13.3.1 Esempio 2

Resitisci nome e reddito dei padri i cui figli guadagnano più di 20

```
SELECT DISTINCT F.Nome, F.Reddito
FROM PERSONE F, PATERNITÀ, PERSONE C
WHERE F.Nome = PATERNITÀ.Padre AND
      PATERNITÀ.Figlio = C.Nome AND
      C.Reddito > 20
```

```
SELECT Nome, Reddito
FROM PERSONE
WHERE Nome IN (SELECT Padre
                  FROM PATERNITÀ
                  WHERE Figlio=ANY(SELECT Nome
                                    FROM PERSONE
                                    WHERE REDDITO>20))
```

Possiamo riscriverlo senza **DISTINCT**, perché non uniremo le tabelle, quindi i nomi dei padri non verranno ripetuti per ogni figlio:

4.4.13.3.2 Esempio 3

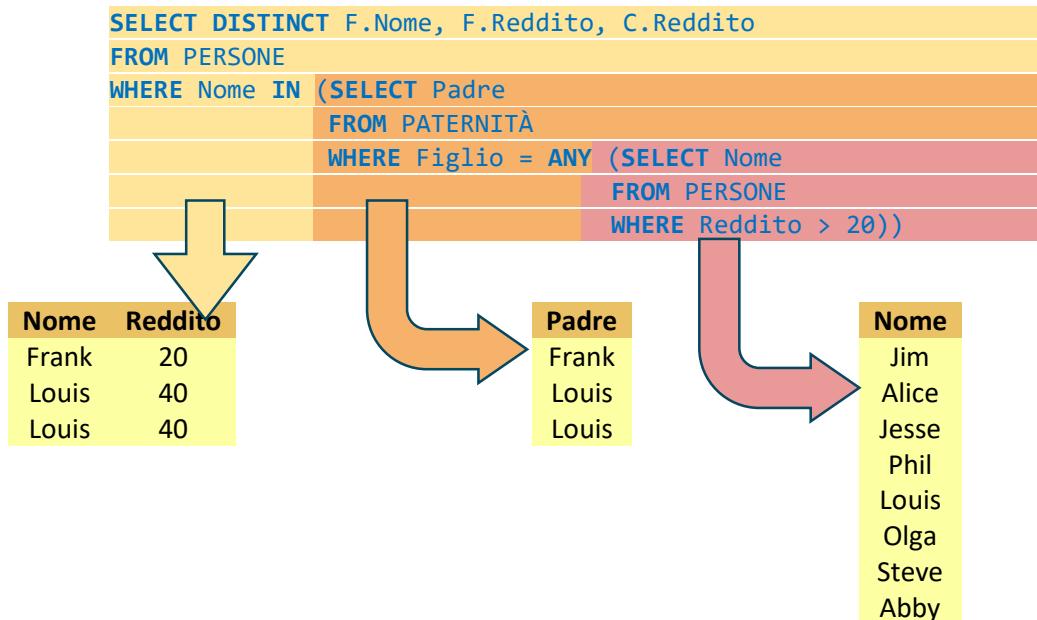
Restituire nome e reddito dei padri i cui figli guadagnano più di 20, e restituire anche il reddito del figlio

```
SELECT DISTINCT F.Nome, F.Reddito, C.Reddito
FROM PERSONE F, PATERNITÀ, PERSONE C
WHERE F.Nome = Padre AND Figlio = C.Nome AND C.Reddito > 20
```

Questa query restituisce la stessa risposta?

```
SELECT DISTINCT F.Nome, F.Reddito, C.Reddito
FROM PERSONE
WHERE Nome IN (SELECT Padre
                FROM PATERNITÀ
                WHERE Figlio = ANY (SELECT Nome
                                     FROM PERSONE
                                     WHERE Reddito > 20))
```

Significato di ANY: la condizione è vera se il valore di Figlio è uguale a uno qualsiasi dei valori restituiti dalla query nidificata



42. **Risposta:** no, perché per ogni padre noi non vediamo il reddito del figlio.

4.4.13.4 Considerazioni sulle query annidate:

- Regole di visibilità
 - Non è possibile riferirsi a variabili dichiarate in un blocco interno
 - Se il nome di una variabile è omesso assumiamo di prendere quello dichiarato “più vicino”
- Possiamo riferirci a una variabile definita:
 - Nello scope della query nella quale viene definita (eg., blocchi esterni)
 - Oppure nello scope della query annidata, a qualsiasi livello (eg., blocco interno) al suo interno

4.4.13.5 Semantica delle query nidificate con variabili

La query più interna viene eseguita una volta per ogni tupla all'interno della query esterna.

L'unico modo per evitare ciò è creando una vista, che, tuttavia, modifica lo schema del database.

4.4.14 Quantificatore esistenziale

EXISTS (Expr)

Il predicato è vero se *Expr* ritorna almeno una tupla

Di solito *Expr* è una query annidata

Utile con una variabile di collegamento tra la query esterna e la query nidificata

4.4.14.1.1 Esempio 1

Persone che hanno almeno un figlio

```
SELECT *
FROM PERSONE
WHERE EXISTS (SELECT *
               FROM PATERNITÀ
               WHERE Padre = Nome)
OR
EXISTS (SELECT *
        FROM MATERNITÀ
        WHERE Madre = Nome)
```

La colonna **Nome** è presa dalla relazione dichiarata nel blocco esterno

4.4.14.1.2 Esempio 2

Padri i cui figli guadagnano tutti più di 20

```
SELECT DISTINCT Padre
FROM PATERNITÀ Z
WHERE NOT EXISTS (SELECT *
                   FROM MATERNITÀ, W
                   PERSONE
                   WHERE W.Padre = Z.Padre
                         AND W.Figlio = Nome
                         AND Reddito <= 20)
```

Z.Padre è preso dalla relazione ottenuta nel blocco esterno

4.4.14.1.3 Errore

Persone della stessa età e reddito

```
SELECT *
FROM PERSONE
WHERE EXISTS (SELECT *
               FROM PERSONE
               WHERE Età = Età AND
                     Reddito = Reddito)
```

```
SELECT *
FROM PERSONE P
WHERE EXISTS (SELECT *
               FROM PERSONE
               WHERE P.Nome ≠ Nome AND
                     P.Età = Età AND
                     P.Reddito = Reddito)
```

Regola di scope: **Età** e **Reddito**, senza riferimento alla tabella si riferiscono implicitamente alla clausola **FROM** più vicina.

4.4.14.1.4 Visibilità sbagliata

```
SELECT *
FROM IMPIEGATI
WHERE Dept IN (SELECT Nome
          FROM DIPARTIMENTO D1
          WHERE Nome='Produzione')
OR
Dept IN (SELECT Nome
          FROM DIPARTIMENTO D2
          WHERE D2.Città = D1.Città)
```

Sbagliato perché nell'ultima **SELECT** la città di **D1** non è visibile.

4.4.14.2 Impostare le differenze e query annidate

```
SELECT Nome
FROM IMPIEGATI
EXCEPT
SELECT Cognome AS Nome
FROM IMPIEGATI

SELECT I.Nome
FROM IMPIEGATI I
WHERE NOT EXISTS (SELECT *
          FROM IMPIEGATI
          WHERE Cognome = I.Nome)
```

4.4.14.3 Posizioni di query annidate

- # clausola **WHERE**: utilizzo standard, ne abbiamo visto molti usi
- # clausola **FROM**: è necessaria una nuova fonte di dati (eg. una relazione), l'alternativa sarebbe creare una vista che, però, modificherebbe lo schema del database
- # clausola **SELECT**: utilizzo insolito, equivale al join. Ha bisogno di una tupla come risultato

4.4.14.3.1 Query annidate nella clausola FROM

*Fornire il nome e il reddito
dei figli di Jim*

```
SELECT Nome, Reddito
FROM PERSONE P, (SELECT Figlio)
          FROM PATERNITÀ
          WHERE Padre = 'Jim'
          AS FIGLIOJIM
WHERE Nome = FIGLIOJIM.Figlio
```

4.4.14.3.2 Query annidate nella clausola SELECT

*Restituire le spese di spedizione per ciascun cliente
nella tabella dei clienti*

```
SELECT CLIENTI.Nome
(SELECT SUM(CostiSpedizione)
          FROM ORDINI
          WHERE CLIENTI.Num =
ORDINI.Num)
          AS TotCostiSpedizione
FROM CLIENTI
```

La query può essere riscritta semplicemente
usando il join tra le due tabelle

4.4.15 Funzione 'Aggregate'

Nella target list possiamo inserire espressioni che calcolano valori da un insieme di tuple attraverso le funzioni aggregate

- # Aggr: COUNT | MIN | MAX | AVG | SUM
- # Sintassi base:

*Aggr([DISTINCT] *)
Aggr([DISTINCT] Attribute)*

4.4.15.1 COUNT

Il numero dei figli di Frank

```
SELECT COUNT(*) AS NumFigliFrank
FROM PATERNITÀ
WHERE Padre = 'Frank'
```

La funzione aggregata (COUNT) è applicata alle tuple del seguente risultato:

```
SELECT *
FROM PATERNITÀ
WHERE Padre = 'Frank'
```

PATERNITÀ	
Padre	Figlio
Steve	Frank
Louis	Olga
Louis	Phil
Frank	Jim
Frank	James

NumFigliFrank
2

4.4.15.2 COUNT DISTINCT

PERSONE

Nome	Età	Reddito
Jim	27	30
James	25	24
Alice	55	36
Jesse	50	36

```
SELECT COUNT(*)
FROM PERSONE
```

COUNT(*)
4

```
SELECT COUNT(DISTINCT Reddito)
FROM PERSONE
```

COUNT(DISTINCT Reddito)
3

4.4.15.3 Altre funzioni aggregate

- # SUM, AVG, MAX, MIN
- # Media del reddito dei figli di Frank

```
SELECT AVG(Reddito)
FROM PERSONE JOIN PATERNITÀ ON
    Nome = Figlio
WHERE Padre = 'Frank'
```

4.4.15.4 Funzioni aggregate con valori NULL

PERSONE

Nome	Età	Reddito
Jim	27	30
James	25	NULL
Alice	55	36
Jesse	50	36

```
SELECT COUNT(*)
FROM PERSONE
```

COUNT(*)
4

```
SELECT COUNT(Reddito)
FROM PERSONE
```

COUNT(Reddito)
3

```
SELECT COUNT(Reddito)
FROM PERSONE
```

COUNT(DISTINCT Reddito)
2

```
SELECT AVG(Reddito) AS AvgRed
FROM PERSONE
```

AvgRed
2

4.4.15.5 Funzioni aggregate e Target List

- Una query errata

```
SELECT Nome, MAX(Reddito)
FROM PERSONE
```

- Qual è il nome?

Non possiamo estrarre il nome che ha il reddito maggiore. La Target List deve avere attributi tutti dello stesso tipo.

```
SELECT MIN(Età), MAX(Reddito)
FROM PERSONE
```

4.4.15.6 Massimo e query annidate

Restituire le persone che hanno lo stesso reddito massimo

```
SELECT *
FROM PERSONE
WHERE Reddito = (SELECT MAX(Reddito)
                  FROM PERSONE)
```

4.4.16 Funzioni aggregate e Grouping

Le funzioni aggregate possono operare su gruppi di relazioni attraverso il comando **GROUP BY**:

GROUP BY AttrList

<i>Il numero dei figli dei padri</i>		PATERNITÀ		Padre	
Padre	Figlio	Padre	NumeroDiFigli	Padre	NumeroDiFigli
Steve	Frank	Steve	1	Louis	Olga
Louis	Olga	Louis	2	Louis	Phil
Louis	Phil	Frank	2	Frank	Jim
Frank	Jim	Frank		Frank	James
Frank	James				

SELECT Padri, COUNT(*) AS NumeroDiFigli
FROM PATERNITÀ
GROUP BY Padre

SEMANTICA:

1. Eseguire la query senza funzioni aggregate e senza operatori aggregati

```
SELECT *
FROM PERSONE
```
2. Poi eseguire il raggruppamento e applicare le funzioni aggregate su ogni gruppo

4.4.16.1 Grouping e Target List

Errata:

```
SELECT Padre, AVG(C.Reddito), F.Reddito
FROM PERSONE C JOIN PATERNITÀ ON C.Nome = Figlio
JOIN PERSONE F ON Padre = F.Nome
GROUP BY Padre
```

Corretta:

```
SELECT Padre, AVG(C.Reddito), F.Reddito
FROM PERSONE C JOIN PATERNITÀ ON C.Nome = Figlio
JOIN PERSONE F ON Padre = F.Nome
GROUP BY Padre, F.Reddito
```

4.4.16.2 Condizioni sui gruppi

Fornire i padri i cui figli hanno un reddito medio maggiore di 25
restituire il reddito medio del padre e dei loro figli

```
SELECT Padre, AVG(C.Reddito)
FROM PERSONE C JOIN PATERNITÀ ON C.Nome = Figlio
GROUP BY Padre
HAVING AVG(C.Reddito) > 25
```

4.4.16.3 WHERE vs. HAVING

Fornire i padri i cui figli minori di 30 anni hanno un reddito medio maggiore di 20

```
SELECT Padre, AVG(C.Reddito)
FROM PERSONE C JOIN PATERNITÀ ON C.Nome = Figlio
WHERE C.Età < 30
GROUP BY Padre
HAVING AVG(C.Reddito) > 20
```

4.4.16.4 Raggruppamenti e NULL

R	A	B
1	11	
2	11	
3	NULL	
4	NULL	

SELECT B, COUNT(*)
FROM R GROUP BY B

B	COUNT(*)
11	2
NULL	2

SELECT A, COUNT(*)
FROM R GROUP BY A

A	COUNT(*)
1	1
2	1
3	1
4	1

SELECT A, COUNT(B)
FROM R GROUP BY A

A	COUNT(B)
1	1
2	1
3	0
4	0

4.4.16.5 Sintassi SELECT: Riassunto

```
SELECT AttList1 + Exprs
FROM TableList + Joins
[ WHERE Condition ]
[ GROUP BY AttList2 ]
[ HAVING AggrCondition ]
[ ORDER BY OrderingAttr1 ]
```

4.5 OPERATORI DI AGGIORNAMENTO

Gli operatori di aggiornamento sono:

> **INSERT**

> **DELETE**

> **UPDATE**

... di una o più tuple nella tabella

... sulla base di un predicato che potrebbe coinvolgere altre relazioni

4.5.1 Insert

```
INSERT INTO Table [ (AttList) ]
VALUES( Vals )
```

oppure

```
INSERT INTO Table [ (AttList) ]
SELECT ...
```

4.5.1.1 Esempi

```
INSERT INTO PERSONE( Nome, Età, Reddito )
VALUES( 'Jack', 25, 25 )
```

```
INSERT INTO PERSONE VALUES( 'John', 25, 52 )
```

```
INSERT INTO PERSONE( Nome, Reddito )
VALUES( 'Robert', 55 )
```

argomenti.

Se l'elenco degli attributi viene omesso, assumiamo che tutti gli attributi siano considerati e che ogni valore **corrisponda a un attributo specifico come dichiarato nello schema della relazione**.

Se l'elenco degli attributi non contiene tutti gli attributi della relazione, vengono posizionati **un valore NULL o un valore predefinito**.

L'ordinazione degli attributi e dei valori è rilevante.

Entrambe gli elenchi **dovrebbero avere lo stesso numero di**

4.5.2 Delete

```
DELETE FROM Table
[ WHERE Condition ]
```

4.5.2.1 Esempi

```
DELETE FROM PERSONE
WHERE Età < 35
```

```
DELETE FROM PATERNITÀ
WHERE Figlio NOT IN ( SELECT Nome
FROM PERSONE )
```

```
DELETE FROM PATERNITÀ
```

Rimuovere le tuple che

soddisfano una condizione data

43.Può causare la rimozione di altre tuple (se i vincoli sono stati definiti usando **CASCADE**)

44.Se non viene fornita alcuna condizione, come deve essere inteso **WHERE TRUE**

4.5.3 Update

```
UPDATE TableName
```

```
SET Attribute = < Expr | SELECT ... | NULL | DEFAULT >
[ WHERE Condition ]
```

PERSONE

Nome	Età	Reddito
Jim	27	30
James	25	15
Bob	55	36

```
UPDATE PERSONE
SET Reddito = 45
WHERE Nome = Bob
```

```
UPDATE PERSONE
SET Reddito = Reddito*1.1
WHERE Età < 30
```

```
UPDATE PERSONE
SET Reddito =
( SELECT Reddit FROM
PERSONE WHERE Nome=Jim )
```

```
WHERE Nome = Bob
```

PERSONE

Nome	Età	Reddito
Jim	27	30
James	25	15
Bob	55	45

Nome	Età	Reddito
Jim	27	33
James	25	16.5
Bob	55	36

Nome	Età	Reddito
Jim	27	30
James	25	15
Bob	55	30

5 SQL AVANZATO

5.1 CARATTERISTICHE EVOLUTE DI DEFINIZIONE DEI DATI

Dopo aver descritto i comandi base per la definizione dei dati e la scrittura delle interrogazioni in SQL, completiamo la rassegna dei componenti di uno schema. Descriviamo quindi la clausola **CHECK**, le asserzioni e le primitive per la definizione di viste.

5.1.1 Vincoli di integrità generici

CHECK specifica i vincoli sulle tuple (e possibilmente ancora più complessi che non sono sempre supportati in tutte le implementazioni SQL)

CHECK (Predicato)

Esempio 1:

```
CREATE TABLE IMPIEGATI (
    Numero      integer, primary key,
    Cognome     character(20),
    Nome        character(20),
    Gender      character not null
        CHECK (Gender in ('M','F')),
    Salario     integer
        CHECK (Salario >= 0),
    Supervisore integer,
        CHECK (Salario <= (SELECT Salario
                            FROM IMPIEGATI J
                           WHERE Supervisore = J.Numero))
)
```

Un comando **SELECT** nel vincolo **CHECK** non è completamente supportato in ogni implementazione SQL

Esempio 2:

```
CREATE TABLE IMPIEGATI (
    Numero      integer, primary key,
    Cognome     character(20),
    Nome        character(20),
    Gender      character not null
        CHECK (Gender in ('M','F')),
    Salario     integer,
    Ritenuta    integer,
    Netto       integer,
    Supervisore character(6),
        CHECK (Netto = Salario - Ritenuta)
)
```

Ok

Esempio 3:

insert into IMPIEGATI values (1,'Doe','John','','100,20,80);	Sesso non è né 'M' né 'F'
insert into IMPIEGATI values (2,'Lee','Jim','M',100,10,80);	Netto (80) non è 100 - 10
insert into IMPIEGATI values (3,'Hill','Sam','M',70,20,50);	Questo update è ammesso

5.1.2 Asserzioni

Definiscono vincoli al livello dello schema

```
CREATE ASSERTION NomeCome CHECK (Predicato)
```

```
CREATE ASSERTION AlmenoUnImpiegato
```

```
  CHECK (1 <= (SELECT COUNT(*)  
    FROM IMPIEGATI))
```

Un comando **SELECT** con il vincolo **CHECK** non è totalmente supportato in tutte le implementazioni di SQL

5.1.3 Viste

```
CREATE VIEW NomeVista [ ( AttList ) ] AS  
  SelectStaement [ WITH [ LOCAL | CASCDED ] CHECK OPTION ]
```

```
CREATE VIEW IMPIEGATIADMIN  
  (Nome, Cognome, Stipendio) AS  
    SELECT Nome, Cognome, Stipendio  
      FROM IMPIEGATI  
        WHERE Dipart = 'Amministrazione' AND  
          Stipendio > 10
```

5.1.3.1 Aggiornamento viste

Solitamente questo tipo di aggiornamenti sono ammessi per viste definite su una singola relazione.

