

Dispensa di

Basi di Dati

Prof. Nicola Tonellotto

A.A. 2022-2023

Università di Pisa

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Triennale Ingegneria Informatica

A cura di

Federico Nardi



Sommario

Premessa	3
Lezione 01: Introduzione	4
Lezione 02: Modello Relazionale	15
Lezione 03: Algebra e Calcolo Relazionale.....	25
Lezione 04: Metodologie e Modelli Progettuali	50
Lezione 05: Progettazione Concettuale	69
Lezione 06: Progettazione Logica.....	84
Lezione 07: Dipendenze Funzionali.....	102
Lezione 08: Normalizzazione.....	118
Lezione 09: Gestione del buffer e della memoria secondaria	132
Lezione 10: Ottimizzazione delle Interrogazioni.....	149
Lezione 11: Gestione Transazioni	155
Lezione 12: Gestione della Concorrenza.....	177

Premessa

La dispensa di Basi di Dati è stata reiscritta dalle slides del professor Tonellotto aggiungendo qualche piccolo appunto preso a lezione.

Non ne voglio prendere nessun merito, la lascio disponibile gratuitamente a tutta la componente studentesca del corso di Ingegneria Informatica ed a chi verrà da fuori.

Riconoscimenti:

Andrea Covelli

Simone Guercini

Federico Massini

Lorenzo Melani

Lezione 01: Introduzione

Che cos'è l'informatica?

L'informatica è la scienza del trattamento razionale dell'informazione, specialmente per mezzo di macchine automatiche, considerata come supporto alla conoscenza umana ed alla comunicazione.

Notiamo che questa definizione è composta da due anime:

- **metodologica**
- **tecnologica**

Sistema Organizzativo ed Informativo

Il **sistema organizzativo** è costituito da risorse e regole per lo svolgimento coordinato di attività (chiamati processi) per perseguire gli scopi propri di un'organizzazione (azienda o ente).

Risorse: possono essere persone, denaro, materiali, informazioni.

Invece, il **sistema informativo** è la componente del sistema organizzativo che acquisisce, elabora, conserva, produce le informazioni di interesse (utili al perseguimento degli scopi). Inoltre, esegue e gestisce i processi informativi (quindi i processi che coinvolgono informazioni).

Gestione delle Informazioni

Le operazioni di gestione delle informazioni sono:

- Raccolta, acquisizione;
- Archiviazione, conservazione
- Elaborazione, trasformazione, produzione
- Distribuzione, comunicazione, scambio

Sistema Informativo ed Automazione

Il concetto di **sistema informativo** è indipendente da qualsiasi automatizzazione: esistono organizzazioni la cui ragion d'essere è la gestione di informazioni e che operano da secoli senza impiegare automatizzazioni.

Esempio: servizi anagrafici e banche

La parte del sistema informativo che gestisce informazioni con tecnologia informatica si chiama sistema informativo automatizzato (**Sistema Informatico**).

Sistema Informatico

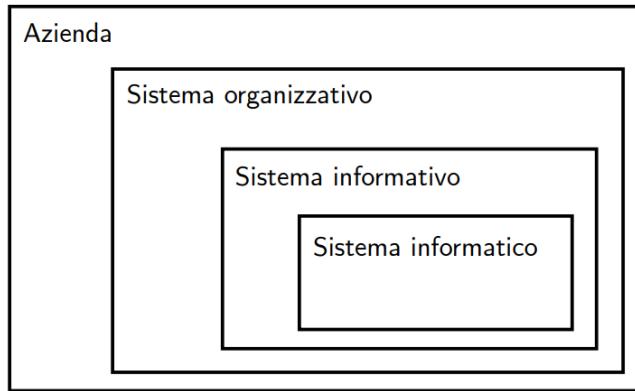


Figura 1: Sistema Informatico di esempio

Gestione delle Informazioni

Nelle attività umane, le informazioni vengono gestite in forme diverse:

- Idee formali
- Linguaggio naturale
- Disegni, grafici, schemi
- Numeri e codici

Su vari supporti come:

- Mente umana
- Carta
- Dispositivi elettronici

Informazioni e Dati

L'informazione è una notizia, un dato o elemento che consente di avere conoscenza più o meno esatta di fatti, situazioni, modi di essere

Il dato è ciò che è immediatamente presente alla conoscenza, prima di ogni elaborazione.

In informatica il dato è l'elemento di informazione costituito da simboli che debbono essere elaborati.

Dati ed Informazioni



Lun-Ven



Sabato



Festivo

- che cosa significano questi numeri?
- cartelli stradali, in Finlandia; sono orari!
- ma la differenza?
- senza "interpretazione" il dato serve a ben poco!

Gestione delle Informazioni

I dati sono spesso il risultato di forme di organizzazione e codifica delle informazioni

Esempio ripreso: nei servizi anagrafici facciamo riferimento a persone con le seguenti informazioni:

- *descrizioni discorsive*
- *nome e cognome*
- *estremi anagrafici*
- *codice fiscale*

Perché i dati?

La rappresentazione precisa di forme più ricche di informazione e conoscenza è difficile. Quindi i dati costituiscono spesso una risorsa strategica, perché sono più stabili nel tempo di altre componenti (processi, tecnologie, ruoli umani)

Basi di Dati

Il cuore di un sistema informativo automatizzato è la base di dati (database), cioè un insieme organizzato di dati utilizzati per rappresentare le informazioni di interesse
La definizione generica **metodologica**:

è un insieme organizzato di dati utilizzati per il supporto allo svolgimento delle attività di un ente (azienda, ufficio, persona).

La definizione generica **tecnologica** e **metodologica**:

un insieme di dati gestito da un DBMS.

Quindi le basi di dati:

- Hanno dimensioni (molto) maggiori della memoria centrale dei sistemi di calcolo utilizzati;
- Hanno un tempo di vita indipendente dalle singole esecuzioni dei programmi che le utilizzano (persistenza dei dati).

Sistema di Gestione di Basi di Dati

Il sistema di gestione di Basi di Dati (Database Management System, DBMS) gestisce collezioni di dati:

- **Grandi**: che hanno dimensioni molto maggiori della memoria centrale dei sistemi di calcolo utilizzati ed il limite deve essere solo quello fisico dei dispositivi. *Esempio di dimensioni molto grandi*:
 - *500 Gigabyte (dati transazionali)*;
 - *10 Terabyte (dati decisionali)*;
 - *500 Terabyte (dati scientifici)*;
 - *2.25 miliardi di pagine Web*.
- **Persistenti**: cioè che hanno un tempo di vita indipendente dalle singole esecuzioni dei programmi che le utilizzano.
- **Condivise**: cioè che ogni organizzazione è divisa in settori o comunque svolge diverse attività e ciascun settore/attività ha un sottosistema informativo non necessariamente disgiunto.

Garantisce quindi:

- privacy
- affidabilità
- efficienza
- efficacia

Esempio

Corso di Studi in Ingegneria Informatica			
ORARIO DELLE LEZIONI PER L'ANNO ACADEMICO 1999-2000			
INSEGNAMENTO	Docente	Aula	Orario
Analisi matematica I	Luigi Neri	N1	8:00-9:30
Basi di dati	Piero Rossi	N2	9:45-11:15
Chimica	Nicola Mori	N1	9:45-11:30
Fisica I	Mario Bruni	N1	11:45-13:00
Fisica II	Mario Bruni	N3	9:45-11:15
Sistemi informativi	Piero Rossi	N3	8:00-9:30

Corso di Studi in Ingegneria Informatica		
Orario di ricevimento dei docenti		
DOCENTE	INSEGNAMENTI	ORARIO
Mario BRUNI	Fisica I Fisica II	Martedì' 10-12
Luigi NERI	Analisi matematica I	Lunedì' 12-13
Piero ROSSI	Basi di dati Sistemi informativi	Giovedì' 11-13
Nicola MORI	Chimica	Martedì' 16-18

Figura 2: Esempio di una base di dati contenente informazioni

Problemi

Ci sono due problemi legati alle basi di dati:

- **Ridondanza:** ovvero informazioni ripetute;
- **Rischio di incoerenza:** ovvero che versioni possono non coincidere.

Archivi e Basi di Dati

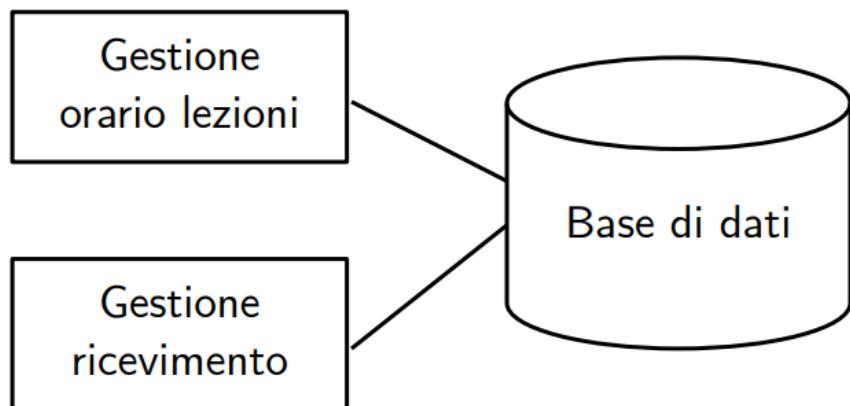


Figura 3:Tabelle contenute in una base di dati

Le Basi di Dati sono condivise

Una base di dati è una risorsa integrata, condivisa tra applicazioni. Ci sono delle conseguenze riguardo a ciò:

- Attività diverse su dati condivisi, devono essere gestiti da meccanismi di autorizzazione.
- Accessi di più utenti ai dati condivisi, deve esserci quindi il controllo della concorrenza.

I Database garantiscono privacy

Si possono definire meccanismi di autorizzazione come:

- L'utente A è autorizzato a leggere tutti i dati e modificare X;
- L'utente B è autorizzato a leggere i dati X ed a modificare Y.

I Database garantiscono affidabilità

Quindi l'affidabilità è una *resistenza a malfunzionamenti hardware e software*.

Una base di dati è una risorsa pregiata quindi deve essere conservata a lungo termine.

Introduciamo una tecnica fondamentale per i database: gestione delle transizioni.

Transazione

La transazione è l'insieme delle operazioni da considerare indivisibile (atomiche), corretto anche in presenza di concorrenza e con effetti definitivi.

Le transazioni possono essere quindi:

- **Atomiche**, ovvero una sequenza di operazioni correlate devono essere eseguite per intero o per niente.
Esempio: trasferimento di fondi da un conto A ad un conto B, quindi o si fanno il prelevamento da A ed il versamento su B o nessuno dei due.
- **Concorrenti**, ovvero che l'effetto di transizione deve essere coerente.
Esempio: se due agenzie richiedono lo stesso posto libero su un treno si deve evitare di assegnarlo due volte.

La conclusione positiva di una transazione corrisponde ad un **impegno (commit)** a mantenere traccia del risultato in modo definitivo, anche in presenza di guasti e di esecuzione concorrente.

I DBMS devono essere efficienti

I DBMS cercano di usare al meglio le risorse di spazio di memoria (principale e secondaria) e tempo (di esecuzione e risposta).

Quindi contenente molte funzioni, rischiano l'inefficienza e per questo ci sono grandi investimenti e competizione.

L'efficienza è anche il risultato della qualità delle applicazioni.

I DBMS devono essere efficaci

I DBMS cercano di rendere produttive le attività dei loro utilizzatori, offrendo funzionalità articolate, potenti e flessibili.

Il sistema informatico deve essere adeguatamente dimensionato e la base di dati ben progettata e realizzata.

DBMS vs File System

La gestione di insiemi di dati grandi e persistenti è possibile anche attraverso sistemi più semplici, come i file system dei sistemi operativi.

Quest'ultimi prevedono forme rudimentali di condivisione, ovvero "tutto o niente"

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

Descrizione dei Dati

Nei programmi tradizionali che accedono a file, ogni programma contiene una descrizione della struttura del *file* stesso, con i conseguenti rischi di incoerenza fra le descrizioni (ripetute in ciascun programma) ed i file stessi.

Nei DBMS esiste una porzione della base di dati che contiene una descrizione centralizzata dei dati, che può essere utilizzata dai vari programmi.

Descrizione dei dati nei DBMS

I programmi fanno riferimento ai dati, ma la loro struttura in memoria deve poter essere modificata senza dover modificare i programmi.

Viene introdotto il concetto di **modello dei dati**: è un insieme di costrutti utilizzati per organizzare i dati di interesse e descriverne la dinamica.

Il modello dei dati fornisce ai programmi applicativi una **vista astratta** dei dati.

Differenza tra Schema ed Istanza

In ogni base di dati esistono due concetti molto importanti, tra cui:

- **Schema**, che ne scrive la struttura e rimane invariata nel tempo sostanzialmente.
- **Istanza**, che sono i valori attuali che possono cambiare rapidamente nel tempo..

Lo **schema** è l'intestazione della tabella.

Insegnamento	Docente	Aula	Ora
Analisi Matem. I	Luigi Neri	N1	8:00
Basi di Dati	Piero Rossi	N2	9:45
Chimica	Nicola Mori	N1	9:45
Fisica I	Mario Bruni	N1	11:45
Fisica II	Mario Bruni	N3	9:45
Sistemi Inform.	Piero Rossi	N3	8:00

L'**istanza** sono le righe di ciascuna tabella

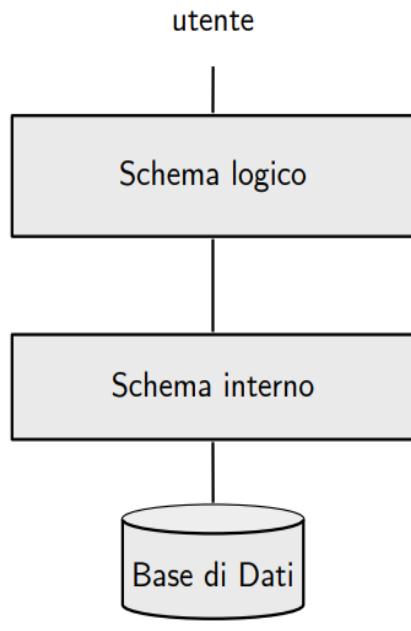
Figura 4: Modello ed Istanza in un DBMS

Modelli dei Dati

Ci sono diversi modelli dei dati come:

- **Modelli Logici**: sono adottati nei DBMS esistenti per l'organizzazione dei dati. Caratteristiche principali sono:
 - Utilizzati dai programmi;
 - Indipendenti dalle strutture fisiche.
Esempio: relazionale, reticolare, gerarchico, ad oggetti.
- **Modelli Concettuali**: permettono di rappresentare i dati in modo indipendente da ogni sistema. Caratteristiche principali sono:
 - cercano di descrivere i concetti nel mondo reale;
 - sono utilizzati nelle fasi preliminari di progettazione.
Esempio: il modello Entity-Relationship (ER).

Architettura Semplificata di un DBMS



Schema Logico: è la descrizione della base di dati nel modello logico.
Esempio: Struttura della tabella

Schema Interno (fisico): è la rappresentazione dello schema logico per mezzo di strutture di memorizzazione (*file*).
Esempio: record con puntatori

Figura 5: Architettura semplice di un DBMS

Il livello **Logico** è indipendente da quello **Fisico**, perché una tabella è utilizzata nello stesso modo qualunque sia la sua realizzazione fisica (che può cambiare nel tempo).
Noi vedremo solo lo schema Logico, non quello fisico.

Linguaggi per Basi di Dati

La disponibilità di vari linguaggi ed interfacce per la definizione di schemi per la lettura/modifica di istanze contribuisce all'efficacia del DBMS:

- Linguaggi testuali interattivi (SQL);
- Comandi (SQL) immersi in un linguaggio ospite (Java, C++, ...);
- Interfacce amichevoli (senza linguaggio testuale come Access).

Dobbiamo fare ora una distinzione terminologica dei linguaggi usati nel DBMS:

- **Data Definition Language (DDL)** viene usata per la definizione di schemi logici e fisici.
- **Data Manipulation Language (DML)** viene usata per l'interrogazione ed aggiornamento di istanze di badi di dati.

Architettura a tre livelli per DBMS

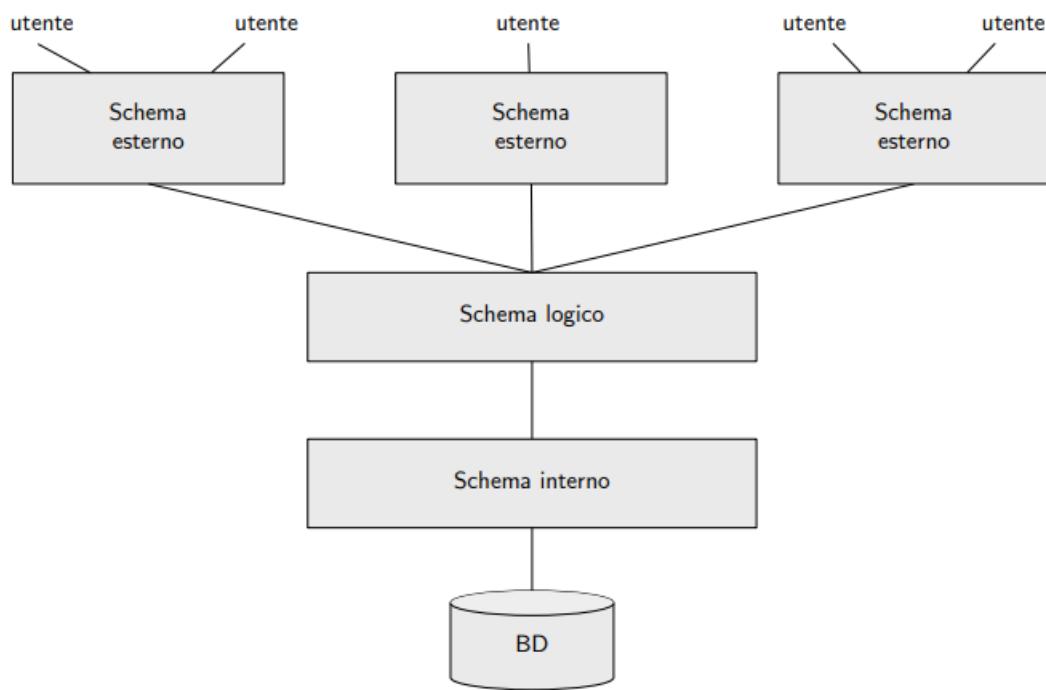


Figura 6: Architettura DBMS a 3 livelli

Quindi ricapitoliamo:

- **Schema Logico** rilascia una descrizione dell'intera base di dati nel modello Logico "principale" del DBMS;
- **Schema interno** (fisico) viene usato per la rappresentazione dello schema logico per mezzo di strutture fisiche di memorizzazione;
- **Schema esterno** dà una descrizione di parte della base di dati in un modello logico ("viste" parziali, derivate anche in modelli diversi).

Indipendenza dei Dati

L'accesso ai dati avviene solo tramite il livello esterno (che può coincidere con quello logico). Dobbiamo fare due distinzioni importanti (*vedi Figura 6*):

- **Indipendenza Fisica**: dove il livello Logico e quello esterno sono indipendenti da quello Fisico. Che vuol dire?
 - Una tabella è utilizzata nello stesso modo qualunque sia la sua realizzazione fisica;
 - La realizzazione fisica può cambiare senza che debbano essere modificati i programmi.

- **Indipendenza Logica:** dove il livello esterno è indipendente da quello Logico.
Che vuol dire?
 - Aggiunte o modifiche alle viste non richiedono modifiche a livello logico.
 - Modifiche allo schema logico che lasciano inalterato lo schema sono trasparenti

Personaggi

I personaggi che troveremo durante il corso di Basi di Dati sono i seguenti:

- Progettisti e realizzatori di DBMS;
- Progettisti della base di dati ed amministratori della base di dati;
- Progettisti e programmati di applicazioni;
- Utenti:
 - Utenti finali: che eseguono applicazioni predefinite (transazioni)
 - Utenti casuali: che eseguono operazioni non previste a priori, usando linguaggi interattivi

Vantaggi dei DBMS

I vantaggi legati alla nostra base di dati sono :

- Dati usati come risorsa comune, quindi base di dati come modello della realtà
- Gestione centralizzata con possibilità di standardizzazione ed “economia di scala”
- Disponibilità di servizi integrati
- Riduzione di ridondanze ed inconsistenze
- Indipendenza dei dati che favorisce lo sviluppo e la manutenzione delle applicazioni.

Svantaggi dei DBMS

Gli svantaggi legati alla nostra base di dati sono :

- Costo dei prodotti e della transizione verso di essi
- Non sopportabilità delle funzionalità (con riduzione di efficienza)

Lezione 02: Modello Relazionale

Modelli Logici

Riprendiamo il discorso dei modelli Logici e rifacciamo una classificazione:

- **Gerarchico e Reticolare**, che utilizzano riferimenti esplicativi (puntatori) tra record;
- **Relazionale (*quello che andremo a studiare*)**, che è basato su valori. Noteremo che anche i riferimenti fra dati in strutture diverse (chiamate relazioni) sono rappresentati per mezzo dei valori stessi.

Studenti

Matricola	Cognome	Nome	Nascita
6554	Rossi	Mario	05/12/78
8765	Neri	Paolo	03/11/76
9283	Verdi	Luisa	12/11/79
3456	Rossi	Maria	01/02/79

Esami

Studente	Voto	Corso
3456	30	4
3456	24	2
9283	28	1
6554	26	1

Corsi

Codice	Titolo	Docente
1	Analisi	Bruni
2	Chimica	Peri
4	Chimica	Fino

Il **Modello Relazionale** si basa sul concetto di Relazione (con una variante).

Le **relazioni** hanno naturale rappresentazione per mezzo di tabella.

Figura 7: Esempio di Modello Relazionale

Esempi di Relazione Matematica

- **Insieme:** $D_1 = \{a, b\}$ e $D_2 = \{x, y, z\}$
- **Prodotto Cartesiano:** $D_1 \times D_2 = \{(a,x), (a,y), (a,z), (b,x), (b,y), (b,z)\}$
- **Relazione:** $R = \{(a,x), (a,z), (b,y)\} \subseteq D_1 \times D_2$

Relazione Matematica

Siano dati n insiemi (anche non distinti) D_1, D_2, \dots, D_n

Definiamo il loro prodotto cartesiano $D_1 \times D_2 \times \dots \times D_n$ come l'insieme di tutte le **n -uple** (d_1, d_2, \dots, d_n) tali che $d_1 \in D_1, d_2 \in D_2, \dots, d_n \in D_n$

Una **relazione matematica** è un sottoinsieme di $D_1 \times D_2 \times \dots \times D_n$

Dobbiamo fare attenzione quindi ad eventuali caratteristiche:

1. Non c'è un ordinamento tra le *n-uple*, dette anche **tuple**
2. Non esistono *n-uple uguali*
3. Ogni *n-upla* è **ordinata**, ovvero che l'*i*-esimo valore d_i proviene dall' *i-esimo* insieme D_i
4. Gli insiemi D_1, \dots, D_n sono i **domini** delle relazioni

Struttura Posizionale

Sia $Partite \subseteq \text{String} \times \text{String} \times \mathbb{N} \times \mathbb{N}$

Juventus	Lazio	3	1
Lazio	Milan	2	0
Juventus	Roma	0	2
Roma	Milan	0	1

Ciascuno dei domini ripetuti ha ruoli diversi che sono distinguibili attraverso la **posizione**.

La **struttura** si dice che è **posizionale**.

Figura 8: Esempio di Struttura Posizionale

Struttura Non Posizionale

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

Quindi a ciascun dominio si associa un nome unico nella tabella, detto **attributo**, che ne descrive il ruolo.

Figura 9: Esempio di Struttura Non Posizionale

Il Modello Relazionale è basato sui valori

I riferimenti fra dati in **relazioni diverse** sono rappresentati per mezzo di **valori** dei domini che compaiono nelle *n-uple*.

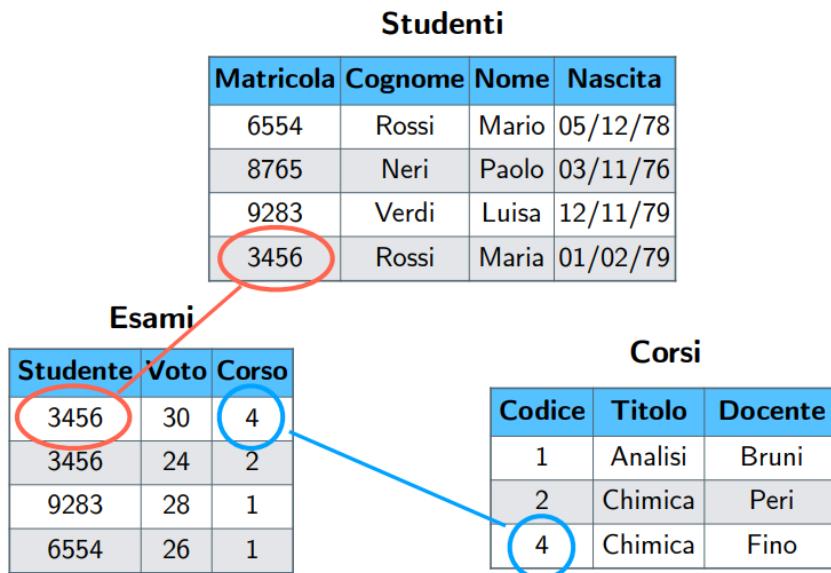


Figura 10: Relazioni tra i record degli attributi

Definizioni Schemi, Relazioni

È importante distinguere bene:

- **Schema di relazione:** dove un simbolo R detto nome della relazione ed un insieme di (*nomi di*) **attributi** $X = \{A_1, \dots, A_n\}$ solitamente indicato con $R(X)$. A ciascun attributo è associato un **dominio**.
- **Schema di base di dati:** che è un insieme di **schemi di relazione** con nomi diversi solitamente indicato come $R = \{R_1(X_1), \dots, R_m(X_m)\}$

Possiamo anche aggiungere che:

- una **n-upla** su un insieme di attributi X è una funzione che associa a ciascun attributo $A \in X$ un valore nel dominio A ;
- Il simbolo $t[A]$ denota il valore della *n-upla* t sull'attributo A .

Aggiungiamo anche le definizioni di **istanze**:

- (*Istanza di*) **relazione** su uno schema $R(X)$:
 - insieme r di *n-uple* su X
- (*Istanza di*) **base di dati**: su uno schema $R = \{R_1(X_1), \dots, R_m(X_m)\}$
 - insieme di relazioni $r = \{r_1, \dots, r_m\}$ dove ogni r_i è una relazione sullo schema $R_i(X_i)$.

Vediamo ora un esempio per rendere più chiaro

Studenti

Matricola	Cognome	Nome	Nascita
6554	Rossi	Mario	05/12/78
8765	Neri	Paolo	03/11/76
9283	Verdi	Luisa	12/11/79
3456	Rossi	Maria	01/02/79

Esami

Studente	Voto	Corso
3456	30	4
3456	24	2
9283	28	1
6554	26	1

Corsi

Codice	Titolo	Docente
1	Analisi	Bruni
2	Chimica	Peri
4	Chimica	Fino

- $X_1 = \{ \text{Matricola}, \text{Cognome}, \text{Nome}, \text{Nascita} \}$
- $X_2 = \{ \text{Studente}, \text{Voto}, \text{Corso} \}$
- $X_3 = \{ \text{Codice}, \text{Titolo}, \text{Docente} \}$

- $R_1 = \text{Studenti}$
- $R_2 = \text{Esami}$
- $R_3 = \text{Corsi}$

$R = \{R_1(X_1), R_2(X_2), R_3(X_3)\} = \{$
 $\text{Studenti}(\text{Matricola}, \text{Cognome}, \text{Nome}, \text{Nascita}),$
 $\text{Esami}(\text{Studente}, \text{Voto}, \text{Corso}),$
 $\text{Corsi}(\text{Codice}, \text{Docente}, \text{Titolo})\}$

Relazione su un Singolo Attributo

Studenti

Matricola	Cognome	Nome	Nascita
6554	Rossi	Mario	05/12/78
8765	Neri	Paolo	03/11/76
9283	Verdi	Luisa	12/11/79
3456	Rossi	Maria	01/02/79

Studenti Lavoratori

Matricola
6554
3456

Informazione Incompleta

Il **modello relazionale impone ai dati una struttura rigida**: infatti, le informazioni sono rappresentate da *n-uple* il cui formato deve corrispondere esattamente agli schemi di relazione. È anche vero che i dati disponibili possono non corrispondere al formato previsto.

Persone

Nome	Secondo Nome	Cognome
Franklin	Delano	Roosevelt
Winston		Churchill
Charles		De Gaulle
Josip		Stalin

Figura 11: Esempio di Informazioni incomplete

Ci sono un paio di *consigli* da seguire in questo caso:

1. **Non conviene usare valori particolari¹ del dominio**, perché potrebbero non esistere valori “non utilizzati”.
Questi “valori non utilizzati” potrebbero diventare significativi.
In fase di utilizzo nei programmi sarebbe necessario ogni volta tener conto del “significato” di questi valori.
2. **Quindi usiamo un valore distinto aggiunto a tutti i domini:**
 - a. Il **valore nullo** denota l'**assenza** di un valore del dominio.
 - b. Se i valori dell’attributo A appartengono al dominio D_a allora $t[A]$ è un valore del dominio oppure il valore nullo **NULL**.

Si possono e si debbono imporre restrizioni sulla presenza di valori nulli in una relazione. Abbiamo tre casi differenti di valore nullo:

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

NB: i DBMS non distinguono i tipi di valore nullo.

¹ valori particolari: 0, stringa nulla, “99”

Riportiamo a seguito un esempio di tabelle contenenti troppi valori NULL.