Possiamo forzare il DataBase a eseguire alcuni controlli

```
CREATE VIEW IMPIEGATIADMINPOVERI AS  
  SELECT *  
    FROM IMPIEGATIADMIN  
      WHERE Stipendio < 50  
        WITH CHECK OPTION
```

CHECK OPTION permette di aggiornare la vista, ma solo se le tuple inserite appartengono alla vista
(l'utente non può avere uno stipendio maggiore di 50)

5.1.3.2 Operazioni non consentite

```
CREATE VIEW IMPIEGATIADMINPOVERI AS  
  SELECT *  
    FROM IMPIEGATIADMIN  
      WHERE Stipendio < 50  
        WITH CHECK OPTION
```

```
UPDATE IMPIEGATIADMINPOVERI  
  SET Stipendio = 60  
  WHERE Nome = 'Ann'
```

5.1.3.3 Alterare una vista: Local e Cascaded

- › **local** (nel caso di viste su viste): l'aggiornamento delle tuple dev'essere eseguito solo all'ultimo livello della vista
- › **cascaded** (nel caso di viste su viste): l'aggiornamento delle tuple dev'essere eseguito su tutte le viste e le relazioni sottostanti

5.1.4 Interrogare una vista

Le viste possono essere interrogate come qualunque altra relazione nel database, per esempio:

```
SELECT * FROM IMPIEGATIADMIN
```

Corrisponde a eseguire la seguente query

```
SELECT Nome, Cognome, Stipendio
FROM IMPIEGATI
WHERE Dipart = 'Amministrazione' AND
Stipendio > 10
```

5.1.4.1 Un'interrogazione SQL

Fornire la media di uffici per dipartimento

Query errata:

```
SELECT AVG(COUNT(DISTINCT Uffici))
FROM IMPIEGATI
GROUP BY Dipart
```

La query è errata perché la sintassi SQL non ammette operatori aggregati annidati

Query corretta:

```
CREATE VIEW DIPARTUFFICI(NomeDip, NroUffici) AS
SELECT Dipart, count(distinct Uffici)
FROM IMPIEGATI
GROUP BY Dipart

SELECT AVG(NroUffici)
FROM DIPARTUFFICI
```

5.1.5 Viste ricorsive

Per ogni persona, fornire i suoi antenati, dato:

```
PATERNITÀ(Padre, Figlio)
```

Dobbiamo usare la ricorsione:

```
WITH RECURSIVE ANTENATI(Antenato, Discendente) AS
(
  SELECT Padre, Figlio
  FROM PATERNITÀ
  UNION ALL
  SELECT ANTENATI, PATERNITÀ
  WHERE Discendente = Padre
)
SELECT *
FROM ANTENATI
```

PATERNITÀ	
Padre	Figlio
Carl	Frank
Louis	Olga
Louis	Bob
Frank	Alex
Frank	Alfred

WITH definisce la vista **ANTENATI**, che viene costruita ricorsivamente usando **PATERNITÀ**

5.2 FUNZIONI SCALARI

SQL mette a disposizione anche le funzioni scalari, che possono essere usate all'interno delle espressioni del linguaggio. Le funzioni ricevono come argomento una o più espressioni, che restituiscono valori di un dominio elementare in corrispondenza di ogni tupla su cui viene valutata la query; le funzioni, a loro volta, restituiscono un valore semplice per ogni diversa tupla.

5.2.1 Famiglie di funzioni

SQL fornisce alcune famiglie di funzioni. I sistemi spesso arricchiscono l'insieme di funzioni di ogni famiglia.

- 45. Funzioni temporali: servizi di utilità per la gestione di informazioni temporali
 - > **Current_date()** ritorna la data presente
 - > **Extract(yearExpression)** estrae parte della data di un'espressione in input (mese, giorno, ora, anno...)

- 46. Funzioni di manipolazione di stringhe:
 - > **char_length** ritorna la lunghezza della stringa
 - > **lower** converte i caratteri della stringa a minuscoli
- 47. Funzioni di conversione di dominio:
 - > **cast** permette la conversione di un valore in un altro dominio
- 48. Funzioni condizionali:
 - > le vediamo adesso

5.2.2 Funzioni condizionali

Queste funzioni non estendono il potere espressivo del linguaggio ma permettono di realizzare comandi SQL in modo più compatto e facile da comprendere, evitando per esempio di costruire interrogazioni composte da unioni di tante interrogazioni più semplici.

Coalesce La funzione **COALESCE** ammette come argomento una sequenza di espressioni e restituisce il primo valore non nullo. La funzione può quindi essere usata per convertire i valori nulli in valori definiti dal programmatore.

Nullif La funzione **NULLIF** richiede come argomento un'espressione e un valore costante; se l'espressione è pari al valore costante la funzione restituisce il valore nullo, altrimenti restituisce il valore dell'espressione.

Case La funzione **CASE** permette di specificare strutture condizionali, il cui risultato dipende dalla valutazione del contenuto delle tabelle. È usata per fornire una logica del tipo "if-then-else"

Valutare le tasse del veicolo per il suo tipo e l'anno di registrazione (dal 1975)

Per ogni impiegato ritornare o un numero di cellulare valido oppure il suo numero di telefono

IMPIEGATI(Numero, Dipart, Cell, TelCasa)

**SELECT Numero, COALESCE(Cell, TelCasa)
FROM IMPIEGATI**

Estrarre i cognomi e i dipartimenti a cui gli impiegati appartengono, ritornare un valore NULL per il dipartimento quando l'attributo Dipart ha valore 'Sconosciuto'

**SELECT Cognome,
NULLIF(Dipart, 'Sconosciuto')
FROM IMPIEGATI**

VEICOLO(Targa, Tipo, Anno, Kwatt,
Lunghezza, NAssi)

**SELECT Targa
(CASE Tipo
WHEN 'Auto' THEN 2.58 * Kwatt
WHEN 'Moto' THEN (22.00 + 1.00 *
Kwatt)
ELSE null
END) AS Tax
FROM VEICOLO
WHERE Anno > 1975**

5.3 CONTROLLO DELL'ACCESSO

La presenza di meccanismi di protezione dei dati è rilevante in molte applicazioni. Uno dei compiti più importanti di un amministratore di basi di dati è quello di scegliere e implementare opportune politiche di controllo di accesso. SQL prevede che ogni utente sia identificato in modo univoco dal sistema.

5.3.1 Privilegi

SQL permette di garantire per ogni utente specifici privilegi (lettura, scrittura, ...) sia per tutti il DB che per solo una sua parte.

I privilegi possono essere garantiti su relazioni, attributi, viste o domini

Quindi, di default, c'è almeno un amministratore per il quale possiede tutti i privilegi

Qualunque utente che crei una specifica risorsa ne guadagna automaticamente tutti i privilegi

Ogni privilegio è caratterizzato da:

- 49. Una specifica **risorsa**
- 50. Un utente che **fornisce** il privilegio
- 51. Un utente **al quale è fornito** il privilegio
- 52. Una specifica **operazione**
- 53. La possibilità di **propagare** il privilegio o no

I privilegi disponibili sono i seguenti:

- **insert**: permette di inserire nuove tuple

- **update**: permette di modificare tuple già esistenti
- **delete**: permette di eliminare tuple
- **select**: permette di leggere una risorsa
- **references**: permette di definire un vincolo di integrità referenziale
- **usage**: permette di usare una risorsa

5.3.2 Comandi per concedere e revocare privilegi

Concedere privilegi

```
GRANT < Privilegi | all privileges >
    ON Risorsa
    TO Utenti [ with grant option ]
```

grant option permette all'utente di propagare il suo privilegio ad altri utenti

```
GRANT SELECT ON DIPARTIMENTO TO Jack
```

I privilegi devono essere revocati dallo stesso utente che li ha concessi

- **restrict** (default) la revoca non coinvolge altri utenti che hanno ricevuto i privilegi dall'utente corrente
- **cascade** la revoca viene estesa agli altri utenti

Revocare privilegi

```
REVOKE Privilegi ON Risorsa
    FROM Utenti [ restrict|cascade ]
```

Attenzione: **cascade** causa una "reazione a catena"

L'implementazione SQL deve nascondere (senza dare indizio) la parte del database che non è accessibile all'utente. Per esempio può essere che:

- 54. La tabella **IMPIEGATI** non esiste
- 55. La tabella **IMPIEGATI** esiste, ma l'utente non vi ci possa accedere

Potremmo utilizzare una vista per mostrare a un utente solo delle tuple specifiche

- 56. La vista è definita con un predicato di selezione
- 57. Il privilegio viene garantito per una vista specifica

5.3.3 Autorizzazioni

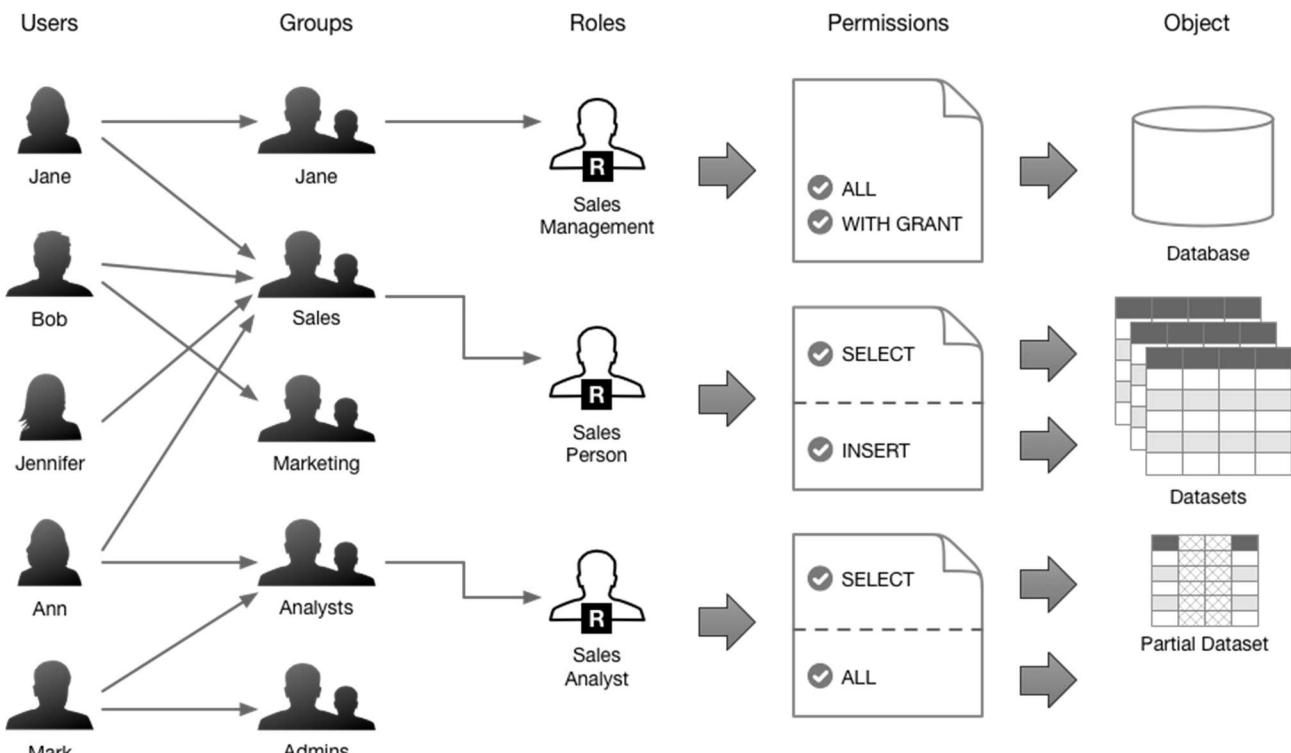
SQL-3 ha introdotto RBAC (Role-Based Access Control, **RBAC**)

Ogni **role** agisce come un contenitore di più privilegi che possono essere concessi attraverso il comando **grant**

In qualsiasi momento, ogni utente ha:

58. Privilegi “individuali” concessi, associati direttamente a lui

59. Privilegi concessi al suo ruolo tramite RBAC



I seguenti comandi creano un nuovo ruolo *Nome*

CREATE ROLE *Nome*

Questo comando fornisce il ruolo *Nome* all’utente corrente

SET ROLE *Nome*

Esempio:

Fornire il comando **CREATE TABLE** a un’utente specifico, attraverso il ruolo **Impiegato**

- Creare il nuovo ruolo

CREATE ROLE Impiegato

- Concedere il privilegio al ruolo definito in precedenza

GRANT CREATE TABLE TO Impiegato

- Concedere il privilegio a un utente specifico

GRANT Impiegato TO utente

- Revocare il privilegio fornito in precedenza

REVOKE CREATE TABLE FROM Impiegato

5.4 TRANSAZIONI

Una transazione è un programma in esecuzione che forma un'unità logica di elaborazione del database (operazione atomica)

Tutto il codice eseguito all'interno di una transazione gode di proprietà particolari dette *proprietà acide* delle transazioni: *atomicità*, *consistenza*, *isolamento* e *persistenza* (**ACID**) :

60. **Atomicity**
61. **Consistency**

62. **Isolation**
63. **Durability**

Atomicità Una transazione è un'unità atomica di processamento, dovrebbe essere eseguita o non eseguita affatto

*Trasferimento di soldi da un conto corrente A ad uno B:
il denaro viene ritirato da A e trasferito a B, o non viene eseguita alcuna operazione*

Consistenza Una transazione deve conservare la coerenza: ovvero l'esecuzione non deve violare i vincoli di integrità definiti. Se una transazione li viola il sistema deve intervenire per annullare la transazione o per correggere la violazione del vincolo.

Isolamento L'esecuzione di una transazione dev'essere indipendente dall'esecuzione contemporanea di altre transazioni. Si richiede che il risultato dell'esecuzione concorrente di un insieme di transazioni sia analogo al risultato che le stesse transazioni otterrebbero qualora ciascuna di esse fosse eseguita da sola.

Per esempio, il caso in cui due versamenti allo stesso conto corrente avvengano nello stesso momento.

Persistenza (Durable effect) L'effetto di una transazione che ha eseguito il commit correttamente non venga più perso. Un database deve garantire che nessun dato venga perso per nessun motivo.

5.4.1.1.1 Esempio:

Ritirare 10€ dal conto 42177 e versarli sul conto 12202

START TRANSACTION

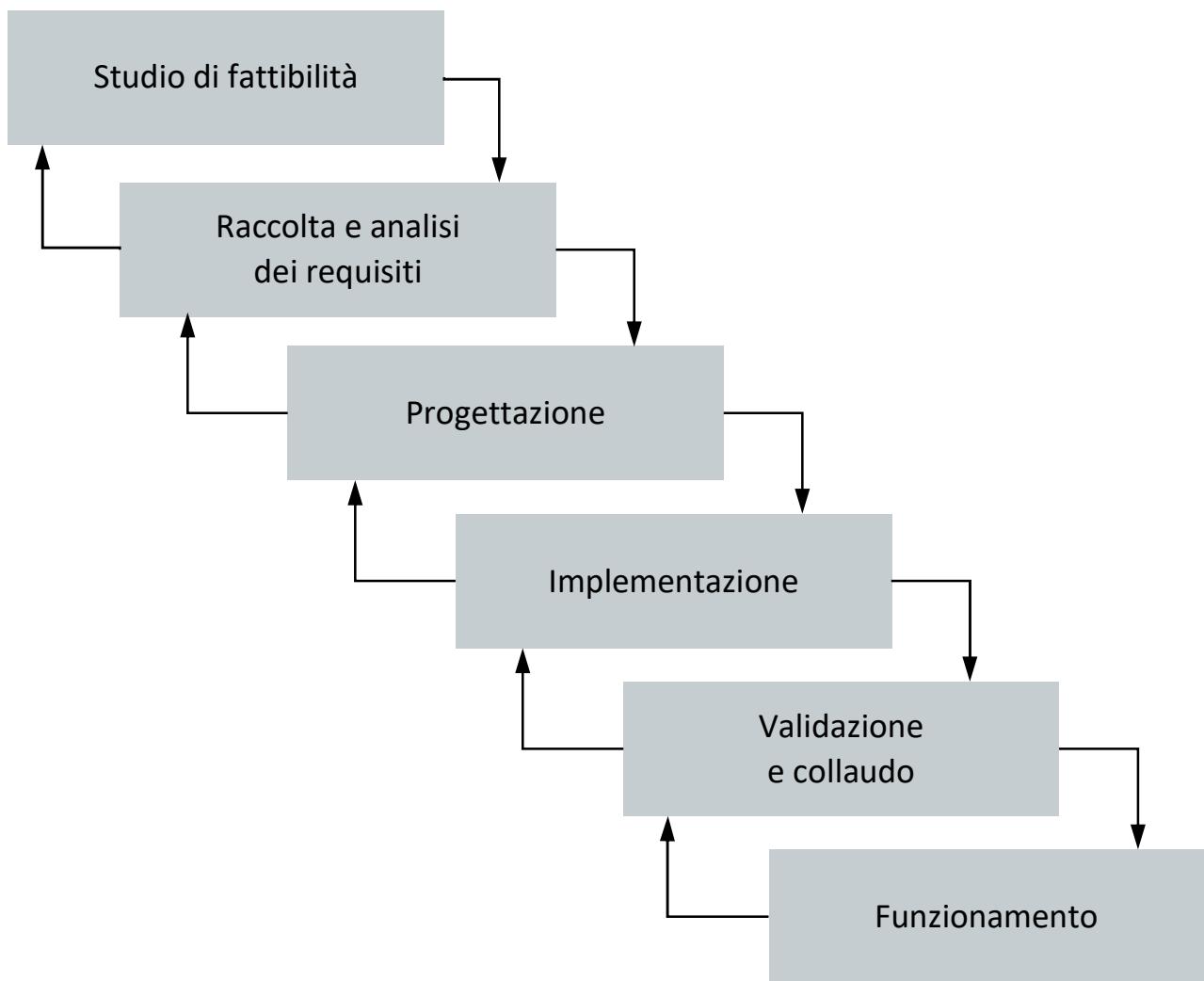
```
UPDATE CONTOCORRENTE
  SET Saldo = Saldo - 10
 WHERE NumConto = 42177;
UPDATE CONTOCORRENTE
  SET Saldo = Saldo + 10
 WHERE NumConto = 12202;
COMMIT WORK;
```


6 MODELLAZIONE CONCETTUALE DEI DATI

6.1 INTRODUZIONE ALLA PROGETTAZIONE

La progettazione di un database costituisce solo una delle componenti del processo di sviluppo di un sistema informativo complesso e va inquadrata in un contesto più ampio, quello del *ciclo di vita* dei sistemi informativi.

6.1.1 Ciclo di vita dei sistemi informativi



1. **Studio di fattibilità.** Per definire i costi delle varie alternative possibili e a stabilire le priorità di realizzazione delle varie componenti del sistema
2. **Raccolta e analisi dei requisiti.** È l'individuazione e lo studio delle proprietà e delle funzionalità che il sistema informativo dovrà avere. Richiede un'interazione con gli utenti del sistema.
3. **Progettazione.** Generalmente si divide in *progettazione dei dati* e *progettazione delle applicazioni*. Nella prima si individua la struttura e l'organizzazione che i dati dovranno avere, nell'altra si definiscono le caratteristiche dei programmi applicativi.
4. **Implementazione.** Realizzazione del sistema informativo.
5. **Validazione e collaudo.** Per verificare il corretto funzionamento e la qualità del sistema informativo.
6. **Funzionamento.** Il sistema diventa operativo ed esegue i compiti per i quali era stato progettato.

6.1.2 Metodologie di progettazione

Nei database si è consolidata una metodologia di progetto articolata in tre fasi principali:

1. Progettazione concettuale
Schema concettuale
2. Progettazione logica
Schema logico
3. Progettazione fisica
Schema fisico

Una collezione di componenti viene usata per categorizzare dati rilevanti e descrivere operazioni su di essi.

I costruttori sono componenti cruciali i costruttori svolgono lo stesso ruolo delle definizioni dei tipi di dato nei linguaggi di programmazione.

Per esempio, il modello relazionale definisce il costruttore “relazione” per set uniformi di tuple (record)

Per ogni database abbiamo:

- Uno **schema**, che non varia nel tempo, che definisce la struttura dati
- Un’**istanza**, i valori correnti, che possono cambiare nel tempo e molto velocemente

6.1.2.1 Due tipi di modelli

Modelli logici: organizzano i dati con un DBMS

- Livello di astrazione dei dati utilizzato dai software
- Indipendenti dalla progettazione fisica

Per esempio: relazionali, grafi, gerarchie, oggetti

Modelli concettuali: permettono la rappresentazione dei dati indipendentemente dal sistema

- Provano a descrivere concetti del mondo reale
- Vengono usati nelle fasi preliminari di progettazione

Il più utilizzato è il modello **Entità-Relazione**

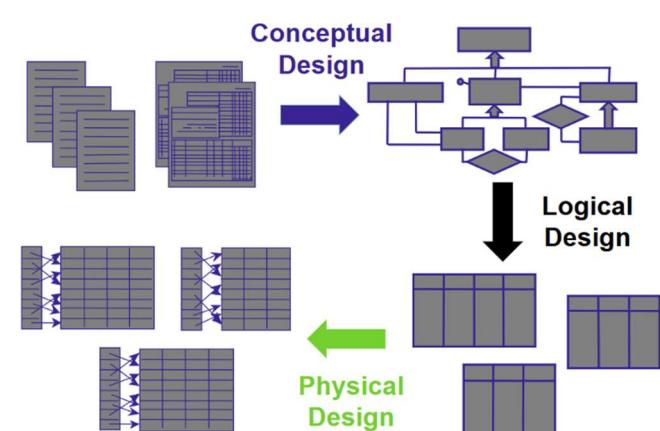
Ma perché si usano i modelli concettuali?

Proviamo a costruire un database relazionale direttamente dal modello logico

- Da dove iniziamo?
- C’è il rischio di approfondire ulteriori dettagli non utili
- Dobbiamo definire relazioni tra le tabelle
- Il modello relazione è troppo “rigido” per essere usato come modello

Li usiamo perché:

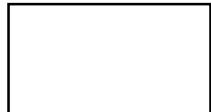
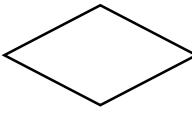
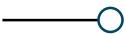
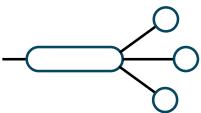
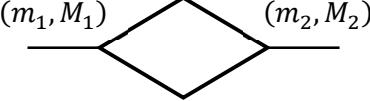
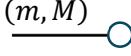
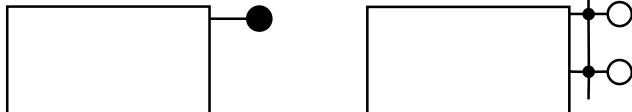
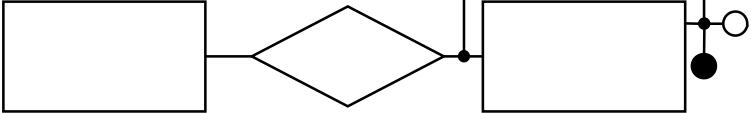
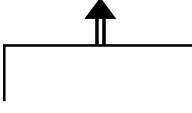
- Consentono di ragionare sulla realtà di interesse definendo un “modello” indipendente dall’attuazione
- Permettono di definire classi di oggetti di interesse e le loro relazioni
- Provvedono una rappresentazione visiva efficiente (utile per la documentazione)



6.2 IL MODELLO ENTITÀ-RELAZIONE

Il modello Entità-Relazione (E-R) è un modello *concettuale* di dati che fornisce una serie di strutture dette *costrutti* atte a descrivere la realtà in maniera facile.

Questi costrutti vengono utilizzati per definire *schemi* che descrivono l'organizzazione e la struttura delle *occorrenze* dei dati.

Costrutti	Rappresentazione grafica
Entità	
Relazione	
Attributo semplice	
Attributo composto	
Cardinalità di relazione	
Cardinalità di attributo	
Identificatore interno	
Identificatore esterno	
Generalizzazione	
Sottoinsieme	

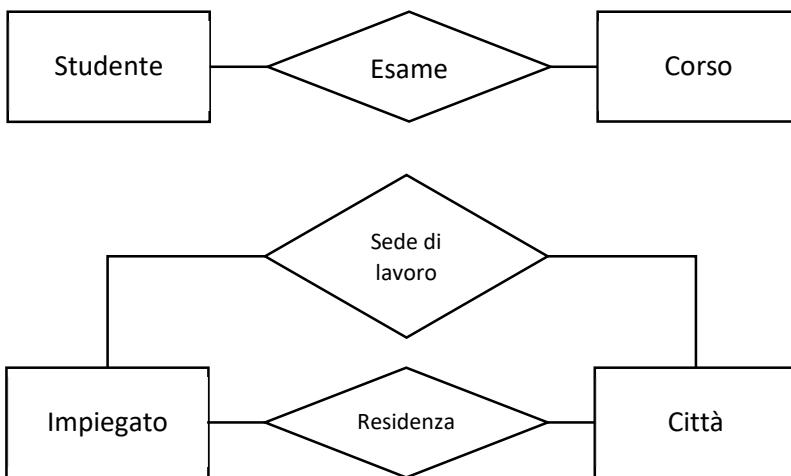
6.2.1 Entità

Rappresentano classi di oggetti (fatti, cose e persone) che hanno proprietà comuni ed esistenza “autonoma” ai fini dell’applicazione di interesse: **Città**,

Dipartimento, **Impiegato**, **Acquisto** e **Vendita** sono esempi di entità di un’applicazione aziendale. Una occorrenza di un’entità è un oggetto della classe che l’entità rappresenta. Le città di Roma, Milano e Bologna, sono esempi di occorrenze dell’entità Città, gli impiegati Marini e Ferrari sono invece esempi di occorrenze dell’entità **Impiegato**.

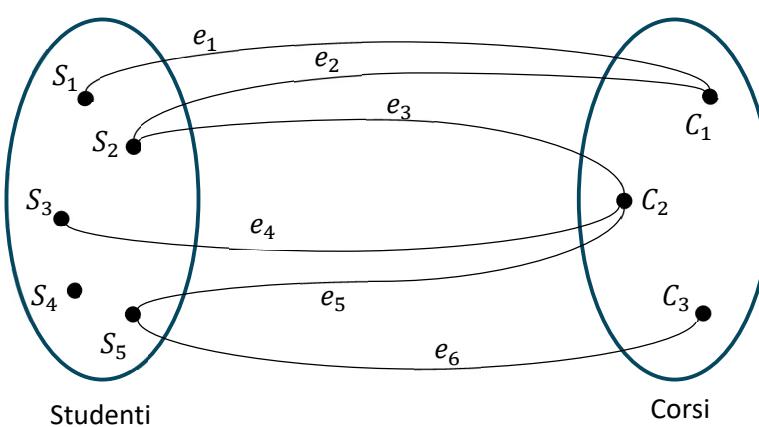
Un’occorrenza di entità non è un valore che identifica un oggetto (eg. cognome dell’impiegato o il suo codice fiscale), ma è l’oggetto stesso (l’impiegato “in carne e ossa”).

In uno schema, ogni entità ha un nome che la identifica univocamente e viene rappresentata graficamente mediante un rettangolo con il nome dell’entità all’interno.



Si possono avere relazioni diverse che coinvolgono le stesse entità, come le relazioni **Residenza** e **Sede di lavoro** tra le entità **Impiegato** e **Città**.

Esempi di occorrenze della relazione **Esame** tra le entità **Studente** e **Corso** sono le coppie e_1, e_2, e_3, e_4, e_5 ed e_6 nella figura, dove vengono anche raffigurate le occorrenze delle entità coinvolte.



6.2.2 Relazioni

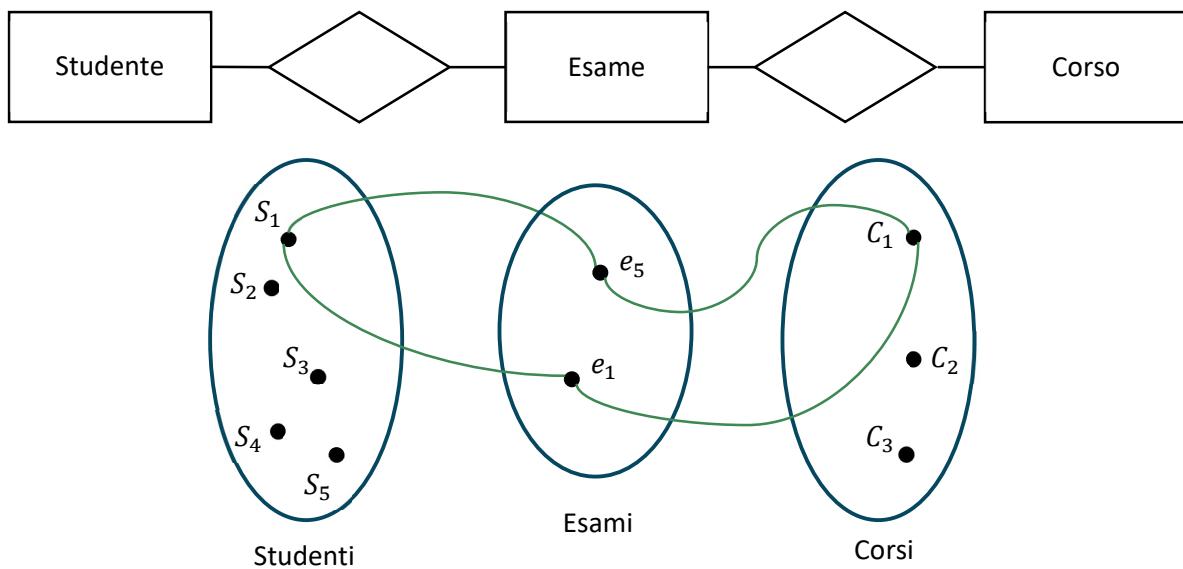
Rappresentano legami logici, significativi per l’applicazione di interesse tra due o più entità.

Residenza è un esempio di relazione che può sussistere tra le entità **Città** e **Impiegato** mentre **Esame** è un esempio -di relazione che può sussistere tra le entità **Studente** e **Corso**. Un’occorrenza di relazione è un’ n -upla (coppia nel caso più frequente di relazione binaria) costituita da occorrenze di entità, una per ciascuna delle entità coinvolte.

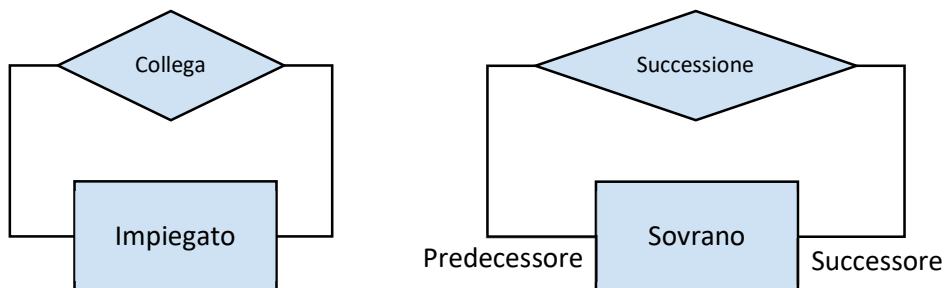
L’insieme delle occorrenze di una relazione del modello E-R è una relazione matematica tra le occorrenze delle entità coinvolte, ovvero, è un sottoinsieme del loro prodotto cartesiano. Quindi tra tutte le occorrenze non ci possono essere n -uple ripetute.

Nell’esempio la relazione **Esame** non è in grado di descrivere che uno studente ha sostenuto più volte lo stesso esame (ciò porterebbe a n -uple identiche) In questo caso, anche l’esame va rappresentato con un’entità collegata tramite relazioni alle entità **Studente** e **Corso**.

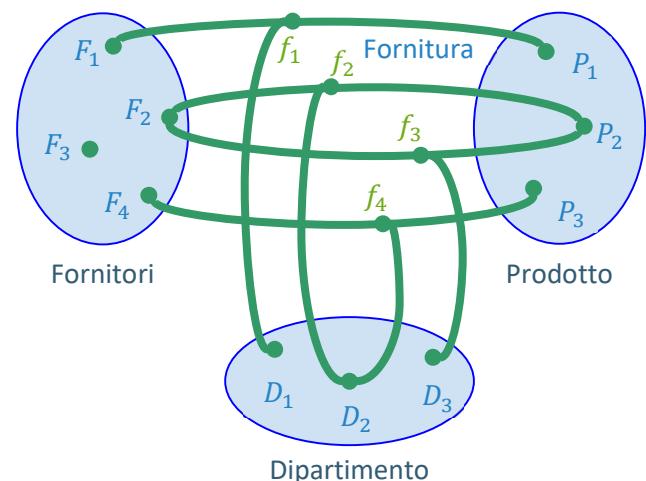
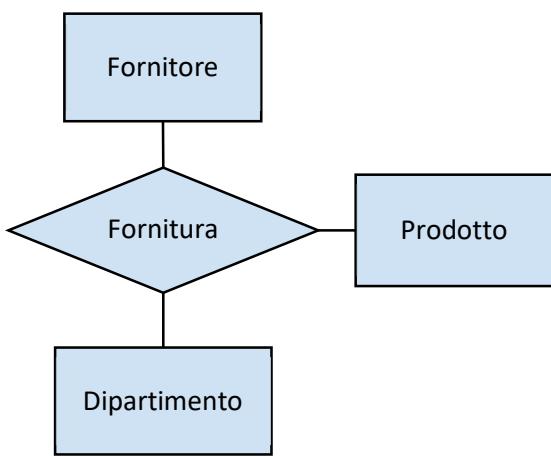
Trasformiamo quindi **Esame** da relazione a entità:



È possibile anche avere relazioni *ricorsive*, ovvero relazioni tra un'entità e sé stessa. Va notato che, a differenza della prima relazione, la relazione **Successione** non è simmetrica. In questo caso è necessario stabilire due *ruoli* che l'entità coinvolta gioca nella relazione. Questo può essere fatto associando degli identificatori (nel nostro caso Predecessore e Successore) alle linee uscenti dalla relazione ricorsiva.

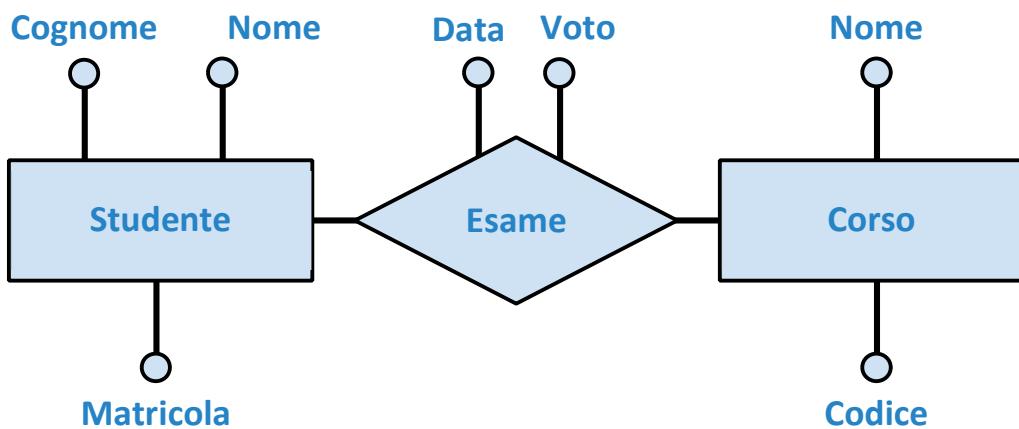


Infine è possibile avere relazioni *n*-arie, ovvero relazioni che coinvolgono più di due entità, l'esempio descrive il fatto che un fornitore rifornisce un dipartimento di un certo prodotto. Un possibile insieme di occorrenze di questa relazione potrebbe essere che la ditta Pinto fornisce stampanti al dipartimento Vendite, e computer al dipartimento Sviluppo, mentre la ditta Sami fornisce computer al dipartimento Vendite e fotocopiatrici al dipartimento Sviluppo.



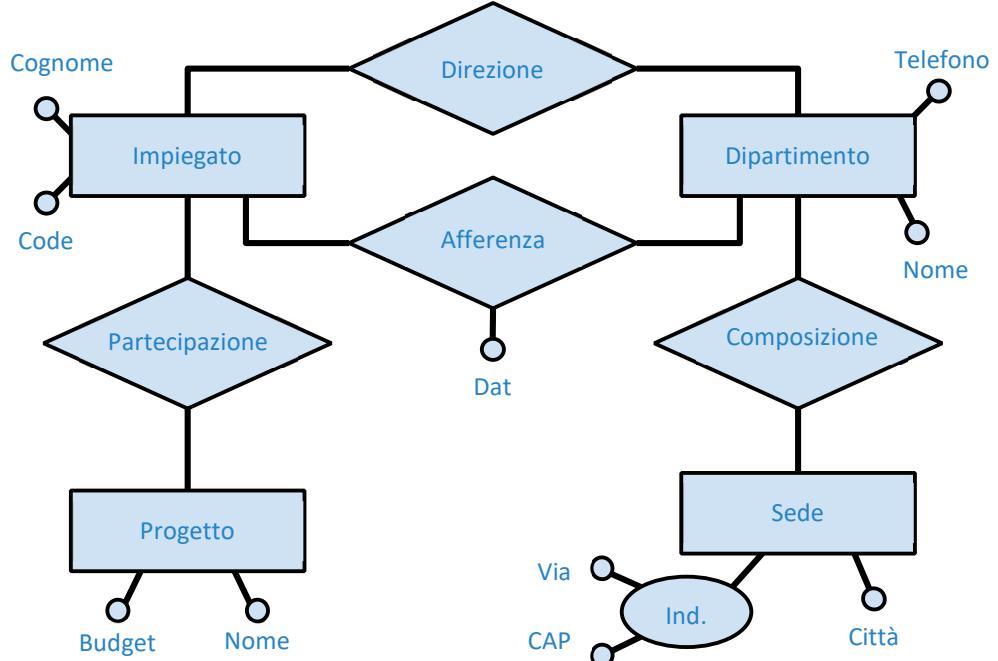
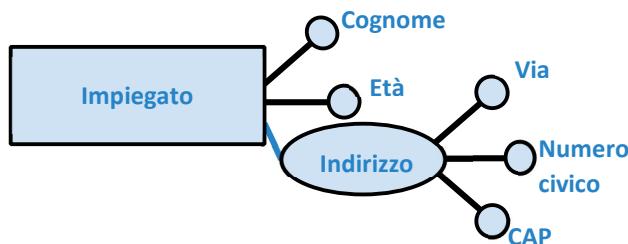
6.2.3 Attributi

Descrivono le proprietà elementari di entità o relazioni. Per esempio, **Cognome**, **Stipendio** ed **Età** sono possibili attributi dell'entità **Impiegato**, mentre **Data** e **Voto** lo sono per la relazione **Esame** tra **Studente** e **Corso**.



Un attributo associa a ciascuna occorrenza di entità (o di relazione) un valore appartenente a un insieme, detto *dominio*, che contiene i valori ammissibili per l'attributo.

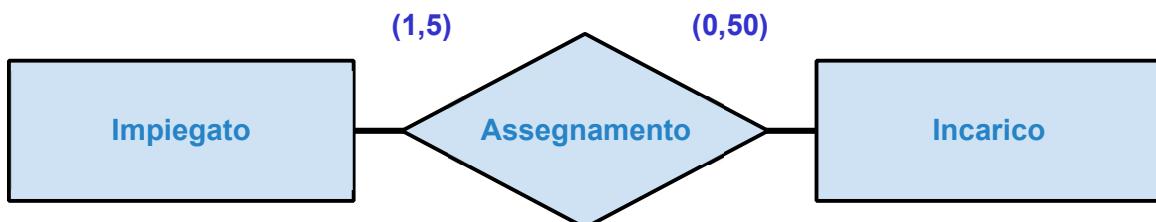
Può risultare utile, a volte, raggruppare attributi di una medesima entità o relazione che presentano affinità nel loro significato o uso: l'insieme di attributi che si ottiene così viene detto *attributo composto*. Possiamo, per esempio, raggruppare gli attributi **Via**, **Numero civico** e **CAP** dell'entità **Persona** per formare l'attributo composto **Indirizzo**.



6.2.4 Altri costrutti del modello

Esaminiamo i rimanenti costrutti del modello E-R: le **cardinalità** delle relazioni e degli attributi, gli **identificatori** delle entità e le **generalizzazioni**. Come vedremo solo l'ultimo costrutto è “nuovo”; gli altri costituiscono dei *vincoli di integrità* su costrutti già visti, ovvero proprietà che occorrenze di entità e di relazioni devono soddisfare per essere considerate “ valide”.

Cardinalità delle relazioni. Vengono specificate per ciascuna partecipazione di entità a una relazione e descrivono il numero minimo e massimo di occorrenze di relazione a cui una occorrenza dell'entità può partecipare. Quindi dicono quante volte, in una relazione tra entità, un'occorrenza di una di queste entità può essere legata a occorrenze delle altre entità coinvolte. Per esempio, se in una relazione **Assegnamento** tra le entità **Impiegato** e **Incarico** specifichiamo per la prima entità una cardinalità minima paria a 1 e una cardinalità massima paria a 5, vogliamo indicare che un impiegato può partecipare a un minimo di una occorrenza e a un massimo di cinque occorrenze della relazione **Assegnamento**. In altre parole, vogliamo dire che, nella nostra applicazione a un impiegato dev'essere assegnato almeno un incarico ma non più di 5. Se per l'entità **Incarico** specifichiamo una cardinalità minima pari a 0 e massima pari a 50 imponiamo che a un certo incarico può partecipare o nessuna occorrenza o al massimo 50 occorrenze della relazione **Assegnamento**.



Per semplicità utilizzeremo solo tre simboli:

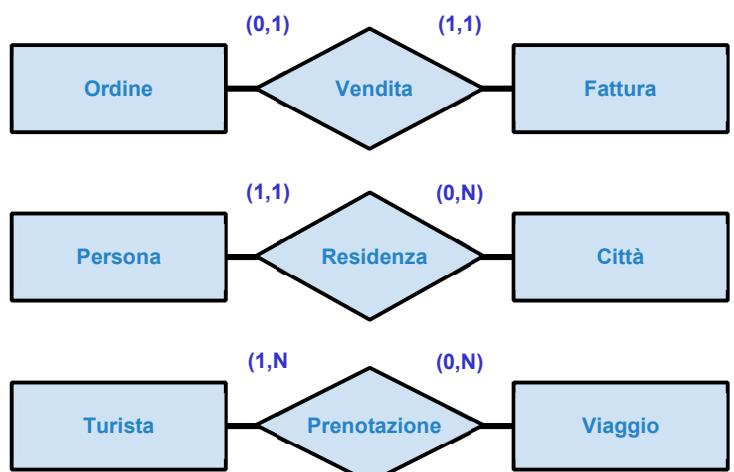
- # 0 e 1 per la cardinalità minima
 - 0 = partecipazione *opzionale*
 - 1 = partecipazione *obbligatoria*
- # 1 e N per la cardinalità massima
 - N non impone alcuna restrizione

Osservano le cardinalità massime è possibile classificare le relazioni binarie in base al tipo di corrispondenza che viene stabilita tra le occorrenze delle entità coinvolte.