Studenti			
Matricola	Cognome	Nome	Nascita
6554	Rossi	Mario	05/12/78
8765	Neri	Paolo	03/11/76
9283	Verdi	Luisa	12/11/79
NULL	Rossi	Maria	01/02/79

Esami		
Studente	Voto	Corso
NULL	30	NULL
NULL	24	2
9283	28	1
6554	26	1

Corsi		
Codice	Titolo	Docente
1	Analisi	Bruni
2	NULL	NULL
4	Chimica	Fino

Figura 12: Esempio di troppi valori nulli

Basi di Dati Scorrette

Esistono istante di basi di dati che, pur scritte **sintatticamente corrette**, non rappresentano informazioni possibili per l'applicazione di interesse.

In questo caso stiamo parlando di basi di dati scorrette.

Esami			
Studente	Voto	Lode	Corso
276545	32		1
276545	30	e lode	2
787643	27	e lode	3
739430	24		4

Studenti		
Matricola	Cognome	Nome
276545	Rossi	Mario
787643	Neri	Paolo
787643	Verdi	Luisa

Figura 13: Esempio di Base di dati Scorretta

Vincoli di Integrità

I **vincoli di integrità** sono delle *proprietà* che si devono associare alla base di dati che esprimono la sua correttezza (se soddisfatte).

Che cosa permettono di fare?

- Permettono una descrizione più accurata della realtà
- Danno un contributo alla “qualità dei dati”
- Sono utili nella progettazione
- Sono usati dai DBMS nella esecuzione delle interrogazioni

I vincoli corrispondono quindi a proprietà nel mondo reale modellato dalla base di dati ed **interessano tutte le istanze**.

I vincoli sono associati allo schema e si considerano **corrette** le sue istanze che **soddisfano tutti i vincoli**.

Addentriamoci nei vincoli di integrità dicendo subito che è una funzione booleana (predicato) che associa ad ogni istanza delle base di dati, il valore vero o falso:

- Se il predicato associa **vero** allora la proprietà del vincolo è **soddisfatta**.
- Se il predicato associa **falso** allora la proprietà del vincolo **non è soddisfatta**.

Esistono due tipi di vincoli di integrità:

- **intra-relazionali**
- **inter-relazionali**

Vincoli Intrarelazionali

Il suo soddisfacimento è definito rispetto ad una singola relazione della base di dati.
Esistono delle tipologie di vincoli **intrarelazionali**:

- **Vincolo di n-upla**: può essere valutato su ciascuna *n-upla* indipendentemente dalle altre.
Esempio: (Voto = 30) OR NOT (Lode= "e lode")
- **Vincolo di dominio**: vincolo di *n-upla* che coinvolge un solo attributo
Esempio: (Voto>= 18) AND (Voto <=30)

Vincoli Interrelazionali

Il suo soddisfacimento è definito rispetto a **più relazioni** della base di dati.

Vediamo un esempio scritto:

"Un numero di matricola può comparire nella relazione Esami solo se compare nella relazione Studenti"

Identificazione delle n-uple

Matricola	Cognome	Nome	Corso	Nascita
27655	Rossi	Mario	Ing. Inf.	05/12/78
78763	Rossi	Mario	Ing. Inf.	03/11/76
65432	Neri	Piero	Ing. Mecc.	12/11/79
87654	Neri	Mario	Ing. Inf.	03/11/76
67653	Rossi	Piero	Ing. Mecc.	05/12/78

Non ci sono due *n-uple* con lo stesso valore sull'attributo Matricola.

Non ci sono due *n-uple* uguali su tutti e tre gli attributi Cognome, Nome e Nascita.

Figura 14: Distinzioni di alcuni valori di Attributi

Chiave e Superchiave

La **chiave** è un insieme di attributi che identificano univocamente le n-uple.

Diciamolo formalmente:

- Un insieme di K di attributi è una **superchiave** per una relazione r, se r non contiene due *tuple distinte* t_1 e t_2 con $t_1[K]=t_2[K]$
- K è una **chiave** per r se è una *superchiave minimale* di r (cioè non contiene l'altra superchiave)

Matricola	Cognome	Nome	Corso	Nascita
27655	Rossi	Mario	Ing. Inf.	05/12/78
78763	Rossi	Mario	Ing. Inf.	03/11/76
65432	Neri	Piero	Ing. Mecc.	12/11/79
87654	Neri	Mario	Ing. Inf.	03/11/76
67653	Rossi	Piero	Ing. Mecc.	05/12/78

Matricola è una **chiave**, perché:
è una superchiave e contiene un solo attributo, quindi è minimale

Cognome, Nome, Nascita è un'altra chiave perché soddisfa i requisiti:

- è una superchiave perché r non contiene due *tuple distinte*
- è minimale perché non contiene l'altra superchiave

Perché è **minimale**?

Matricola	Cognome	Nome	Corso	Nascita
27655	Rossi	Mario	Ing. Inf.	05/12/78
78763	Rossi	Mario	Ing. Inf.	03/11/76
65432	Neri	Piero	Ing. Mecc.	12/11/79
87654	Neri	Mario	Ing. Inf.	03/11/76
67653	Rossi	Piero	Ing. Mecc.	05/12/78

Cognome e Nome non distinguono n-uple 1 e 2

Cognome e Nascita non distinguono n-uple 1 e 5

Nome e Nascita non distinguono le n-uple 2 e 5

Esistenza delle chiavi

Una relazione contiene *n-uple* tutte diverse tra loro.

(Ricorda che è un insieme).

Ogni relazione ha come **superchiave** l'insieme degli attributi su cui è definita.

Quindi ha *almeno* una chiave.

Importanza delle chiavi

L'esistenza delle chiavi garantisce l'accessibilità a ciascun dato della base di dati.

Le chiavi permettono di correlare i dati in relazioni diverse.

Chiavi e Valori Nulli

In presenza di valori nulli, i valori della chiave non permettono:

- Di identificare le *n-uple*.
- Di realizzare facilmente i riferimenti da altre relazioni.

Matricola	Cognome	Nome	Corso	Nascita
NULL	NULL	Mario	Ing. Inf.	05/12/78
78763	Rossi	Mario	Ing. Inf.	03/11/76
65432	Neri	Piero	Ing. Mecc.	12/11/79
87654	Neri	Mario	Ing. Inf.	NULL
NULL	Rossi	Piero	NULL	05/12/78

- La presenza di valori nulli nella chiave deve essere limitata.

Dipendenze Funzionali

Le **dipendenze funzionali** sono vincoli di chiave sono particolari tipi di vincoli, che fanno parte di una categoria più vasta.

Formalmente:

Dati due insiemi di attributi X e Y, si dice che X, determina Y, e si scrive $X \rightarrow Y$, se e solo se date due *n-uple* distinte t_1 e t_2 , se $t_1[X] = t_2[X]$ allora $t_1[Y] = t_2[Y]$.

Le dipendenze funzionali possono essere usate per garantire opportune proprietà di una base di dati.

Integrità Referenziale

Le **integrità referenziali** sono informazioni in relazioni diverse che possono essere correlate attraverso valori comuni. In particolare, i valori delle chiavi.

Esempio: Infrazioni

Codice	Data	Vigile	Prov	Numero
34321	1/2/15	3987	MI	39548K
53524	4/3/15	3295	TO	E39548
64521	5/4/16	3295	PR	839548
73321	5/2/18	9345	PR	839548

Vigili

Matricola	Cognome	Nome
3987	Rossi	Luca
3295	Neri	Piero
9345	Neri	Mario
7543	Mori	Gino

Figura 15: Esempio integrità referenziale

Vincolo di Integrità Referenziale

Un **vincolo di integrità** referenziale è una regola fra gli attributi X di una relazione R₁ ed un'altra relazione R₂ che impone ai valori su X in R₁ di comparire come valori della chiave primaria di R₂.

NOTA BENE: L'ordine degli attributi tra cui è stabilito il vincolo è **significativo**.

Integrità Referenziale e Valori Nulli

In presenza di valori nulli i vincoli possono essere resi meno restrittivi.

Il vincolo non è fra ogni valore degli attributi X di una relazione R₁ e la chiave primaria della relazione R₂, ma tra i valori di X diversi da **NULL** e la chiave primaria di R₂.

Impiegati

Matricola	Cognome	Progetto
34321	Rossi	IDEA
53524	Neri	XYZ
64521	Verdi	NULL
73032	Bianchi	IDEA

Progetti

Codice	Inizio	Durata	Costo
IDEA	2000	36	200
XYZ	2001	24	120
BOH	2001	24	150

Reazione alla Violazione di Vincoli

Cosa succede quando si tenta di compiere un'operazione che viola un vincolo, ad esempio si cerca di inserire nella base di dati un valore non consentito per quell'attributo?

Sono possibili meccanismi per il supporto alla gestione delle violazioni che si chiamano **azioni compensative**.

Domanda: "Cosa succede se viene eliminata una n-upla, causando una violazione?"

Risposta: "Il comportamento standard sarebbe 'il rifiuto dell'operazione'."

Interagiscono in questo caso le **azioni compensative**, che sono :

- **Eliminazione in cascata**
- **Introduzione di valori nulli**

Impiegati

Matricola	Cognome	Progetto
34321	Rossi	IDEA
53524	Neri	NULL
64521	Verdi	NULL
73032	Bianchi	IDEA

Progetti

Codice	Inizio	Durata	Costo
IDEA	2000	36	200
BOH	2001	24	150

Impiegati

Matricola	Cognome	Progetto
34321	Rossi	IDEA
64521	Verdi	NULL
73032	Bianchi	IDEA

Progetti

Codice	Inizio	Durata	Costo
IDEA	2000	36	200
BOH	2001	24	150

Figura 16: Esempi di azioni compensative

Lezione 03: Algebra e Calcolo Relazionale

Linguaggi per le basi di dati

Avevamo detto che ci sono diversi linguaggi per le basi di dati a seconda di ciò che abbiamo intenzione di fare, come:

- operazioni sullo schema: **Data Definition Language (DDL)**;
- operazioni sui dati: **Data Manipulation Language (DML)**, dove possiamo:
 - interrogare la nostra base di dati (*query*);
 - aggiornare la nostra base di dati (*update*);

Linguaggi di interrogazione per le basi di dati

- *Cosa intendiamo per interrogazione?*

Intendiamo svolgere un'operazione di lettura sulla base di dati che può richiedere l'accesso a più di una tabella.

- *Cosa è necessario fare per specificare il significato di una interrogazione?*

Dobbiamo fare due formalismi a riguardo:

- **Modo dichiarativo:** si specificano le proprietà del risultato (“che cosa”).
- **Modo procedurale:** si specificano le modalità di generazione del risultato (“come”)

Definiamo quindi il **comportamento delle interrogazioni** in modo procedurale utilizzando le espressioni dell'**Algebra Relazionale**.

Definiamo invece **qual è il risultato di un'interrogazione** in modo dichiarativo utilizzando le espressioni del **Calcolo Relazionale**.

Impariamo bene che il **Calcolo Relazionale è l'effettiva semantica del linguaggio**, dove le espressioni sono espresse ad alto livello e non viene applicato nessun concetto di costo.

Invece con **Algebra Relazionale si definisce il modo in cui il DBMS esegue un'interrogazione.**

Algebra Relazionale

Ricordiamo che l'algebra in sé per sé è composta da dati+operatori.

Nell'Algebra Relazionale invece dobbiamo chiarire chi sono:

- **Dati:** sono le relazioni;
- **Operatori:** operazioni su relazioni, che producono relazioni e che possono essere composti.

Operatori dell'Algebra Relazionale

Dobbiamo fare una distinzione importante tra:

- Operatori su insiemi:
 - unione
 - intersezione
 - differenza
- Operatori su relazioni:
 - ridenominazione;
 - selezione;
 - proiezione;
 - **join:**
 - naturale;
 - prodotto cartesiano;
 - theta.

Operatori su Insiemi

Le **relazioni** sono **insiemi**.

I **risultati** devono essere **relazioni**.

Si possono applicare gli **operatori su insiemi** solo a **relazioni** definite sugli stessi **attributi**, in modo che il **risultato** sia una **relazione** sugli stessi **attributi**.

Unione

L'unione di due relazioni sullo stesso insieme di attributi X è una relazione su X che contiene le n -uple sia dell'una che dell'altra relazione originarie.

Laureati			Specialisti		
Matricola	Nome	Età	Matricola	Nome	Età
7274	Rossi	32	9297	Neri	33
7432	Neri	24	7432	Neri	24
9824	Verdi	25	9824	Verdi	25

Laureati \cup Specialisti

Matricola	Nome	Età
7274	Rossi	32
7432	Neri	24
9824	Verdi	25
9297	Neri	33

Figura 17: Esempio di operatore di Unione

Intersezione

L'intersezione di due relazioni sullo stesso insieme di attributi X è una relazione su X che contiene le n -uple appartenenti ad entrambe le relazioni originarie.

Laureati			Specialisti		
Matricola	Nome	Età	Matricola	Nome	Età
7274	Rossi	32	9297	Neri	33
7432	Neri	24	7432	Neri	24
9824	Verdi	25	9824	Verdi	25

Laureati \cap Specialisti		
Matricola	Nome	Età
7432	Neri	24
9824	Verdi	25

Figura 18: Esempio di operatore di Intersezione

Differenza

La differenza tra due relazioni sullo stesso insieme di attributi X è una relazione su X che contiene le n -uple appartenenti alla prima relazione che non appartengono anche alla seconda-

Laureati			Specialisti		
Matricola	Nome	Età	Matricola	Nome	Età
7274	Rossi	32	9297	Neri	33
7432	Neri	24	7432	Neri	24
9824	Verdi	25	9824	Verdi	25

Laureati – Specialisti		
Matricola	Nome	Età
7274	Rossi	32

Figura 19: Esempio di operatore di Differenza

Unione Impossibile

Sebbene abbia senso, come posso effettuare l'unione delle due relazioni seguenti?

Paternità		Maternità	
Padre	Figlio	Madre	Figlio
Adamò	Abele	Eva	Abele
Adamò	Caino	Eva	Set
Abramo	Isacco	Sara	Isacco

Paternità ∪ Maternità
???

Figura 20: Esempio di unione impossibile

Risposta: Usa la ridenominazione!!

Ridenominazione

È un **operatore monadico**² e modifica lo schema dell'operando, lasciandone inalterata l'istanza.

Data una relazione R, in generale, questo operatore si scrive come:

$$\rho_{B_1 B_2 \dots \leftarrow A_1 A_2 \dots} (R)$$

Da leggersi:

- L'attributo A_1 viene sostituito dall'attributo B_1
- L'attributo A_2 viene sostituito dall'attributo B_2

² **Operatore monadico:** contente 1 solo operando

Riportiamo un esempio più pratico dell'operatore di ridenominazione

Paternità		Maternità	
Padre	Figlio	Madre	Figlio
Adamo	Abele	Eva	Abele
Adamo	Caino	Eva	Set
Abramo	Isacco	Sara	Isacco

$\rho_{\text{Genitore} \leftarrow \text{Padre}}(\text{Paternità})$

$\rho_{\text{Genitore} \leftarrow \text{Madre}}(\text{Maternità})$

Paternità		Maternità	
Genitore	Figlio	Genitore	Figlio
Adamo	Abele	Eva	Abele
Adamo	Caino	Eva	Set
Abramo	Isacco	Sara	Isacco

Infine, avremo...

$\rho_{\text{Genitore} \leftarrow \text{Padre}}(\text{Paternità}) \cup \rho_{\text{Genitore} \leftarrow \text{Madre}}(\text{Maternità})$

Genitore	Figlio
Adamo	Abele
Adamo	Caino
Abramo	Isacco
Eva	Abele
Eva	Set
Sara	Isacco

Selezione

È un **operatore monadico** e produce un risultato che:

- Ha lo stesso schema dell'operando
- Contiene un sottoinsieme delle n-uple dell'operando
- Solo quelle n-uple che soddisfano una condizione fissata

Usiamo qualche formalismo per indicare l'operatore di selezione.

Data una relazione R(X), in generale, questo operatore si scrive come:

$$\sigma_F(R)$$

Dove:

F è una **espressione Booleana** ottenuta componendo con gli operatori logici **AND**, **OR** e **NOT** delle condizioni atomiche.

Una **condizione atomica** ha la forma:

- $A * B$, dove A e B sono attributi di X con **domini compatibili** e * è un **operatore di confronto**.
- $A * k$, dove A è un attributo di X, k è una **costante** con dominio compatibile con A e * è un **operatore di confronto**.

Proviamo a fare una selezione:

voglio gli Impiegati che guadagnano più di 50000 euro

Impiegati

Matricola	Cognome	Filiale	Stipendio
7309	Rossi	Roma	55
5998	Neri	Milano	64
9553	Milano	Milano	32
5698	Neri	Napoli	40

$\sigma_{\text{Stipendio} > 50}(\text{Impiegati})$

Matricola	Cognome	Filiale	Stipendio
7309	Rossi	Roma	55
5998	Neri	Milano	64

Adesso però:

voglio gli impiegati che guadagnano più di 50000 euro e lavorano a Milano.

$\sigma_{\text{Stipendio} > 50 \text{ AND Filiale} = \text{'Milano'}}(\text{Impiegati})$

Matricola	Cognome	Filiale	Stipendio
5998	Neri	Milano	64

*Figura 21: Esempi di selezione dalla tabella Impiegato. Ricorda di moltiplicare lo stipendio *1000.*

Selezione e Valori Nulli

La condizione atomica è vera solo per valori non nulli in qualsiasi attributo.

Riportiamo un esempio:

Impiegati

Matricola	Cognome	Filiale	Stipendio
7309	Rossi	Roma	55
5998	Neri	Milano	NULL
9553	Milano	Milano	32
5698	Neri	Napoli	40

$\sigma_{\text{Filiale}='\text{Milano}'}(\text{Impiegati})$

Matricola	Cognome	Filiale	Stipendio
9553	Milano	Milano	32

Per riferirsi a valori nulli esistono condizioni apposite: **IS NULL** e **IS NOT NULL**.

Impiegati

Matricola	Cognome	Filiale	Stipendio
7309	Rossi	Roma	55
5998	Neri	Milano	NULL
9553	Milano	Milano	32
5698	Neri	Napoli	40

$\sigma_{(\text{Filiale}='\text{Milano}') \text{ OR } (\text{Stipendio IS NULL})}(\text{Impiegati})$

Matricola	Cognome	Filiale	Stipendio
5998	Neri	Milano	NULL
9553	Milano	Milano	32

Proiezione

La proiezione è un operatore monadico e produce un risultato che:

- Ha un sottoinsieme degli attributi dell'operando
- Contiene tutte le n-uple cui contribuiscono tutti i valori esistenti dell'operando.

Possiamo dire formalmente:

Data una relazione $R(X)$ ed un insieme di attributi $Y \subseteq X$, in generale, questo operatore si scrive come:

$$\pi_Y(R)$$

Il risultato è una relazione su Y che contiene l'insieme delle *n-uple* di R ristrette ai soli attributi di Y. C'è da ricordarsi che il risultato è un insieme che **non** può contenere n-uple uguali. Vediamo degli esempi:

Calcolare la matricola e cognome di tutti gli impiegati

Impiegati				$\pi_{\text{Matricola,Cognome}}(\text{Impiegati})$	
Matricola	Cognome	Filiale	Stipendio	Matricola	Cognome
7309	Neri	Napoli	55	7309	Neri
5998	Neri	Milano	64	5998	Neri
9553	Rossi	Roma	32	9553	Rossi
5998	Rossi	Roma	40	5998	Rossi

Calcolare Nome e Filiale di tutti gli Impiegati

Impiegati				$\pi_{\text{Cognome,Filiale}}(\text{Impiegati})$	
Matricola	Cognome	Filiale	Stipendio	Cognome	Filiale
7309	Neri	Napoli	55	Neri	Napoli
5998	Neri	Milano	64	Neri	Milano
9553	Rossi	Roma	32	Rossi	Roma
5998	Rossi	Roma	40	Rossi	Roma

Figura 22: Esempio di utilizzo della proiezione sbagliato

L'esempio riportato è errato, vediamo ora la versione corretta:

Impiegati				$\pi_{\text{Cognome}, \text{Filiale}}(\text{Impiegati})$	
Matricola	Cognome	Filiale	Stipendio	Cognome	Filiale
7309	Neri	Napoli	55	Neri	Napoli
5998	Neri	Milano	64	Neri	Milano
9553	Rossi	Roma	32	Rossi	Roma
5998	Rossi	Roma	40		

Figura 23: Esempio di utilizzo della proiezione corretto

Cardinalità delle Proiezioni

Una proiezione può contenere al più tante n-uple quante ne ha l'operando.

Può contenerne di meno.

Se X è una superchiave di R allora $\pi_X(R)$ contiene esattamente tante tuple quante ne ha R.

Proiezione e Selezione

Per chiarire meglio la differenza possiamo dire:

- Selezione σ decomposizione orizzontale
- Proiezione π decomposizione verticale

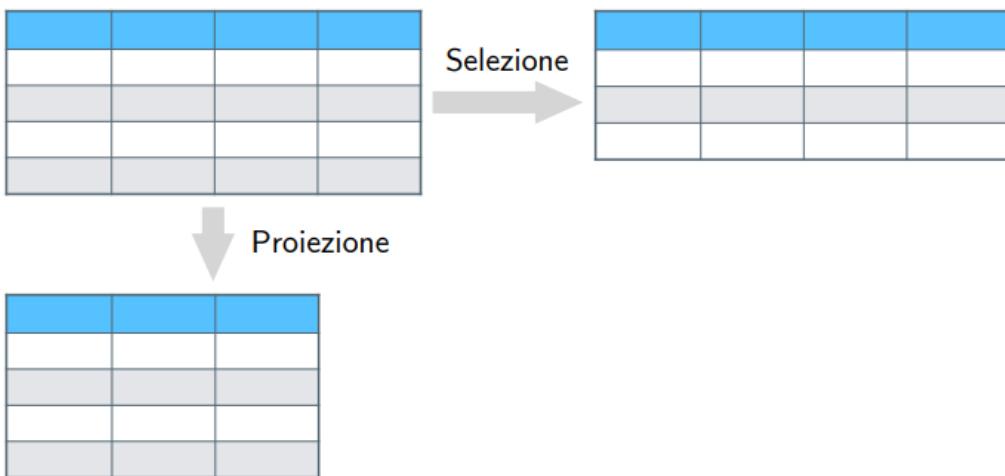


Figura 24: Differenza tra selezione e proiezione

Vediamo un esempio per mettere in chiaro i concetti:

Calcolare Matricola e Cognome degli Impiegati che guadagnano più di 50000 euro

Impiegati

Matricola	Cognome	Filiale	Stipendio
7309	Rossi	Roma	55
5998	Neri	Milano	64
9553	Milano	Milano	32
5698	Neri	Napoli	40

$\pi_{\text{Matricola}, \text{Cognome}}(\sigma_{\text{Stipendio} > 50}(\text{Impiegati}))$

Matricola	Cognome
7309	Rossi
5998	Neri

Figura 25: Ricorda che lo Stipendio va moltiplicato *1000

Se combiniamo selezione e proiezione possiamo estrarre informazioni da una sola soluzione. Fai attenzione a queste due regole importanti:

- Non possiamo combinare informazioni presenti in relazioni diverse.
- Non possiamo combinare informazioni presenti in *n-uple* diverse della stessa relazione.

Join Naturale

È un operatore con due operandi (generalizzabile) e produce un risultato:

- sull'unione degli attributi degli operandi;
- contiene le n-uple costruite ciascuna a partire da una n-upla di ognuno degli operandi.

Date due relazioni $R_1(X_1)$ e $R_2(X_2)$, in generale questo operatore si scrive come

$$R_1 \bowtie R_2$$

Il risultato è una relazione R definita su $X_1 X_2$ come:

$$\begin{aligned} R(X_1 \cup X_2) = R_1(X_1) \bowtie R_2(X_2) = \\ \{t \mid \text{esistono } t_1 \in R_1 \text{ e } t_2 \in R_2 \text{ con } t[X_1] = t_1 \text{ e } t[X_2] = t_2\} \end{aligned}$$

Vediamo degli esempi di utilizzo del Join

Impiegato	Reparto		Reparto	Capo		Impiegato	Reparto	Capo
Rossi	A		A	Mori		Rossi	A	Mori
Neri	B		B	Bruni		Neri	B	Bruni
Bianchi	B					Bianchi	B	Bruni

Figura 26: Esempio di join completo

Impiegato	Reparto		Reparto	Capo		Impiegato	Reparto	Capo
Rossi	A		B	Mori		Neri	B	Mori
Neri	B		C	Bruni		Bianchi	B	Mori
Bianchi	B							

Impiegato	Reparto		Reparto	Capo		Impiegato	Reparto	Capo
Rossi	A		D	Mori				
Neri	B		C	Bruni				
Bianchi	B							

Impiegato	Reparto		Reparto	Capo		Impiegato	Reparto	Capo
Rossi	B		B	Mori		Rossi	B	Mori
Neri	B		B	Bruni		Rossi	B	Bruni
						Neri	B	Mori

Cardinalità del Join

Il join di R_1 e R_2 contiene un numero di n-uple compreso tra 0 ed il prodotto di $|R_1|$ e $|R_2|$.

- Se il join coinvolge una chiave di R_2 allora il numero di *n-uple* è compreso tra 0 ed $|R_1|$
- Se il join coinvolge una chiave di R_2 ed un vincolo di integrità referenziale allora il numero delle *n-uple* è uguale a $|R_1|$

Quindi il Join di $R_1(A,B)$ e $R_2(B,C)$ contiene un numero di n-uple:

$$0 \leq |R_1 \bowtie R_2| \leq |R_1| \times |R_2|$$

- Se B è una chiave di R_2 allora il numero di *n-uple* è $0 \leq |R_1 \bowtie R_2| \leq |R_1|$;
- Se B è una chiave di R_2 ed esiste un vincolo di integrità referenziale fra B (in R_1) ed R_2 allora il numero delle *n-uple* è $|R_1 \bowtie R_2| = |R_1|$

Una difficoltà del Join è invece:

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B



Reparto	Capo
B	Mori
C	Bruni

Impiegato	Reparto	Capo
Neri	B	Mori
Bianchi	B	Mori

Figura 27: Esempio dove alcune tuple vengono tagliate fuori

Join Esterno

Il join Esterno estende, con valori nulli, le n-uple che verrebbero tagliate fuori da un join (interno). Ne esistono 3 versioni di join esterno:

- **Sinistro**: mantiene tutte le n-uple del primo operando, estendendole con valori nulli se necessario (*left join*)
- **Destro**: mantiene tutte le n-uple del secondo operando, estendendole con valori nulli se necessario (*right join*)
- **Completo**: mantiene tutte le n-uple di entrambi gli operandi, estendendole con valori nulli se necessario. (*full join*)

Join Esterno Sinistro

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B



Reparto	Capo
B	Mori
C	Bruni

Join Esterno Destro

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B



Reparto	Capo
B	Mori
C	Bruni

Impiegato	Reparto	Capo
Neri	B	Mori
Bianchi	B	Mori
Rossi	A	NULL

Impiegato	Reparto	Capo
Neri	B	Mori
Bianchi	B	Mori
NULL	C	Bruni

Join Esterno Completo

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B



Reparto	Capo
B	Mori
C	Bruni

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

Reparto	Capo
B	Mori
C	Bruni

Impiegato	Reparto	Capo
Neri	B	Mori
Bianchi	B	Mori
Rossi	A	NULL
NULL	C	Bruni

Impiegato	Reparto
Neri	B
Bianchi	B

Reparto	Capo
---------	------

Join e Proiezioni

Date due relazioni $R_1(X_1)$ e $R_2(X_2)$

$$\pi_{X_1} (R_1 \bowtie R_2) \subseteq R_1$$

Date una relazione $R(X)$ con $X = X_1 \cup X_2$

$$(\pi_{X_1}(R) \bowtie \pi_{X_2}(R)) \supseteq R$$

Impiegato	Reparto	Capo
Neri	B	Mori
Bianchi	B	Bruni
Verdi	A	Bini

Impiegato	Reparto	Reparto	Capo
Neri	B	B	Mori
Bianchi	B	B	Bruni
Verdi	A	A	Bini

Impiegato	Reparto	Capo
Neri	B	Mori
Neri	B	Bruni
Bianchi	B	Mori
Bianchi	B	Bruni
Verdi	A	Bini

Figura 28: Esempio di Join e Proiezioni

Prodotto Cartesiano

Date due relazioni $R_1(X_1)$ e $R_2(X_2)$ senza attributi a comune, cioè $X_1 \cap X_2 = \emptyset$, la definizione di Join naturale funziona ugualmente.

La relazione risultante contiene sempre un numero di *n-uple* pari al **prodotto delle cardinalità degli operandi**. Tutte le *n-uple* sono combinabili tra loro.

Per questo diciamo che la relazione risultante corrisponde al **prodotto cartesiano delle relazioni**:

$$R = R_1 \bowtie R_2 = R_1 \times R_2$$

Impiegato	Reparto	Codice	Capo
Rossi	A	A	Mori
Neri	B	B	Bruni
Bianchi	B	B	Bruni

Impiegato	Reparto	Codice	Capo
Rossi	A	A	Mori
Rossi	A	B	Bruni
Neri	B	A	Mori
Neri	B	B	Bruni
Bianchi	B	A	Mori
Bianchi	B	B	Bruni

Figura 29: Esempio di Prodotto Cartesiano

Theta Join

Nella pratica, il prodotto cartesiano ha senso (quasi) solo se seguito da una selezione:
 $\sigma_F(R_1 \times R_2)$

Questa composizione di operatori è un operatore derivato chiamato **theta-join** ed indicato come: $R_1 \bowtie_F R_2$

La condizione F è spesso una **congiunzione (AND) di atomi di confronto** $A_1 \vartheta A_2$ dove

- ϑ è un **operatore di confronto** ($>$, $<$, $=$, ...)
- A_1 e A_2 sono **attributi** di relazioni diverse

Se l'operatore di confronto è l'uguaglianza (=) allora si parla di **equi-join**.