Le relazioni aventi cardinalità massima paria a 1 per entrambe le entità coinvolte, come la relazione **Vendita**, definiscono una corrispondenza uno a uno tra le occorrenze di tali entità e vengono chiamate **relazioni uno a uno**.

In maniera analoga, le relazioni aventi un'entità con cardinalità massima pari a 1 e l'altra con cardinalità massima pari a N, come la relazione **Residenza** sono chiamate **relazioni uno a molti**.

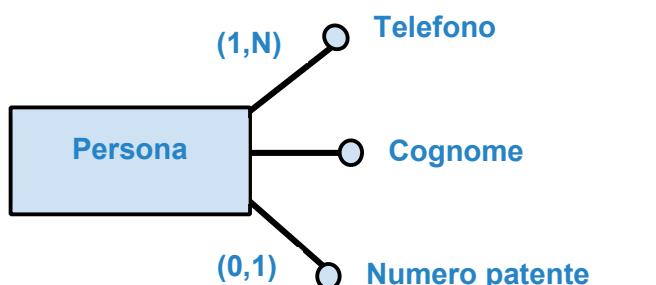
Infine, le relazioni che hanno cardinalità massima pari a N per entrambe le entità coinvolte, come la relazione **Prenotazione**, vengono chiamate **relazioni molti a molti**.



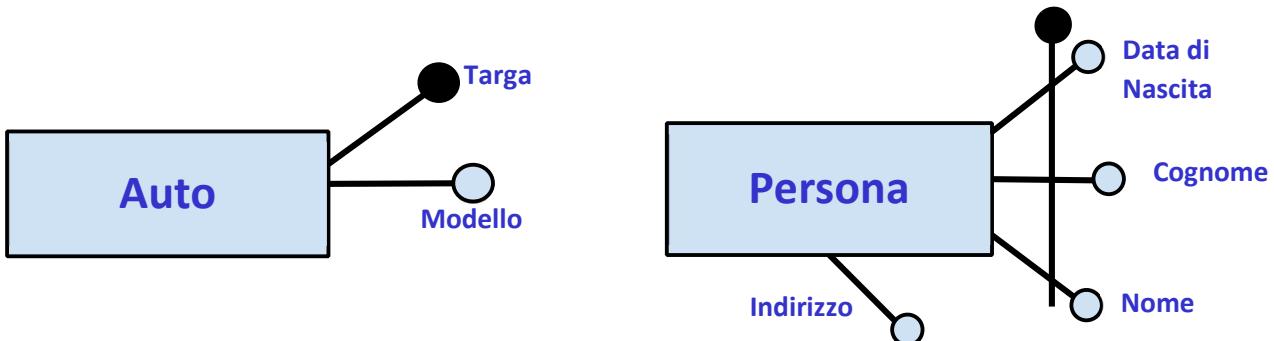
Cardinalità degli attributi. Possono essere specificate per gli attributi di entità o relazioni e descrivono il numero minimo e massimo di valori dell'attributo associati a ogni occorrenza di entità o relazione. Nella maggior parte dei la cardinalità di un attributo è pari a (1,1) e viene omessa. In questi casi l'attributo rappresenta una funzione che associa a ogni occorrenza di entità un solo valore dell'attributo.

Il valore per un certo attributo può essere però nullo, oppure possono esistere diversi valori di un certo attributo per un'occorrenza di entità. Queste situazioni si possono rappresentare associando all'attributo in questione una cardinalità minima pari a 0 nel primo caso e massima pari a N nel secondo.

Sulla base della cardinalità risulta che una persona ha uno e un solo cognome, può avere o non avere un numero di patente (ma se lo ha è unico) e ha almeno uno recapito telefonico (e può averne più di uno).

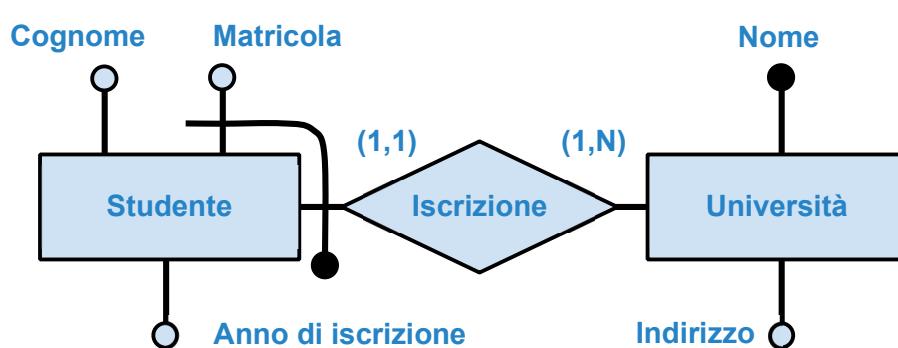


Identificatori di entità. Vengono specificati per ciascuna entità di uno schema e descrivono i concetti (attributi/entità) dello schema che permettono di identificare in maniera univoca le occorrenze delle entità. In molti casi, uno o più attributi di un'entità sono sufficienti a individuare un identificatore: si parla in questo caso di identificatore interno.



Per esempio, un identificatore interno per l'entità **Automobile** con attributi **Modello**, **Targa** e **Colore** è l'attributo **Targa**, in quanto non possono esistere due automobili con gli stessi valori sull'attributo **Targa**. Allo stesso modo un identificatore interno per l'entità **Persona** con attributi **Nome**, **Cognome**, **Indirizzo** e **Data di Nascita** può essere l'insieme degli attributi **Nome**, **Cognome**, **Data di Nascita**, avendo assunto che nella nostra applicazione non esistono due persone con lo stesso nome e cognome e con la stessa data di nascita.

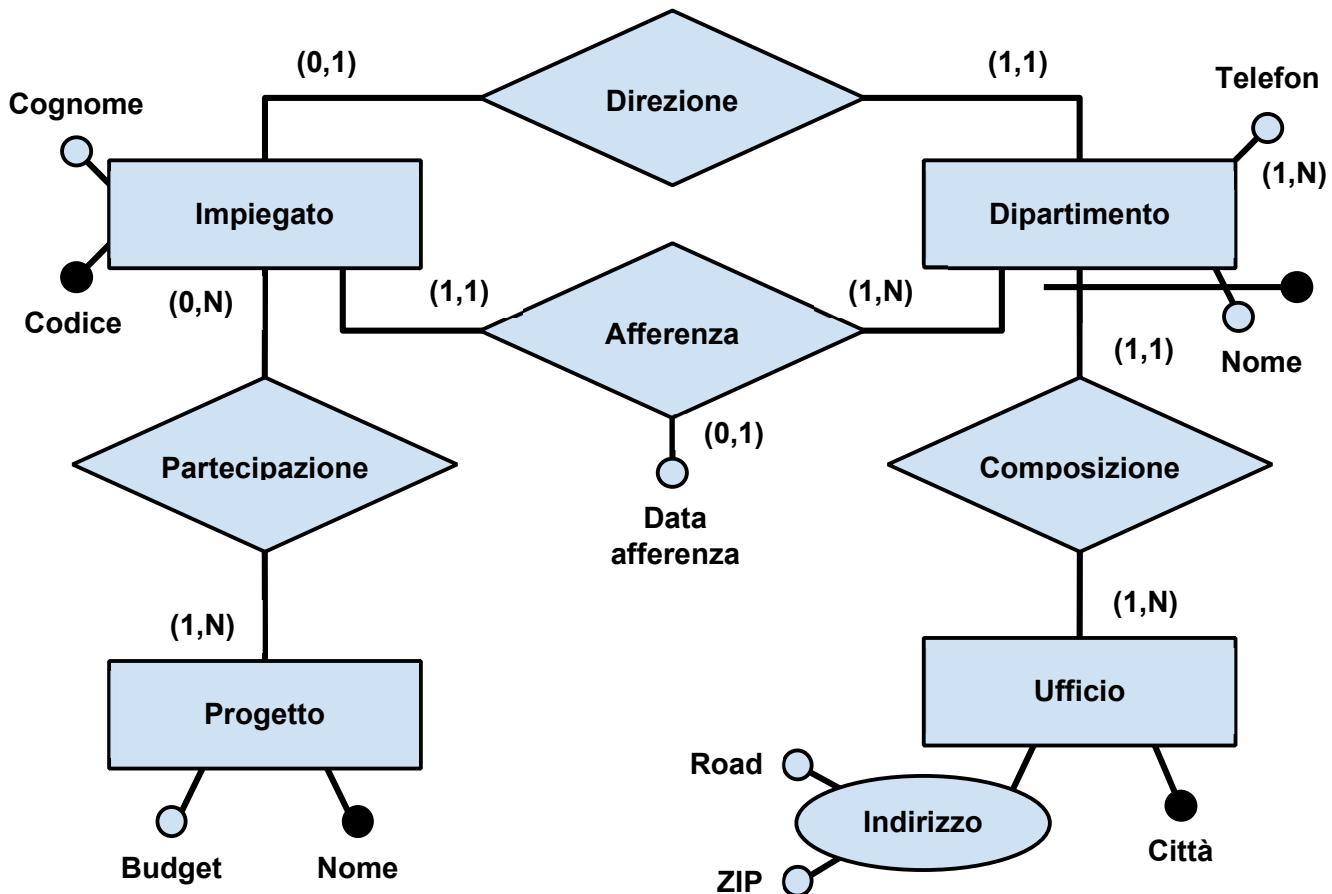
Può succedere però che gli attributi di un'entità non sono sufficienti a identificare univocamente le sue occorrenze. Si consideri per esempio l'entità dello **studente** nello schema in figura. L'attributo **Matricola**



e l'entità **Università** formano l'identificatore esterno, perché possono esserci studenti con la stessa matricola ma che appartengono a università differenti.

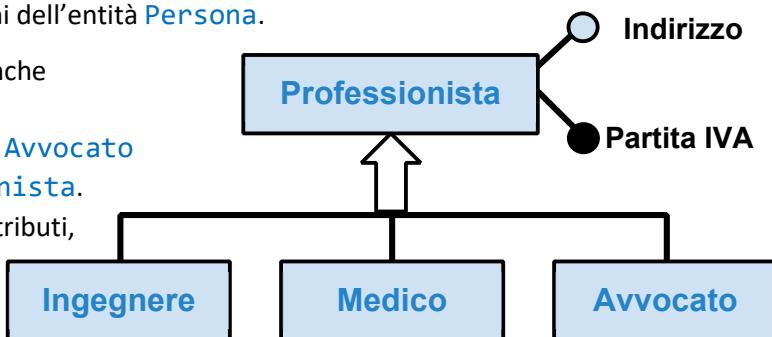
Ogni entità deve avere almeno un identificatore!

Un'identificazione esterna è possibile solo attraverso una relazione con una cardinalità (1,1)



Generalizzazioni. Rappresentano legami logici tra un'entità E , detta *genitore*, e una o più entità E_1, \dots, E_n , dette entità *figlie*, di cui E è più generale, nel senso che le comprende come caso particolare. Si dice in questo caso che E è *generalizzazione* di E_1, \dots, E_n e che le entità E_1, \dots, E_n sono *specializzazioni* dell'entità E . Per esempio, l'entità **Persona** è una generalizzazione dell'entità **Uomo** e **Donna**, mentre **Professionista** è una generalizzazione delle entità **Ingegnere**, **Medico** e **Avvocato**. Per contro le entità **Uomo** e **Donna** sono specializzazioni dell'entità **Persona**.

- Ogni occorrenza di un'entità figlia è anche un'occocrenza dell'entità genitore.
Per esempio un'occocrenza dell'entità **Avvocato** è anche un'occocrenza di **Professionista**.
- Ogni proprietà dell'entità genitore (attributi, identificatori, relazioni e altre generalizzazioni) è anche una proprietà delle entità figlie.



Per esempio, se l'entità **Professionista** ha attributi **Indirizzo** e **Partita IVA**, anche le entità **Avvocato**, **Ingegnere** e **Medico** possiedono questi attributi. Inoltre, l'identificatore di **Professionista** è valido anche per le entità figlie, questa proprietà si è nota come *ereditarietà*.

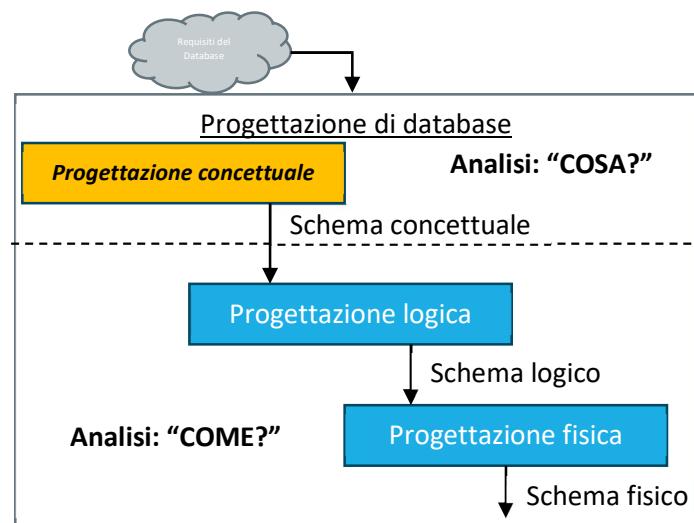
Le generalizzazioni possono essere classificate sulla base di due proprietà tra loro ortogonali.

- Una generalizzazione è **totale** se ogni occorrenza dell'entità genitore è un'occorrenza di almeno una delle entità figlie
La generalizzazione **Persona**, **Uomo/Donna** è Totale (gli uomini e le donne costituiscono tutte le persone)
- Una generalizzazione è **esclusiva** se ogni occorrenza dell'entità genitore è al più un'occorrenza di una delle entità figlie, altrimenti è **sovrapposta**.
La generalizzazione **Professionista** è invece esclusiva. (esistono anche altre professioni)

7 PROGETTAZIONE CONCETTUALE

La progettazione concettuale consiste nella costruzione di uno schema Entità-Relazione in grado di descrivere al meglio le specifiche sui dati di un'applicazione. Lo schema che si ottiene può contenere molti concetti correlati in maniera complicata. Quindi la progettazione dello schema finale è un processo graduale: lo schema concettuale viene progressivamente raffinato e arricchito attraverso una serie di trasformazioni ed eventuali correzioni.

Vedremo ora le strategie che è possibile seguire in questo processo di sviluppo di uno schema concettuale.



Analizzeremo prima la parte di raccolta e analisi dei requisiti che non è una fase completamente separata dalla progettazione, ma procede a volte parallelamente a essa.

Parleremo poi di alcuni criteri generali per tradurre specifiche informali in concetti del modello E-R. Guarderemo le principali strategie di progettazione per poi analizzare le qualità che uno schema concettuale ben progettato deve possedere. Concluderemo

7.1 RACCOLTA E ANALISI DEI REQUISITI

Per *raccolta dei requisiti* si intende la completa individuazione dei problemi che l'applicazione da realizzare deve risolvere e le caratteristiche del che dovrà avere.

I requisiti di un'applicazione provengono da fonti diverse. Le principali fonti di informazioni in genere sono:

- Gli *utenti dell'applicazione*. Le informazioni si acquisiscono attraverso:
 - Opportune interviste, anche ripetute
 - Una documentazione scritta che gli utenti hanno predisposto (*ad hoc*)
- Tutta la *documentazione esistente* che ha attinenza con il problema in studio:
 - Moduli
 - Regolamenti interni
 - Procedure aziendali e normative
- Eventuali *realizzazioni preesistenti*.

Quindi nella fase di acquisizione è importante l'interazione con gli utenti del sistema. Durante questa interazione può succedere che utenti diversi forniscano informazioni diverse, spesso complementari ma qualche volta contradditorie. In genere gli utenti a livello più alto possiedono una visione più ampia, ma meno dettagliata. Possono però indirizzare verso gli esperti dei singoli sottoproblemi.

Nel corso delle interviste è opportuno effettuare con l'utente verifiche di comprensione e consistenza sulle informazioni che si stanno raccogliendo; magari attraverso esempi (generali e relativi ai casi limite) oppure richiedendo definizioni e classificazioni precise. È inoltre molto importante distinguere gli aspetti essenziali rispetto a quelli marginali e procedere per raffinamenti successivi.

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

Regole empiriche:

- Scegliere il corretto livello di astrazione
- Standardizzare la struttura delle frasi
- Evitare frasi contorte
- Distinguere frasi "data" dalle frasi "function"

Regole generali:

- Costruire un glossario dei termini
- Individuare sinonimi/omonimi e unificare i termini
- Rendere esplicito il riferimento ai termini
- Ordinare le frasi per concetti

Esempio:

Vediamo quali sono le principali modifiche da apportare al nostro testo. *luogo* di nascita dei partecipanti (r4) è un omonimo (termini uguali che indicano concetti diversi) del luogo in cui si tengono le lezioni e va sostituito da *città* di nascita, così come *posto* (r5) che va sostituito con *datore di lavoro*. Va poi chiarito che a riga 6 l'*indirizzo* e il *numero di telefono* fanno

<u>Società di formazione</u>	
1	<i>Si vuole realizzare una base di dati per una società che eroga corsi, di cui vogliamo rappresentare i dati dei partecipanti ai corsi e dei docenti. Per i partecipanti (circa 5000), identificati da un codice, si vuole memorizzare il codice fiscale, il cognome, l'età, il sesso, il luogo di nascita, il nome dei loro attuali datori di lavoro, i posti dove hanno lavorato in precedenza insieme al periodo, l'indirizzo e il numero di telefono, i corsi che hanno frequentato (i corsi sono in tutto circa 200) e il giudizio finale. Rappresentiamo anche i seminari che stanno attualmente frequentando e, per ogni giorno, i luoghi e le ore dove sono tenute le lezioni. I corsi hanno un codice, un titolo e possono avere varie edizioni con date di inizio e fine e numero di partecipanti. Se gli studenti sono liberi professionisti, vogliamo conoscere l'area di interesse e, se lo possiedono, il titolo. Per quelli che lavorano alle dipendenze di altri, vogliamo conoscere invece il loro livello e la posizione ricoperta. Per gli insegnati (circa 300), rappresentiamo cognome, l'età, il posto dove sono nati, il nome del corso che insegnano, quelli che hanno insegnato nel passato e quelli che possono insegnare. Rappresentiamo anche tutti i loro recapiti telefonici. I docenti possono essere dipendenti interni alla società o collaboratori esterni.</i>

riferimento ai datori di

lavoro dei partecipanti. Il giudizio (riga 7) deve essere interpretato come *votazione in decimi*, mentre *periodo* (riga 6) va interpretato come *date di inizio e fine rapporto*. Bisogno inoltre specificare che i partecipanti frequentano o hanno frequentato specifiche *edizioni* di corsi. Per quanto riguarda gli altri termini che fanno riferimento ai corsi: *seminario* (riga 8) è un sinonimo e va sostituito da *edizione di corso*; *giorno* (riga 9), riferito alle lezioni, è troppo astratto e va utilizzato *giorno della settimana*; *luogo* (riga 9) è un omonimo, che va sostituito da *aula*. Il termine *studente* (riga 11) va sostituito con *partecipante*. Per *titolo* (riga 13) di un partecipante che è libero professionista si intende il suo *titolo professionale*. Per *posto* (riga 15) indica la *città* di nascita. Il *nome* del corso che insegnano (riga 16) è un sinonimo del *titolo* del corso e il *recapito telefonico* (righe 17-18) è sinonimo di *numero di telefono*.

Termino	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

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. Otteniamo così la strutturazione delle specifiche sui dati.

Frasi di carattere generale
<i>Si vuole realizzare una base di dati per una società che eroga corsi, di cui vogliamo rappresentare i dati dei partecipanti ai corsi e dei docenti.</i>
Frasi relative ai partecipanti
<i>Per i partecipanti (circa 5000), identificati da un codice, rappresentiamo il codice fiscale, il cognome, l'età, il sesso, la città di nascita, i nomi dei loro attuali datori di lavoro e di quelli precedenti (insieme alle date di inizio e fine rapporto), le edizioni dei corsi che stanno attualmente frequentando e quelli che hanno frequentato in passato, con la relativa votazione finale in decimi.</i>
Frasi relative ai datori di lavoro
<i>Relativamente ai datori di lavoro presenti e passati dei partecipanti, rappresentiamo il nome, l'indirizzo e il numero di telefono.</i>
Frasi relative ai corsi
<i>Per i corsi (circa 180), rappresentiamo il titolo e il codice, le varie edizioni con date di inizio e fine, per ogni edizione, rappresentiamo il numero di partecipanti e il giorno della settimana, le aule e le ore dove si sono tenute le lezioni.</i>
Frasi relative a tipi specifici di partecipanti
<i>Per i partecipanti che sono liberi professionisti, rappresentiamo l'area di interesse e, se lo possiedono, il titolo professionale. Per i partecipanti che sono dipendenti, rappresentiamo invece il loro livello e la posizione ricoperta.</i>
Frasi relative ai docenti
<i>Per i docenti (circa 300), rappresentiamo il cognome, l'età la città di nascita, tutti i numeri di telefono, il titolo del corso che insegnano, di quelli che hanno insegnato in passato e di quelli che possono insegnare. I docenti possono essere dipendenti interni alla società di formazione o collaboratori esterni.</i>

Naturalmente, accanto alle specifiche sui dati vanno raccolte le specifiche sulle operazioni da effettuare su questi dati. Bisogna cercare di impiegare la medesima terminologia usata per i dati (facendo 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. Per la nostra applicazione, le operazioni sui dati saranno le seguenti.