Impiegati		Reparti	
Impiegato	Reparto	Codice	Capo
Rossi	A	A	Mori
Neri	B	B	Bruni
Bianchi	B	B	Bruni

Impiegati $\bowtie_{\text{Reparto}=\text{Codice}}$ **Reparti**

Impiegato	Reparto	Codice	Capo
Rossi	A	A	Mori
Neri	B	B	Bruni
Bianchi	B	B	Bruni

Figura 30: Esempio di Theta Join

Equivalenza di Espressioni

Due **espressioni** sono **equivalenti** se producono lo stesso risultato qualunque sia l'istanza attuale della base di dati.

L'equivalenza è importante nella pratica perché i DBMS cercano di eseguire **espressioni equivalenti** a quelle date, ma **meno costose**. Elenchiamo ora le due equivalenze importanti:

Push selections down

$$\sigma_{A=k}(R_1 \bowtie R_2) \equiv R_1 \bowtie \sigma_{A=k}(R_2)$$

- A è un attributo di R_2
- k è una costante sul dominio di A .

Push projections down

$$\pi_{X_1Y_2}(R_1 \bowtie R_2) \equiv R_1 \bowtie \pi_{Y_2}(R_2)$$

- X_1 sono gli attributi di R_1
- X_2 sono gli attributi di R_2
- gli attributi X_2-Y_2 però non sono coinvolti nel join.

Utilizziamo quindi queste due espressioni di **Push selections down** e **Push projection down** perché riducono in modo significativo la dimensione del risultato intermedio e quindi il costo dell'operazione.

Ottimizzazione delle Interrogazioni

Query processor (od ottimizzatore): un modulo del DBMS.

Più importante nei sistemi attuali che in quelli “vecchi” (gerarchici e reticolari).

Responsabile di prendere una query SQL e generare un piano di esecuzione ottimizzato, cercando di minimizzare il tempo di esecuzione e l'utilizzo delle risorse. Le interrogazioni sono espresse ad alto livello (ricordare il concetto di indipendenza dei dati):

- insiemi di *n-uple*
- Poca proceduralità

L'ottimizzatore scegli la strategia realizzativa (di solito fra diverse alternative), a partire dall'istruzione SQL.

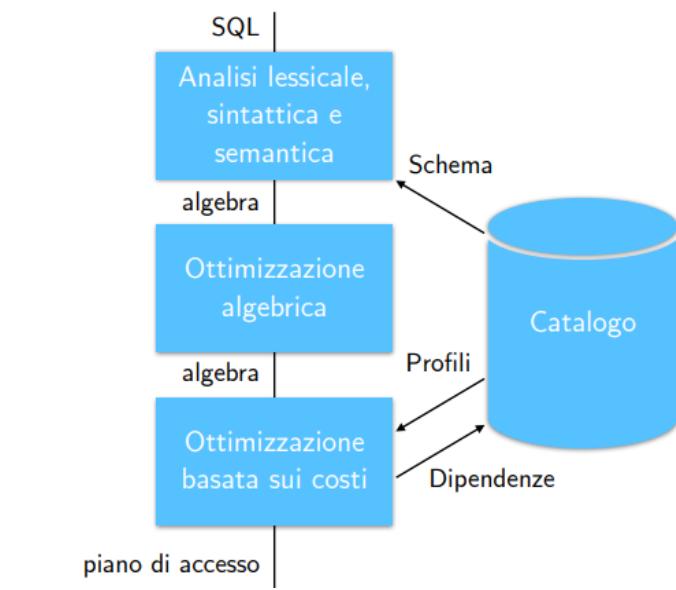


Figura 31: Schema dell'esecuzione delle Interrogazioni

Profili delle Relazioni

Informazioni quantitative:

- Cardinalità di ciascuna relazione
- Dimensioni delle *n-uple*
- Dimensioni dei valori
- Numero di valori distinti degli attributi
- Valore minimo e massimo di ciascun attributo

Sono memorizzate nel “catalogo” e aggiornate con comandi del tipo *update statistics* e sono utilizzate nella **fase finale** dell’ottimizzazione, per **stimare** le dimensioni dei **risultati intermedi**.

Ottimizzazione Algebrica

Il termine ottimizzazione è improprio (anche se efficace) perché il processo utilizza euristiche e si basa sulla nozione di equivalenza:

Due **espressioni** si dicono **equivalenti** se producono lo stesso risultato qualunque sia l’istanza attuale della base di dati.

I DBMS cercano di eseguire espressioni equivalenti a quelle date, ma “**meno costose**”.

Euristica fondamentale: selezioni e proiezioni il più presto possibile (per **ridurre le dimensioni dei risultati intermedi**):

- **push selections down**
- **push projections down**

Grafo

Un **grafo** $G=(V,E)$ consiste in:

- un insieme V di vertici (o nodi)
- un insieme E di coppie di vertici, detti archi ed ogni arco connette due vertici.

Dobbiamo fare una distinzione importante tra:

- **Grafo orientato (o diretto)**: ogni arco è orientato e rappresenta relazioni orientate tra coppie di oggetti.
- **Grafo non orientato (o non diretto)**: gli archi non hanno un’orientazione e rappresentano relazioni simmetriche tra coppie di oggetti.

Cammino e Ciclo

Un **cammino** in un grafo $G=(V,E)$ da un vertice x ad un vertice y è dato da una sequenza di vertici (v_0, v_1, \dots, v_k) di V con $v_0=x$ e $v_k=y$ tale che per ogni $1 \leq i \leq k$, l'arco $(v_{i-1}, v_i) \in E$.

Un **cammino** (v_0, v_1, \dots, v_k) tale che $v_0=v_k$ è detto **ciclo**.

Un grafo diretto è detto **aciclico** se **non** contiene cicli.

Albero

Un grafo non orientato si dice **connesso** se esiste un cammino tra ogni coppia di vertici.

Un **albero** è un **grafo non orientato** nel quale due vertici qualsiasi sono connessi da uno ed un solo cammino.

Rappresentazione Interna delle Interrogazioni

Possiamo dire che gli **Alberi** sono composti da:

- **Foglie:** dati (relazioni, file)
- **Nodi intermedi:** operatori (operatori algebrici, poi effettivi operatori di accesso ai dati)

Vediamo un paio di esempi:

$$\sigma_{A=10}(R_1 \bowtie R_2)$$

$$R_1 \bowtie \sigma_{A=10}(R_2)$$



Procedura Euristica di Ottimizzazione

- Decomporre le selezioni congiuntive in successive selezioni atomiche
- Anticipare il più possibile le selezioni
- In una sequenza di selezioni, anticipare le più selettive
- Combinare prodotti cartesiani e selezioni per formare join
- Anticipare il più possibile le proiezioni (anche introducendone di nuove)

Vediamo ora un esempio di procedura euristica di ottimizzazione.

Siano:

$R_1(ABC)$, $R_2(DEF)$, $R_3(GHI)$

Interrogazione:

```
SELECT A, E  
FROM R1, R2, R3  
WHERE  
B > 100 AND H = 7 AND I > 2 AND C = D AND F = G
```

dove:

- FROM: prodotto cartesiano
- WHERE: selezione
- SELECT: proiezione

$$\pi_{AE} (\sigma_{B>100 \text{ AND } H=7 \text{ AND } I>2 \text{ AND } C=D \text{ AND } F=G} (r_1 \bowtie r_2 \bowtie r_3))$$

Procediamo per step e vediamone la sua evoluzione:

L'espressione

$$\pi_{AE} (\sigma_{B>100 \text{ AND } H=7 \text{ AND } I>2 \text{ AND } C=D \text{ AND } F=G} (r_1 \bowtie r_2 \bowtie r_3))$$

diventa (passi 1, 2, 3 e 4)

$$\pi_{AE} (\sigma_{B>100}(r_1) \bowtie_{C=D} r_2) \bowtie_{F=G} \sigma_{I>2} (\sigma_{H=7}(r_3))$$

diventa (passo 5)

$$\pi_{AE} (\pi_{AEF} ((\pi_{AC} (\sigma_{B>100}(r_1))) \bowtie_{C=D} r_2) \bowtie_{F=G} \pi_G (\sigma_{I>2} (\pi_{GI} (\sigma_{H=7}(r_3)))))$$

Relazioni Derivate

Relazioni di base: contenuto autonomo

Relazioni derivate: contenuto funzione del contenuto di altre relazioni

- Rappresentazioni diverse per gli stessi dati
- Definire per mezzo di interrogazioni
- Possono essere definite su altre relazioni derivate ma ne esistono due tipi:

Esistono due tipi di **relazioni derivate**:

- **Viste materializzate.**
- **Viste virtuali**, o semplicemente **viste**.

Vediamo un esempio di vista:

Afferenza		Direzione	
Impiegato	Reparto	Reparto	Capo
Rossi	A	A	Mori
Neri	B	B	Bruni
Bianchi	B	C	Leoni
Verdi	C		

- Una vista:

$$\text{Supervisione} = \pi_{\text{Impiegato}, \text{Capo}} (\text{Afferenza} \bowtie \text{Direzione})$$

Figura 32: Esempio di vista

Viste Materializzate

Le viste materializzate sono relazioni derivate memorizzate nella base di dati.

Vantaggi:

- Immediatamente disponibili per le interrogazioni

Svantaggi:

- Ridondanti
- Appesantiscono gli aggiornamenti
- Sono raramente supportate dai DBMS

Viste Virtuali

Le viste virtuali sono relazioni derivate non memorizzate nella base di dati.

- Sono supportate da tutti i DBMS
- Una interrogazione su una vista è eseguita “ricalcolando” la vista (o quasi)

Interrogazioni su viste

Le interrogazioni su viste sono eseguite sostituendo alla vista la sua definizione.

L'interrogazione $\sigma_{\text{Capo}='Leoni'} (\text{Supervisione})$

È eseguita come

$$\begin{aligned}\sigma_{\text{Capo}='\text{Leoni}'}(\text{Supervisione}) &= \\ &= \sigma_{\text{Capo}='\text{Leoni}'}(\\ &\quad \pi_{\text{Impiegato}, \text{Capo}}(\text{Afferenza} \bowtie \text{Direzione}) \\ &)\end{aligned}$$

Perché usiamo le viste?

Le viste sono uno **strumento di programmazione**:

- Si può semplificare la scrittura di interrogazioni con espressioni complesse e sotto-espressioni ripetute.
- L'uso delle viste virtuali **non influisce sull'efficienza** delle interrogazioni.

Vediamo subito un esempio:

Supponiamo di avere le seguenti relazioni:

$$R_1(ABC), R_2(DEF), R_3(GH)$$

e di definire la seguente vista R :

$$R = \sigma_{A>D}(R_1 \bowtie R_2)$$

Un'interrogazione può essere definita:

- Senza vista:

$$\sigma_{B=G} \left(\sigma_{A>D} (R_1 \bowtie R_2) \bowtie R_3 \right)$$

- Con vista:

$$\sigma_{B=G} (R \bowtie R_3)$$

Viste ed aggiornamenti

Aggiornare una vista significa **modificare le relazioni** di base in modo che la vista, "ricalcolata", rispecchi l'aggiornamento.

L'aggiornamento sulle relazioni di base corrispondente a quello specificato sulla vista deve essere univoco, in generale però non lo è.

Nota Bene: pochi aggiornamenti sono ammissibili sulle viste.

Convenzione

Ignoriamo il join naturale, ovvero vale a dire che non consideriamo implicitamente condizioni su attributi con nomi uguali. Per riconoscere attributi con lo stesso nome gli premettiamo il nome delle relazione seguita da “.”

Usiamo “assegnazioni”, cioè viste, per ridenominare le relazioni e gli attributi solo quando serve l'unione. Vediamone subito un esempio:

Trovare gli impiegati che guadagnano più del proprio capo, mostrando matricola, nome e stipendio dell'impiegato e del capo.

$$\begin{aligned} & \pi_{\text{Matr}, \text{Nome}, \text{Stip}, \text{MatrC}, \text{NomeC}, \text{StipC}} (\\ & \sigma_{\text{Stip} > \text{StipC}} (\\ & \rho_{\text{MatrC}, \text{NomeC}, \text{StipC}, \text{EtàC} \leftarrow \text{Matr}, \text{Nome}, \text{Stip}, \text{Età}}^{\text{(Imp)}} \\ & \bowtie \\ & (\text{Sup} \bowtie_{\text{Imp} = \text{Matr}} \text{Imp}))) \\ & \text{Capi := Imp} \end{aligned}$$

$$\begin{aligned} & \pi_{\text{Imp.Matr}, \text{Imp.Nome}, \text{Imp.Stip}, \text{Capi.Matr}, \text{Capi.Nome}, \text{Capi.Stip}} (\\ & \sigma_{\text{Imp.Stip} > \text{Capi.Stip}} (\\ & \text{Capi} \\ & \bowtie_{\text{Capi.Matr} = \text{Capo}} \\ & (\text{Sup} \bowtie_{\text{Imp} = \text{Imp.Matr}} \text{Imp}))) \end{aligned}$$

Calcolo Relazionale

Il calcolo relazionale fa parte della famiglia di *linguaggi dichiarativi* basati sul calcolo dei predicati del primo ordine. Ne esistono diverse versioni:

- **Calcolo relazione sui domini**, cioè il calcolo sui domini
- **Calcolo su *n-uple*** con dichiarazione di *range*, ovvero calcolo sulle *n-uple*.

Calcolo sui domini

Sintassi: le espressioni hanno la forma: $\{A_1 : x_1, \dots, A_k : x_k \mid f\}$ dove:

f è una **formula** (con connettivi Booleani e quantificatori)

- A_i è un nome di attributo
- x_i è un nome di variabile
- $A_1: x_1, \dots, A_n : x_n$ è chiamata **target list** e descrive il risultato.

Semantica: il risultato è una relazione su A_1, \dots, A_k che contiene *n-uple* di valori per x_1, \dots, x_k che rendono vera la formula f rispetto ad un'istanza di base di dati a cui l'espressione è applicata.

Formula

Sia f una **formula** secondo le seguenti regole. Esistono formule atomiche:

- $R(A_1: x_1, \dots, A_p : x_p)$ dove $R(A_1, \dots, A_p)$ è uno schema di relazione e x_1, \dots, x_p sono **variabili**.
- $x\theta y$ o $x\theta c$, dove x e y sono variabili, c è una costante, e θ è un **operatore di confronto**.

Se f_1 e f_2 sono formule, allora lo sono anche $f_1 \wedge f_2$, $f_1 \vee f_2$ e $\neg f_1$ e si possono usare le **parentesi**.

Se f è una formula e x una variabile, allora anche $\exists x(f)$ e $\forall x(f)$ dove \exists e \forall sono **quantificatori**.

Basi di dati per gli esempi

Impiegato (Matr, Nome, Età, Stipendio)

Supervisione (Capo, Impiegato)

Trovare matricola, nome, età e stipendio degli impiegati che guadagnano più di 40

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

{Matr: m , Nome: n , Età: e, Stipendio: s |

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

Trovare matricola e nome degli impiegati che guadagnano più di 40

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

{Matr: m , Nome: n |

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

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

{Matr: m , Nome: n |

Impiegati(Matr: c , Nome: n , Età: e , Stipendio: s) \wedge s $\forall m' \forall n' \forall e' \forall s'$:

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

Calcolo sui domini: discussione

- **Pregi:**
 - Dichiaratività
- **Difetti:**
 - Verbosità (tante variabili)
 - Possibilità di scrivere espressioni senza senso (dipendenti dal dominio)
 - $\{A : x, B : y \mid R(A : x) \wedge y = y\}$
 - Nel risultato compaiono tuple per qualsiasi valore del dominio di B
 - $\{A : x \mid R(A : x)\}$
 - Nel risultato compaiono tuple per qualsiasi valore del dominio di A che compaiono in R

Nell'algebra tutte le espressioni hanno un senso (indipendenti dal dominio).

Calcolo sulle *n-uple*: discussione

Nel calcolo sulle *n-uple* le variabili rappresentano tuple quindi si ha minore verbosità. Le espressioni hanno la forma:

$\{T \mid L \mid f\}$

- T è la target list, con elementi del tipo
- $Y : x.Z$
- $x.Z \equiv Z : x.Z$
- $x.* \equiv X : x.X$ (asterisco quando prendo tutti gli attributi)
- x è una variabile
- Y e Z sono liste di attributi

Gli attributi di Z devono comparire nello schema della relazione che costituisce il campo di variabilità, o range, di x . L è la range list, che elenca le variabili libere della formula f con i relativi campi di variabilità, o range. $-f$ è una formula.

Alcune interrogazioni importanti non si possono esprimere, in particolare le unioni: $R_1(AB) \cup R_2(AB)$. Perché?

Ogni variabile nel risultato ha un solo *range*, mentre vorremmo *n-uple* sia dalla prima relazione che dalla seconda. Però possiamo dire che **intersezione** e **differenza** sono esprimibili.

Per questa ragione SQL (che è basato su questo calcolo) prevede un **operatore esplicito di unione**, ma non tutte le versioni prevedono intersezione e differenza.

Calcolo ed Algebra: limiti

Calcolo ed algebra sono sostanzialmente **equivalenti**, infatti:

- per ogni espressione del calcolo relazionale che sia indipendente dal dominio esiste un' espressione nell'algebra relazionale equivalente ad essa.
- per ogni espressione dell'algebra relazionale esiste un'espressione del calcolo relazionale equivalente ad essa (e quindi indipendente dal dominio).

Ci sono però **interrogazioni** interessanti non **esprimibili**:

- calcolo di **valori derivati**: possiamo solo **estrarre valori**, non calcolarne nuovi:
 - a livello di *n-upla* o di singolo valore (conversioni somme, differenze, etc.)
 - su insiemi di *n-uple* (somme, medie, etc.)
- interrogazioni inerentemente ricorsive, come la **chiusura transitiva**.

Vediamo un esempio:

Per ogni impiegato trovare tutti i superiori.

(quindi il capo, il capo del capo e così via...)

Impiegato	Capo
Rossi	Lupi
Neri	Bruni
Lupi	Falchi

Impiegato	Superiore
Rossi	Lupi
Neri	Bruni
Lupi	Falchi
Rossi	Falchi

Figura 33: Esempio di chiusura transitiva

Nell'esempio precedente basterebbe eseguire il join della relazione con se stessa senza opportuna ridenominazione. Vediamo ora il solito esempio aggiungendo una tupla:

Impiegato	Capo
Rossi	Lupi
Neri	Bruni
Lupi	Falchi
Falchi	Leoni

Impiegato	Superiore
Rossi	Lupi
Neri	Bruni
Lupi	Falchi
Falchi	Leoni
Rossi	Falchi
Lupi	Leoni
Rossi	Leoni

Non esiste la possibilità di esprimere l'interrogazione che calcoli la chiusura transitiva di una relazione qualunque.

In algebra relazionale l'operazione si simulerebbe con un numero di **join illimitato**.

Divisione

Dati due insiemi di **attributi disgiunti** X_1 e X_2 , una relazione r su $X_1 \cup X_2$ ed una relazione r_2 su X_2 , la **divisione** $r_1 \div r_2$ è una relazione su X_1 che contiene le n -uple ottenute come "proiezione" di n -uple di r che si combinano con tutte le n -uple di r_2 per formare n -uple di r :

$$r \div r_2 = \{ t_1 \text{ su } X_1 \mid \text{per ogni } t_2 \in r_2 \text{ esiste } t \in r \text{ con } t[X_1] = t_1 \text{ e } t[X_2] = t_2 \}$$

Sedi

Filiale	Ufficio
Roma	Acquisti
Roma	Vendite
Roma	Studi
Milano	Acquisti
Milano	Vendite
Milano	Studi
Napoli	Acquisti
Napoli	Vendite

Uffici
Ufficio
Acquisti
Vendite
Studi

Sedi \div Uffici
Filiale
Milano
Roma

L'operatore divisione è derivato perché può essere espresso con altri operatori nel seguente modo:

$$r \div r_2 = \pi_{X_1}(r) - \pi_{X_1}((\pi_{X_1}(r) \times r_2) - r)$$

dove:

- $\pi_{X_1}(r) \times r_2$ contiene le n -uple di $\pi_{X_1}(r)$ "estese" con tutti i possibili valori di r_2
- $(\pi_{X_1}(r) \times r_2) - r$ contiene le "estensioni" di $\pi_{X_1}(r)$ che non compaiono in r
- $\pi_{X_1}((\pi_{X_1}(r) \times r_2) - r)$ contiene le n -uple di $\pi_{X_1}(r)$ per le quali un qualche "completamento" con r_2 non compare in r
- Togliendo queste ultime n -uple a $\pi_{X_1}(r)$ otteniamo le n -uple di $\pi_{X_1}(r)$ che si "combinano" con tutte le n -uple di r_2 , cioè il risultato della divisione

Lezione 04: Metodologie e Modelli Progettuali

Proviamo a **modellare un'applicazione** definendo direttamente lo **schema logico** della base di dati: da dove cominciamo?

- Rischiamo di **perderci** subito **nei dettagli**
- Dobbiamo quindi pensare subito a come **correlare** le varie **tabelle** (chiavi, etc.)
- Abbiamo che il **modello relazionale** è **rígido**.

Progettazione delle basi di dati

È una delle attività del processo di sviluppo dei sistemi informativi.

Va quindi inquadrata in un contesto più generale:

- il ciclo di vita (*lifecycle*) dei sistemi informativi:
 - **insieme ed organizzazione temporale** delle **attività** svolte da **analisti, progettisti, utenti**, nello **sviluppo** e nell'uso dei sistemi informativi.
 - **attività iterativa**, quindi ciclo.

Un buon progetto

I passi del ciclo di vita per essere ben fatti richiedono in generale un linguaggio/modello per descrivere il sistema da progettare.

Per le basi di dati, quindi, la metodologia di progetto deve essere basata su

- Modelli per rappresentare i dati che siano facili da usare
- Decomposizione delle attività in fasi (e/o livelli)
- Strategie e criteri di scelta nei vari passi

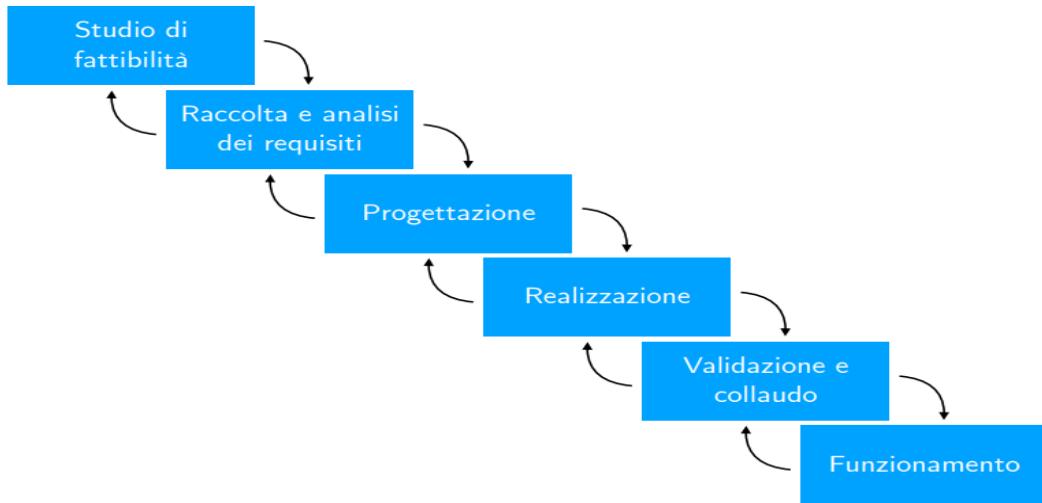
Modello per il ciclo di vita

il primo modello da scegliere è quello per il ciclo di vita.

Esistono vari modelli, il più vecchio è il **modello a cascata** (*waterfall model*)

Nel modello a cascata le fasi sono **ordinate** e “**non ripetibili**”.

Fasi del ciclo di vita



Studio di fattibilità: definizioni costi e priorità

Raccolta ed analisi dei requisiti: studio delle proprietà del sistema

Progettazione: dati e funzioni

Realizzazione: implementazione

Validazione e collaudo: sperimentazione

Funzionamento: il sistema diventa operativo in produzione (**shipping**)

Raccolta ed analisi dei requisiti

Ci sono **due sottofasi**:

- **acquisizione dei requisiti:** il reperimento dei requisiti è un'attività difficile e non standardizzabile.
- **analisi dei requisiti:** l'attività di analisi inizia con i primi requisiti raccolti e spesso indirizza verso altre acquisizioni.

Linguaggi per definire i requisiti: un esempio è l'**UML**.

Ma come si acquisiscono i requisiti?

Direttamente dagli utenti:

- interviste
- documentazione apposita

Da documentazione esistente:

- normative(leggi, regolamenti di settore)
- regolamenti interni, procedure aziendali
- realizzazioni preesistenti

Interazione con gli utenti

Problemi:

- **utenti diversi** possono fornire informazioni diverse
- **utenti a livello più alto** hanno spesso una visione più ampia ma meno dettagliata
- spesso l'acquisizione dei requisiti avviene “*per raffinamenti successivi*”

Spunti:

- effettuare spesso verifiche di comprensione e coerenza
- verificare anche per mezzo di esempi (*generali e relativi a casi limite*)
- richiedere definizioni e classificazioni
- far evidenziare gli aspetti essenziali rispetto a quelli marginali (*ranking dei requisiti*)

Interazione con gli utenti tramite documentazione

Regole generali:

- **standardizzare la struttura delle frasi**
- **separare le frasi sui dati da quelle sulle funzioni**
- **organizzare termini e concetti**
 - costruire un **glossario di termini**
 - **unificare i termini** (individuare i sinonimi)
 - **rendere esplicito il riferimento** fra termini
 - **riorganizzare le frasi per concetti**

Esempio:

Società di formazione (1)	Società di formazione (2)
<p>Si vuole realizzare una base di dati per una società che eroga corsi: di ogni corso vogliamo rappresentare i dati dei partecipanti e dei docenti. Per gli studenti (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 già frequentato (le materie sono in tutto circa 200) e il giudizio finale.</p>	<p>Rappresentiamo anche i corsi attualmente attivi 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.</p>

Società di formazione (3)
<p>Per gli insegnanti (circa 300), rappresentiamo il 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 della società o collaboratori esterni.</p>

Glossario dei termini

Termine	Descrizione	Sinonimi	Collegamenti
Partecipante	Persona che partecipa ai corsi.	Studente	Corso Società
Docente	Docente dei corsi. Può essere esterno.	Insegnante	Corso
Corso	Corso organizzato dalla società. Può avere più edizioni	Materia	Docente
Datore di lavoro	Ente presso cui i partecipanti lavorano o hanno lavorato.	Posto	Partecipante

Strutturazione dei requisiti in gruppi di frasi omogenee

Frasi di carattere generale

Si vuole realizzare una base di dati per una società che eroga corsi: di ogni corso vogliamo rappresentare i dati dei partecipanti e dei docenti.

Frasi relative ai partecipanti

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 nel passato, con la relativa votazione finale in decimi.

Frasi relative ai datori di lavoro

Relativamente ai datori di lavoro presenti e passati dei partecipanti, rappresentiamo il nome, l'indirizzo e il numero di telefono.

Frasi relative ai corsi

Per i corsi (circa 200), rappresentiamo il titolo e il codice, le varie edizioni con date di inizio e fine e, per ogni edizione, rappresentiamo il numero di partecipanti e il giorno della settimana, le aule e le ore dove sono tenute le lezioni.

Frasi relative a tipi specifici di partecipanti

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.

Frasi relative ai docenti

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 nel passato e di quelli che possono insegnare. I docenti possono essere dipendenti interni della società di formazione o collaboratori esterni.

Progettazione

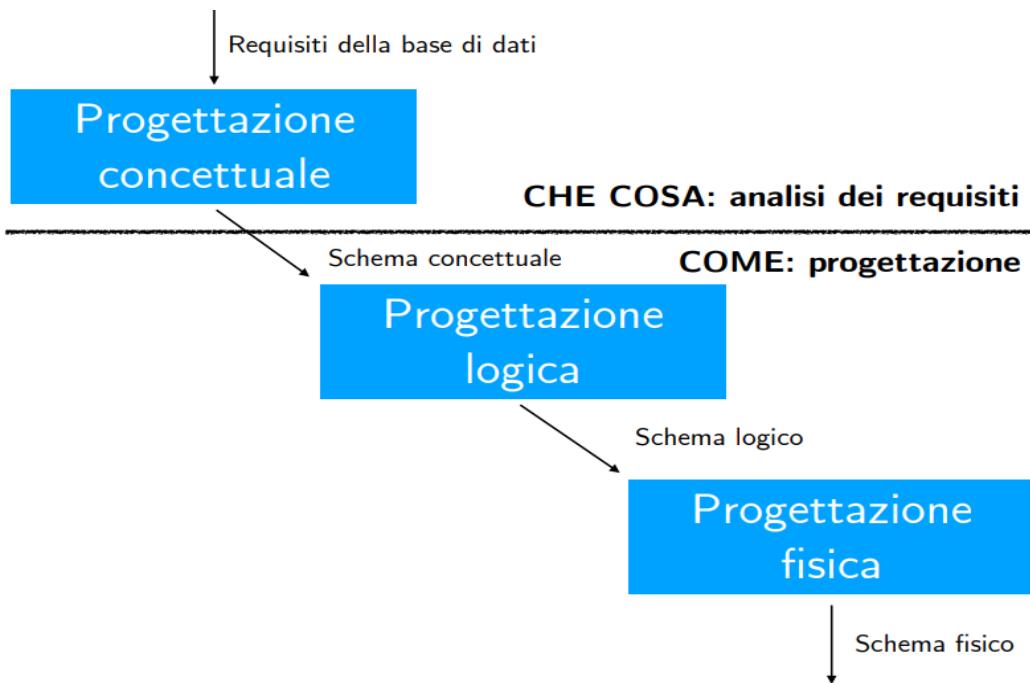
La **progettazione** è una fase del ciclo di vita. Per un sistema software la progettazione consta fondamentalmente di due aspetti:

- **progettazione dei dati**
 - nel caso di sistemi informativi, il progetto dei dati ha un ruolo centrale
- **progettazione delle applicazioni**

Progettare per livelli di astrazione

- **Livello concettuale:** esprime i requisiti di un sistema in una descrizione adatta all'analisi dal punto di vista esterno.
- **Livello Logico:** evidenzia l'organizzazione dei dati dal punto di vista del loro contenuto informativo, descrivendo la struttura di ciascun record ed i collegamenti tra record diversi.
- **Livello fisico:** a questo livello la base di dati è vista come un insieme di blocchi fisici su disco. Qui viene decisa l'allocazione dei dati e le modalità di memorizzazione dei dati sul disco.

Illustriamo con uno schema quello che abbiamo descritto finora.



Modello dei dati

Il modello dei dati è un insieme di costrutti usati per organizzare i dati di interesse e descriverne la dinamica.

La componente fondamentale è l'insieme di **meccanismo di strutturazione** (o *costruttori di tipo*).

Come nei linguaggi di programmazione esistono meccanismi che permettono di definire nuovi tipi, così ogni modello dei dati prevede alcuni costruttori.

Esempio: il modello relazionale prevede il costruttore relazione, che permette di definire insiemi di record omogenei

Schemi ed Istanze

In ogni base di dati esistono:

- lo schema, sostanzialmente invariante nel tempo, che ne descrive la struttura.
 - Nel modello relazionale sono le intestazioni delle tabelle
- l'istanza, i valori attuali, che possono cambiare anche molto rapidamente.
 - Nel modello relazionale sono il corpo di ciascuna tabella.

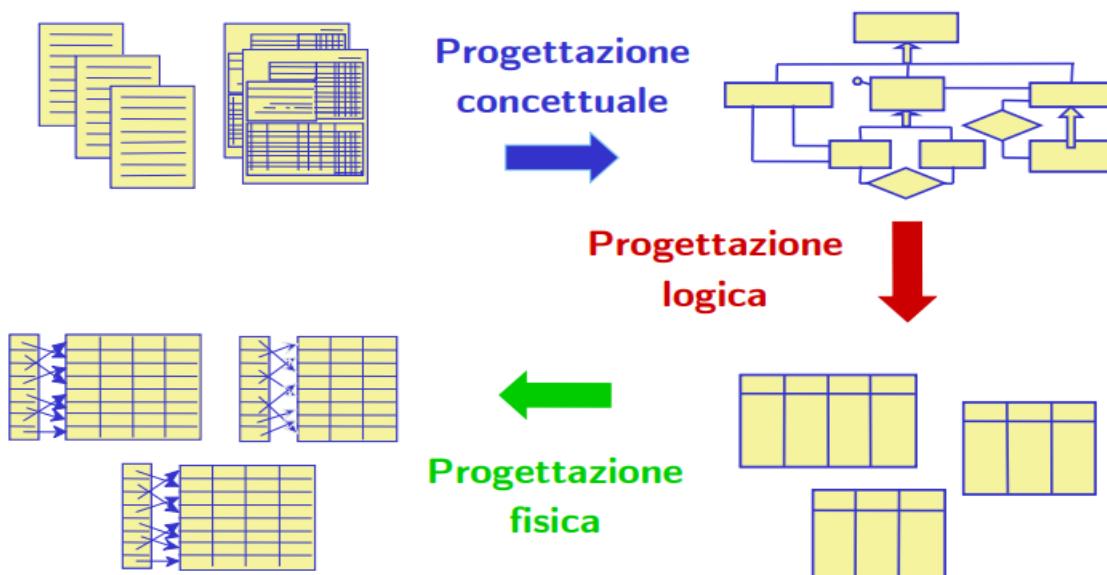
Principali Tipi di Modelli

Esistono eventuali tipi di modelli:

- **Modelli Logici** che sono utilizzati nei DBMS esistenti per l'organizzazione dei dati. Infatti, sono:
 - utilizzati dai programmi

- indipendenti dalle strutture fisiche
Esempi: relazionale, reticolare, gerarchico, ad oggetti.
- **Modelli concettuali** che permettono di rappresentare i dati in modo indipendente da ogni sistema. Infatti:
 - cercano di descrivere i concetti nel mondo reale
 - sono utilizzati nelle fasi preliminari di progettazione
 - il più noto è il **modello Entità-Relazione (Entity-Relationship)**.

Passaggi Tra Modelli



Modello E-R

Il modello E-R si è ormai affermato nelle metodologie di progetto e nei sistemi software di ausilio alla progettazione anche se in una versione leggermente diversa da quella originaria.

Entità

Le entità sono una classe di oggetti (fatti, persone, cose) della applicazione di **interesse** con proprietà **comuni** e con esistenza **autonoma**.

Esempi: impiegato, città, conto corrente, ordine, fattura

Invece le **occorrenze** (o istanza) di entità è un elemento della classe (l'oggetto, la persona, non un valore dei fatti legati all'oggetto).

Esempio: un “impiegato”, non so niente di lui ma esiste con proprietà note.

Graficamente possiamo esprimere così:



Figura 34: Rappresentazione grafica delle entità

Caratteristiche delle entità

Ogni entità ha un **nome** che la identifica **univocamente** nello schema:

- nomi espressivi
- opportune convenzioni
- singolare

Relationship

Legame logico fra due o più entità, rilevante nell'applicazione di interesse.

Esempi:

- Residenza (fra persone e città)
- Esame (fra studente e corso)

Chiamata anche **relazione, correlazione, associazione**

Graficamente si esprimono così:

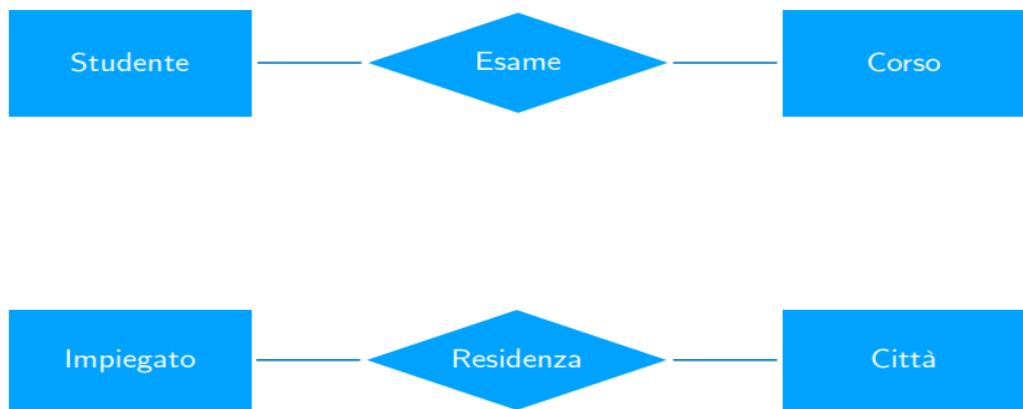


Figura 35: Rappresentazione grafica delle Relationship

Caratteristiche delle relationship

Ogni relationship ha un nome che la identifica **univocamente** nello schema:

- nomi espressivi
- opportune convenzioni
 - singolare
 - sostantivi invece di verbi (se possibile)
 - per non dare un verso alla *relationship*

Occorrenze delle relationship

Una **occorrenza** di una relationship binaria è coppia di occorrenze di entità, ma per ciascuna entità coinvolta.

Una occorrenza di una relationship **n-aria** è una **n-upla** di **occorrenze di entità**, una per ciascuna delle **n entità coinvolte**.

Nell'ambito di una relationship non ci possono essere occorrenze (coppie, n-uple) ripetute.

Esempio:

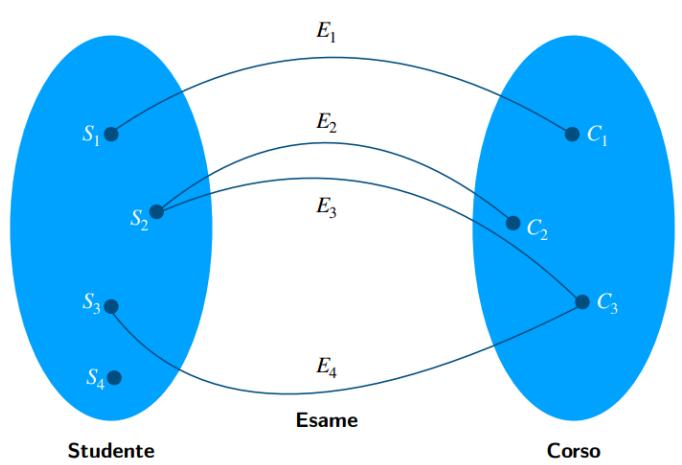


Figura 36: Esempio di occorrenze

Per esempio, vogliamo progettare una base di dati per il libretto elettronico. Che uso vogliamo fare di questa base di dati?

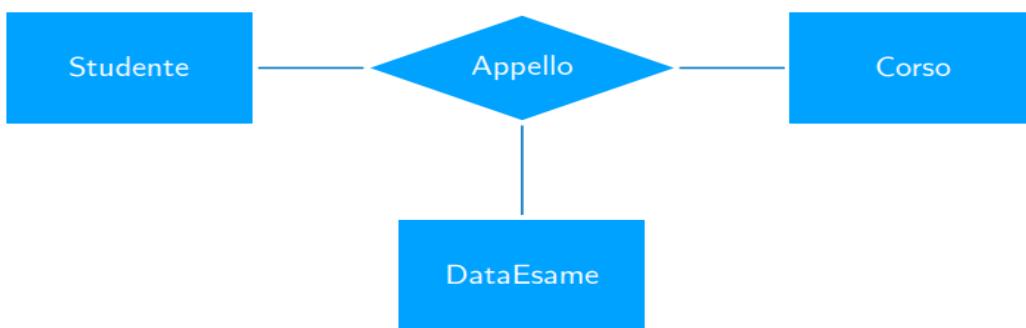
Supporto al servizio statini con la possibilità di calcolare statistiche.

Partiamo con la **prima rappresentazione**

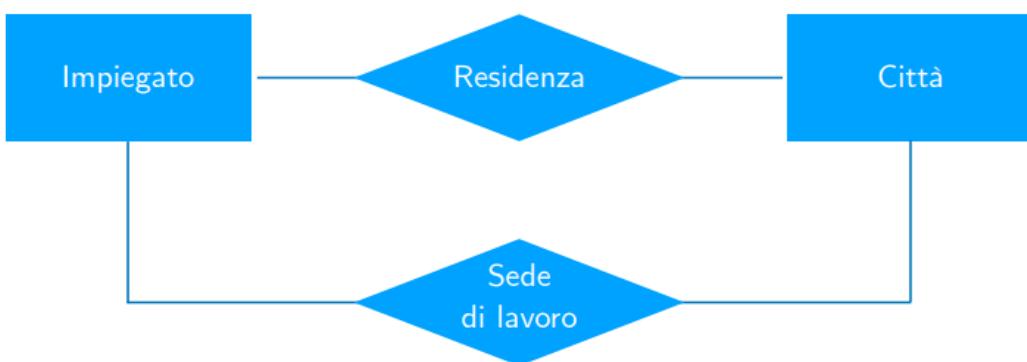


E le **statistiche**? Per esempio, numero di studenti di un corso che sostengono l'esame in un dato appello?

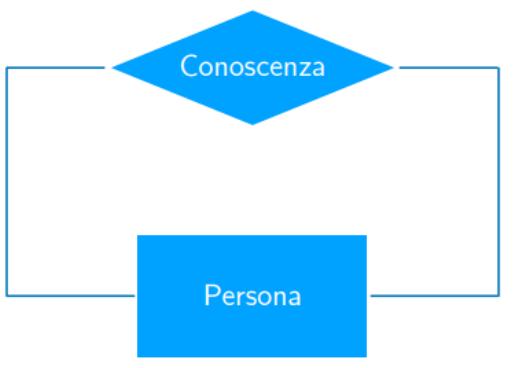
Vediamo ora nella **seconda rappresentazione**



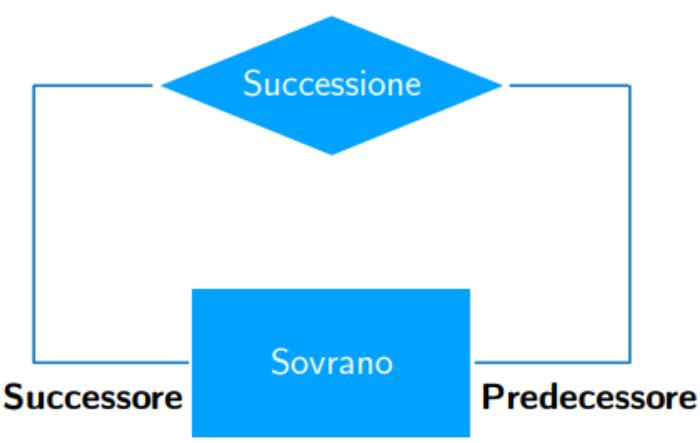
Relationship diverse sulle stesse entità



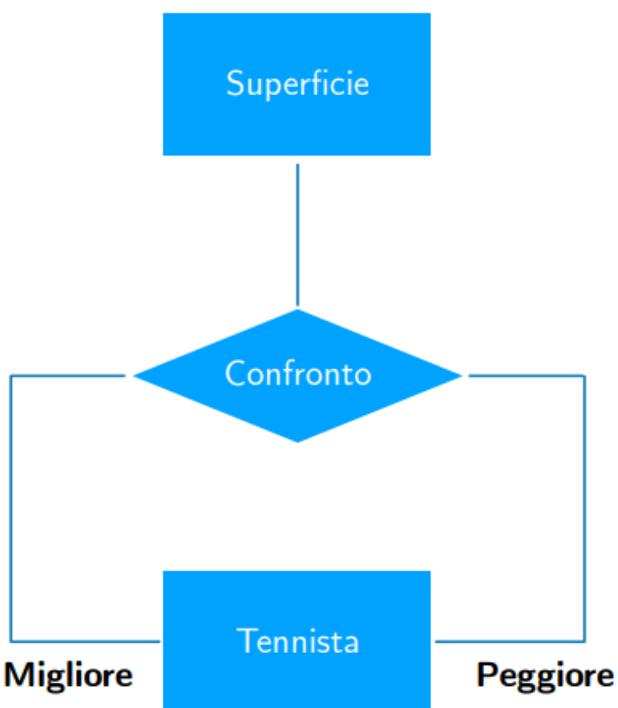
Relationship Ricorsiva



Relationship Ricorsiva con i ruoli



Relationship Mista



Attributo

L'attributo è una **proprietà elementare** di un'**entità** o di una **relationship**, di **interesse** ai fini dell'applicazione.

Quindi associa **ad ogni occorrenza** di entità o relationship un **valore** appartenente ad un insieme detto **dominio dell'attributo**.

Graficamente li esprimiamo così:

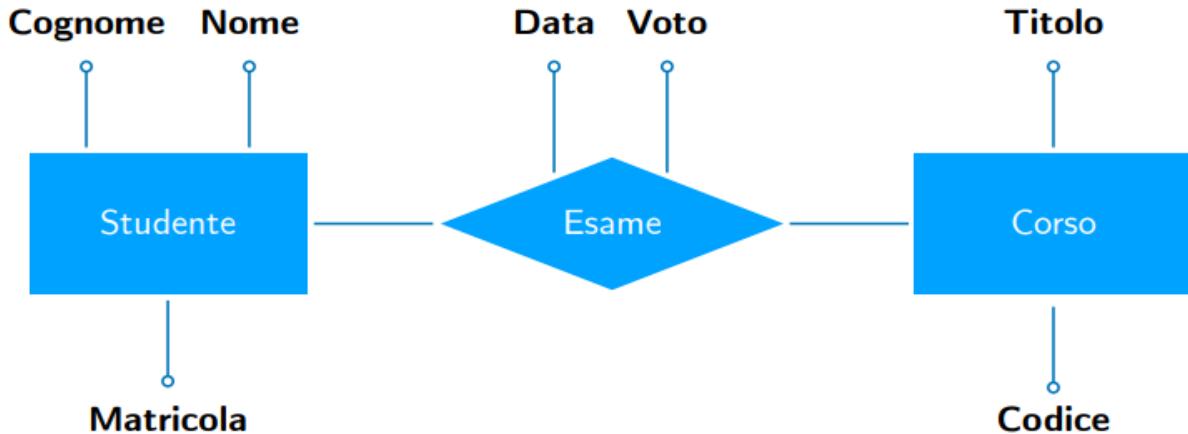


Figura 37: Rappresentazione Grafica di Attributi

Attributi Composti

Gli attributi composti raggruppano attributi di una **medesima** entità o relationship che presentano **affinità** nel loro significato o uso.

Esempio: Via, numero civico e CAP formano un indirizzo

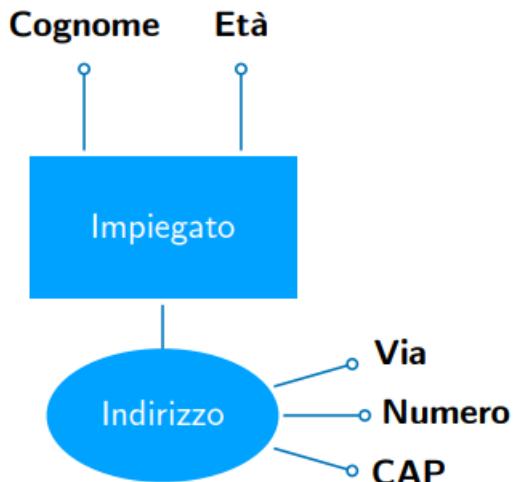
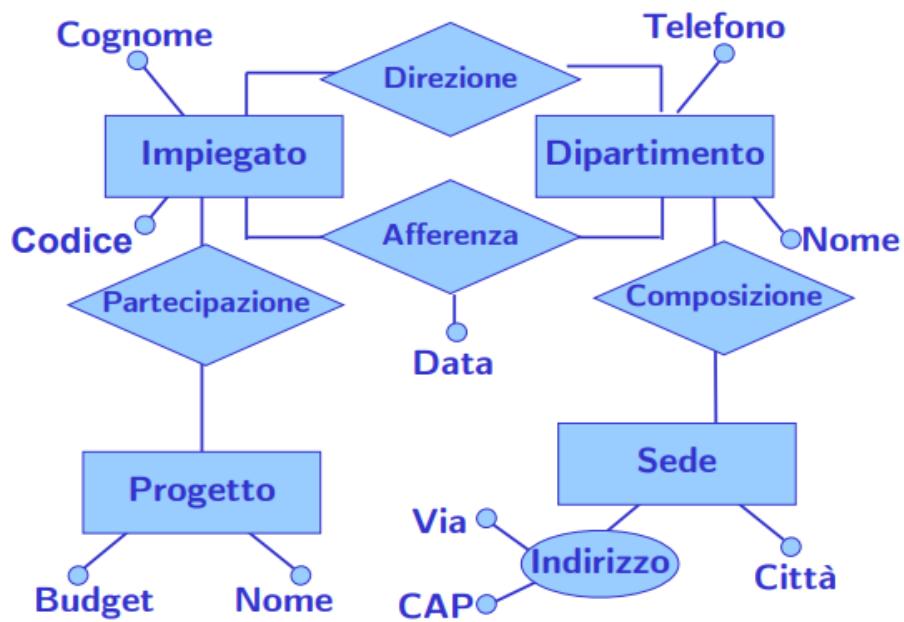


Figura 38: Rappresentazione Grafica Attributi Composti

Schema E-R con solo i costrutti base

Si vuole descrivere l'organizzazione di un'azienda:

- con sedi diverse
- ogni sede è composta di vari dipartimenti
- gli impiegati dell'azienda afferiscono ai vari dipartimenti ed un impiegato li dirige
- Gli impiegati lavorano su progetti
- Ogni entità o *relationship* può avere vari attributi



Concetti Inesprimibili

- Un dipartimento ha un solo direttore?
- Un impiegato può afferire ad un solo dipartimento?
- Il direttore di un dipartimento afferisce a quel dipartimento?...

Altri costrutti del Modello E-R

- **Cardinalità:**
 - di relationship
 - di attributo
- **Identificatore:**
 - interno
 - esterno

Cardinalità di relationship

Sono una coppia di valori associati ad ogni entità che partecipa ad una relationship. Quindi specificano il numero **minimo** e **massimo di occorrenze** della *relationship* cui ciascuna occorrenza di entità può partecipare.



Per semplicità usiamo solo 3 simboli:

- **0 e 1** per la **cardinalità minima**;
 - **0** = “*partecipazione opzionale*”
 - **1** = “*partecipazione obbligatoria*”
- **1 e N** per la **massima**, quindi non pone alcun limite



Tipi di relationship

Col riferimento alle **cardinalità massime**, abbiamo relationship:

- **uno a uno**



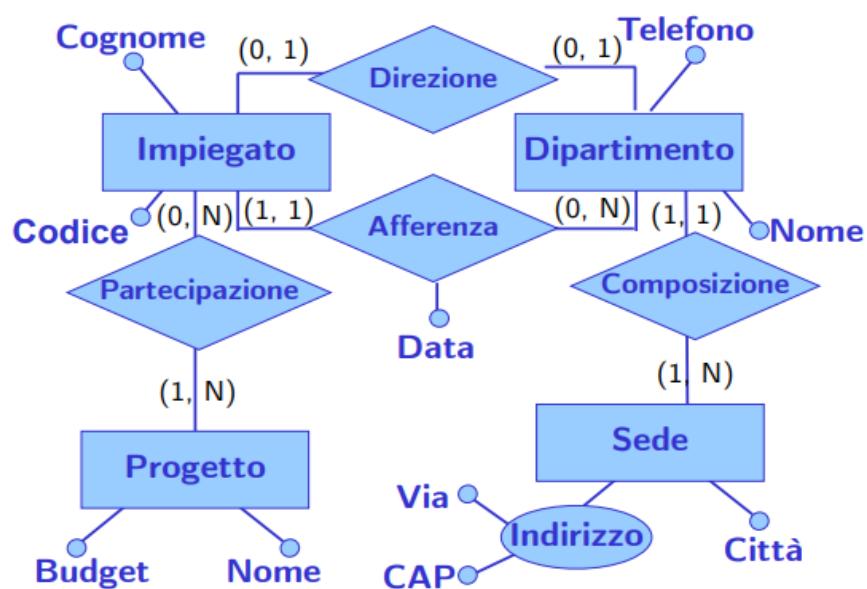
- **uno a molti**



- **molti a molti**

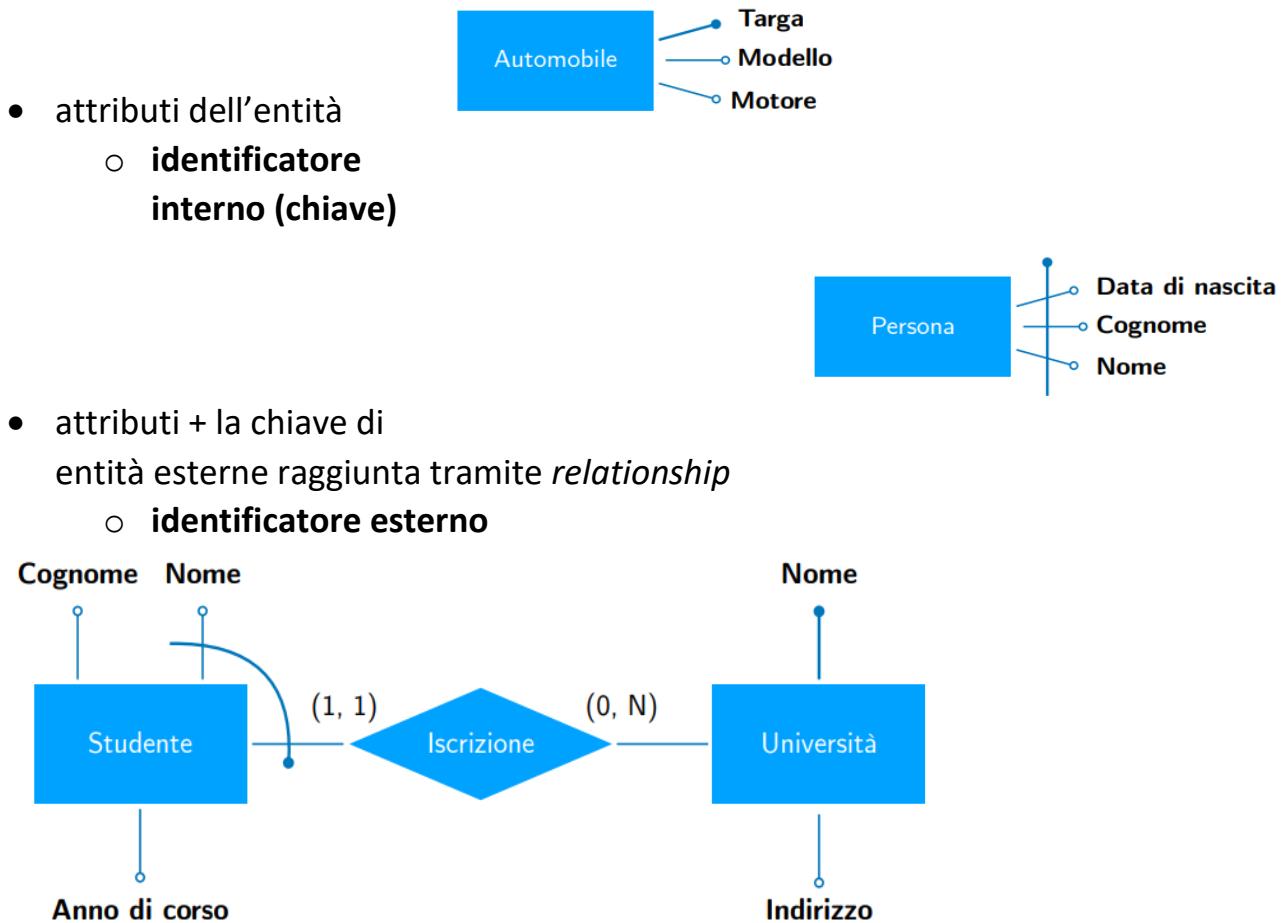


Esempio finale



Identificatore di entità

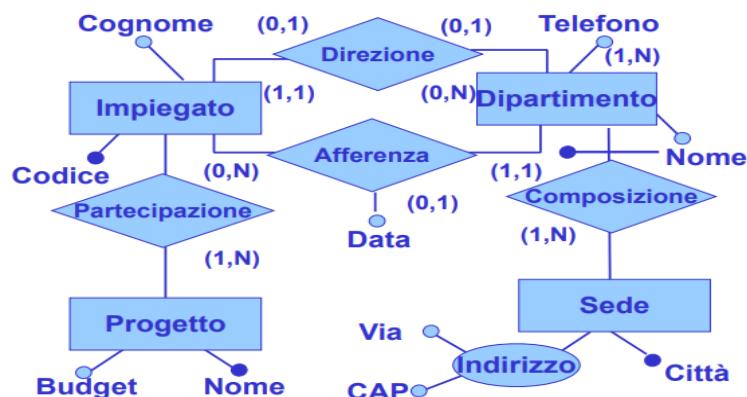
L'**identificatore di entità** è uno strumento per l'identificazione univoca delle occorrenze di un'entità. Esso è costituito da:



Caratteristiche degli identificatori

- Ogni entità deve possedere almeno un **identificatore**, ma può averne in generale più di uno.
- Una **identificazione esterna** è possibile solo attraverso una *relationship* a cui l'entità da identificare partecipa con **cardinalità (1,1)**

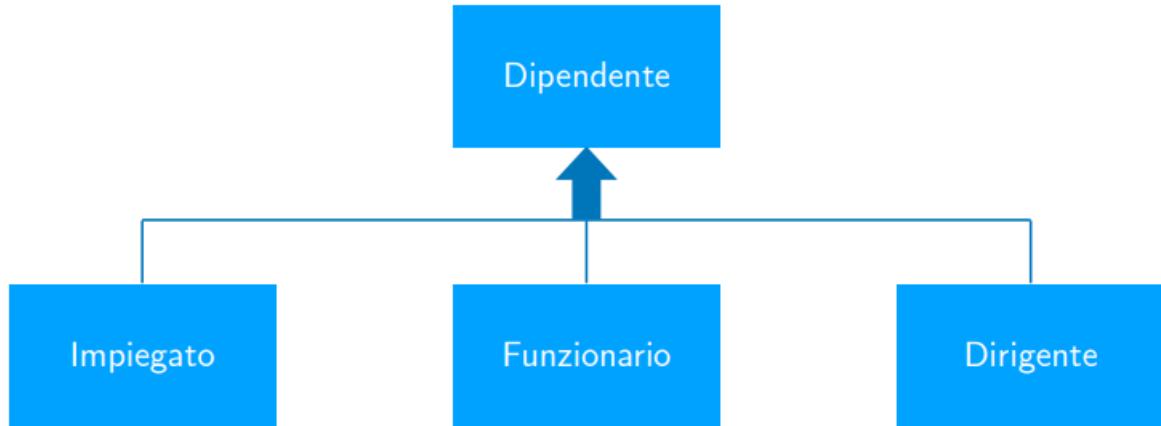
Esempio:



Generalizzazione

La **generalizzazione** mette in relazione **una o più entità** E_1, E_2, \dots, E_n con una entità E che le comprende come **casi particolari**.

- E è una **generalizzazione** di E_1, E_2, \dots, E_n
- E_1, E_2, \dots, E_n sono **specializzazioni** (o sottotipi) di E .