Operazione 1: inserisci un nuovo partecipante indicando tutti i suoi dati (operazione da effettuare in media 40 volte al giorno).

Operazione 2: assegna un partecipante a un'edizione di corso (circa 50 volte al giorno).

Operazione 3: inserisci un nuovo docente indicando tutti i suoi dati e i corsi che può insegnare (2 volte al giorno).

Operazione 4: assegna un docente abilitato a un'edizione di un corso (15 volte al giorno)

Operazione 5: stampa tutte le informazioni sulle edizioni passate di un corso con titolo, orari lezioni e numero partecipanti (10 volte al giorno).

Operazione 6: stampa tutti i corsi offerti, con informazioni sui docenti che possono insegnarli (20 volte al giorno).

Operazione 7: Per ogni docente, trova i partecipanti a tutti i corsi da lei/lui insegnati (5 volte a settimana)

Operazione 8: effettua una statistica su tutti i partecipanti a un corso con tutte le informazioni su di essi, sull'edizione alla quale hanno partecipato e sulla rispettiva votazione (10 volte al mese).

Dopo questa strutturazione dei requisiti, siamo pronti ad avviare la prima fase della progettazione che consiste nella costruzione di uno schema concettuale in grado di descrivere in maniera adeguata tutte le specifiche dei dati raccolte.

7.2 RAPPRESENTAZIONE CONCETTUALE DI DATI

Cerchiamo di stabilire alcune buone pratiche per una corretta rappresentazione concettuale dei dati.

Iniziamo da alcuni criteri generali di rappresentazione per poi passare a una rassegna di alcuni classici *design pattern*, ovvero soluzioni progettuali a problemi comuni della progettazione concettuale dei dati.

7.2.1 Criteri generali di rappresentazione

Precisiamo che non esiste una rappresentazione univoca di un insieme di specifiche, perché le stesse informazioni possono essere rappresentate in modi differenti e non comparabili. Comunque, quando ci si trova davanti a diverse possibilità, è utile avere delle indicazioni sulle scelte più opportune. Nel caso della progettazione concettuale conviene, in buona sostanza, seguire le “regole concettuali” del modello E-R.

- *Se un concetto ha proprietà significative e/o descrive classi di oggetti con esistenza autonoma, è opportuno rappresentarlo con un'entità.*

Per esempio, il concetto di *docente* è naturale rappresentarlo con un'entità, in quanto possiede diverse proprietà (cognome, età e città di nascita) e la sua esistenza è indipendente dagli altri concetti. Chiaramente lo stesso discorso vale anche per concetti astratti come, per esempio, *corso*.

- *Se un concetto ha una struttura semplice e non possiede proprietà rilevanti associate, è opportuno rappresentarlo con un attributo di un altro concetto a cui si riferisce.*

Per esempio, il concetto di *età* è certamente da rappresentare come un attributo. In effetti anche il concetto di *città* può risultare autonomo e strutturato ma va rappresentato nella nostra applicazione come attributo perché, oltre al nome, non è di interesse nessun'altra sua proprietà.

- *Se sono state individuate due (o più) entità e nei requisiti compare un concetto che le associa, questo concetto può essere rappresentato come una relazione.*

Per esempio, il concetto di *partecipazione a un corso* è certamente rappresentabile da una relazione tra entità (*partecipanti* e *corsi*). È importante sottolineare il fatto che questo vale nel caso in cui il concetto in questione non abbia, esso stesso, le caratteristiche dell'entità. Un esempio tipico è il concetto di *visita* relativo a pazienti e medici: è assai improbabile che questo concetto possa essere rappresentato da una relazione tra paziente e medico. Perché di una visita sono di interesse diverse proprietà, come l'orario e la diagnosi. Ma soprattutto perché, per poter rappresentare il fatto molto plausibile che lo stesso paziente può sostenere più visite con lo stesso medico, allora la visita deve essere per forza rappresentata da una entità collegata da relazioni uno a molti con le entità che rappresentano i pazienti e i medici.

- *Se uno o più concetti risultano essere casi particolari di un altro, è opportuno rappresentarli facendo uso di una generalizzazione.*

Nella nostra applicazione è evidente che i concetti di *professionista* e *dipendente* costituiscono dei casi particolari del concetto di *partecipante* ed è quindi indicato definire una generalizzazione tra le entità che rappresentano questi concetti.

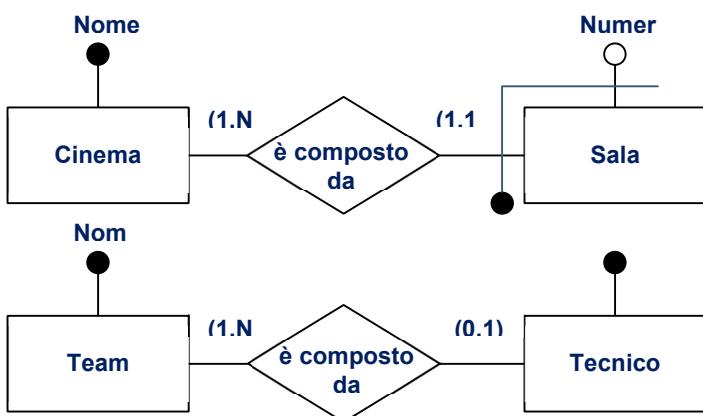
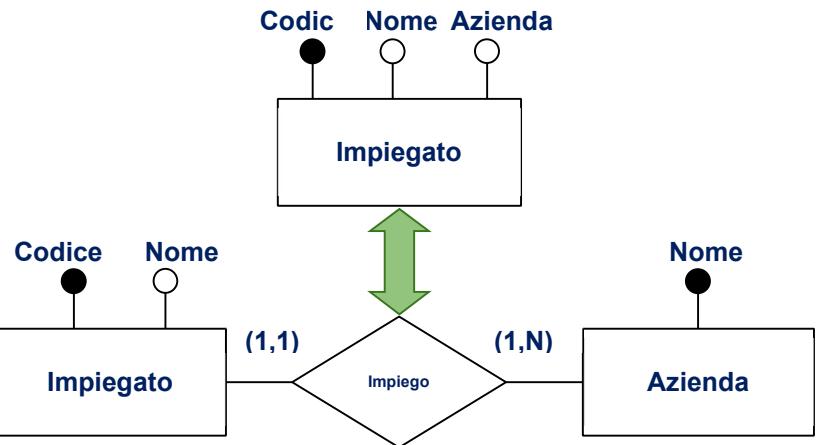
I criteri visti hanno validità generale, sono cioè indipendenti dalla strategia di progettazione scelta. Come vedremo nel prossimo paragrafo, infatti, in ogni strategia esiste priamo o poi un momento in cui va presa una decisione sul costrutto da scegliere per rappresentare una certa specifica.

7.2.2 Pattern di progetto

Cominciamo da un caso semplice: quello in cui si individua nelle specifiche un concetto autonomo con proprietà associate, le chiare caratteristiche di un'entità del modello E-R.

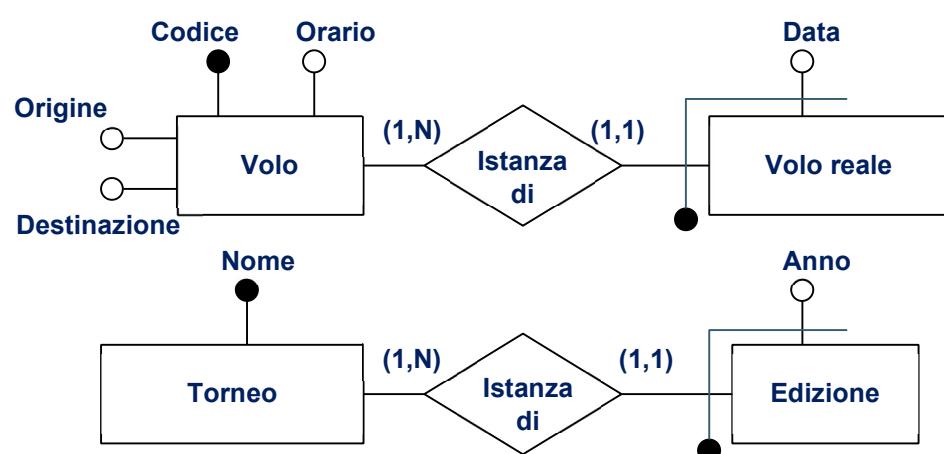
Nel caso, per esempio di un impiegato di cui sono di interesse un codice, il nome e l'azienda nella quale lavora, otteniamo il primo, semplice schema con una sola entità.

Con questa soluzione non stiamo rappresentando anche il concetto di *azienda*: qui l'azienda è solo un attributo, ovvero niente di più che una stringa che assegniamo a un'occorrenza di **Impiegato**. Per poter rappresentare esplicitamente il concetto di *azienda* dobbiamo reificare (considerare come concreto ciò che è astratto) l'attributo, facendolo diventare un'entità.



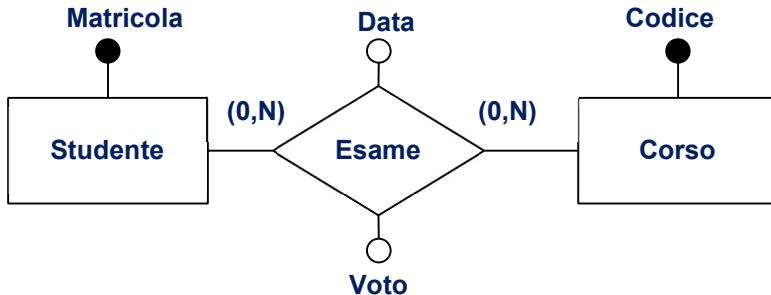
Osserviamo ora dei semplici pattern che coinvolgono le relazioni. Un caso frequente di uso di questo costrutto è quello in cui si vuole rappresentare il fatto che un'entità è *parte di* un'altra entità, come avviene negli schemi qui sotto. Queste relazioni sono tipicamente uno a molti e si presentano in due forme. Nel primo caso, l'esistenza di un'occorrenza dell'entità "parte" dipende dall'esistenza di un'occorrenza dell'entità che la contiene (la sala di un cinema multisala) e richiede un'identificazione esterna. Nel secondo, l'entità contenuta nell'altra (il tecnico di un team) ha esistenza autonoma, come indicato dalla partecipazione opzionale alla relazione.

Un'altra situazione piuttosto comune è quella illustrata qua sotto, negli esempi le occorrenze di un'entità della relazione sono *istanze* di occorrenze dell'altra entità. Nel primo caso abbiamo un'entità che descrive il concetto astratto di *volo* presente sull'orario di una compagnia aerea, con un codice (per esempio "AZ610") un'origine (Roma), una destinazione (New York) e un orario (14:15), e un'altra entità che rappresenta il volo "reale", vale a dire l'istanza di un certo volo in un certo giorno (per esempio il volo "AZ610" del

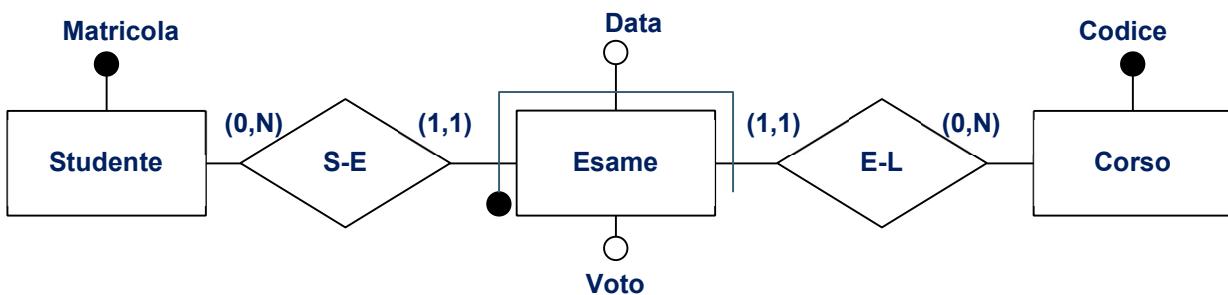


15/12/2024). È facile far confusione tra questi due concetti che però vanno tenuti distinti perché giocano ruoli diversi nell'applicazione. L'identificazione del volo reale avviene attraverso la data e, esternamente, il volo di cui è istanza (si assume dunque che lo stesso volo non può essere ripetuto lo stesso giorno).

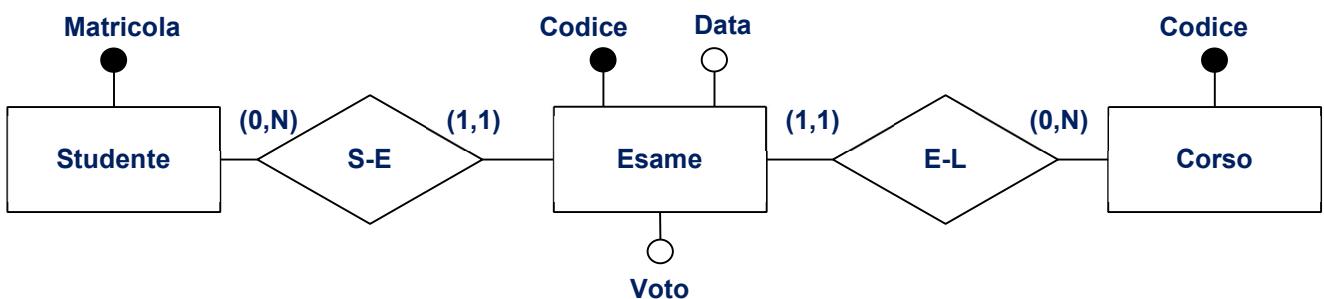
Consideriamo ora il caso in cui si utilizza una relazione, tipicamente molti a molti, per descrivere un concetto che lega altri due concetti, come avviene nell'esempio qua sotto, nel quale l'esame è rappresentato da una relazione tra lo studente e il corso. Questa soluzione è valida solo se ogni studente può sostenere una sola volta un certo esame perché, per definizione, un'occorrenza della relazione **Esame** è un insieme di coppie studente-corso, senza duplicati.



Il fatto che questo concetto abbia degli attributi associati non cambia la situazione, ci suggerisce piuttosto che, soprattutto nel caso in cui lo studente può sostenere più volte lo stesso esame, la soluzione corretta è lo schema in figura, nel quale abbiamo rettificato la relazione **Esame** di prima rappresentandola come entità. In questo caso, l'identificazione di un esame avviene attraverso lo studente, il corso e la data dell'esame.

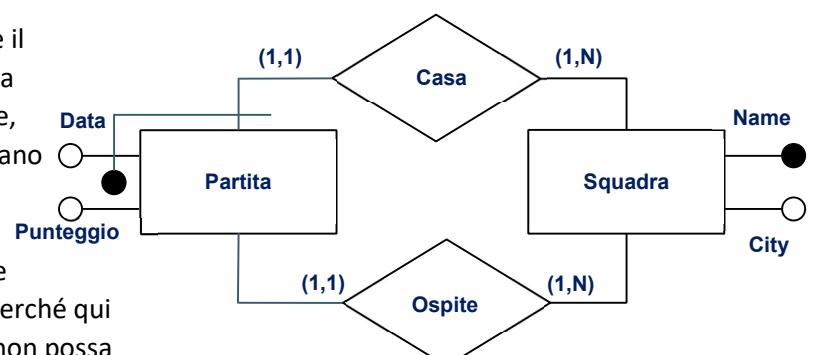


Una soluzione alternativa che non richiede un'identificazione esterna complessa è costituita dal seguente schema, nel quale è stato introdotto un codice identificativo. Questa scelta semplifica le cose ma bisogna tenere conto del fatto che il codice è un concetto nuovo, non presente nelle specifiche e che quindi dovrà essere opportunamente gestito dal sistema informativo in via di sviluppo.

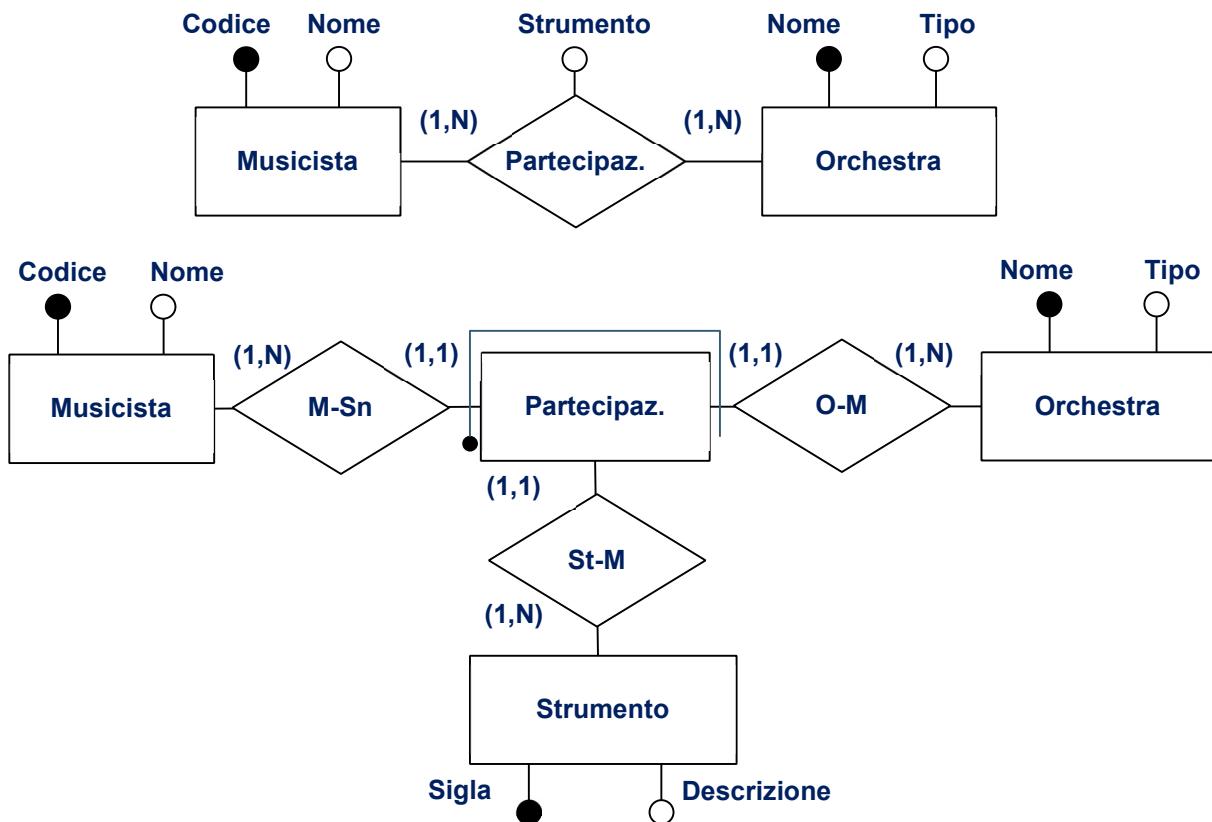


Un altro pattern comune è il seguente, dove il concetto di *parità* può essere visto come una relazione ricorsiva sull'entità Squadra. Ma se, come spesso accade, in un torneo si incontrano più volte, è necessario reificare la relazione binaria e ottenere lo schema figura.

L'identificazione dell'entità partita coinvolge solo la data e la squadra che gioca in casa perché qui evidentemente si assume che una squadra non possa giocare due partite nello stesso giorno.

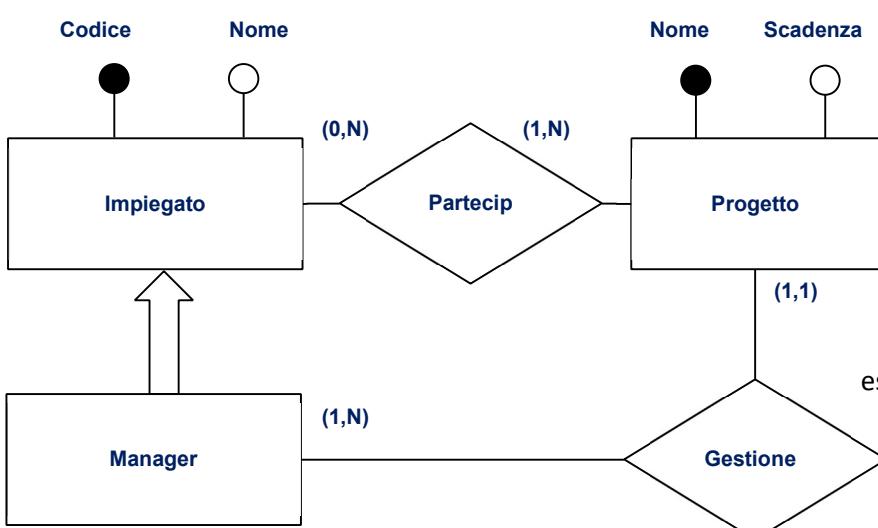


Consideriamo ora la relazione molti a molti che rappresenta la partecipazione di un musicista a un'orchestra con un certo strumento. In base a quanto esposto prima, se il musicista può suonare strumenti diversi ma suona, per ogni orchestra, sempre lo stesso strumento, lo schema è corretto ed è sufficiente una relazione con l'attributo Strumento. Il difetto di questo schema è semmai un altro e ha a che fare con quanto detto per lo schema Impiegato-Azienda: non stiamo rappresentando esplicitamente il concetto di *strumento* che qui è solo una stringa. Se lo strumento è un concetto rilevante per l'applicazione, dobbiamo reificare l'attributo della relazione e, per poterlo fare, dobbiamo reificare anche la relazione. Si ottiene in questo modo lo schema successivo.