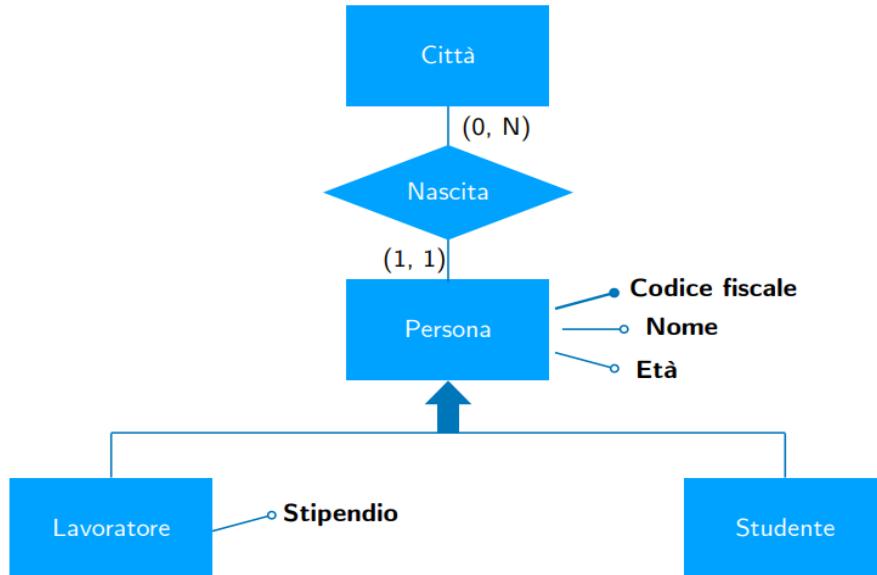


Proprietà delle Generalizzazioni

Se E (genitore) è generalizzazione di E_1, E_2, \dots, E_n (figlie)

- allora ogni proprietà di E è significativa per E_1, E_2, \dots, E_n
- allora ogni occorrenza di E_1, E_2, \dots, E_n è occorrenza anche di E

Esempio:



Caratteristiche delle generalizzazioni

- **Ereditarietà:** tutte le **proprietà** (attributi, relationship, altre generalizzazioni) dell'entità genitore vengono **ereditate** dalle entità figlie e **non rappresentate esplicitamente**.
- **Generalizzazione totale:** se ogni occorrenza dell'entità genitore è occorrenza di almeno una delle entità figlie, altrimenti è **parziale**
- **Generalizzazione esclusiva:** se ogni occorrenza dell'entità genitore è occorrenza di al più una delle entità figlie, altrimenti è **sovraposta**.

Generalizzazione Totale e Parziale

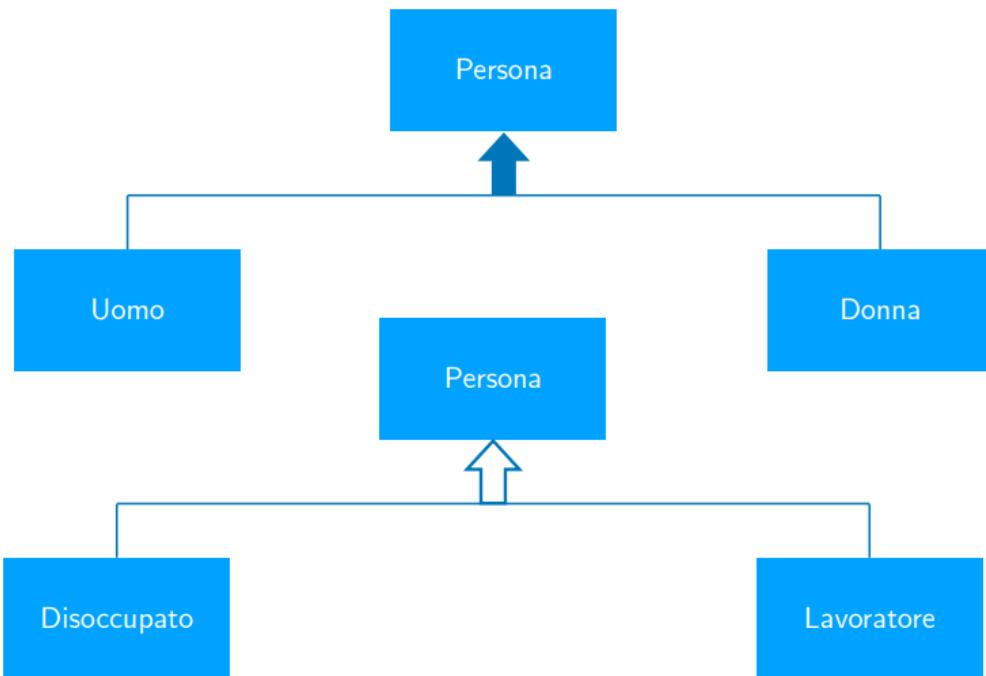
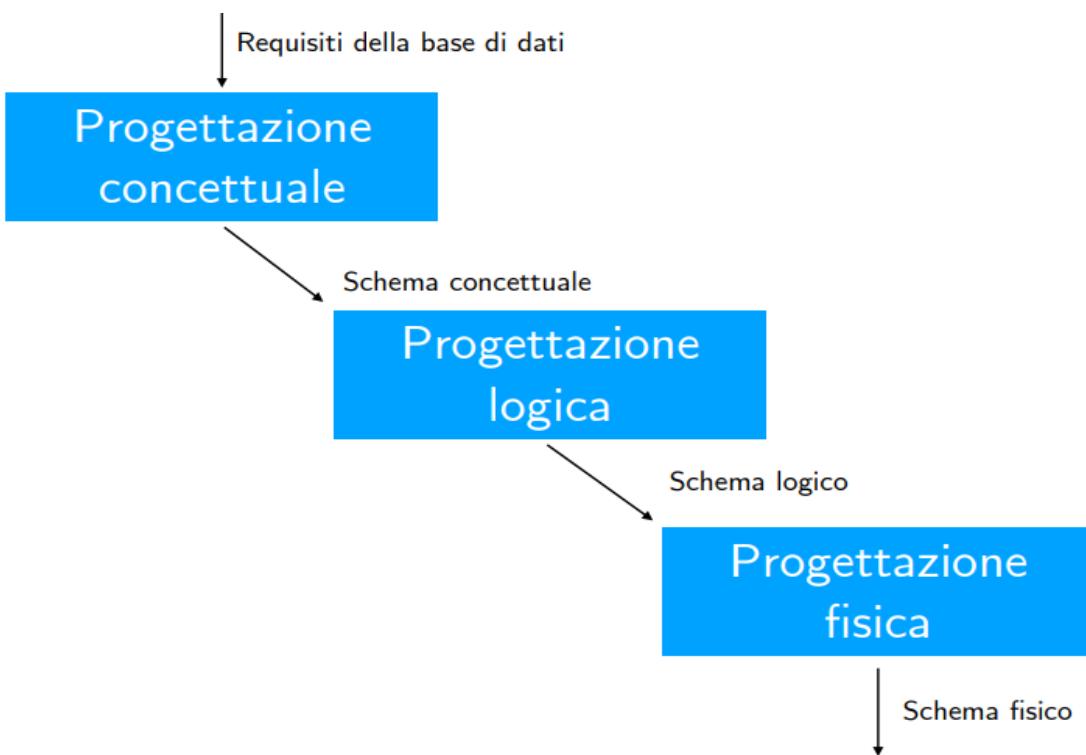


Figura 39: Esempio di Generalizzazione Totale e Parziale

Altre proprietà

- Possono esistere **gerarchie a più livelli** e **multiple generalizzazioni** allo stesso livello.
- **Un'entità** può essere inclusa **in più gerarchie**, come genitore e/o figlia.
- Se una generalizzazione ha solo un'entità figlia si parla di **sottoinsieme**
- Il genitore di una generalizzazione totale può **non avere identificatore**, purché...

Lezione 05: Progettazione Concettuale



Quale costrutto E-R va utilizzato per rappresentare un concetto presente nelle specifiche?

Bisogna basarsi sulle definizioni dei costrutti del modello E-R.

- se ha proprietà significative e descrive oggetti con esistenza autonoma:
 - **entità**
- se è semplice e non ha proprietà
 - **attributo**
- se correla due o più concetti
 - **relationship**
- se è caso particolare di un altro
 - **generalizzazione**

Design Pattern

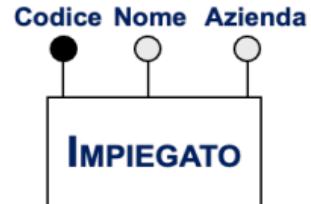
Soluzioni progettuali a problemi comuni

Largamente usati nell'ingegneria del software

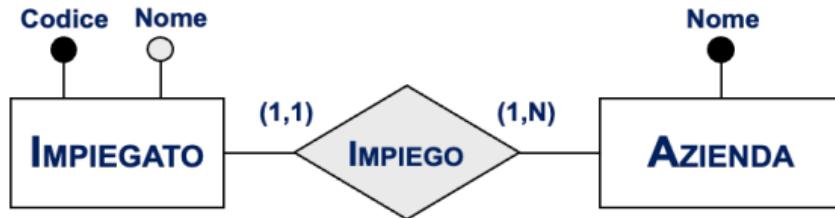
Vediamo alcuni pattern comuni nella progettazione concettuale di basi di dati.

Reificazione³ di un attributo di entità

Nel primo caso Azienda è un semplice attributo, una semplice stringa di caratteri che assegniamo a un'occorrenza di Impiegato.



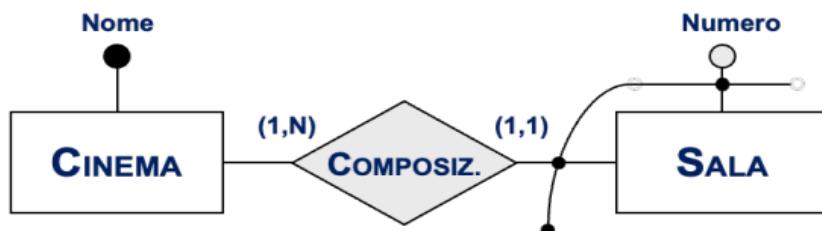
Nel secondo caso rappresentiamo esplicitamente il concetto di Azienda, facendolo diventare un'entità.



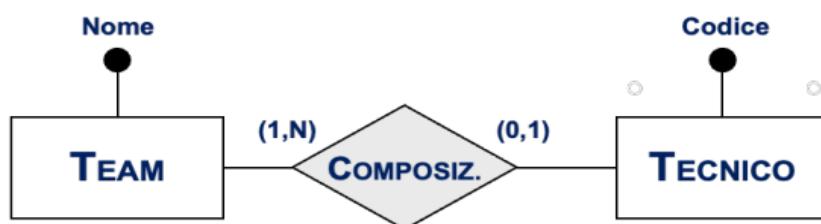
Parte-di

Ora si vuole rappresentare il fatto che un'entità è parte di un'altra entità. 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 e richiede un'identificazione esterna.



Nel secondo, l'entità contenuta nell'altra ha esistenza autonoma, come indicato dalla partecipazione opzionale alla relazione.

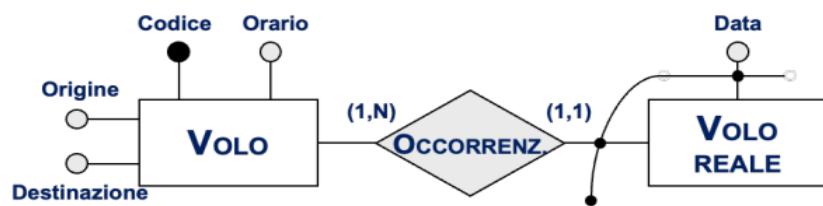


³**Reificazione:** il processo tramite cui un concetto astratto viene trasformato in un modello dei dati o altri oggetti creati tramite un linguaggio di programmazione.

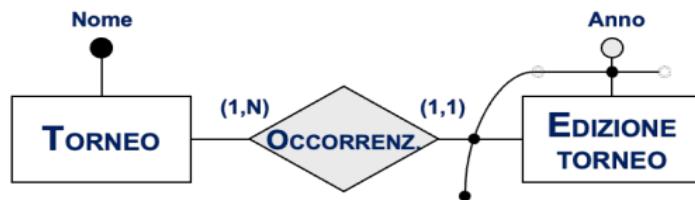
Istanza-di

Le occorrenze di un'entità della relazione sono istanze di occorrenze dell'altra entità.

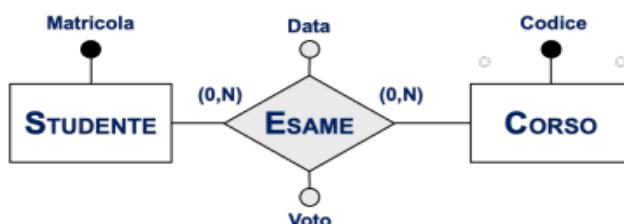
Nel primo caso abbiamo un'entità che descrive il concetto astratto di volo presente sull'orario di una compagnia aerea, con un codice, un'origine, una destinazione e un orario, e un'altra entità che rappresenta il volo "reale", vale a dire l'istanza di un certo volo in una certa data. L'identificazione del volo reale avviene attraverso la data, e esternamente il volo di cui è istanza.



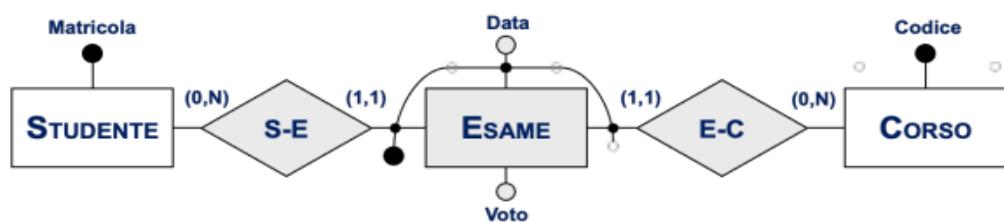
Un caso analogo è l'altro schema con cui viene rappresentato il concetto di torneo sportivo e una sua edizione.



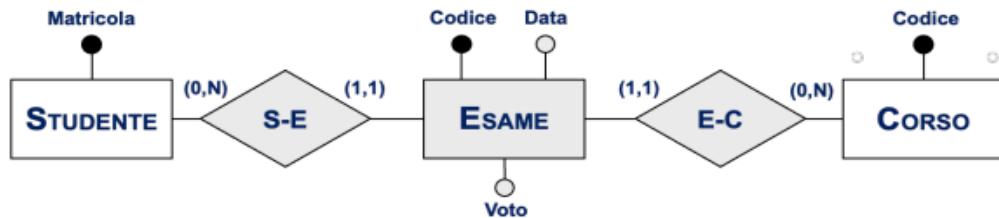
Reificazione di relationship binaria



Nei casi in cui si utilizza una relazione, tipicamente molti a molti, per descrivere un concetto che lega altri due concetti.

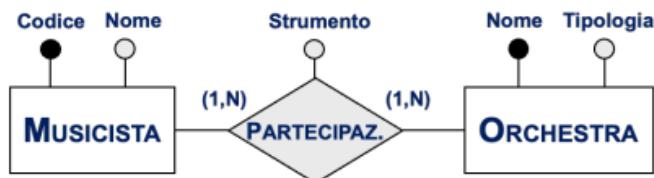


Questa soluzione è valida solo se ogni studente può sostenere una sola volta un certo esame perché, per definizione, una occorrenza della relazione Esame è un insieme di coppie Studente-Corso univoche. Dobbiamo quindi reificare la relazione Esame e renderla un'entità. In questo caso l'identificazione di un Esame avviene attraverso lo Studente, il Corso e la data dell'Esame.

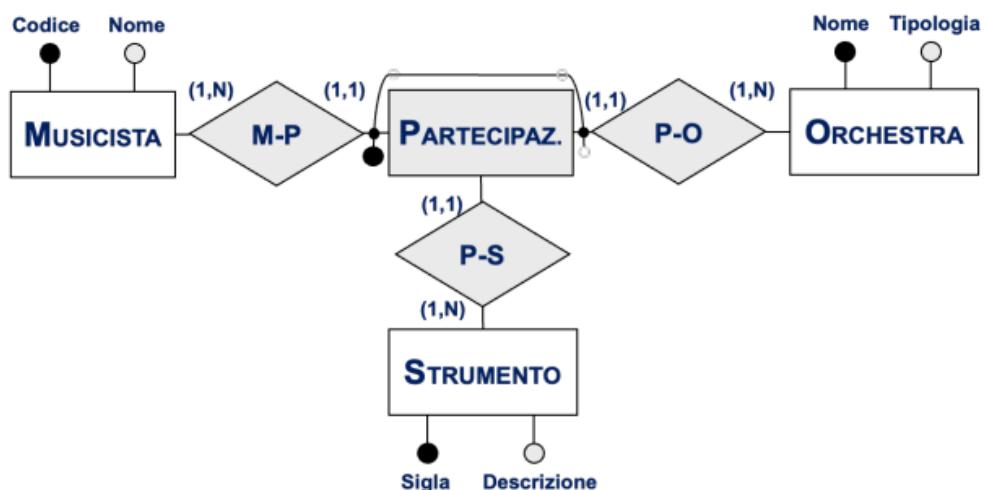


Nel secondo caso abbiamo introdotto un codice identificativo Codice. Questa scelta semplifica le cose perché non richiede un'identificazione esterna complessa, ma bisogna tenere conto del fatto che il codice è un concetto nuovo non presente nelle specifiche che dovrà quindi essere opportunamente gestito dal sistema informativo in via di sviluppo.

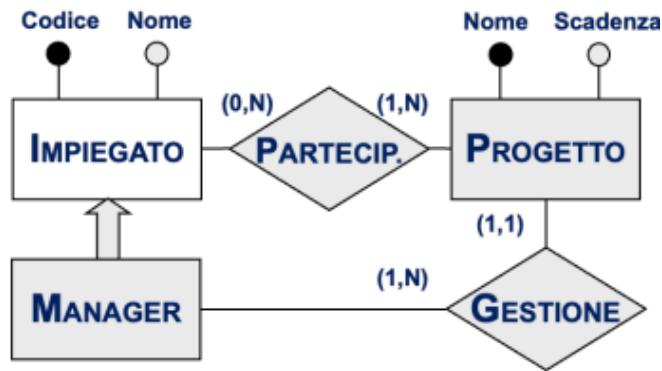
Reificazione di attributo di *relationship*



La prima figura rappresenta la partecipazione di un musicista a un'orchestra con un certo strumento. Questa relazione è corretta se il musicista suona strumenti diversi ma suona, per ogni orchestra, sempre lo stesso strumento. Se lo strumento è un concetto rilevante per l'applicazione dobbiamo reificare l'attributo della relazione e quindi reificare la relazione stessa.



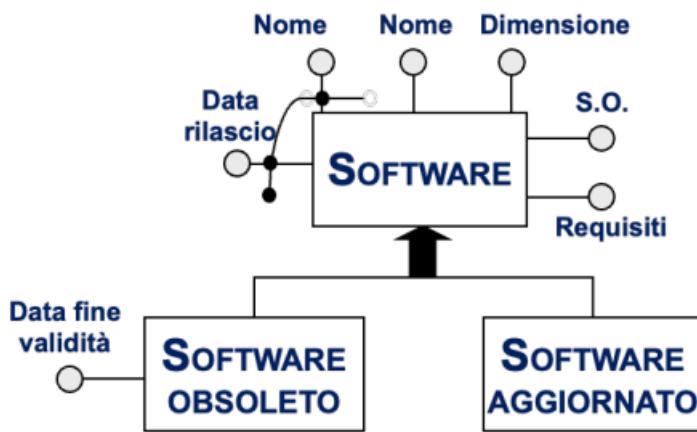
Caso particolare di entità



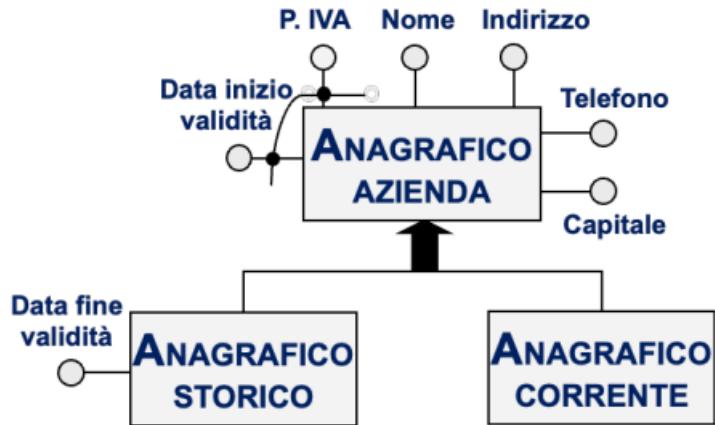
In questo schema con una generalizzazione è ragionevole assumere che un Manager può gestire solo un Progetto al quale partecipa. Questo implica che ogni coppia Manager-Prodotto che compare tra le occorrenze della relazione Gestione deve comparire anche tra le occorrenze della relazione Partecipazione. Questo vincolo però non può essere espresso direttamente sullo schema con un apposito costrutto e va quindi aggiunta una regola alla documentazione dello schema.

Storicizzazione di concetto

Si tratta di uno schema nel quale vogliamo memorizzare le informazioni correnti di un'azienda (software), tenendo traccia dei dati che sono variati.

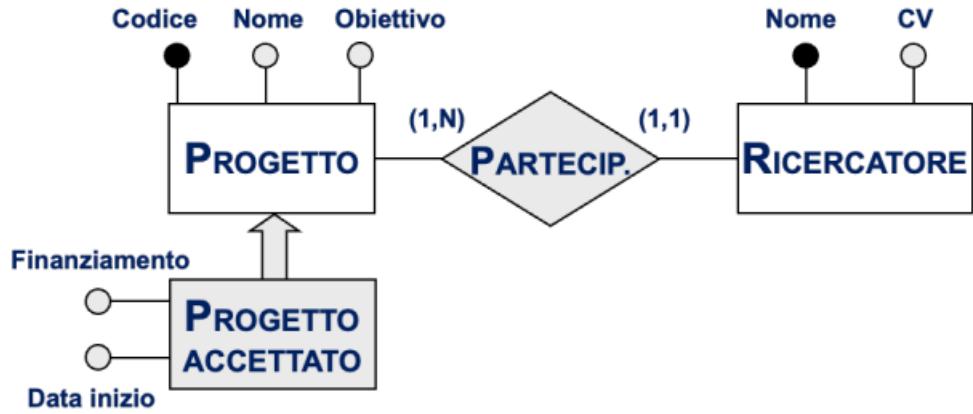


Come suggerito dallo schema, una soluzione piuttosto efficace consiste nell'utilizzare allo scopo due entità con gli stessi attributi dove una rappresenta il concetto di interesse con le informazioni aggiornate, l'altra lo storico.



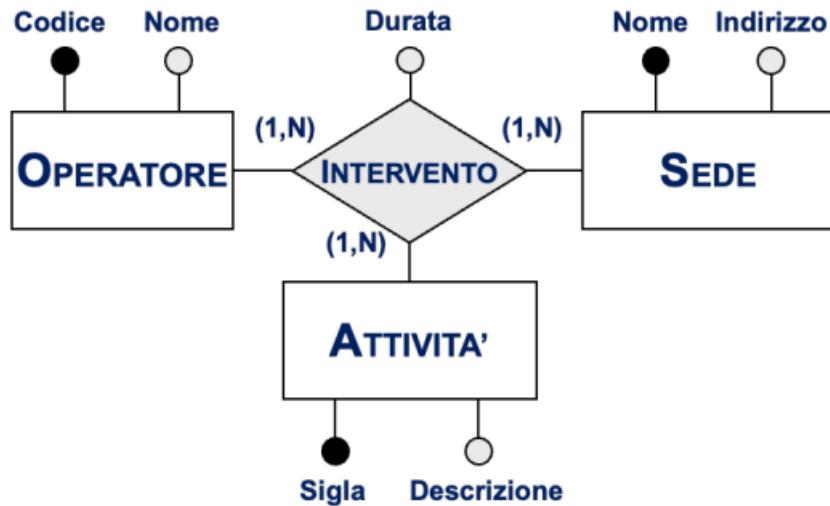
Prendendo un esempio simile notiamo che le proprietà di queste entità vengono messe a fattor comune mediante una generalizzazione la cui entità genitore rappresenta tutte le informazioni anagrafiche delle aziende, sia quelle correnti sia quelle passate. Vengono inoltre introdotti degli attributi per definire l'intervallo di validità dei dati (Data Inizio-Data Fine). L'identificazione si ottiene aggiungendo all'identificatore "naturale" Partita Iva, la data di inizio validità delle informazioni, ovvero il momento in cui esse sono state introdotte: questo diventerà anche la data di fine validità delle informazioni che vengono soppiantate

Evoluzione di concetto



In questo esempio 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 Progetti che vengono proposti con l'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. Come si vede dallo schema in figura, il concetto di generalizzazione si presta bene a modellare questa situazione.

Relationship ternaria



Nell'esempio è stata scelta correttamente una **relationship 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 e in Sedi diverse.

Reificazione di relationship ternaria

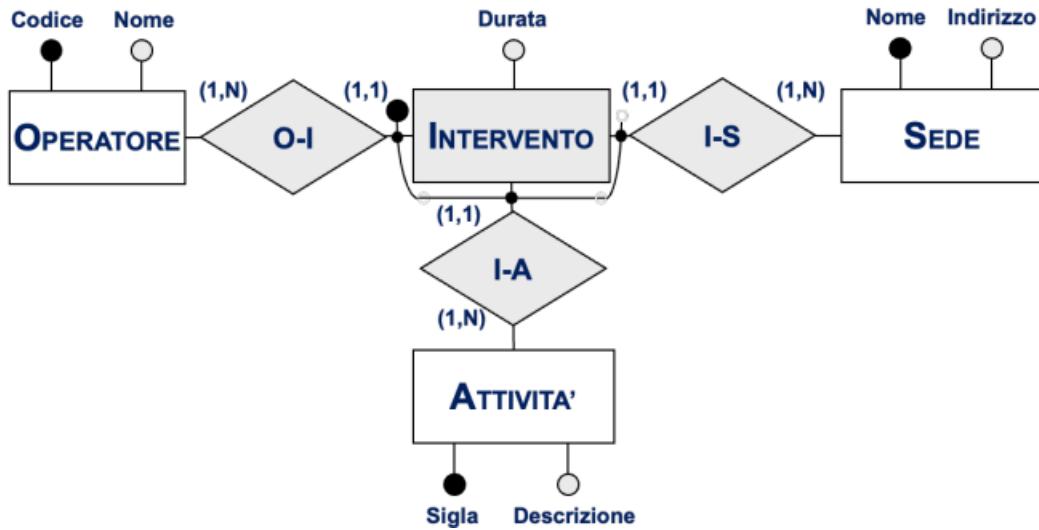


Figura 40: Esempio di Reificazione di Relationship ternaria

Anche questa relazione può essere *reificata* e questa operazione si rende necessaria quando la realtà da modellare è diversa da quella appena descritta, il nuovo schema (Figura 40) modella esattamente la situazione dello schema originario perché la nuova entità risulta identificata da tutte le entità originarie. Cambiando opportunamente l'identificazione siamo però in grado di modellare con questo pattern altre situazioni per le quali la relationship ternaria non sarebbe corretta.

In particolare, se in ogni Sede ogni Operatore svolgesse sempre la stessa Attività, l'entità Intervento sarebbe identificata solo dalle entità Sede e Operatore.

Se, viceversa, in ogni Sede ogni Attività venisse svolta sempre dallo stesso Operatore, l'entità Intervento sarebbe identificata solo dalle entità Attività e Sede.

Se, infine, ogni Operatore svolgesse ogni Attività in una sola Sede, l'entità Intervento sarebbe identificata solo dalle entità Operatore e Attività.

Reificazione di relationship ternaria

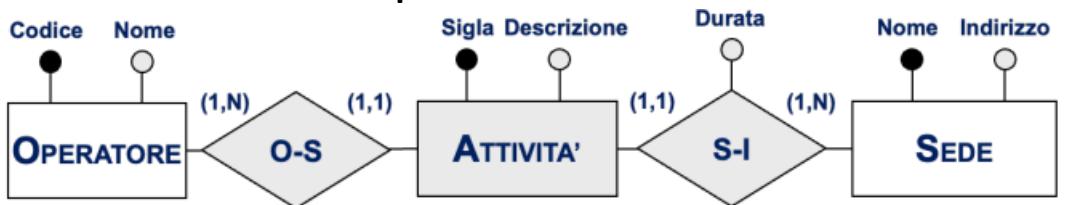


Figura 41: Esempio di Reificazione di relationship ternaria

L'ultimo schema descrive nel modo migliore la situazione in cui la sola entità Attività è identificante: cioè ogni Attività viene svolta in una sola Sede da un solo Operatore. In questo caso lo schema si semplifica perché il legame tra l'Attività e la Sede si può rappresentare separatamente da quello tra l'Attività e l'Operatore.

Strategie di progetto

Come procediamo con tante specifiche anche dettagliate? Come ci orientiamo? Strategie:

- **top-down**
- **bottom-up**
- **inside-out**

Strategia **top-down**

Si parte da uno **schema iniziale** che viene **successivamente raffinato ed integrato** per mezzo di **primitive** che lo trasformano in una serie di schemi **intermedi** per arrivare allo schema E-R finale. Mediante opportune trasformazioni elementari, dette **primitive di trasformazione top-down**, aumentiamo il dettaglio dei vari concetti presenti.

Primitive di raffinamento

- Da entità ad associazione tra entità
- Da entità a generalizzazione
- Da associazione ad insiemi di associazioni
- Da associazione ad entità con associazioni
- Introduzione di attributi su entità ed associazioni

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 (si osservi che le primitive agiscono su singoli concetti). Questo è possibile solo quando si possiede, sin dall'inizio, una visione globale e astratta di tutte le componenti del sistema, ma ciò è estremamente difficile quando si ha a che fare con applicazioni di una certa complessità.

Strategia **bottom-up**

Si parte delle **specifiche iniziali** e si suddividono in componenti sempre più piccole fino a dare specifica ad una **componente minima** di cui si dà lo schema E-R.

Gli schemi prodotti vengono **fusi ed integrati** fino ad ottenere, attraverso una completa integrazione di tutte le componenti, lo **schema concettuale finale**.

A differenza della strategia top-down, con questa strategia i vari concetti presenti nello **schema finale** vengono via via introdotti durante le varie fasi.

Anche in questo caso, lo schema finale si ottiene attraverso alcune trasformazioni, dette **primitive di trasformazione 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.

Primitive di trasformazione bottom-up:

- Generazione di entità
- Generazione di associazione
- Generazione di generalizzazione

Nella pratica cosa succede?

Si procede di solito con una **strategia mista** che cerca di combinare i vantaggi della strategia **top-down** con quelli della strategia **bottom-up**. Il progettista suddivide i requisiti in **componenti separate**, come nella strategia bottom-up, ma allo stesso tempo definisce uno **schema scheletro** contenente, a livello astratto, i concetti principali dell'applicazione (*più citati o perché indicati esplicitamente come cruciali e li si organizza in un semplice schema concettuale*). Questo schema ci fornisce una visione unitaria, sia pure astratta, dell'intero progetto e favorisce le fasi di integrazione degli schemi sviluppati. Sulla base dello schema scheletro si può **decomporre, poi raffinare, espandere e integrare**. Riassumiamo il tutto in step:

- si individuano i **concetti principali** e si realizza uno **schema scheletrico**
- sulla base di questo si può **decomporre**
- poi si **raffina, si espande, si integra**

Una metodologia

1. Analisi dei requisiti:

- Analizzare i requisiti ed eliminare le ambiguità
- Costruire un glossario dei termini
- Raggruppare i requisiti in insiemi omogenei

2. Passo base

- Definire uno schema scheletrico con i concetti più rilevanti

3. Passo di decomposizione (*da effettuare se appropriato o necessario*)

- Effettuare una decomposizione dei requisiti con riferimento ai concetti presenti nello schema scheletro

4. Passo iterativo (*da ripetere finché non si è soddisfatti*)

- Raffinare i concetti presenti sulla base delle loro specifiche
- Aggiungere concetti per descrivere specifiche non descritte

5. Passo di integrazione (*da effettuare se è stato eseguito il passo 3*)

- Integrare i vari sottoschemi in uno schema generale facendo riferimento allo schema scheletro

6. Analisi di qualità (*ripetuta e distribuita*)

- Verificare le qualità dello schema e modificarlo

Qualità di uno schema concettuale

- **Correttezza:** uno schema è corretto quando utilizza propriamente i **costrutti** messi a disposizione dal **modello concettuale** di riferimento. Gli errori possono essere sintattici o semantici. I primi riguardano un uso non ammesso di costrutti come per esempio una generalizzazione tra relazioni invece che tra entità. I secondi riguardano invece un uso di costrutti che non rispetta la loro definizione, per esempio l'uso di una relazione per descrivere il fatto che un'entità è specializzazione di un'altra. La correttezza si può verificare per ispezione, confrontando i concetti presenti nello schema in via di costruzione con le specifiche e con le definizioni dei costrutti del modello concettuale usato
- **Completezza:** uno schema è completo quando rappresenta tutti i dati di interesse e quando tutte le operazioni possono essere eseguite a partire dai concetti descritti nello schema. La completezza si può verificare controllando che tutte le specifiche sui dati siano rappresentate da qualche concetto presente nello schema che stiamo costruendo, e che tutti i concetti coinvolti in un'operazione presente nelle specifiche siano raggiungibili "navigando" attraverso lo schema.

- **Leggibilità:** uno schema è leggibile quando rappresenta i requisiti in maniera naturale e facilmente comprensibile. Necessario rendere lo schema autoesplicativo, per esempio, mediante una scelta opportuna dei nomi da dare ai concetti. La leggibilità dipende da criteri puramente estetici. Alcuni consigli sono: -disporre i costrutti su una griglia scegliendo come elementi centrali quelli con più relazioni con altri; -tracciare solo linee perpendicolari e cercare di minimizzare le intersezioni; -disporre le entità che sono genitori di generalizzazioni sopra le relative entità figlie. La leggibilità si può verificare facendo delle prove di comprensione con gli utenti.
- **Minimalità:** uno schema è minimale quando tutte le specifiche sui dati sono rappresentate una sola volta nello schema. Uno schema quindi non è minimale quando esistono delle ridondanze, ovvero concetti che possono essere derivati da altri (presenza di cicli dovuta alla presenza di relazioni/generalizzazioni). La minimalità può essere verificata per ispezione, controllando se esistono concetti che possono essere eliminati dallo schema che stiamo costruendo senza inficiare la sua completezza.

Esempio:

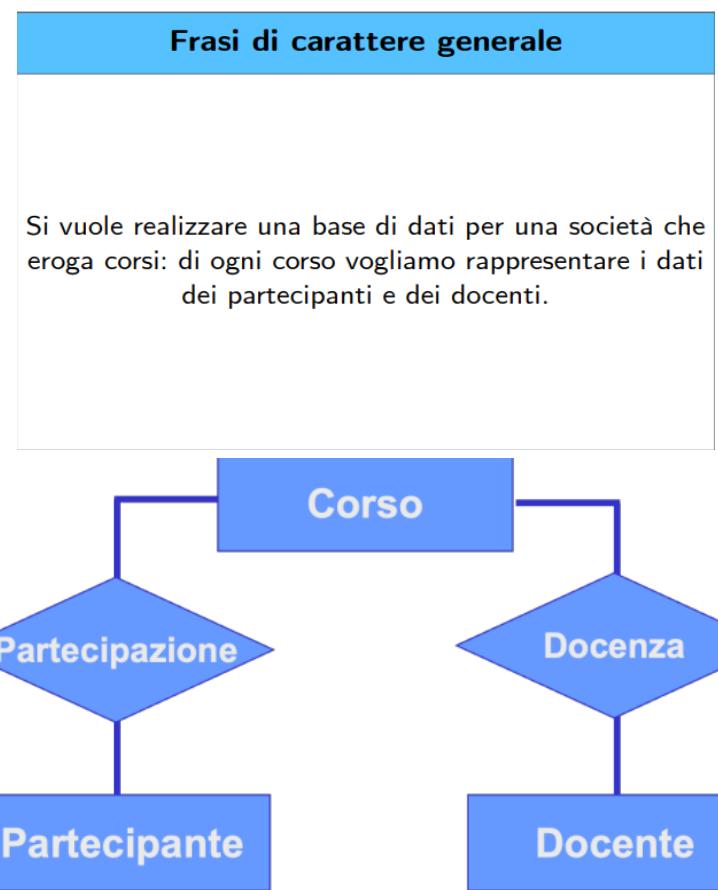


Figura 42: Esempio di Schema Scheletrico

Esempio continua...

Frasi relative ai partecipanti

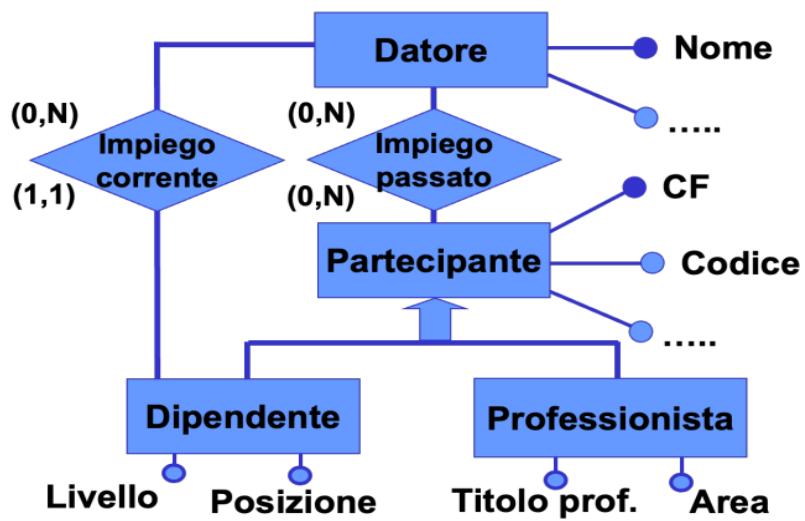
Per i partecipanti (circa 5000), identificati da un codice, rappresentiamo il codice fiscale, il cognome, l'età, il sesso, la città di nascita, i nomi dei loro attuali datori di lavoro e di quelli precedenti (insieme alle date di inizio e fine rapporto), le edizioni dei corsi che stanno attualmente frequentando e quelli che hanno frequentato nel passato, con la relativa votazione finale in decimi.

Frasi relative ai datori di lavoro

Relativamente ai datori di lavoro presenti e passati dei partecipanti, rappresentiamo il nome, l'indirizzo e il numero di telefono.

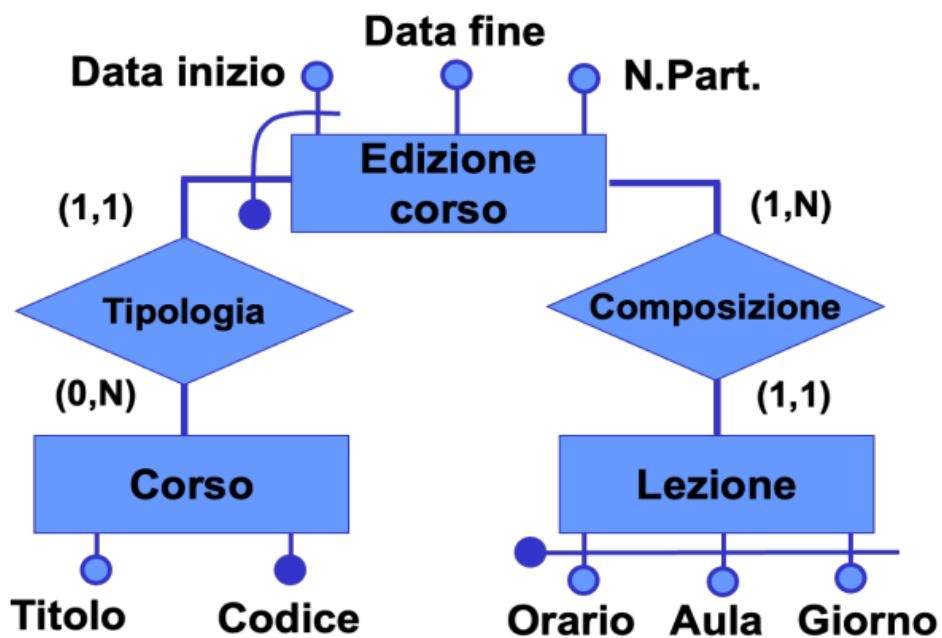
Frasi relative a tipi specifici di partecipanti

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.



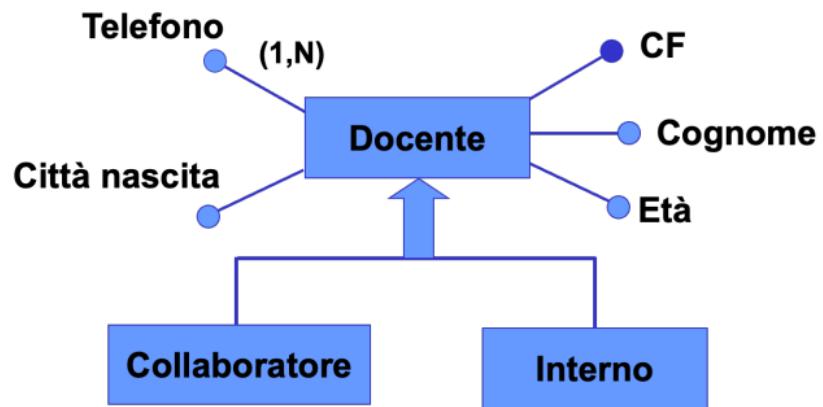
Frasi relative ai corsi

Per i corsi (circa 200), rappresentiamo il titolo e il codice, le varie edizioni con date di inizio e fine e, per ogni edizione, rappresentiamo il numero di partecipanti e il giorno della settimana, le aule e le ore dove sono tenute le lezioni.

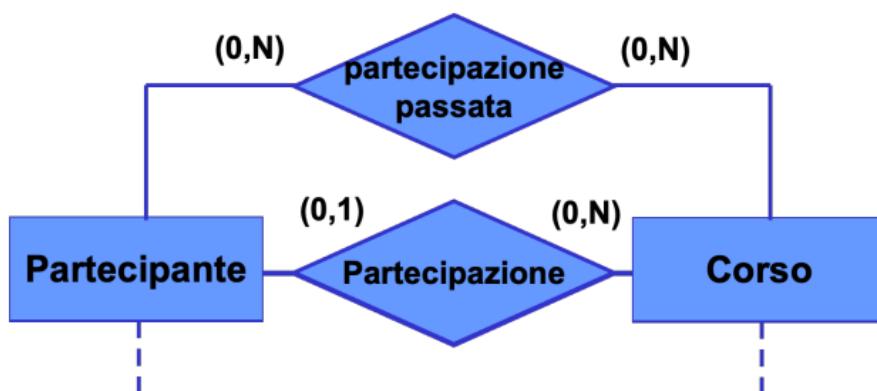
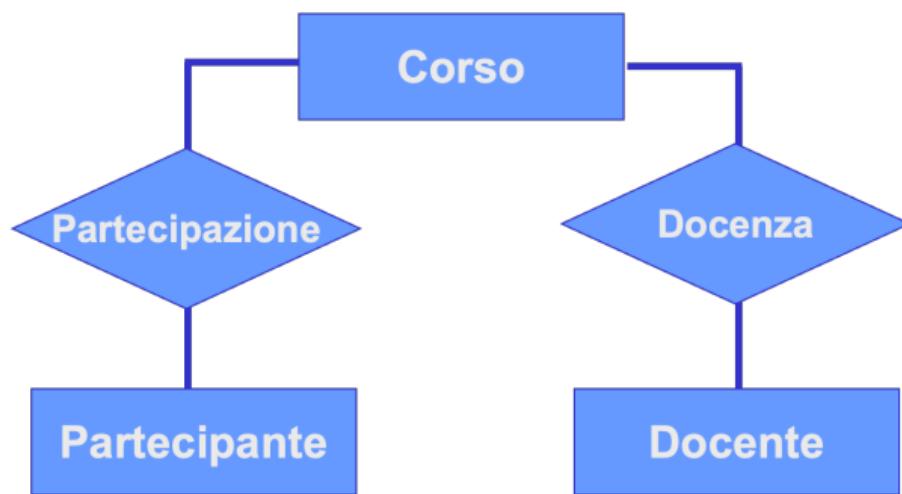


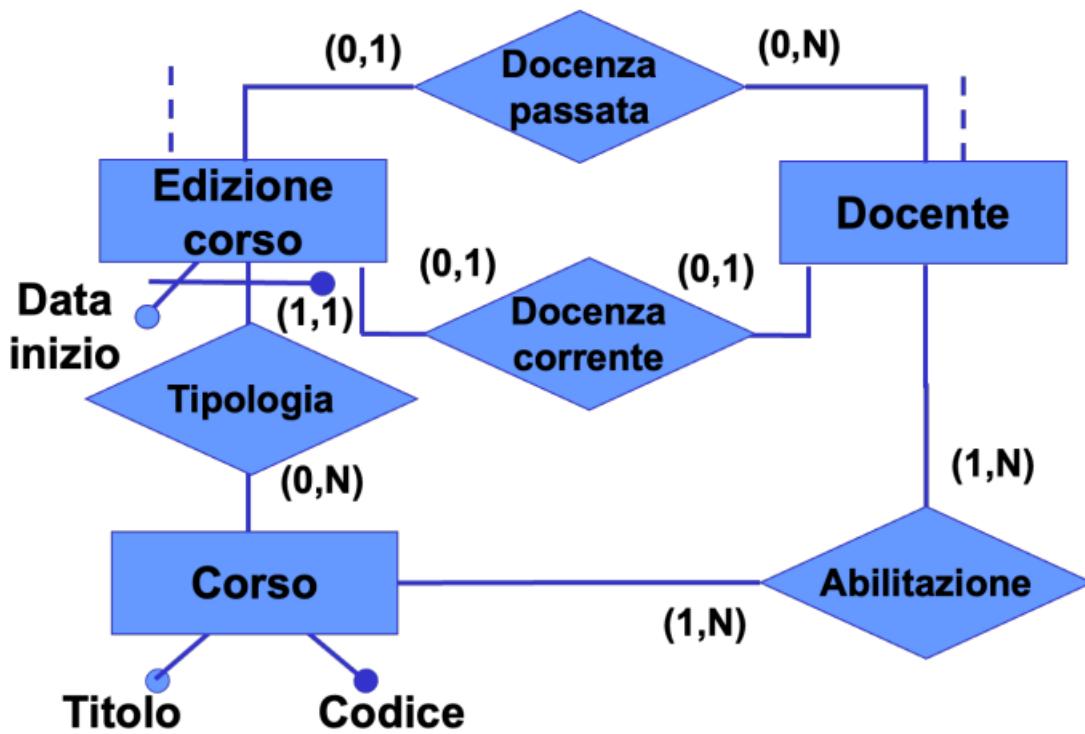
Frasi relative ai docenti

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 nel passato e di quelli che possono insegnare. I docenti possono essere dipendenti interni della società di formazione o collaboratori esterni.



Integrazione





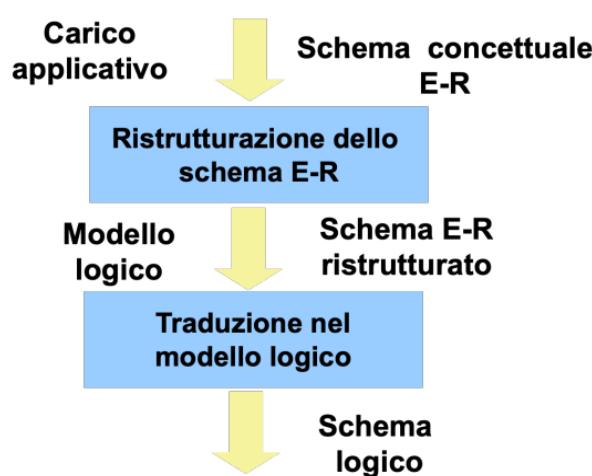
Lezione 06: Progettazione Logica



Obiettivo

L'obiettivo di questa fase è “**Tradurre**” lo **schema concettuale** in uno **schema logico** che rappresenti gli stessi dati in maniera **corretta ed efficiente**.

- **Dati in ingresso:**
 - schema concettuale
 - informazioni sul **carico applicativo**⁴
 - modello logico
- **Dati in uscita:**
 - schema logico
 - documentazione associata



⁴ **Carico applicativo:** inteso come dimensione dei dati

Ristrutturazione dello schema E-R

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

Le motivazioni sono:

- **semplificare** la traduzione
- **ottimizzare** le prestazioni

Uno **schema E-R ristrutturato** non è (più) uno schema concettuale nel **senso stretto** del termine

Indicatori per valutare le prestazioni

Consideriamo degli **indicatori** dei parametri che caratterizzano le prestazioni:

- **spazio**: il numero di occorrenze previste
- **tempo**: il numero di occorrenze (di *entità e relationship*) visitate per portare a termine un'operazione

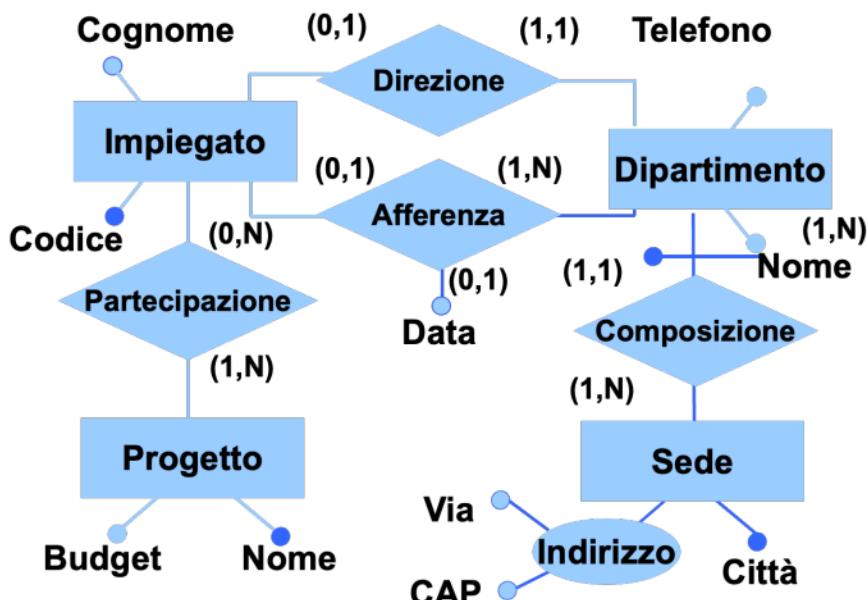


Tavola dei Volumi

Concetto	Tipo	Volume
Sede	E	10
Dipartimento	E	80
Impiegato	E	2000
Progetto	E	500
Composizione	R	80
Afferenza	R	1900
Direzione	R	80
Partecipazione	R	6000

Indicatori per valutare le prestazioni

Operazione: Trova tutti i dati di un impiegato, del dipartimento nel quale lavora e dei progetti ai quali partecipa.

Si costruisce una **tavola degli accessi** basata su uno **schema di navigazione**

Schema di navigazione:

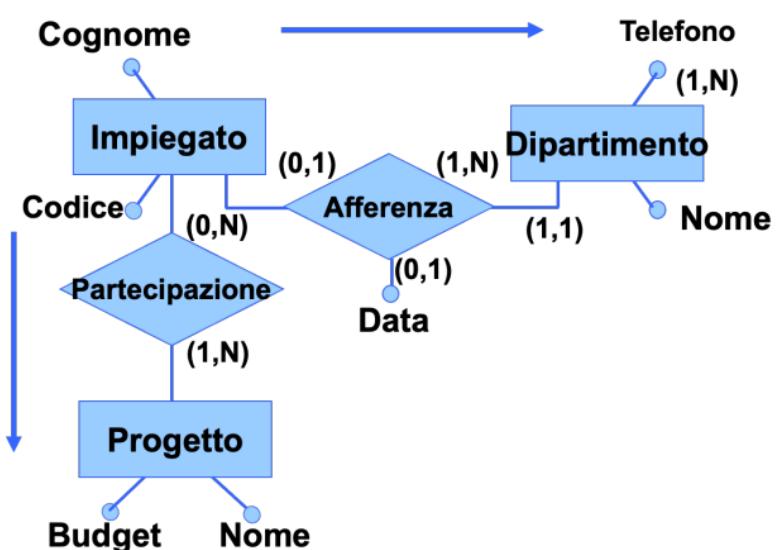


Tavola degli accessi:

Concetto	Costrutto	Accessi	Tipo
Impiegato	Entità	1	L
Afferenza	Relationship	1	L
Dipartimento	Entità	1	L
Partecipazione	Relationship	3	L
Progetto	Entità	3	L

Figura 43: Esempio di Tavola degli Accessi

Accediamo a un'occorrenza di Impiegato per accedere a un'occorrenza dell'associazione Afferenza e attraverso questa a un'occorrenza dell'entità Dipartimento (Solo 1 perché ogni Impiegato afferisce al più a un Dipartimento). Successivamente per conoscere i dati dei Progetti ai quali lavora, dobbiamo accedere a 3 occorrenze dell'associazione Partecipazione e attraverso queste a 3 occorrenze dell'entità Progetto (3 perché in media un impiegato lavora su 3 Progetti). L indica accesso in lettura, S indica accesso in scrittura.
 Distinzione che va fatta perché generalmente le operazioni di Scrittura sono più onerose di quelle in lettura (in quanto devono essere eseguite in modo esclusivo e possono richiedere l'aggiornamento di indici, che sono strutture ausiliarie per l'accesso efficiente ai dati).

Attività di ristrutturazione

- **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 relationship:** si decide se è opportuno partizionare concetti dello schema (entità e/o associazioni) in più concetti o, viceversa, accoppare concetti separati in un unico concetto.
- **Scelta degli identificatori primari:** si seleziona un identificatore per quelle entità che ne hanno più di uno.

1. Analisi delle ridondanze

Una **ridondanza** in uno schema E-R è una informazione significativa ma derivabile da altre.

In questa fase si decide se **eliminare** le ridondanze eventualmente presenti o **mantenerle** (o anche di **introdurne** di nuove).

Vantaggi delle ridondanze:

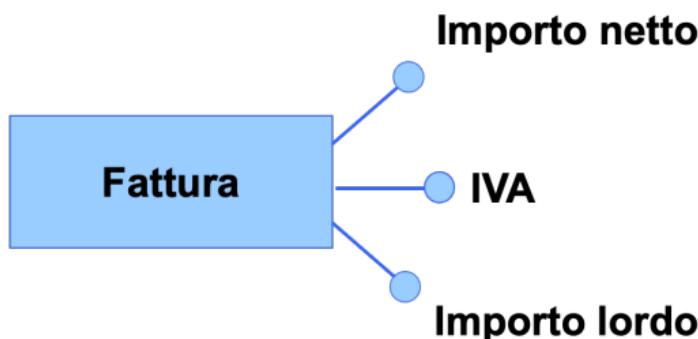
- **semplificazione** delle interrogazioni.

Svantaggi delle ridondanze:

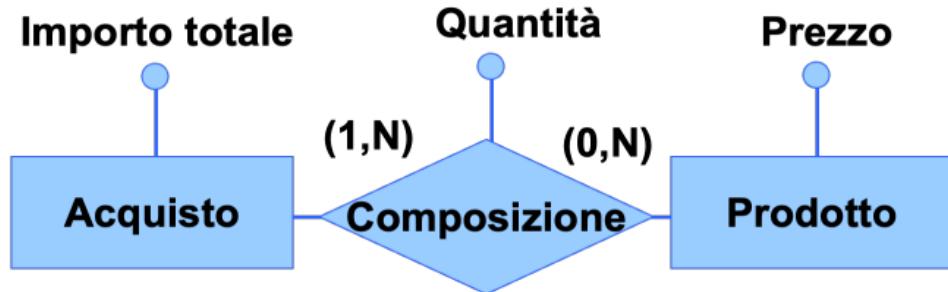
- **appesantimento** degli aggiornamenti
- maggiore occupazione di **spazio**

Forme di ridondanza in uno schema E-R

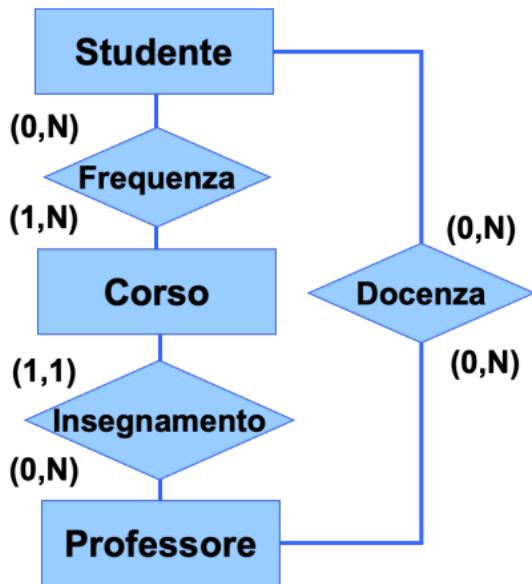
- **Attributi derivabili:**
 - **da altri attributi della stessa entità** (o *relationship*) (occorrenza per occorrenza) un attributo è deducibile dagli altri attraverso un'operazione di somma/differenza.



- **da attributi di altre entità o relationship:** l'importo totale si può derivare dall'attributo Prezzo sommando i prezzi dei prodotti di cui un acquisto è composto.



- **Relationship derivabili dalla composizione di altre** (cicli di *relationship* nello schema qui rappresentato, l'associazione Docenza può essere derivata dalle associazioni Frequenza e Insegnamento. Va precisato che la presenza di cicli non genera necessariamente ridondanze.



- **Attributo derivabile da operazioni di conteggio di occorrenze:** Per esempio nello schema composto da Persona - Residenza Relationship - Città (Numero Abitanti), il numero abitanti si può derivare contando le occorrenze dell'associazione Residenza a cui tale Città partecipa. La decisione di mantenere o eliminare una ridondanza va presa confrontando costo di esecuzione delle operazioni che coinvolgono il dato ridondante e relativa occupazione di memoria,

nei casi di presenza e assenza della **ridondanza**.

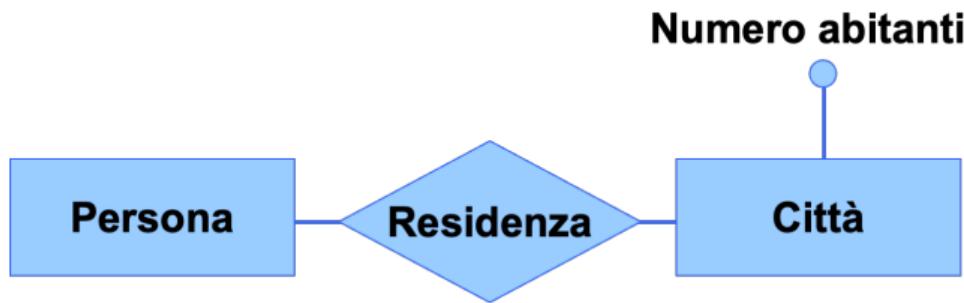


Tavola dei volumi ed operazioni

Concetto	Tipo	Volume
Città	E	200
Persona	E	1000000
Residenza	R	1000000

- **Operazione 1:** memorizza una nuova Persona con la relativa Città di Residenza (500 volte al giorno)
- **Operazione 2:** stampa tutti i dati di una Città (incluso il numero di Abitanti) (2 volte al giorno)

Distinguiamo i casi in presenza e assenza di ridondanza.