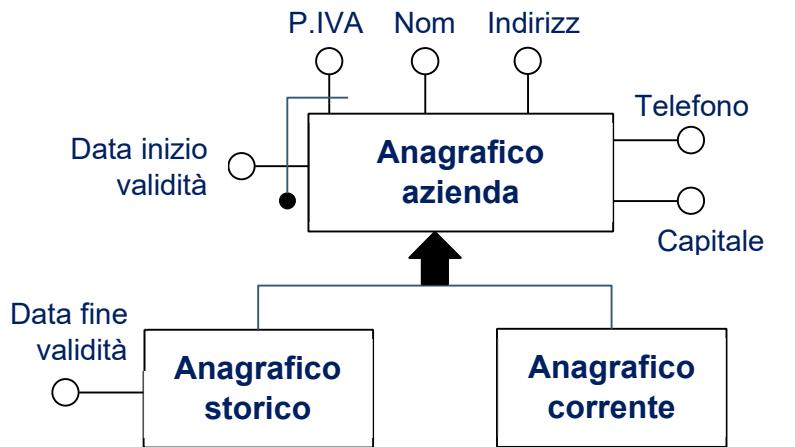


Passiamo ora ad alcuni pattern che coinvolgono le generalizzazioni. Un primo esempio è il seguente, dove si vuole rappresentare un caso particolare di un altro, nell'esempio il sottoinsieme degli impiegati che sono dei manager. Si noti come sia possibile specializzare in questo modo i vari ruoli all'interno di un progetto (l'altra entità dello schema): la gestione è a carico solo dei manager. In questo schema è ragionevole assumere che

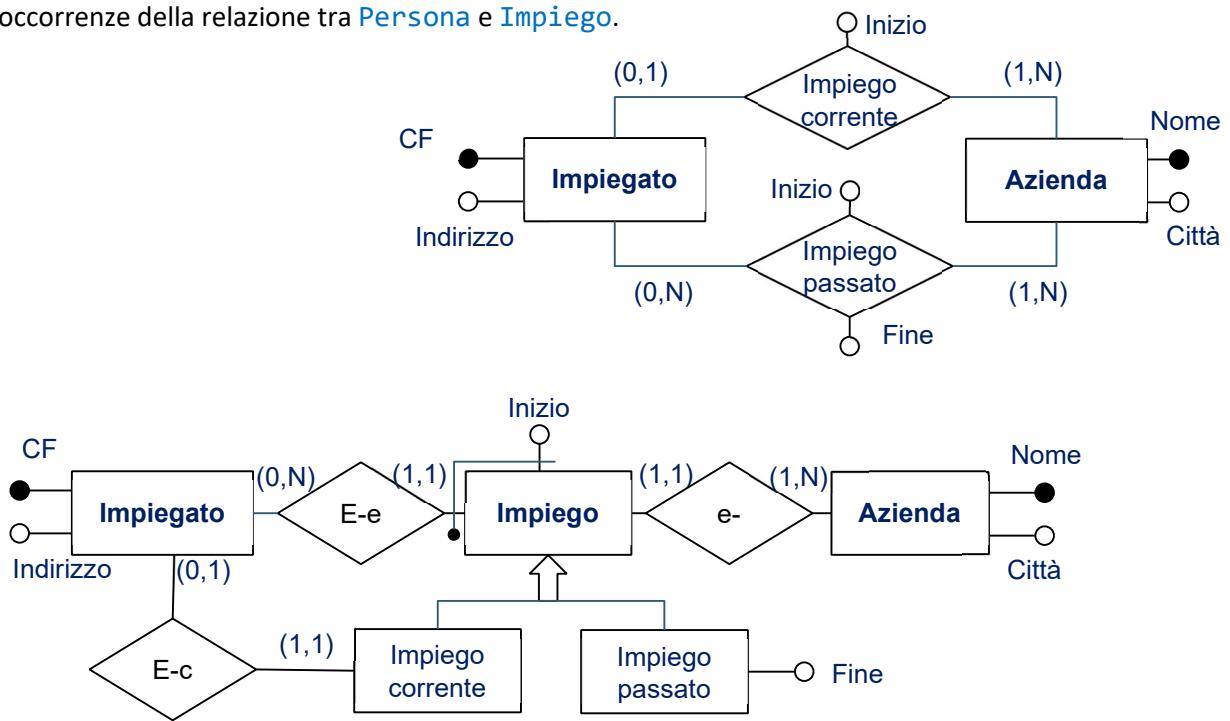
un manager può gestire solo un progetto al quale partecipa.
Questo implica che ogni coppia manager-progetto che compare tra le occorrenze della relazione **Gestione** deve comparire anche tra le occorrenze della relazione **Partecipazione**. Questo vincolo però non può essere espresso direttamente sullo schema con un apposito costrutto e va quindi aggiunta una regola alla documentazione dello schema.



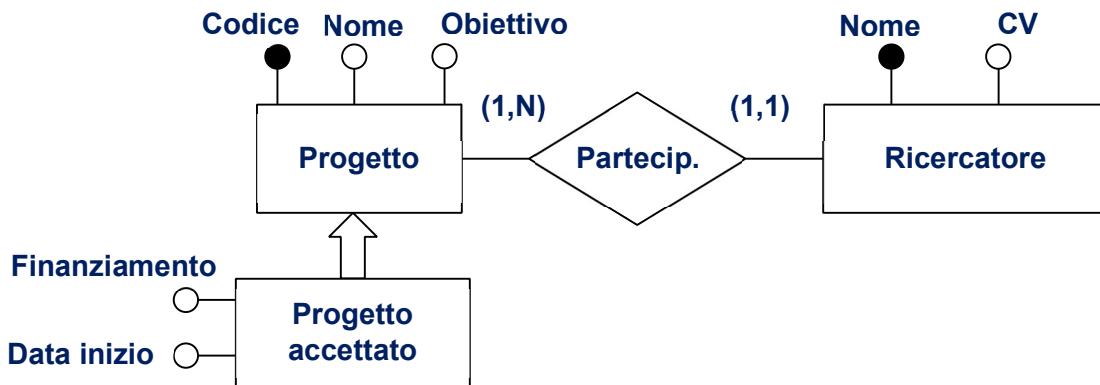
Lo schema in figura mostra un altro caso di uso comune del costrutto di generalizzazione. Si tratta di uno schema nel quale si vuole gestire la "storicizzazione" un concetto: nel caso particolare, di un'entità. Nell'esempio vogliamo memorizzare le informazioni correnti di un'azienda tenendo però traccia dei dati che sono variati. Come suggerito dallo schema, una soluzione piuttosto efficace consiste nell'utilizzare allo scopo due entità con gli stessi attributi: una rappresenta il concetto di interesse con le informazioni aggiornate, l'altra lo "storico". Le proprietà di queste entità vengono messe a fatto comune mediante una generalizzazione la cui entità genitore rappresenta tutte le informazioni anagrafiche delle aziende, sia quelle correnti sia quelle passate. Vengono inoltre introdotti degli attributi per definire l'intervallo di validità dei dati (data inizio e data fine). L'identificazione si ottiene aggiungendo all'identificatore "naturale" la data di inizio di validità delle informazioni, ovvero il momento in cui esse sono state introdotte: questo istante diventerà anche la data di fine validità delle informazioni che vengono soppiantate.



Un caso analogo è mostrato in figura. In questo caso si vuole storicizzare un concetto rappresentato da una relazione tra entità, nell'esempio gli impieghi presenti e passati di una persona (il suo curriculum lavorativo). Come nel caso precedente, una possibile soluzione consiste nel rappresentare separatamente i dati correnti e i dati storici e introdurre opportuni nuovi attributi per specificare gli intervalli di validità delle informazioni. Notare le differenti cardinalità delle partecipazioni dell'entità **Persona** alle due relazioni. Un'analisi attenta di quest'ultimo schema ci fa comprendere che qui non possiamo rappresentare il fatto che una persona possa aver lavorato, in periodi diversi, per la stessa azienda. Avremmo infatti in questo caso due occorrenze identiche della relazione **Impiego passato**. La soluzione in questo caso è, ancora una volta, la reificazione delle relazioni. Otteniamo così lo schema in figura che consente di utilizzare una generalizzazione. Anche in questo caso risulta necessario l'inserimento di un vincolo esterno allo schema che impone a tutte le occorrenze della relazione tra **Persona** e **Impiego corrente** compaiano anche tra le occorrenze della relazione tra **Persona** e **Impiego**.

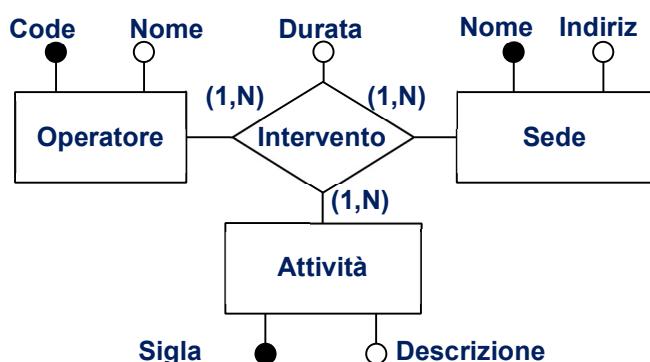


L'ultimo esempio d'uso comune del costrutto di generalizzazione è quello riportato di seguito. In questo schema vogliamo rappresentare il fatto che un certo concetto subisce un'evoluzione nel tempo che può essere diversa per le diverse occorrenze del concetto. Nell'esempio abbiamo dei progetti che vengono proposti con obiettivo di ottenere un finanziamento. Solo alcuni di questi vengono accettati e, per questi, vanno aggiunte ulteriori informazioni quali la data di inizio ufficiale del progetto e il finanziamento effettivamente assegnato. Il costrutto di generalizzazione si presta bene a modellare questa situazione.



Infine, consideriamo la relazione ternaria in figura. Come già accennato nel capitolo 6, negli schemi E-R le relazioni ternarie si incontrano raramente e relazioni che coinvolgono più di tre entità sono fortemente sconsigliate, perché tipicamente cercano di rappresentare, con un unico costrutto, concetti tra loro indipendenti. Nell'esempio in figura è stata scelta correttamente una relazione ternaria perché si vuole modellare il caso in cui un operatore può effettuare operazioni che consistono in attività diverse svolte in sedi diverse. Inoltre, in ogni sede possono operare operatori diversi svolgendo attività diverse. Infine, le

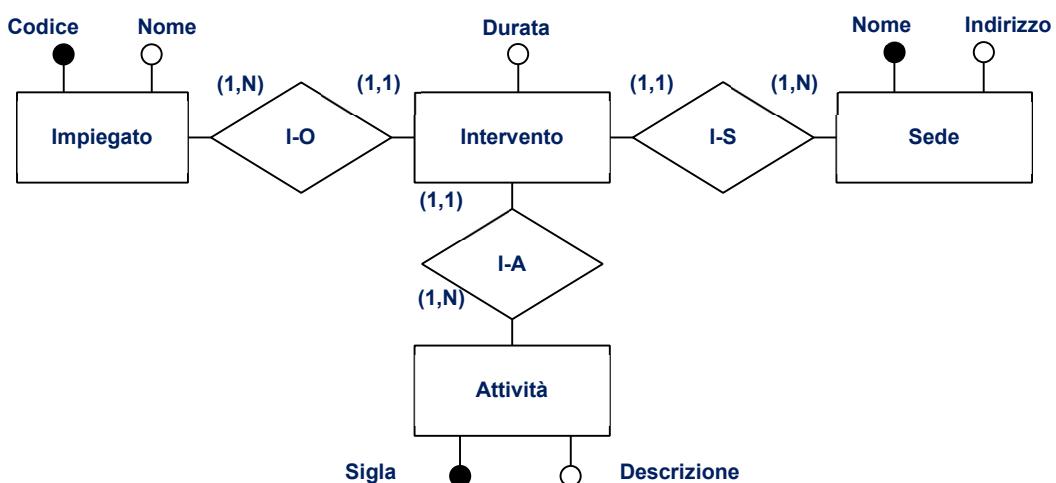
attività possono essere svolte da operatori diversi in sedi diverse.



Anche questa relazione, come tutte le altre, può essere reificata e questa operazione si rende necessaria quando la realtà da modellare è diversa da quella appena descritta. Il nuovo schema modella esattamente la situazione dello schema originario perché la nuova entità risulta identificata da tutte le entità originarie.

Cambiando opportunatamente l'identificazione,

siamo però in grado di modellare con questo pattern altre situazioni per le quali la relazione ternaria non sarebbe corretta.



7.3 STRATEGIE DI PROGETTO

Come possiamo mappare i nostri requisiti in istanze dello schema E-R?

Strategie:

- Top-Down

- Bottom-Up
- Inside-Out

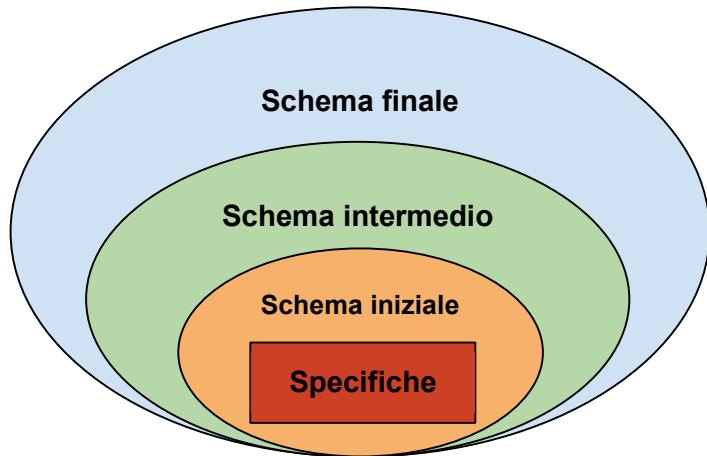
7.3.1 Strategia top-down

In questa strategia lo schema concettuale viene prodotto mediante una serie di raffinamenti successivi, a partire da uno schema iniziale che descrive tutte le specifiche con pochi concetti molto astratti. Lo schema viene poi via via raffinato mediante opportune trasformazioni che aumentano il dettaglio dei vari concetti presenti.

Nel passaggio da un livello di raffinamento a un altro, lo schema viene modificato facendo uso di alcune trasformazioni elementari che vengono denominate *primitive di trasformazione top-down*.

Esempi di primitive di trasformazione top-down sono:

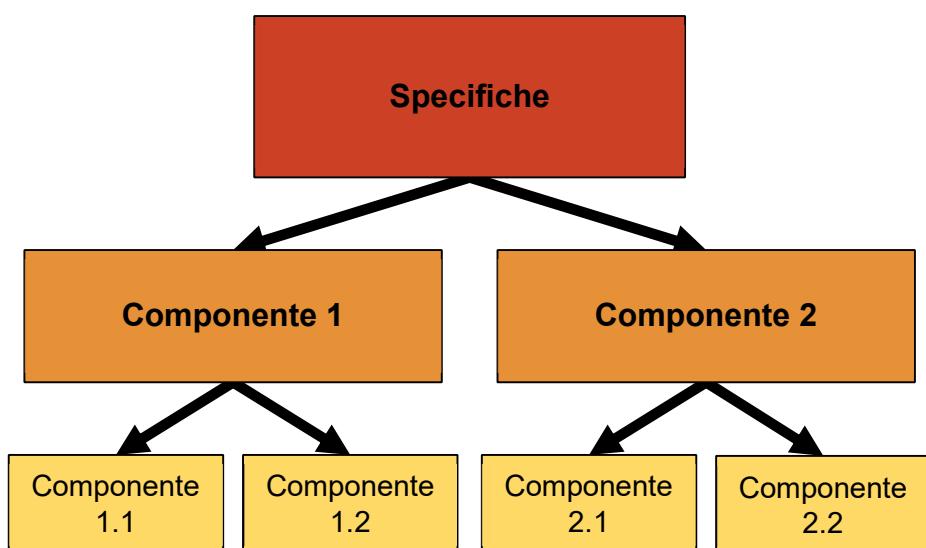
- La definizione degli attributi di un'entità o di una relazione; per esempio, la specifica, per un'entità **Persona**, di tutti gli attributi di interesse quali **Codice Fiscale**, **Cognome**, **Età**, **Sesso** e **Città di Nascita**.
- La reificazione di un attributo o di un'entità che viene trasformato come nel primo schema a pagina 72 (impiegato passa da entità a relazione).
- La decomposizione di una relazione in due relazioni distinte; per esempio, quella che consente di giungere da uno schema con una sola relazione **Impiego** tra le entità **Persona** e **Azienda** allo schema al penultimo schema di pagina 75.
- La trasformazione di un'entità in una gerarchia di generalizzazione; per esempio, quella che consente di giungere al primo schema a pagina 75.



Il vantaggio della strategia top-down è che il progettista può descrivere inizialmente tutte le specifiche dei dati trascurandone i dettagli, per poi entrare nel merito di un concetto alla volta. Questo però è possibile solo quando si possiede, sin dall'inizio una visione globale e astratta di *tutte* le componenti del sistema.

7.3.2 Strategia bottom-up

In questa strategia, le specifiche iniziali sono suddivise in componenti via via sempre più piccole, fino a quando queste componenti descrivono un frammento elementare della realtà di interesse. A questo punto, le varie componenti vengono rappresentate da semplici schemi concettuali che possono consistere anche in singoli concetti.

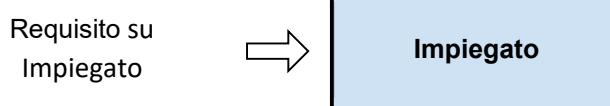


I vari schemi così ottenuti vengono poi fusi fino a giungere, attraverso una completa integrazione di tutte le componenti, allo schema concettuale finale.

Anche in questo caso lo schema finale si ottengono attraverso alcune trasformazioni elementari che vengono denominate **primitive di trasformazioni bottom-up** che introducono in uno schema nuovi concetti non presenti precedentemente e in grado di descrivere aspetti della realtà di interesse che non erano ancora stati rappresentati.

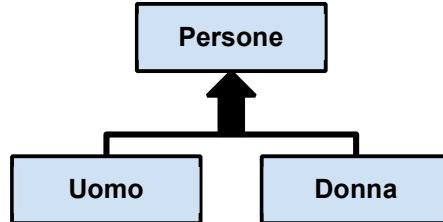
Esempi di primitive bottom-up sono:

- # Introduzione di una nuova entità o di una relazione dall'analisi delle specifiche



- # Aggregazione di una serie di attributi in un'entità o in una relazione

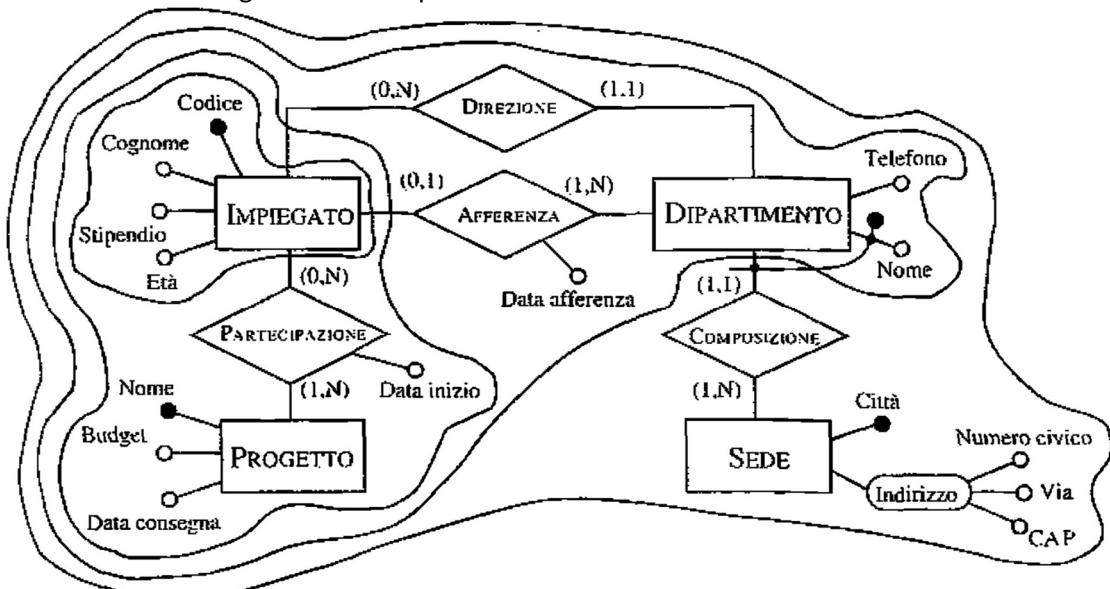
- # Individuazione nelle specifiche di un legame tra diverse entità riconducibile a una generalizzazione



Il vantaggio della strategia bottom-up è che si adatta a una decomposizione del problema in componenti più semplici, facilmente individuabili, il cui progetto può essere affrontato anche da progettisti diversi.