Presenza di ridondanza

Operazione 1

Concetto	Costrutto	Accessi	Tipo
Persona	Entità	1	S
Residenza	Relazione	1	S
Città	Entità	1	L
Città	Entità	1	S

Operazione 2

Concetto	Costrutto	Accessi	Tipo
Città	Entità	1	L

Assenza di ridondanza

Operazione 1

Concetto	Costrutto	Accessi	Tipo
Persona	Entità	1	S
Residenza	Relazione	1	S

Operazione 2

Concetto	Costrutto	Accessi	Tipo
Città	Entità	1	L
Residenza	Relazione	5000	L

Costi

Definiamo i costi delle 2 operazioni

- **Presenza** di ridondanza (**Costi**):
 - **Operazione 1:** 1500 accessi in scrittura e 500 accessi in lettura al giorno
 - **Operazione 2:** trascurabile
 - Contiamo doppi gli accessi in scrittura: totale di 3500 accessi al giorno
- **Assenza** di ridondanza (**Costi**):
 - **Operazione 1:** 1000 accessi in scrittura
 - **Operazione 2:** 10000 accessi in lettura al giorno
 - Contiamo doppi gli accessi in scrittura: totale di 12000 accessi al giorno

2. Eliminazione delle generalizzazioni

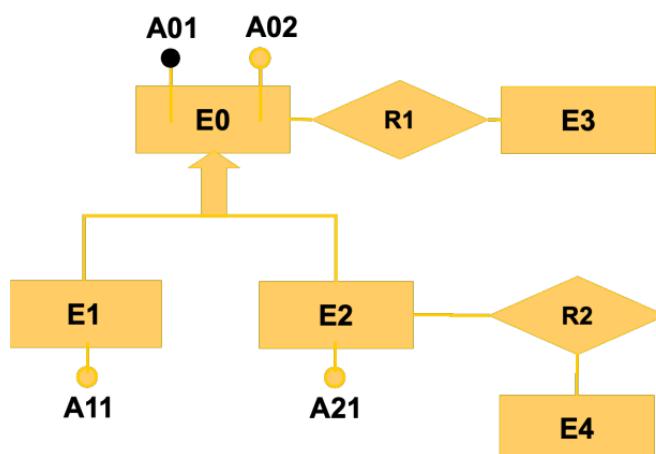
Il modello relazionale non può rappresentare direttamente le **generalizzazioni**.

Entità o **relationship** sono invece direttamente rappresentabili

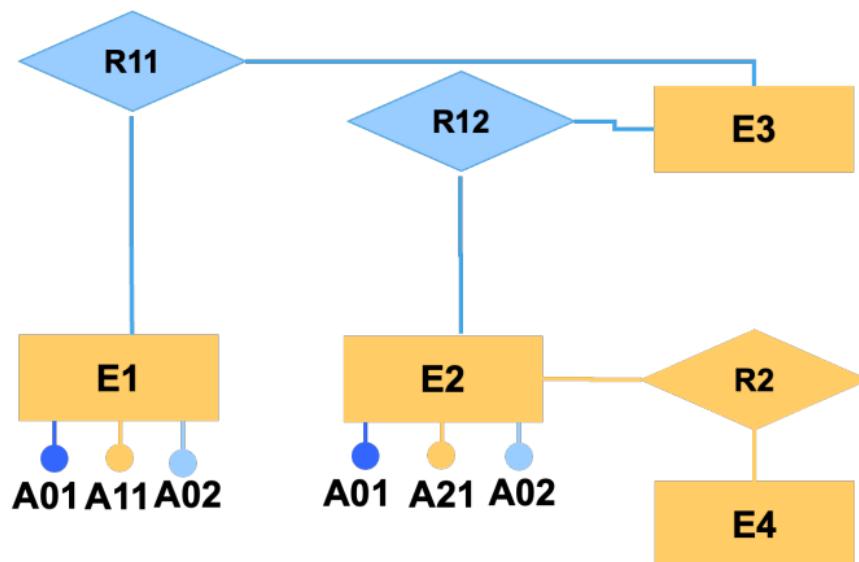
Si **eliminano** perciò le **gerarchie**, sostituendole con *entità* o *relationship*.

Abbiamo diverse possibilità per farlo:

- **Accorpamento delle figlie della generalizzazione nel genitore:** le entità figlie (E_1, E_2) vengono eliminate e le loro proprietà (attributi e partecipazioni a relationship e generalizzazioni) vengono aggiunte all'entità genitore E_0 . A tale entità viene aggiunto un ulteriore attributo che serve a distinguere il Tipo di un'occorrenza, cioè se tale occorrenza apparteneva alle entità figlie (E_1, E_2) o, nel caso di generalizzazione totale, a nessuna di esse. Se per esempio una generalizzazione tra l'entità Persona e le entità Uomo e Donna viene ristrutturata in questo modo, all'entità Persona va aggiunto l'attributo Sesso per mantenere la distinzione tra le occorrenze di tale entità che la generalizzazione originaria rappresentava.

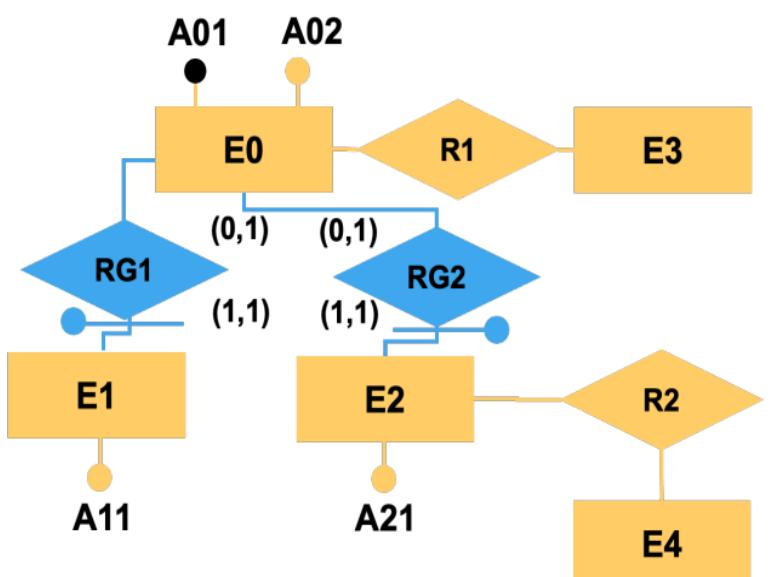


- **Accorpamento del genitore della generalizzazione nelle figlie:** l'entità genitore viene eliminata e, per la proprietà dell'ereditarietà, i suoi attributi, il suo identificatore e le relazioni a cui tale entità partecipava, vengono aggiunti alle entità figlie. Le associazioni R_{11} e R_{12} rappresentano rispettivamente la restrizione dell'associazione R_1 sulle occorrenze dell'entità E_1 ed E_2 . Se per esempio una generalizzazione tra l'entità Persona, avente Cognome ed Età come attributi e Codice Fiscale come identificatore, e le entità Uomo e Donna viene ristrutturata in questo modo, alle entità Uomo e Donna vanno aggiunti gli attributi Cognome ed Età e l'identificatore Codice Fiscale.



- **Sostituzione della generalizzazione con relationship:** la generalizzazione si trasforma in due associazioni uno a uno che legano rispettivamente l'entità genitore alle entità figlie. Non ci sono trasferimenti di attributi o associazioni e le entità figlie sono identificate esternamente dall'entità genitore.

Nello schema ottenuto vanno aggiunti dei vincoli, ogni occorrenza di E_0 non può partecipare contemporaneamente a R_{G1} e R_{G2} e inoltre se la generalizzazione è totale, ogni occorrenza di E_0 deve partecipare o a un'occorrenza di R_{G1} o a un'occorrenza di R_{G2}



Come scegliere?

La scelta fra le varie alternative si può fare **basandosi sul numero e tipo degli accessi** fatti alle singole entità per eseguire le operazioni.

È possibile seguire alcune semplici regole generali:

1. **la prima** conviene se gli accessi al padre ed alle figlie sono contestuali.
2. **la seconda** conviene se gli accessi alle figlie sono distinti.
3. **la terza conviene** se gli accessi alle entità figlie sono separati dagli accessi al padre.
4. ci sono anche possibili soluzioni "**ibride**", soprattutto in gerarchie a più livelli.

3. Partizionamento/accorpamento di entità di relationship

Entità e relationship in uno schema E-R possono essere partizionati o accorpati per garantire una maggior efficienza delle operazioni in base al seguente principio: gli accessi si riducono separando attributi di uno stesso concetto che vengono acceduti da operazioni diverse e raggruppando attributi di concetti diversi che vengono acceduti dalle medesime operazioni.

Si considera sempre che ad ogni accesso si legge l'intera informazione.

Qual è la motivazione?

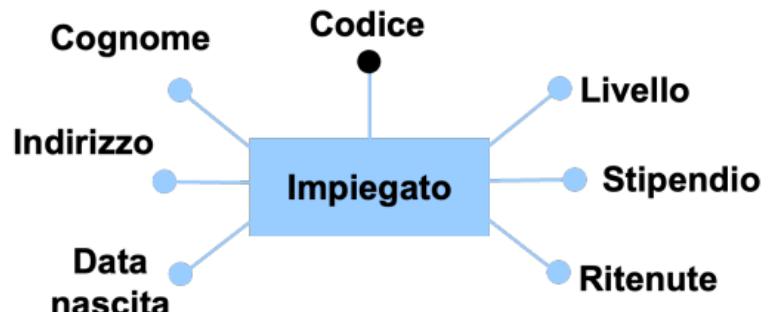
Ristrutturazioni sono effettuate per rendere **più efficienti** le operazioni in base al principio che gli *accessi si riducono*. Come?

- **separando attributi** di un concetto che vengono acceduti separatamente
- **raggruppando attributi** di concetti diversi acceduti insieme

Si considera sempre che ad ogni accesso si **legge l'intera informazione**

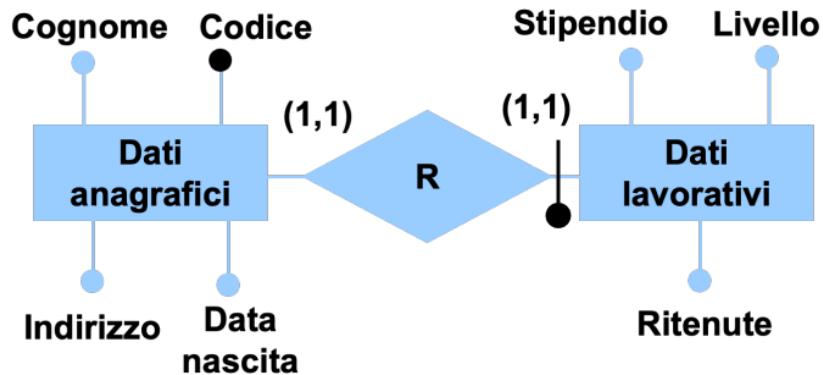
Casi principali

- **Partizionamento verticale di entità:** per esempio l'entità **Impiegato** (Codice, Cognome, Indirizzo, Data nascita, Livello, Stipendio, Ritenute) viene sostituita da 2 entità, collegate da un'associazione uno a uno, che descrivono rispettivamente i Dati Anagrafici (Codice, Cognome, Indirizzo, Data nascita) degli impiegati e i dati relativi alla loro retribuzione, Dati Lavorativi (Livello, Stipendio, Ritenute).



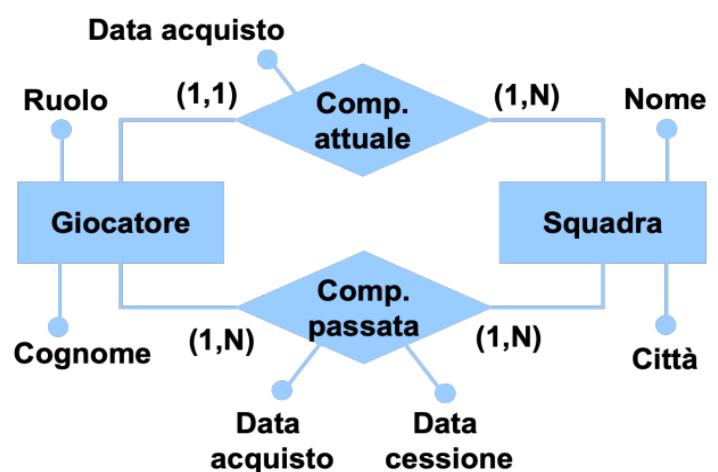
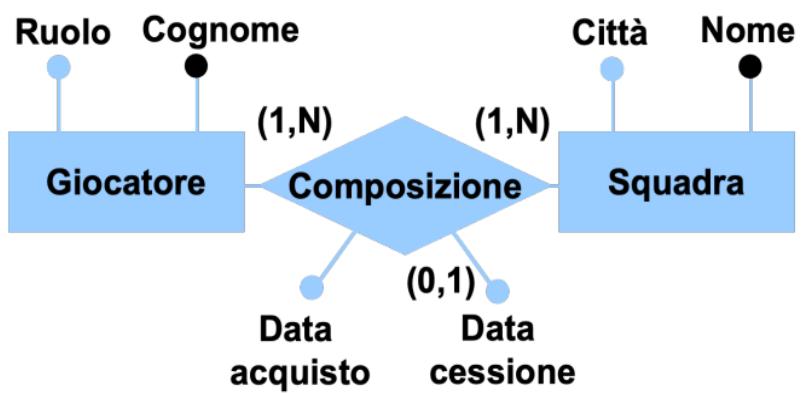
La Ristrutturazione è conveniente se le operazioni che coinvolgono frequentemente l'entità originaria richiedono o solo informazioni anagrafiche o solo informazioni relative alla sua retribuzione.

Si dice "**decomposizione verticale**", nel senso che suddivide il concetto operando sui suoi attributi.

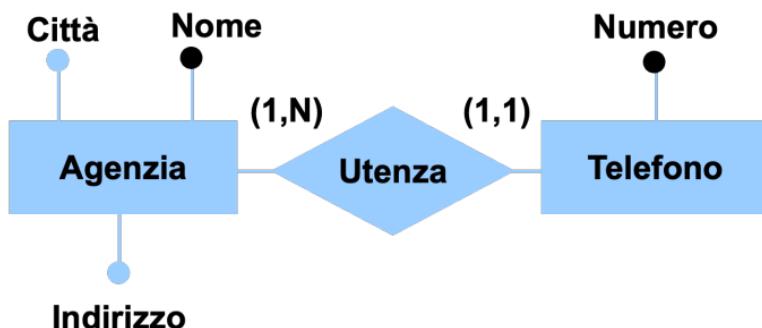
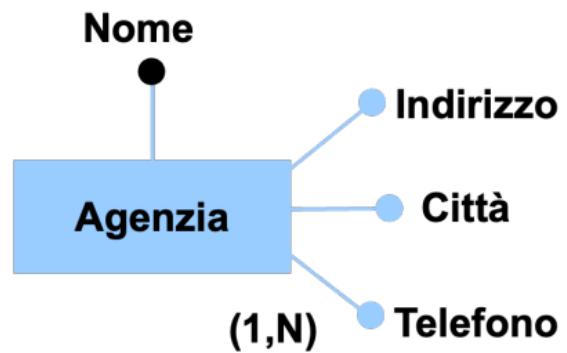


- **Partizionamento orizzontale di relationship:** può convenire in alcuni casi decomporre un'associazione tra due entità in due (o più) associazioni tra le medesime entità, per separare occorrenze dell'associazione originale accedute sempre separatamente.

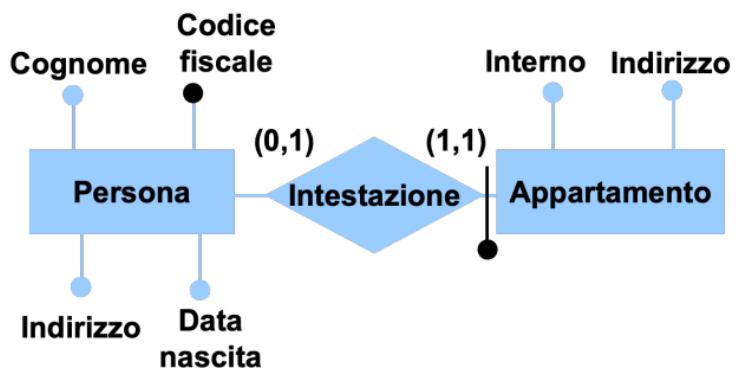
Un esempio di partizionamento di associazioni deriva dallo schema composto da: Giocatore(Ruolo, Cognome) - Composizione Relationship (Data acquisto, Data cessione) - Squadra(Città, Nome), nella quale distinguiamo i giocatori che compongono attualmente una squadra da quelli che ne facevano parte. Lo schema qui sotto rappresenta lo schema partizionato.



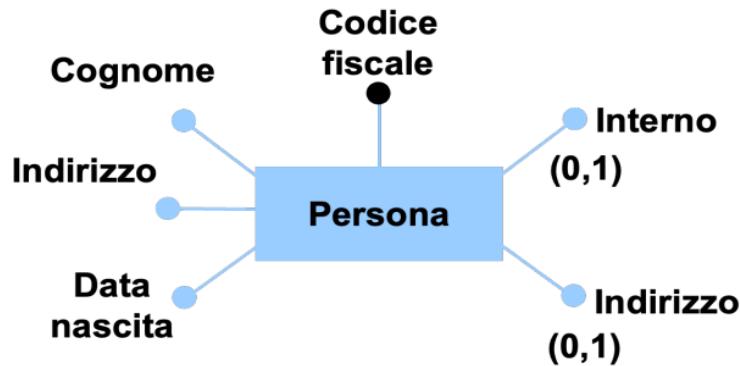
- **Eliminazione di attributi multi-valore:** un particolare tipo di partizionamento. Questa ristrutturazione si rende necessaria perché, come per le generalizzazioni, il modello relazionale non permette di rappresentare in maniera diretta questo tipo di attributo. Per esempio l'entità Agenzia (Nome, Indirizzo, Città, (1,N)Telefono) viene partizionata in due entità: un'entità con lo stesso nome e gli stessi attributi dell'entità originale eccetto l'attributo multivalore; e l'entità Telefono, con il solo attributo Numero, legata mediante associazione uno a molti con l'entità Agenzia.



- **Accorpamento di entità/relationship:** è l'operazione inversa del partizionamento. Un esempio di accorpamento di entità avviene nello schema composto da Persona (Codice Fiscale, Cognome, Indirizzo, Data N)(0,1) - Intestazione REL - (1,1) Appartamento (Interno, Indirizzo), nella quale le entità Persona e Appartamento, legate dall'associazione uno a uno Intestazione, vengono accorpate in un'unica entità contenente gli attributi di entrambi. Questa ristrutturazione può essere suggerita dal fatto che le operazioni più frequenti sull'entità Persona richiedono sempre i dati relativi all'appartamento che occupa, e vogliamo quindi risparmiare gli accessi necessari per risalire a questi dati attraverso l'associazione che li lega. Un effetto collaterale di questa ristrutturazione è la possibile presenza di valori nulli dovuta al fatto che le cardinalità ci dicono che



ci sono persone che non sono intestatarie di nessun appartamento, e quindi non esistono per esse valori per gli attributi Indirizzo e Interno.



4. Scelta degli identificatori primari

Operazione indispensabile per la traduzione nel modello relazionale. E' essenziale nelle traduzioni verso il modello relazionale perché in questo modello le chiavi vengono usate per stabilire legami tra dati in relazioni diverse. Inoltre, i sistemi di gestione di basi di dati richiedono generalmente di specificare una chiave primaria sulla quale vengono costruite automaticamente delle strutture ausiliarie, dette **indici**, per il reperimento efficiente dei dati.

Quindi nei casi in cui esistono entità per le quali sono stati specificati più identificatori, bisogna decidere quale di questi verrà utilizzato come chiave primaria.

Criteri di decisione:

- Gli **attributi con valori nulli** non possono costituire identificatori principali. Tali attributi non garantiscono l'accesso a tutte le occorrenze dell'entità corrispondente.
- Un **identificatore composto** da uno o pochi attributi è da preferire a identificatori costituiti da molti attributi. Questo garantisce che le strutture ausiliarie create per accedere ai dati (**indici**) siano di dimensioni ridotte, permette un risparmio di memoria nella realizzazione dei legami logici tra le varie relazioni e facilita le operazioni di join.
- Un **identificatore interno** con pochi attributi è da preferire a un identificatore esterno che magari coinvolge diverse entità.
- Un **identificatore** che viene utilizzato da molte operazioni per accedere alle occorrenze di un'entità è da preferire rispetto agli altri. In questa maniera infatti tali operazioni possono essere eseguite efficientemente perché possono trarre vantaggio dagli indici creati automaticamente dal DBMS. Se nessuno degli identificatori candidati soddisfa tali requisiti è possibile pensare

di introdurre un ulteriore attributo all'entità che conterrà valori speciali (codici) generati appositamente per identificare le occorrenze delle entità

Se nessuno degli identificatori soddisfa i requisiti visti?

Si introducono nuovi attributi (**codici**) contenenti valori speciali generati appositamente per questo scopo.

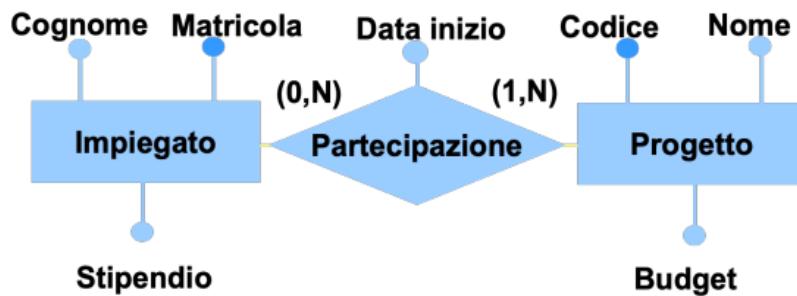
Traduzione verso il modello relazionale

La seconda fase della progettazione logica corrisponde a una traduzione tra modelli di dati diversi: a partire da uno schema E-R ristrutturato si costruisce uno schema logico equivalente, in grado cioè di rappresentare le medesime informazioni. L'idea di base:

- Le **entità diventano relazioni sugli stessi attributi**: una relazione con lo stesso nome avente per attributi i medesimi attributi dell'entità e per chiave il suo identificatore
- Le **relationship diventano relazioni**: una relazione con lo stesso nome avente per attributi gli attributi dell'associazione e gli identificatori delle entità coinvolte. Tali identificatori formano la chiave della relazione.

Entità a relationship molti a molti

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



Impiegato(Matricola, Cognome, Stipendio)

Progetto(Codice, Nome, Budget)

Partecipazione(Matricola, Codice, DataInizio)

Vincoli di integrità referenziale fra:

Matricola in **Partecipazione** e (la chiave di) **Impiegato**

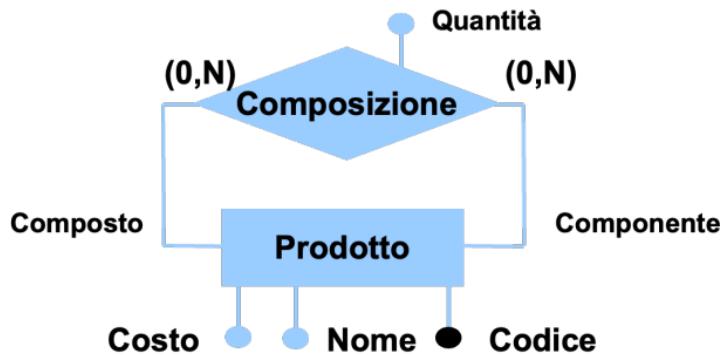
Codice in **Partecipazione** e (la chiave di) **Progetto**

Per rendere più comprensibile il significato dello schema è conveniente effettuare alcune ridenominazioni.

Per esempio Partecipazione(Impiegato, Progetto, DataInizio) nella quale il dominio di Impiegato è un insieme di matricole di Impiegati e quello dell'attributo Progetto è un insieme di codici di progetti.

Relationship Ricorsive

La ridenominazione è necessaria nel caso di associazioni ricorsive.



Questo schema si traduce in:

Prodotto (Codice, Nome Costo)

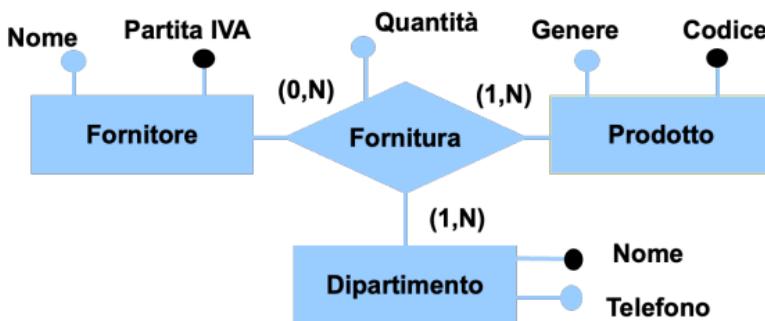
Composizione (Composto, Componente, Quantità)

In questo schema entrambi gli attributi Composto e Componente contengono codici di prodotti.

Esiste un vincolo di integrità referenziale fra questi attributi e l'attributo Codice della relazione Prodotto.

Relationship n-arie

Le associazioni con più entità partecipanti si traducono in maniere analoghe a quanto detto per le associazioni binarie.



Questo schema si traduce in:

Fornitore(PartitaIVA, Nome)

Prodotto(Codice, Genere)

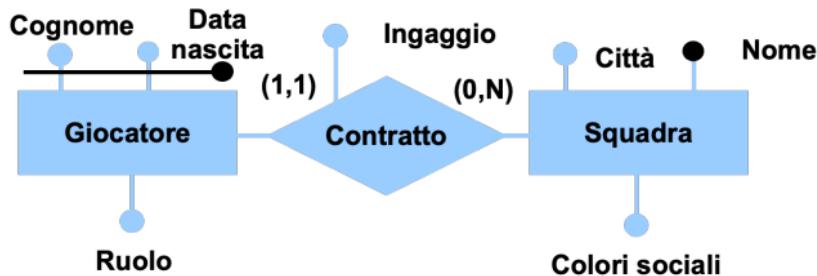
Dipartimento(Nome, Telefono)

Fornitura(Fornitore, Prodotto, Dipartimento, Quantità)

Esistono vincoli di integrità referenziale tra gli attributi Fornitore, Prodotto e Dipartimento della relazione Fornitura e, rispettivamente, l'attributo PartitaIVA di Fornitore, l'attributo Codice di Prodotto e l'attributo Nome di Dipartimento.

Relationship uno a molti

Secondo la regola vista per le associazioni molti a molti la traduzione dello schema dovrebbe essere:



Giocatore(Cognome, DataNascita, Ruolo)

Contratto(CognomeGiocatore, DataNascitaGiocatore, Squadra, Ingaggio)

Squadra(Nome, Città, ColoriSociali)

Va notato però che nella relazione Contratto, la chiave è costituita solo dall'identificatore di Giocatore perché le cardinalità dell'associazione ci dicono che ogni giocatore ha un contratto con una sola squadra. A questo punto le relazioni Giocatore e Contratto hanno la stessa chiave ed è allora possibile fonderle in un'unica relazione (perché esiste una corrispondenza biunivoca tra le rispettive occorrenze). Quindi è preferibile la traduzione che segue, nella quale la relazione Giocatore rappresenta sia l'entità relativa sia l'associazione dello schema E-R originale.

Giocatore(Cognome, DataNascita, Ruolo, Squadra, Ingaggio)

Squadra(Nome, Città, ColoriSociali)

Con vincolo di integrità referenziale fra Squadra in Giocatore e (la chiave di) Squadra.

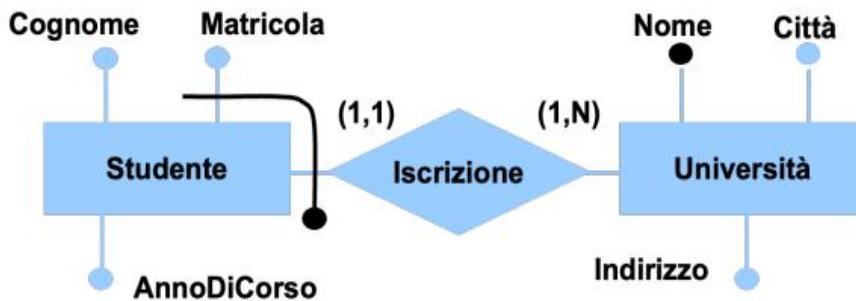
Se la cardinalità minima della relationship è 0, allora Squadra in Giocatore deve ammettere valore nullo.

La traduzione riesce a rappresentare efficacemente la cardinalità minima della partecipazione che ha 1 come cardinalità massima e poi:

- 0 valore nullo ammesso
- 1 valore nullo non ammesso

Entità con identificazione esterna

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



Lo schema relazionale corrispondente a questo schema sopra è:

Studente (Matricola, NomeUniversità, Cognome, AnnoDiCorso)

Università (Nome, Città, Indirizzo)

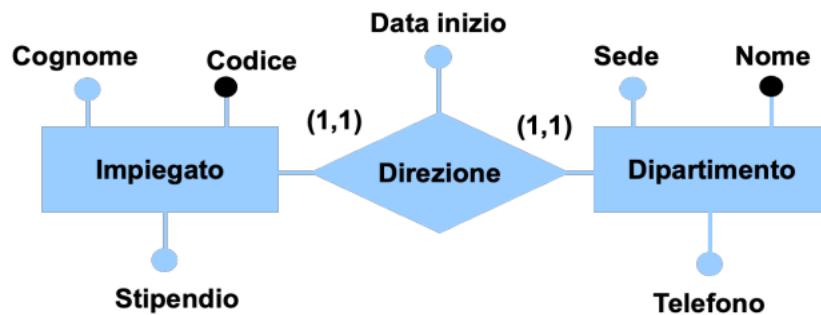
Con **vincolo di integrità referenziale** tra l'attributo NomeUniversità della relazione **Studente** e l'attributo Nome della relazione **Università**.