7.3.3 Strategia inside-out

Questa strategia può essere vista come un caso particolare della strategia bottom-up. Si individuano inizialmente solo alcuni concetti importanti e poi si procede, a partire da questi, a "macchia d'olio". Si rappresentano, cioè, prima i concetti in relazione con i concetti iniziali, per poi muoversi verso quelli più lontani attraverso una navigazione tra le specifiche.



7.3.4 Strategia mista

- Per prima cosa crea uno “sketch” usando le entità più rilevanti
- Poi decomponi lo schema
- Rifinisci (top-down), integra (bottom-up), espandi (inside-out)

Creare uno “sketch” dello schema E-R

Inizia dalle entità più rilevanti, sia perché sono quelle più citate o perché viene specificato che sono rilevanti, e tira giù un primo schema E-R basilare.

7.4 QUALITÀ DI UNO SCHEMA CONCETTUALE

Nella costruzione di uno schema concettuale vanno garantite alcune proprietà generali che uno schema concettuale di buona qualità deve possedere.

Correttezza Uno schema concettuale è *corretto* quando utilizza propriamente tutti i costrutti messi a disposizione dal modello concettuale di riferimento.

Completezza Uno schema concettuale è *completo* quando rappresenta tutti dati di interesse e quando tutte le operazioni possono essere eseguite a partire dai concetti descritti nello schema.

Leggibilità Uno schema concettuale è *leggibile* quando rappresenta i requisiti in maniera naturale e facilmente comprensibile. Per garantire questa proprietà è necessario rendere lo schema autoesplicativo, per esempio, mediante una scelta opportuna dei nomi da dare ai concetti.

Minimalità Uno schema è *minimale* quando tutte le specifiche sui dati sono rappresentate una sola volta nello schema.

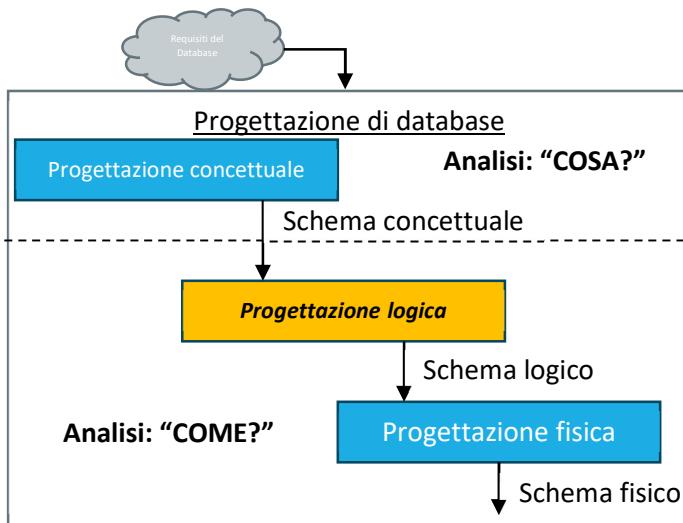
7.5 UNA METODOLOGIA GENERALE

Cerchiamo di tirare le somme su quanto detto relativamente alla progettazione concettuale di database. Per quel che riguarda le strategie di progetto viste va precisato che, in pratica, non accade quasi mai che un progetto proceda *sempre* in maniera top-down o bottom-up. Indipendentemente dalla strategia scelta, nelle situazioni reali capita infatti di modificare lo schema in via di costruzione sia con trasformazioni che raffinano un concetto presente (TD) sia con trasformazioni che aggiungono un concetto non presente (BU). Vediamo quindi una metodologia per la progettazione concettuale con il modello E-R con riferimento alla strategia mista che, come abbiamo detto, fa uso delle tecniche su cui si basano le altre e le comprende come caso particolare.

- | | |
|--|--|
| <ol style="list-style-type: none"> 1. Analisi dei requisiti <ol style="list-style-type: none"> a. Costruzione di un glossario dei termini b. Analizzare i requisiti ed eliminare le ambiguità c. Raggruppare i requisiti in insiemi omogenei
 2. Passo base <ol style="list-style-type: none"> a. Individuare i concetti più rilevanti e rappresentarli in uno schema scheletro.
 3. Passo di decomposizione (se necessario) <ol style="list-style-type: none"> a. Effettuare una decomposizione dei requisiti con riferimento ai concetti presenti nello schema scheletro | <ol style="list-style-type: none"> 4. Passo iterativo (ripeti finché non va bene) <ol style="list-style-type: none"> a. Raffinare i concetti presenti sulla base delle loro specifiche b. Aggiungere nuovi concetti allo schema per descrivere specifiche non ancora descritte
 5. Passo di integrazione <ol style="list-style-type: none"> a. Integrare i vari sottoschemi in uno schema generale facendo riferimento allo schema scheletro
 6. Analisi di qualità <ol style="list-style-type: none"> a. Verificare correttezza, completezza, minimalità e leggibilità |
|--|--|

(senza passi 3 e 5 e con solo raffinamenti nel 4 abbiamo una strategia top-down pura)

8 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.

Non si tratta di una semplice traduzione da un modello a un altro perché, prima di passare allo schema logico, lo schema Entità-Relazione va ristrutturato per soddisfare due esigenze: quella di "semplificare" la traduzione e quella di "ottimizzare" il progetto. La semplificazione dello schema si rende necessaria perché non

tutti i costrutti del modello E-R hanno una traduzione naturale nei modelli logici.

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. Pertanto, è necessario prevedere sia un'attività di *riorganizzazione*, sia un'attività di *traduzione* (dal modello concettuale a quello logico).

8.1 FASI DELLA PROGETTAZIONE LOGICA

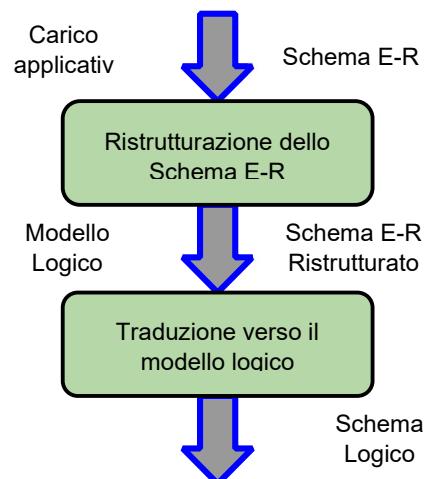
Le fasi principali sono la riorganizzazione dello schema concettuale e la traduzione in modello logico.

- **Ristrutturazione dello schema E-R:**

è una fase indipendente dal modello logico scelto e si basa su criteri di ottimizzazione dello schema e di semplificazione della fase successiva.

- **Traduzione verso il modello logico:**

fa riferimento a uno specifico modello logico (nel nostro caso il modello relazionale) e può includere un'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 è uno schema E-R ristrutturato che tiene conto degli aspetti realizzativi.

La seconda fase prende in input lo schema E-R ristrutturato prodotto prima e assieme al modello logico di riferimento produce lo schema logico del nostro database.

8.2 ANALISI DELLE PRESTAZIONI SU SCHEMI E-R

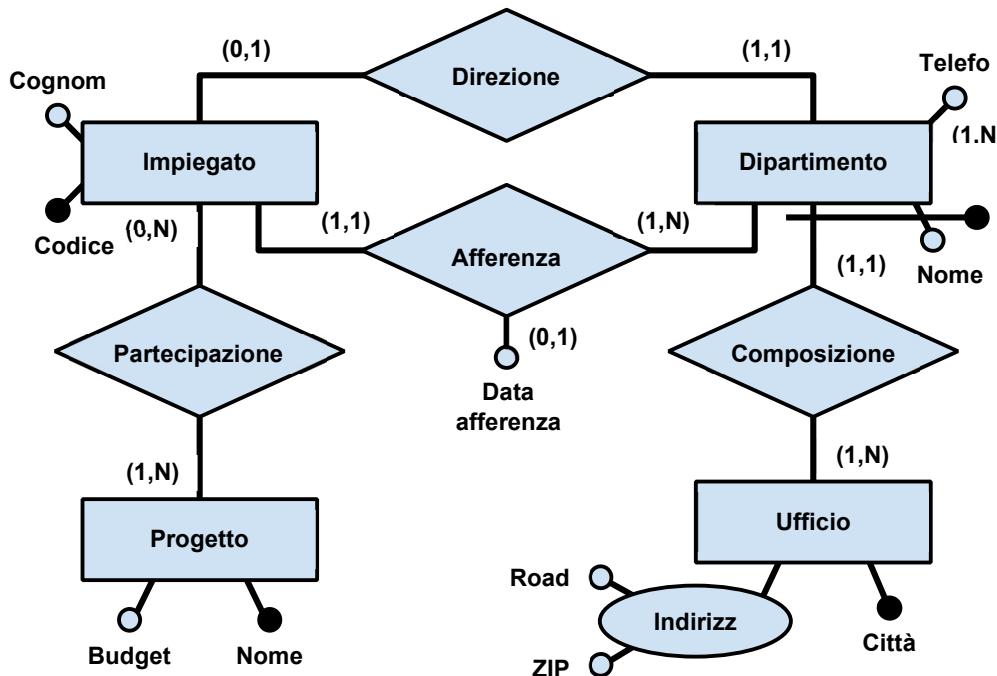
Uno schema E-R può essere ottimizzato per alcuni *indici di prestazione* del progetto. Parliamo di indici di prestazioni e non di prestazioni perché, in realtà, le prestazioni di una base di dati non sono valutabili in maniera precisa in sede di progettazione logica, in quanto dipendenti anche da parametri fisici, dal sistema di gestione di basi di dati che verrà utilizzato e altri fattori. Comunque possiamo effettuare studi di massima dei due parametri che regolano le prestazioni dei sistemi software.

- **Spazio:** numero delle istanze da immagazzinare
- **Tempo:** numero di istanze visitate durante un'operazione

Per studiare questi parametri abbiamo bisogno di conoscere, oltre allo schema, le seguenti informazioni:

- **Volume dei dati.** Ovvero:
 - Numero di occorrenze di ogni entità e associazione dello schema
 - Dimensioni di ciascun attributo (di entità o associazione)
- **Caratteristiche delle operazioni.** Ovvero:
 - Tipo dell'operazione (interattiva o *batch*)
 - Frequenza (numero medio di esecuzioni in un certo intervallo di tempo)
 - Dati coinvolti (entità e/o associazioni)

Per fare un esempio pratico, riprendiamo uno schema già incontrato:



Essendo uno schema riguardante dati sul personale di un'azienda, le operazioni possibili potrebbero essere:

Operazione 1: assegna un impiegato a un progetto

Operazione 2: trova i dati di un impiegato, del dipartimento nel quale lavora e dei progetti a cui partecipa

Operazione 3: trova i dati di tutti gli impiegati di un certo dipartimento

Operazione 4: per ogni sede, trova i suoi dipartimenti con il cognome del direttore e l'elenco degli impiegati del dipartimento

Sebbene un'analisi delle prestazioni che fa riferimento a un numero ristretto di operazioni possa sembrare riduttiva rispetto al reale carico del database, va notato che le operazioni sulle basi di dati seguono la cosiddetta regola "80-20. In base alla quale, l'80% del carico è generato dal 20% delle operazioni. Questo fatto ci consente di valutare adeguatamente il carico concentrando solo sulle operazioni principali previste.

Il volume dei dati e le caratteristiche generali possono essere descritti facendo uso di tabelle come le seguenti. 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).

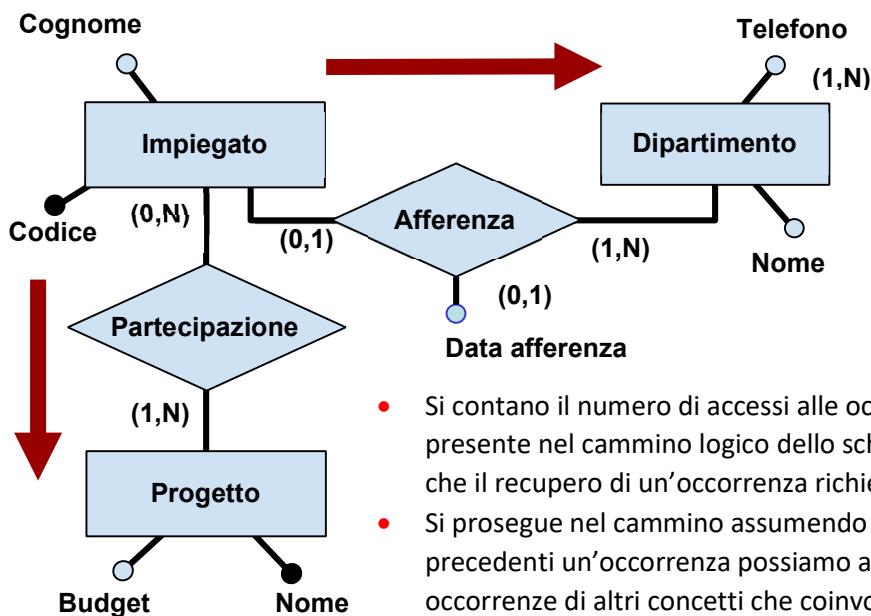
Nella tavola dei volumi, il numero delle occorrenze delle associazioni dipende da due parametri: il numero di occorrenze delle entità coinvolte nelle associazioni e il numero (medio) di partecipazioni di un'occorrenza di entità alle occorrenze di associazioni.

Per esempio, il numero di occorrenze dell'associazione **Composizione** è pari al numero dei dipartimenti, perché le cardinalità ci dicono che un dipartimento appartiene a una sola sede. Il numero di occorrenze dell'associazione **Afferenza** è invece poco meno del numero degli

impiegati, perché dalle cardinalità si evince che ci sono impiegati che non afferiscono a nessun dipartimento.

Infine, assumendo che un impiegato partecipa in media a tre progetti, abbiamo $2000 \times 3 = 6000$ occorrenze per l'associazione partecipazione (e quindi $6000/500 = 12$ impiegati in media su ogni progetto).

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



Un esempio è il seguente, con riferimento all'operazione 2: per ottenere le informazioni d'interesse su un impiegato si parte dall'entità **Impiegato** per accedere, attraverso l'associazione **Afferenza**, al suo dipartimento e, attraverso l'associazione **Partecipazione**, ai progetti ai quali partecipa.

- Si contano il numero di accessi alle occorrenze del primo concetto presente nel cammino logico dello schema di operazione, assumendo che il recupero di un'occorrenza richieda un solo accesso
- Si prosegue nel cammino assumendo che, avendo recupero nei passi precedenti un'occorrenza possiamo accedere tramite a essa a tutte le occorrenze di altri concetti che coinvolgono questa occorrenza.

Torniamo all'operazione 2: facendo riferimento allo schema di operazione della pagina precedente, dobbiamo innanzitutto accedere a un'occorrenza dell'entità **Impiegato** per accedere poi a un'occorrenza dell'associazione **Afferenza** (ogni impiegato afferisce al più a un dipartimento) e, attraverso questa, a un'occorrenza dell'entità **Dipartimento**. Successivamente, per conoscere i dati dei progetti ai quali lavora, dobbiamo accedere a tre occorrenze dell'associazione **Partecipazione** (perché abbiamo detto che in media un impiegato lavora a tre progetti) e, attraverso queste, a tre occorrenze dell'entità **Progetto** (per avere i dati sui progetti). Tutto questo può essere riassunto in una *tavola degli accessi*: (L = lettura; S = scrittura)

Tavola degli accessi (Op. 2)

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

8.3 RISTRUTTURAZIONE DI SCHEMI E-R

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

- **Analisi delle ridondanze.** Si decide se eliminare o mantenere eventuali ridondanze presenti nello schema.
- **Eliminazione delle generalizzazioni.** Tutte le generalizzazioni presenti nello schema vengono analizzate e sostituite da altri costrutti.
- **Partizionamento/accorpamento di entità e associazioni.** Si decide se è opportuno partizionare concetti dello schema (entità e/o relazioni) in più concetti o, viceversa, accorpare concetti separati in un unico concetto.
- **Scelta degli identificatori principali.** Si seleziona un identificatore per quelle entità che ne hanno più di uno.

8.3.1 Analisi delle ridondanze

Una ridondanza in uno schema E-E è un'informazione che è presente ma che può essere derivata da altre. In questa fase dobbiamo decidere se tenere, rimuovere o creare nuove ridondanze.

Pro

- Semplificano le interrogazioni

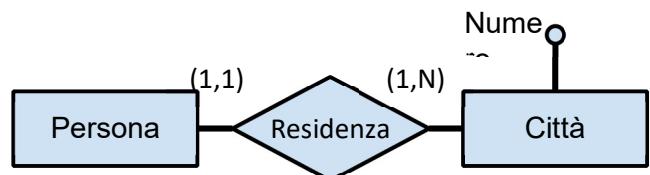
Contro

- Aumentano il tempo di aggiornamento
- Aumentano lo spazio consumato

Tipi di ridondanze:

- Attributi derivati:
 - Da altri attributi nella stessa entità (o relazione)
 - Da attributi di altre entità (o relazioni)
- Relazioni derivabili dalla composizione di relazioni differenti
 - (in generale: cicli di relazioni)

Consideriamo lo schema su persone e città in figura e supponiamo che faccia riferimento a un'applicazione anagrafica di una regione italiana per la quale sono definite le seguenti operazioni principali.



Operazione 1. Memorizza una nuova persona con la relativa città di residenza.

Operazione 2. Stampa tutti i dati di una città (incluso il numero di abitanti).

Tavola dei volumi		
Concetto	Tipo	Volume
Città	E	200
Persona	E	1 000 000
Residenza	R	1 000 000

Tavola delle operazioni		
Operazione	Tipo	Frequenza
Op. 1	I	500/giorno
Op. 2	I	2/giorno

Proviamo ora a valutare gli indici di prestazione in caso di presenza del dato ridondante (attributo **Numero abitanti** nell'entità **Città**).

Assumendo che il numero degli abitanti della città richieda 4byte, abbiamo che il dato ridondante richiede $4 \times 200 = 800$ byte, ovvero meno di un kilobyte di memoria aggiuntiva. Passiamo ora alla stima del costo delle operazioni.

Tavole degli accessi in presenza di ridondanza

Operazione 1			
Concetto	Costrutto	Accesso	Tipo
Persona	E	1	S
Residenza	R	1	S
Città	E	1	L
Città	E	1	S

Operazione 2			
Concetto	Costrutto	Accesso	Tipo
Città	E	1	L

Costi con ridondanza:

- Operazione #1: 1500 scritture + 500 letture / giorno
- Operazione #2: trascurabile

Accesso in scrittura costa il doppio

> Un totale di 3500 accessi al giorno

Tavole degli accessi in assenza di ridondanza

Operazione 1			
Concetto	Costrutto	Accesso	Tipo
Persona	E	1	S
Residenza	R	1	S

Operazione 2			
Concetto	Costrutto	Accesso	Tipo
Città	E	1	L
Residenza	R	5000	L

8.3.2 Eliminazione delle generalizzazioni

I sistemi tradizionali per la gestione delle basi di dati non consentono di rappresentare direttamente una generalizzazione, risulta quindi spesso necessario trasformare questo costrutto in altri costrutti del modello E-R per i quali esiste invece una rappresentazione naturale: le entità e le associazioni.

Abbiamo tre possibili soluzioni:

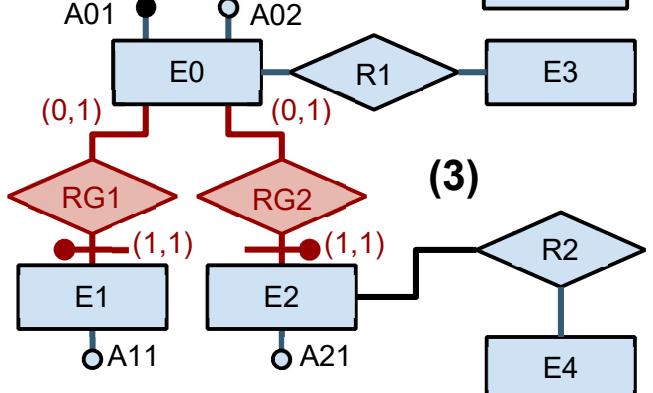
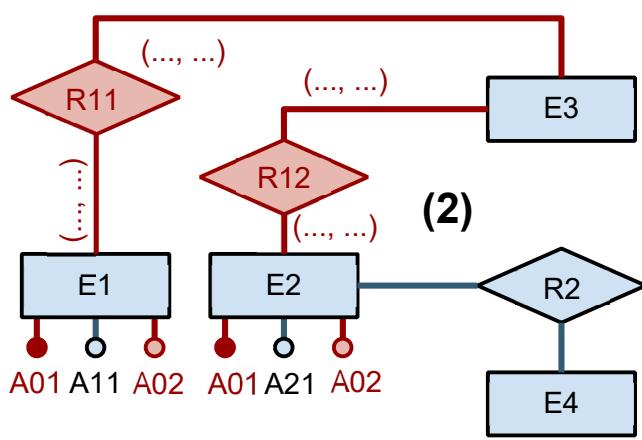
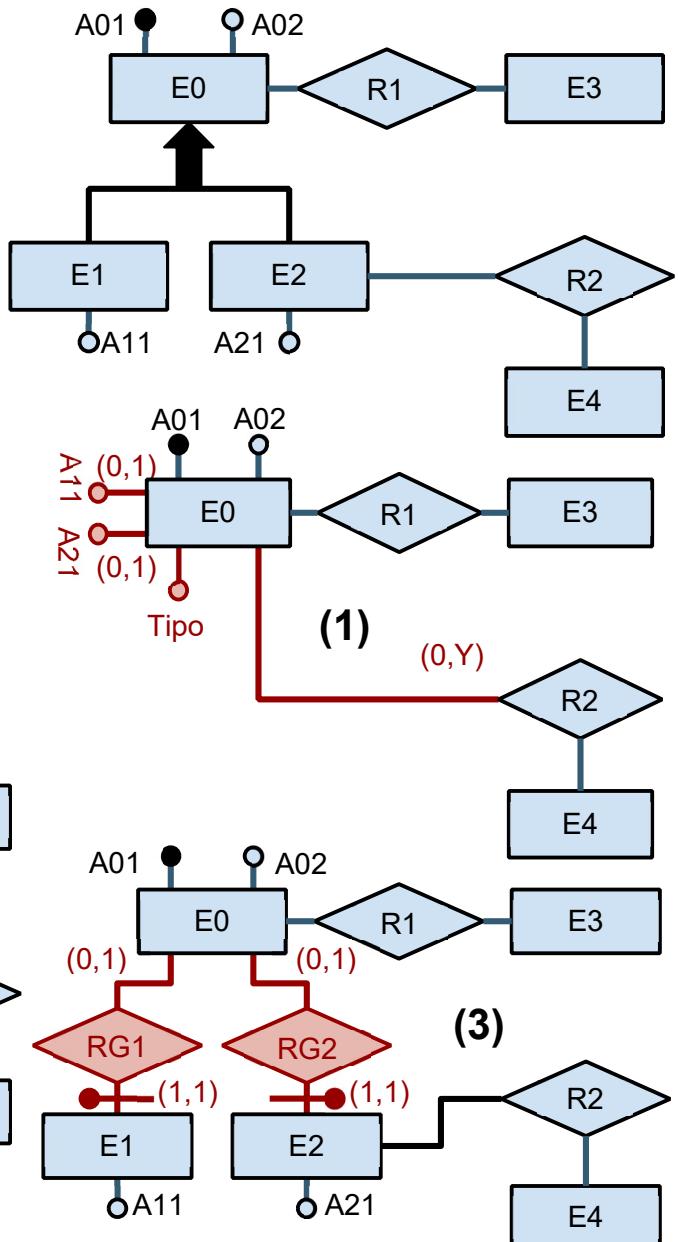
1. **Accorpamento dei figli della generalizzazione nel genitore**
2. **Accorpamento del genitore della generalizzazione nei figli**
3. **Sostituzione della generalizzazione con associazioni**

Costi senza ridondanza

- Operazione #1: 1000 scritture / giorno
- Operazione #2: 10 000 letture / giorno

Accesso in scrittura costa il doppio

> Un totale di 12 000 accessi al giorno



Osservazioni:

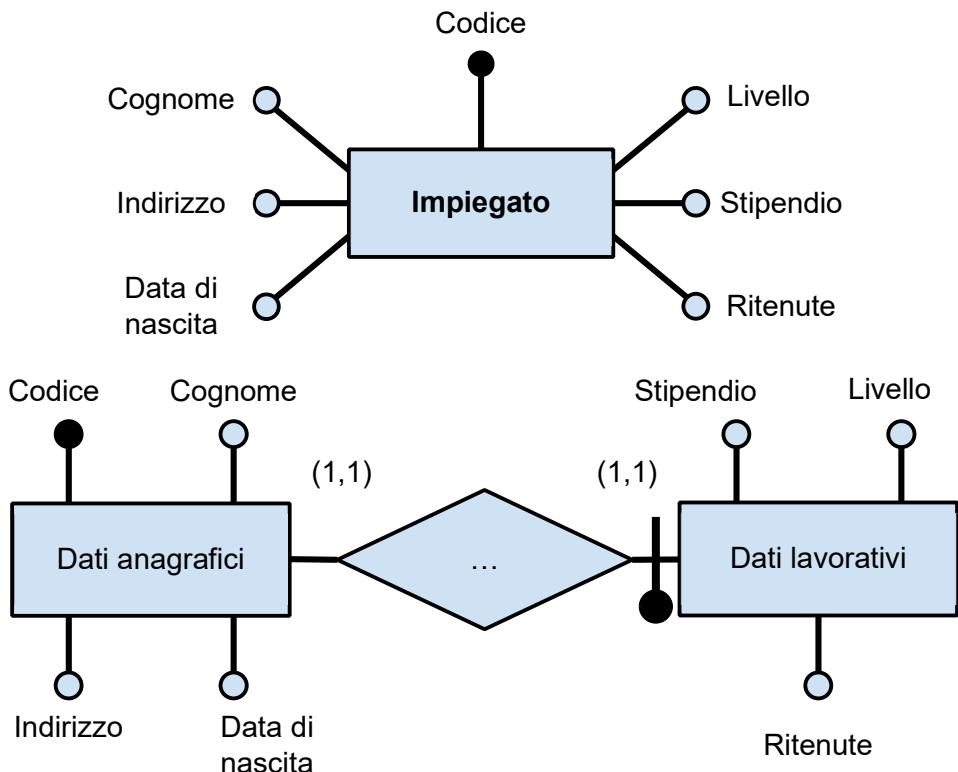
- Possiamo decidere tra tre alternative a seconda della dimensione della tavola degli accessi (considerare solo il numero degli accessi non basta)
- Ora possiamo fornire alcune regole generali che possono essere seguite
 1. **Accorpamento dei genitori**: dev'essere usato quando si accede a padri e figli allo stesso tempo
 2. **Accorpamento dei figli**: dev'essere usato quando i figli vengono visitati senza distinzione
 3. **Utilizzo delle relazioni**: dev'essere usato quando i figli vengono visitati indipendentemente dal padre
- Possiamo usare anche soluzioni "ibride", specialmente con gerarchie a più livelli

8.3.3 Partizionamento/accorpamento di concetti

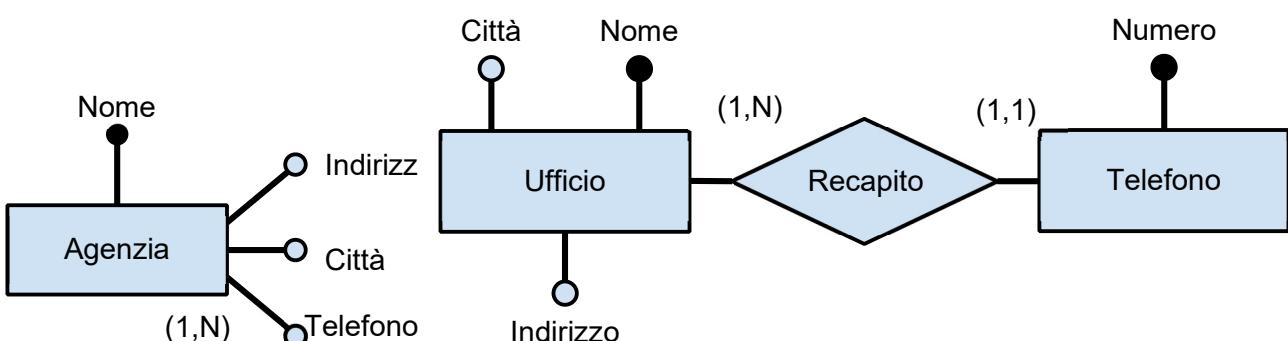
Ristrutturare può fornire operazioni più efficienti semplicemente riducendo il numero di accessi:

- Attributi acceduti separatamente vengono divisi
- Attributi acceduti allo stesso tempo vengono raggruppati, anche quando appartengono a entità/relazioni differenti

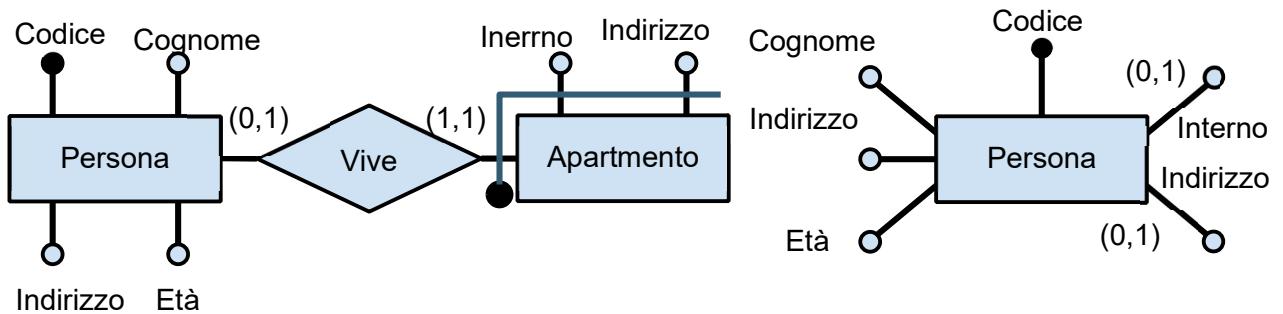
Partizionamenti di entità



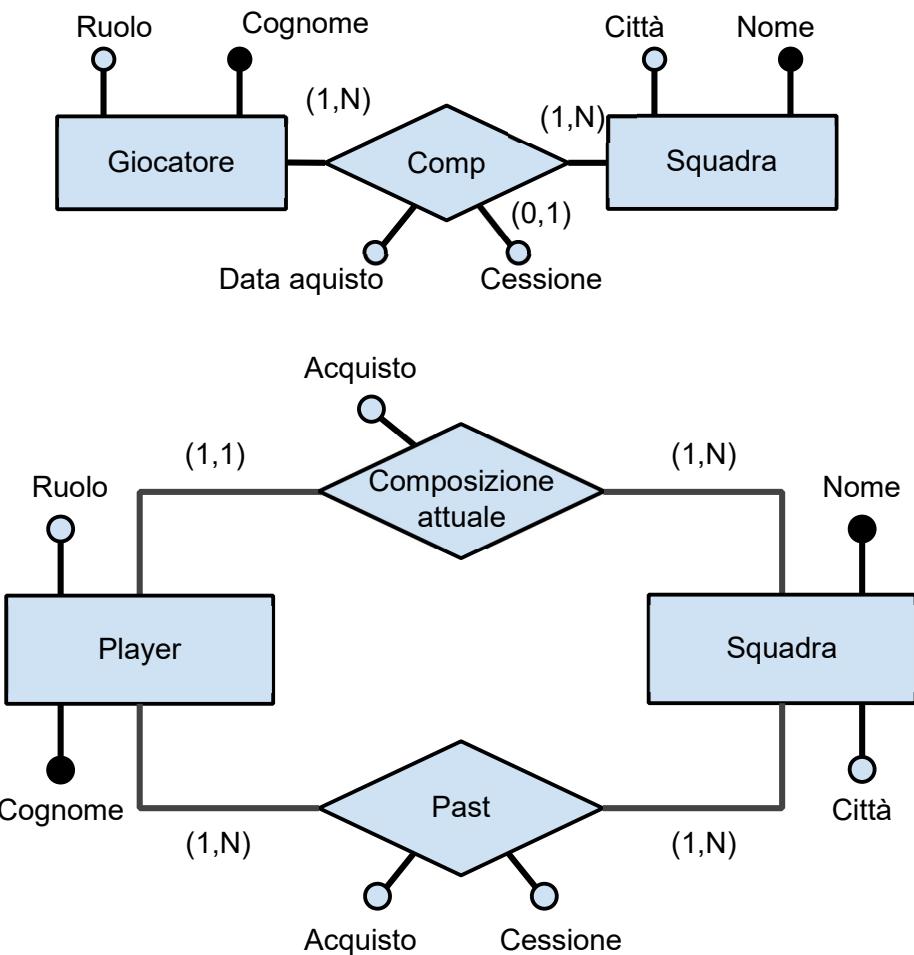
Eliminazione di attributi multivalore



Accorpamento di entità



Altri tipi di partizionamento/accorpamento



8.3.4 Scelta degli identificatori principali

È un'operazione obbligatoria per la traduzione in un modello relazionale

Criteri:

- Informazioni obbligatorie
- Semplicità
- Utilizzato all'interno dell'operazione più frequente/pertinente

E se nessuna delle condizioni è soddisfatta?

Nuovi attributi vengono introdotti utilizzando codici specificamente generati

9 NORMALIZZAZIONE

Le *forme normali* sono proprietà che “certificano” la qualità dello schema di una base di dati relazionale. Vedremo che, quando una relazione non soddisfa una forma normale, allora presenta ridondanze e si presta a comportamenti poco desiderabili durante le operazioni di aggiornamento.

Le forme normali sono generalmente definite sul modello relazionale, ma hanno senso anche in altri contesti, ad esempio nel modello E-R.

Per gli schemi che non soddisfano una forma normale è possibile applicare un procedimento detto *normalizzazione* che consente di trasformare questi schemi non normalizzati in nuovi schemi che garantiscono il soddisfacimento di una forma normale.

9.1 RIDONDANZE E ANOMALIE

Impiegati	Stipendi	Progetto	Budget	Ruolo
Jones	20	Mars	2	Tecnico
Smith	35	Jupiter	15	Designer
Smith	35	Venus	15	Designer
Williams	55	Venus	15	Capo
Williams	55	Jupiter	15	Consulente
Williams	55	Mars	2	Consulente
Brown	48	Mars	2	Capo
Brown	48	Venus	15	Designer
White	48	Venus	15	Designer
White	48	Jupiter	15	Direttore

Consideriamo questa relazione, ha come chiave l'insieme costituito dagli attributi **Impiegato** e **Progetto**. Si può verificare che le tuple hanno le seguenti proprietà:

1. Lo stipendio di ciascun impiegato è unico ed è funzionale del solo impiegato, indipendentemente dai progetti a cui partecipa.
2. Il bilancio di ciascun progetto è unico e dipende dal solo progetto, indipendentemente dagli impiegati che vi partecipano.

Questi fatti hanno delle conseguenze sul contenuto della relazione e sulle operazioni che si possono effettuare su di essa.

- **Ridondanza:** Il valore dello stipendio di ciascun impiegato è ripetuto in tutte le tuple relative a esso
- **Anomalia di aggiornamento:** Se lo stipendio di un impiegato varia bisogna modificarne il valore in tutte le tuple corrispondenti affinché la dipendenza continui a valere.

- **Anomalia di cancellazione:** Se un impiegato interrompe la partecipazione a tutti i progetti senza lasciare l'azienda, e quindi tutte le tuple corrispondenti al suo nome vengono eliminate, non è possibile tenere traccia del suo nome e del suo stipendio.
- **Anomalia di inserimento:** Se si hanno informazioni su un nuovo impiegato, non è possibile inserirlo finché questi non viene assegnato a un progetto.

Perché questa situazione è indesiderabile?

Perché diversi pezzi di informazioni vengono rappresentati nella stessa relazione.

- › Impiegati e i loro stipendi
- › Progetti e i loro budget
- › Il ruolo di ciascun impiegato nel progetto in cui lavorano

9.2 DIPENDENZE FUNZIONALI

Per studiare in maniera sistematica i concetti appena introdotti è necessario far uso di uno strumento di lavoro specifico: la *dipendenza funzionale*. Si tratta di un particolare vincolo di integrità per il modello relazionale che, come suggerisce il nome, descrive legami di tipo funzionale tra gli attributi di una relazione.

Consideriamo sempre la relazione di prima.

Ogni impiegato avrà sempre lo stesso stipendio (anche se partecipa a più progetti)

Ogni progetto avrà sempre lo stesso budget.

Ogni impiegato in ogni progetto avrà sempre lo stesso ruolo (anche se dovessero avere funzioni diverse in progetti differenti)

Def: (*Dipendenze Funzionali*)

Data una relazione r su uno schema $R(X)$ e due sottoinsiemi di attributi non vuoti Y e Z di X , diremo che esiste su r una dipendenza funzionale tra Y e Z se, per ogni coppia di tuple t_1 e t_2 di r aventi gli stessi valori sugli attributi Y , risulta che t_1 e t_2 hanno gli stessi valori anche sugli attributi Z .

Scriveremo:

$$Y \rightarrow Z$$

Per esempio:

$$\begin{array}{l} \text{Impiegato} \rightarrow \text{Stipendio} \\ \text{Progetto} \rightarrow \text{Bilancio} \end{array}$$

Employee	Wage	Project	Budget	Role
Jones	20	Mars	2	Technician
Smith	35	Jupiter	15	Designer
Smith	35	Venus	15	Designer
Williams	55	Venus	15	Chief
Williams	55	Jupiter	15	Consultant
Williams	55	Mars	2	Consultant
Brown	48	Mars	2	Chief
Brown	48	Venus	15	Designer
White	48	Venus	15	Designer
White	48	Jupiter	15	Director

- Impiegato → Stipendio
- Progetto → Budget
- Impiegato Progetto → Proge

Osservazioni:

1. Se $A_1, A_2, \dots, A_k \in Z \Rightarrow$ una relazione soddisfa $Y \rightarrow Z \Leftrightarrow$ tutte le k dipendenze $Y \rightarrow A_1, Y \rightarrow A_2, \dots, Y \rightarrow A_k$

2. La dipendenza funzionale

$$\text{Impiegato Progetto} \rightarrow \text{Progetto}$$

è verificata in quanto due tuple con gli stessi valori sulla coppia di attributi Impiegato e Progetto hanno ovviamente lo stesso valore sull'attributo Progetto.

In generale diremo che una DF $Y \rightarrow A$ è *non banale* se A non compare tra gli attributi di Y .

3. Se prendiamo una chiave K di una relazione r si può facilmente verificare se esiste una dipendenza funzionale tra K e ogni altro attributo dello schema di r .

Abbiamo detto che gli attributi Impiegato e Progetto formano una chiave. Possiamo allora affermare che vale la DF

$$\text{Impiegato Progetto} \rightarrow \text{Funzione}$$

In particolare, esisterà una dipendenza funzionale tra una chiave di una relazione e tutti gli attributi dello schema della relazione. Nel nostro caso abbiamo che:

$$\text{Impiegato Progetto} \rightarrow \text{Stipendio Bilancio Funzione}$$

9.3 FORMA NORMALE DI BOYCE E CODD

9.3.1 Definizione di forma normale di Boyce e Codd

Si possono introdurre proprietà dette *forme normali*, definite con riferimento alle dipendenze funzionali, che sono soddisfatte quando non ci sono anomalie.

La forma normale più importante è quella che prende il nome da Boyce e Codd (BCNF)

Definizione:

Una relazione r è in forma normale di Boyce e Codd se per ogni dipendenza funzionale non banale $X \rightarrow A$ definita su di essa, X contiene una chiave K di r , cioè X è superchiave per r .

9.3.2 Decomposizione in forma normale di Boyce e Codd

Quando una relazione non soddisfa BCNF, nella maggior parte dei casi la rimpiazziamo con due o più relazioni normalizzate che soddisfino la BCNF. Questo processo è chiamato **normalizzazione**.

Questo processo è basato su un semplice criterio: se una relazione rappresenta più concetti indipendenti, allora va decomposta in relazioni più piccole, una per ogni concetto.

Impiegati	Stipendio	Progetto	Budget	Ruolo
Jones	20	Mars	2	Technician
Smith	35	Jupiter	15	Designer
Smith	35	Venus	15	Designer
Williams	55	Venus	15	Chief
Williams	55	Jupiter	15	Consultant
Williams	55	Mars	2	Consultant
Brown	48	Mars	2	Chief
Brown	48	Venus	15	Designer
White	48	Venus	15	Designer
White	48	Jupiter	15	Director

Se sostituiamo alla tabella qua di fianco le tre tabelle sotto, ottenute attraverso delle proiezioni sugli insiemi di attributi corrispondenti ai tre concetti; eliminiamo anomalie e ridondanze.

Le tre relazioni, infatti, sono in forma normale di Boyce e Codd.

Impiegati	Reddito
Jones	20
Smith	35
Williams	55
Brown	48
White	48

Impiegato	Progetto	Ruolo
Jones	Mars	Technician
Smith	Jupiter	Designer
Smith	Venus	Designer
Williams	Venus	Chief
Williams	Jupiter	Consultant
Williams	Mars	Consultant
Brown	Mars	Chief
Brown	Venus	Designer
White	Venus	Designer
White	Jupiter	Director

Progetto	Budget
Mars	2
Jupiter	15
Venus	15