Come si può vedere rappresentando l'identificatore esterno si rappresenta direttamente anche l'associazione tra le due entità.

Ricordiamo infatti che le entità identificate *esternamente* partecipano all'associazione sempre con una cardinalità minima e massima pari a 1.

Relationship uno ad uno

Abbiamo diverse possibilità, cominciamo a vedere le associazioni uno a uno con partecipazioni obbligatorie per entrambe le entità.



Abbiamo due possibilità simmetriche di traduzione:

1. **Impiegato**(Codice, Cognome, Stipendio, DataInizio, DipartimentoDiretto)
Dipartimento(Nome, Sede, Telefono)
per il quale esiste il **vincolo** tra l'attributo DipartimentoDiretto della relazione **Impiegato** e l'attributo Nome della relazione **Dipartimento**.
2. **Impiegato**(Codice, Cognome, Stipendio)
Dipartimento(Nome, Telefono, Sede, Direttore, InizioDirezione)

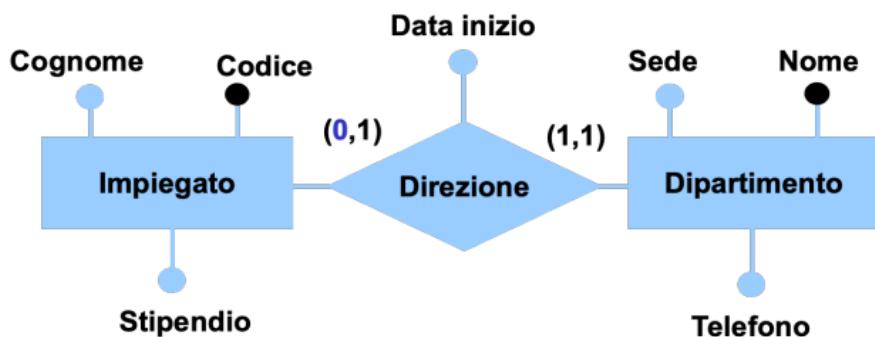
per il quale esiste il **vincolo** tra l'attributo Direttore della relazione Dipartimento e l'attributo Codice della relazione Impiegato.

Trattandosi di una relazione biunivoca tra le occorrenze delle entità sembrerebbe possibile un'ulteriore alternativa nella quale si rappresentano tutti i concetti in un'unica relazione contenente tutti gli attributi in gioco.

Questa alternativa è però da escludere perché non dobbiamo dimenticarci che lo schema che stiamo traducendo è il risultato di una fase di ristrutturazione nella quale sono state effettuate precise scelte anche riguardo l'accorpamento e il partizionamento di entità. Questo significa che, se nello schema E-R ristrutturato abbiamo due entità collegate da un'associazione uno a uno, vuol dire che abbiamo ritenuto conveniente tenere separati i due concetti ed è quindi inopportuno fonderli in sede di traduzione verso il modello relazionale.

Un caso privilegiato

Consideriamo ora il caso di associazione uno a uno con partecipazione opzionale per una sola entità.



In questo caso abbiamo una soluzione preferibile alle altre:

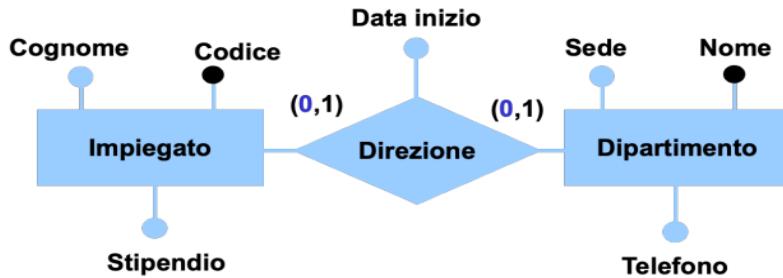
Impiegato(Codice, Cognome, Stipendio)

Dipartimento(Nome, Telefono, Sede, Direttore, InizioDirezione)

Esiste il **vincolo** tra l'attributo Direttore della relazione Dipartimento e l'attributo Codice della relazione Impiegato. Questa alternativa è preferibile rispetto a quella in cui l'associazione viene rappresentata nella relazione Impiegato mediante il nome del dipartimento diretto perché avremmo, per questo caso, possibili valori nulli.

Un altro caso

Consideriamo anche il caso in cui entrambe le entità hanno partecipazione opzionale.



In questo caso esiste un'ulteriore possibilità che prevede tre relazioni separate:

Impiegato(Codice, Cognome, Stipendio)

Dipartimento(Nome, Telefono, Sede)

Direzione(Direttore, Dipartimento, DataInizioDirezione)

Esistono i **vincoli** tra l'attributo Direttore della relazione Direzione e l'attributo Codice della relazione Impiegato e tra l'attributo Dipartimento della relazione Direzione e l'attributo Nome della relazione Dipartimento.

Questa soluzione ha il vantaggio di non presentare mai valori nulli sugli attributi che rappresentano l'associazione.

Per contro, abbiamo bisogno di una relazione in più con un conseguente aumento della complessità della base di dati. Quindi la soluzione con tre relazioni è da prendere in considerazione solo se il numero di occorrenze è molto basso rispetto alle occorrenze delle entità che partecipano all'associazione.

Schema Finale

Impiegato(Codice, Cognome, Dipartimento, Sede, Data)

Dipartimento(Nome, Città, Telefono, Direttore)

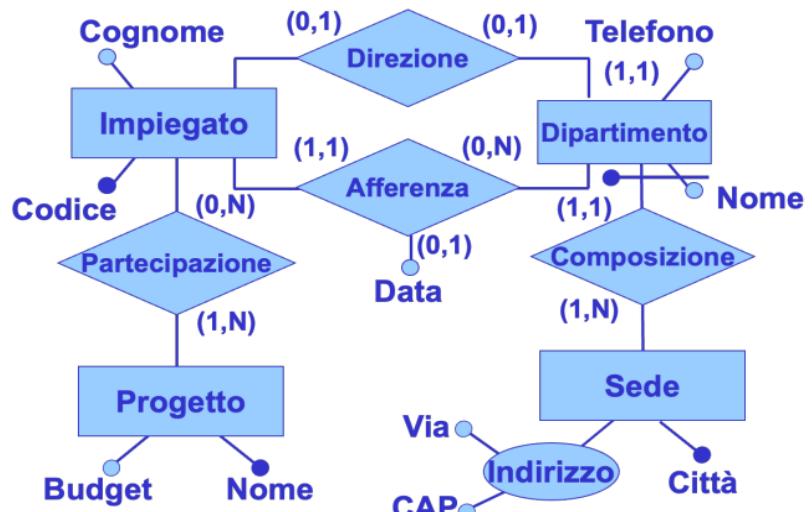
Sede(Città, Via, CAP)

Progetto(Nome, Budget)

Partecipazione(Impiegato,

Progetto)

ATTENZIONE: differenze apparentemente piccole in cardinalità e identificatori possono cambiare di molto il significato



Lezione 07: Dipendenze Funzionali

Progettazione

Abbiamo ipotizzato che gli attributi vengano raggruppati per formare uno schema di relazione **usando il buon senso** del progettista di basi di dati o **traducendo** uno schema di base di dati da un modello di dati concettuale (E-R) , presumibilmente ben fatta.

Ma abbiamo bisogno di **misurare formalmente** se un dato di raggruppamento di attributi è migliore di un altro.

L'**obiettivo** è di valutare la qualità della progettazione degli schemi relazionali

Approccio seguito

- **Top down:** abbiamo iniziato individuando un certo numero di raggruppamenti di attributi per formare relazioni che sussistono come tali nel mondo reale, ad esempio una fattura, un form o un report.

Queste relazioni sono poi state analizzate portando eventualmente a **decomposizioni successive**.

Obiettivi impliciti del progetto logico

- La **conservazione dell'informazione**, cioè il mantenimento di tutti i concetti espressi precedentemente mediante il modello concettuale, inclusi i tipi di attributi, di entità e di associazioni.
- Le **minimizzazioni della ridondanza**, quindi ridurre al meno la ripetizione degli elementi nella base di dati.

Possiamo derivare da questi obiettivi alcune linee guida per il progetto.

Linea guida 1: semplice e bello

- Uno schema di relazione deve essere progettato in modo che sia **semplice spiegarne il significato**.
- Non si devono raggruppare attributi provenienti da più tipi di entità e tipi di relazione in un'unica relazione.
- Intuitivamente, se uno schema di relazione corrisponde ad **un solo tipo di entità o a un solo tipo di relationship**, risulta **semplice spiegarne il significato**.
- In caso contrario, nascerà un'**ambiguità semantica** e quindi lo schema non potrà essere spiegato con facilità.

Linea guida 2: no alle anomalie⁵

- Gli schemi vanno progettati in modo che **non possano presentarsi anomalie** di inserimento, modifica o cancellazione.
- La **mancanza** di anomalie va **certificata**, usando una **descrizione formale della semantica** dei fatti descritti in uno schema relazionale.
- Se possono presentarsi anomalie, vanno chiaramente **rilevate** e si deve assicurare che i programmi che aggiornano la base di dati operino **correttamente**.

Esempio 1

Fattura(CodFatt, CodProd, TotDaPagare, CostoNettoProd, IVA)

Semantica attributi:

CodFatt determina CodProd e TotDaPagare

CodProd determina CostoNettoProd e IVA

CostoNettoProd e IVA determinano TotDaPagare

Spiegazione:

- TotDaPagare deve essere consistente con la regola che lo lega al CostoNettoProd ed all'IVA.
- Inoltre, a CodProd deve essere attribuita la giusta percentuale di IVA.
- Questo secondo legame è esterno al DB, se cambia la legge IVA di un certo prodotto, questo attributo deve essere modificato; però la sua modifica si

⁵ **Anomalie:** in questo caso è tutto ciò che non ci piace

posta dietro un'altra modifica dell'attributo TotDaPagare il cui significato è interno al DB ma è legato ad IVA.

- Per evitare anomalie di inserimento o modifica conviene che TotDaPagare non ci sia nella tabella Fattura.

Esempio 2

Anagrafe(CF, NomePersona, ViaRes, NomeCittaRes, NumAb),
semantica attributi:

CF determina NomePersona, ViaRes e NomeCittaRes

NomeCittaRes determina NumAb.

Spiegazione:

- NumAb è ripetuto per lo stesso NomeCittaRes per quanti sono i residenti.
- Il valore deve essere mantenuto consistente (uguale) per ogni persona di una stessa città.
- Come si può evitare il problema?
 - Trasformando Anagrafe in due schemi separati:
 - Persona(CF, NomePersona, ViaRes, NomeCittaRes)
 - ListaComuni(NomeCitta, NumAb)
 - Con vincolo di integrità referenziale su NomeCittaRes verso NomeCitta e un vincolo aggiuntivo su NumAb

Linea Guida 3: evitare frequenti valori null

Si eviti di porre in una relazione attributi i cui valori possono essere frequentemente nulli. Se i valori nulli sono inevitabili, ci si assicuri che si presentino solo in casi eccezionali rispetto al numero di n-uple di una relazione.

Dipendenza Funzionale

Una **dipendenza funzionale (functional dependency FD)** esprime un **legame semantico** tra **due gruppi di attributi** presenti uno schema di relazione R.

Una dipendenza funzionale è una **proprietà di una relazione**.

- Una *FD* è una proprietà di R, non di una particolare stato valido r di R.
- Una *FD* non può essere dedotta a partire da uno stato valido r, ma deve essere definita esplicitamente da qualcuno che conosce la semantica degli attributi di R.

Forme Normali

Una **forma normale** è una **proprietà** di una base di dati relazionale che ne **garantisce la qualità**, ovvero **sull'assenza di determinati difetti**. Quando una relazione non è normalizzata:

- Possono essere presenti delle ridondanze
- Si presta a comportamenti poco desiderabili durante gli aggiornamenti.

Le forme normali sono di solito definiti sul **modello relazionale**, ma hanno senso in altri contesti.

Esempio: il modello E-R

Normalizzazione

La normalizzazione è una procedura che permette di trasformare schemi non normalizzati in schemi che soddisfano una forma normale.

La normalizzazione va usata come **tecnica di verifica** dei risultati della progettazione di una base di dati.

Non costituisce una metodologia di progettazione.

Le anomalie

<u>Impiegato</u>	<u>Stipendio</u>	<u>Progetto</u>	<u>Bilancio</u>	<u>Funzione</u>
Rossi	20	Marte	2	tecnico
Verdi	35	Giove	15	progettista
Verdi	35	Venere	15	progettista
Neri	55	Venere	15	direttore
Neri	55	Giove	15	consulente
Neri	55	Marte	2	consulente
Mori	48	Marte	2	direttore
Mori	48	Venere	15	progettista
Bianchi	48	Venere	15	progettista
Bianchi	48	Giove	15	direttore

Figura 44: Esempio contenente delle anomalie

Ridondanza

Lo stipendio di ciascun impiegato è ripetuto in tutte le *n-uple* relative, quindi si ha **ridondanza**.

- **anomalia di aggiornamento:** se lo Stipendio di un Impiegato varia, è necessario andarne a modificare il valore in diverse *n-uple*;
- **anomalia di cancellazione:** se un Impiegato interrompe la partecipazione a tutti i Progetti, dobbiamo cancellarlo;
- **anomalia di inserimento:** un nuovo Impiegato senza Progetto non può essere inserito.

Causa dei problemi

Abbiamo usato **un'unica relazione** per rappresentare **informazioni eterogenee**

- Gli impiegati con i relativi stipendi
- I progetti con i relativi bilanci
- Le partecipazioni degli impiegati ai progetti con le relative funzioni

Ora useremo il concetto di **dipendenze funzionali** per studiare meglio questi problemi di anomalie.

Definizione di Dipendenza Funzionale

Siano dati:

una relazione r su $R(X)$

due sottoinsiemi **non vuoti** Y e Z di X .

Esiste in r una **dipendenza funzionale** da Y a Z se, per ogni coppia di n -uple t_1 e t_2 di r con gli stessi valori su Y , risulta che t_1 e t_2 hanno gli stessi valori su Z .

Notazione: $Y \rightarrow Z$ (l'uguaglianza dei valori degli attributi in Y forza quelli in Z)

NOTA BENE: Se $Y \rightarrow Z$ non è detto che esista $Z \rightarrow Y$

Dipendenze Funzionali Particolari

Una dipendenza funzionale è **completa** quando $Y \rightarrow Z$ e, per ogni W incluso in Y , non vale $W \rightarrow Z$.

- Se Y è una **superchiave** di $R(X)$ allora Y determina ogni altro attributo della relazione $Y \rightarrow X$
- Se Y è una **chiave**, allora $Y \rightarrow X$ è una dipendenza funzionale completa

Una dipendenza funzionale è **banale** se è sempre soddisfatta:

- $Y \rightarrow Y$ è banale
- $Y \rightarrow A$ è non banale se A non appartiene Y
- $Y \rightarrow Z$ è non banale se nessun attributo di Z appartiene ad Y

Esempio 1:

Caratterizziamo in termini di dipendenza le informazioni semantiche che abbiamo:

- Ogni impiegato ha un solo stipendio: Impiegato \rightarrow Stipendio
- Ogni progetto ha un solo bilancio: Progetto \rightarrow Bilancio
- Ogni Impiegato ha una sola funzione per progetto: Impiegato, Progetto \rightarrow Funzione

Impiegato	Stipendio	Progetto	Bilancio	Funzione
Rossi	20	Marte	2	tecnico
Verdi	35	Giove	15	progettista
Verdi	35	Venere	15	progettista
Neri	55	Venere	15	direttore
Neri	55	Giove	15	consulente
Neri	55	Marte	2	consulente
Mori	48	Marte	2	direttore
Mori	48	Venere	15	progettista
Bianchi	48	Venere	15	progettista
Bianchi	48	Giove	15	direttore

Legami tra dipendenze funzionali ed anomalie

Impiegato → Stipendio. Ci sono ripetizioni

Progetto → Bilancio. Ci sono ripetizioni

Impiegato, Progetto → Funzione. Non ci sono ripetizioni

Impiegato non è una chiave, Progetto non è una chiave.

Impiegato, Progetto è una chiave.

Le **dipendenze funzionali** sono usate per **verificare l'eventuale presenza di anomalie** in un progetto. Vedremo che sono usate anche per *normalizzare* uno schema.

Data la loro importanza, quando necessario indicheremo con $R(X,F)$ uno schema di relazione $R(X)$ che **verifica un insieme di dipendenza funzionali** F .

Implicazione

Sia F un insieme di dipendenze funzionali definite su $R(Z)$ e sia $X \rightarrow Y$.

Si dice che F **implica (logicamente)** $X \rightarrow Y$, in simboli $F \models X \rightarrow Y$, se **per ogni possibile istanza** r di R che verifica tutte le dipendenze funzionali in F , risulta verificata **anche** la dipendenza funzionale $X \rightarrow Y$

Si dice anche che $X \rightarrow Y$ è **implicata logicamente da** F .

Esempio:

$R(\text{Impiegato}, \text{Categoria}, \text{Stipendio})$.

Le **dipendenze funzionali**:

Impiegato → Categoria e Categoria → Stipendio implicano la dipendenza funzionale Impiegato → Stipendio.

Problema

La definizione di implicazione **non è direttamente utilizzabile** nella pratica.

- Essa prevede una **quantificazione universale** sulle istanze della base di dati (“per ogni istanza”)
- Non **abbiamo un algoritmo** per calcolare tutte le dipendenze funzionali che sono implicate da un dato insieme F .

Armstrong (1974) ha fornito delle **regole di inferenza** che permettono di **derivare costruttivamente tutte** le dipendenze funzionali che sono implicate (non logicamente) da un dato insieme iniziale.

Regole di inferenza di Armstrong

1. **Riflessività** Se $Y \subseteq X$ allora $X \rightarrow Y$
2. **Additività (arricchimento)** Se $X \rightarrow Y$ allora $XZ \rightarrow YZ$ per qualunque Z
3. **Transitività** Se $X \rightarrow Y$ e $Y \rightarrow Z$ allora $X \rightarrow Z$

Derivazione

Siano dati:

- Un **insieme di regole di inferenza RI**
- Un **insieme di dipendenze funzionali F**
- Una **dipendenza funzionale f**

Una **derivazione di f da F secondo RI** è una **sequenza finita** f_1, \dots, f_m dove

- $f_m = f$
- ogni f_i è un elemento di F oppure è ottenuta dalle precedenti dipendenze f_1, \dots, f_{i-1} della derivazione usando una regola di inferenza RI.

Indichiamo con $F \vdash X \rightarrow Y$ il fatto che la **dipendenza funzionale** $X \rightarrow Y$ è **derivabile** da F usando RI.

Regole di derivazione comuni

- **Unione:** $\{X \rightarrow Y, X \rightarrow Z\} \vdash X \rightarrow YZ$
- **Decomposizione:** $\{X \rightarrow YZ\} \vdash X \rightarrow Y$
- **Indebolimento:** $\{X \rightarrow Y\} \vdash XZ \rightarrow Y$
- **Identità:** $\{\} \vdash X \rightarrow X$

Unione : dimostrazione

Unione: $\{X \rightarrow Y, X \rightarrow Z\} \vdash X \rightarrow YZ$

Dimostrazione:

1. $X \rightarrow Y$ per ipotesi
2. $X \rightarrow XY$ per additività da 1
3. $X \rightarrow Z$ per ipotesi
4. $XY \rightarrow YZ$ per additività da 3
5. $X \rightarrow YZ$ per transitività da 2 e 4

Decomposizione: dimostrazione

Decomposizione: $\{X \rightarrow Y Z\} \vdash X \rightarrow Y$

Dimostrazione

1. $X \rightarrow Y Z$ per ipotesi
2. $YZ \rightarrow Y$ per riflessività
3. $X \rightarrow Y$ per transitività da 1 e 2

Indebolimento: dimostrazione

Indebolimento: $\{X \rightarrow Y\} \vdash XZ \rightarrow Y$

Dimostrazione:

1. $XZ \rightarrow X$ per riflessività
2. $X \rightarrow Y$ per ipotesi
3. $XZ \rightarrow Y$ per transitività da 1 e 2

Chiusura di un insieme di attributi

Dato uno schema $R(T,F)$ con $X \subseteq T$, la **chiusura di X rispetto ad F** , indicata col simbolo X^+_F è definita come

$$X_F^+ = \{A \in T \mid F \vdash X \rightarrow A\}$$

Se non vi sono ambiguità scriveremo semplicemente X^+ che indica l'implicazione singola.

Ritorneremo avanti con qualche esempio su questa definizione.

Teorema della chiusura degli attributi

Sia

$$F \vdash X \rightarrow Y \Leftrightarrow Y \subseteq X^+$$

Dimostrazione:

Sia $Y = A_1, \dots, A_k$.

Per la regola di decomposizione abbiamo $F \vdash X \rightarrow A_i$, per definizione di X^+ , $A_i \in X^+$ e quindi anche $Y \subseteq X^+$

Per definizione di X^+ , $F \vdash X \rightarrow A_i$. Per la regola dell'unione, $F \vdash X \rightarrow A_1, \dots, A_k$, cioè $F \vdash X \rightarrow Y$

Correttezza e Completezza

Dato un qualche problema insieme di regole di inferenza RI ed un insieme di dipendenze funzionali F.

- RI è **corretto** se: $F \vdash X \rightarrow Y \Rightarrow F \models X \rightarrow Y$

Applicando RI ad un insieme F di dipendenze funzionali, si ottengono solo *dipendenze logicamente implicate* da F

- Ri è **completo** se $F \models X \rightarrow Y \Rightarrow F \vdash X \rightarrow Y$

Applicando RI ad un insieme F di dipendenze funzionali, si ottengono tutte le *dipendenze logicamente implicate* da F.

Teorema

“Le regole di inferenza di Armstrong sono corrette e complete.”

Dimostrazione:

Supponiamo di avere F un insieme di dipendenze funzionali F su T, ed una dipendenza funzionale $X \rightarrow Y$.

Per prima cosa dimostriamo la **correttezza**

Se $F \vdash X \rightarrow Y$ allora $F \models X \rightarrow Y$

Per seconda cosa dimostriamo la **completezza**

Se $F \models X \rightarrow Y$ allora $F \vdash X \rightarrow Y$

Si procede per *induzione* sulla **lunghezza della derivazione**.

Sia f_1, \dots, f_m la derivazione di $X \rightarrow Y$ da F e supponiamo che il teorema valga per tutte le derivazioni di lunghezza pari a $1, \dots, m-1$.

La dipendenza $f_m = X \rightarrow Y$ è un elemento di F oppure è stata derivata usando una **regola di inferenza di Armstrong**.

- Se è un elemento di F allora è implicata logicamente in maniera banale.
- Se f_m è stata inferita con la regola di riflessività allora $Y \subseteq X$ e l'implicazione logica è banale.
- Se f_m è stata inferita con la regola di additività da una $f_i = X' \rightarrow Y'$, allora per qualche Z si deve avere $X = X'Z$ e $Y = Y'Z$.

Per ipotesi induttiva $F \models f_i$.

Siano t_1 e t_2 due n-uple con $t_1[X'Z] = t_2[X'Z]$.

- per definizione $t_1[X'] = t_2[X']$;
- per f_i , $t_1[Y'] = t_2[Y']$;
- per arricchimento $t_1[Y'Z] = t_2[Y'Z]$.

Quindi $F \models X \rightarrow Y$

Se f_m è stata inferita con la regola di transitività da $f_i = X \rightarrow W$ e $f_j = W \rightarrow Y$ per un qualche W .

Per ipotesi induttiva $F \models f_i$ e $F \models f_j$.

Siano t_1 e t_2 due n-uple con $t_1[X] = t_2[X]$.

Per f_i , $t_1[W] = t_2[W]$ e per f_j , $t_1[Y] = t_2[Y]$. Quindi $F \models X \rightarrow Y$.

Se $F \models X \rightarrow Y$ allora $F \vdash X \rightarrow Y$.

Consideriamo una relazione di n-uple $r = \{t_1, t_2\}$ su T con $t_1[X^+] = t_2[X^+]$ e $t_1[A] \neq t_2[A]$ per ogni $A \in T - X^+$.

Dimostriamo che la relazione r soddisfa F .

Sia $V \rightarrow W \in F$.

- Se $V \not\subseteq X^+$ allora $t_1[V] \neq t_2[V]$ e r soddisfa la dipendenza.
- Se $V \subseteq X^+$ allora $F \vdash X \rightarrow V$ e per transitività $F \vdash X \rightarrow W$, da cui $W \subseteq X^+$ e quindi $t_1[W] = t_2[W]$ e r soddisfa la dipendenza.

Siccome $F \models X \rightarrow Y$, la relazione r soddisfa $X \rightarrow Y$.

Poiché $X \subseteq X^+$ e $t_1[X^+] = t_2[X^+]$, allora $t_1[Y] = t_2[Y]$.

Quindi $Y \subseteq X^+$ e $F \vdash X \rightarrow Y$ per il **teorema di chiusura degli attributi**.

Le regole di inferenza di Armstrong sono corrette e complete.

Questo teorema ci permette di scambiare \models con \vdash ovunque.

In particolare nella definizione di chiusura degli attributi, cioè

$$X_F^+ = \{A \in T \mid F \models X \rightarrow A\}$$

Si può dimostrare che le **regole di inferenza di Armstrong sono minimali**, cioè ignorando anche **una solo** di esse, l'insieme di regole che rimangono **non è più completo**.

Le regole di inferenza di Armstrong non sono l'unico insieme di regole corretto e complesso.

Chiusura di un insieme di dipendenze funzionali

Sia F un insieme di dipendenze funzionali definite su $R(Z)$

La **chiusura di F** è l'insieme F^+ di **tutte** le dipendenze funzionali implicate da F :

$$F^+ = \{X \rightarrow Y \mid F \Rightarrow X \rightarrow Y\}$$

Dato un insieme di dipendenze funzionali F definite su $R(Z)$, un'istanza r di R che soddisfa F soddisfa anche le dipendenze funzionali di F^+ .

Calcolo di F^+

Possiamo usare le **regole di Armstrong** per calcolare F^+ :

Input: $R(T, F)$

Output: F^+

$$F^+ \leftarrow F$$

while (F^+ non cambia) **do**

for each $f \in F^+$ **do**

applicare *riflessività* e *additività* a f e aggiungere a F^+ le dipendenze ottenute

for each $f_1, f_2 \in F^+$ **do**

se possibile, applicare *transitività* a f_1 e f_2 e aggiungere a F^+ la dipendenza ottenuta

return F^+

Esempio 2

Dati $R(ABCDEFGHI)$, $F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$

Alcuni membri di F^+ sono:

- $A \rightarrow H$ per *transitività* da $A \rightarrow B$ e $B \rightarrow H$
- $AG \rightarrow I$ arricchendo $A \rightarrow C$ con G e per *transitività* con $CG \rightarrow I$
- $CG \rightarrow HI$ arricchendo $CG \rightarrow I$ con CG , arricchendo $CG \rightarrow H$ con I e per *transitività*

Calcolo di F^+

Il calcolo di F^+ è **molto costoso**, infatti ha **complessità esponenziale** nel numero di attributi dello schema nel caso peggiore.

Spesso però quello che ci interessa è **verificare** se F^+ contiene una certa dipendenza e **NON generare** l'intera chiusura.

Per fare ciò basta calcolare X^+ per il **teorema di chiusura degli attributi**

$$F \vdash X \rightarrow Y \Leftrightarrow Y \subseteq X^+ \text{ (anche } F)$$

Calcolo di X^+

Input: $R(T, F)$, $X \subseteq T$

Output: X^+

$X^+ \leftarrow X$

```
while ( $X^+$  non cambia) do
  for each  $W \rightarrow V \in F$  do
    if  $W \subseteq X^+$  and  $V \not\subseteq X^+$  then
       $X^+ \leftarrow X^+ \cup V$ 
  return  $X^+$ 
```

Esempio 3

Dati $R(ABCDE)$, $F = \{A \rightarrow B, BC \rightarrow D, B \rightarrow E, E \rightarrow C\}$.

Calcoliamo A^+

$A^+ \leftarrow A$

$A^+ \leftarrow AB$ perché $A \rightarrow B$ e $A \subseteq A^+$

$A^+ \leftarrow ABE$ perché $B \rightarrow E$ e $B \subseteq A^+$

$A^+ \leftarrow ABEC$ perché $E \rightarrow C$ e $E \subseteq A^+$

$A^+ \leftarrow ABCD$ perché $BC \rightarrow D$ e $BC \subseteq A^+$

Possiamo concludere che A è superchiave (e anche chiave)

Chiavi

Dato uno schema $R(T, F)$

- Un insieme di attributi $K \subseteq T$ si dice **superchiave** di R se la dipendenza funzionale $K \rightarrow T$ è implicata da F , ovvero se $K \rightarrow T \in F^+$
- Un insieme di attributi $K \subseteq T$ si dice **chiave** di R se W è una **superchiave** di R e se non esiste alcun sottoinsieme proprio di K che sia una superchiave di R .

Dato che in uno schema ci possono essere più **chiavi**, di solito ne viene scelta una, detta **chiave primaria**, come identificatore delle *n-uple* delle istanze dello schema.

Trovare tutte le chiavi

Il problema di **trovare tutte le chiavi** di una relazione $R(Z)$ richiede un algoritmo di **complessità esponenziale** nel caso pessimo.

Cosa si deve fare:

- Gli attributi che stanno solo a sinistra stanno in tutte le chiavi, chiamiamo N questo insieme.
- Gli attributi che stanno solo a destra non stanno in nessuna chiave

- Si aggiunge ad N un attributo alla volte tra quelli che non stanno solo a destra, poi una coppia di attributi e così via, chiamiamo X_i questo insieme di attributi, ogni volta controlla se la dipendenza $N \cup X_i \rightarrow Z$ esiste.

Verificare una chiave

L'algoritmo per il **calcolo della chiusura di un insieme di attributi** può essere usato per **verificare se un insieme di attributi è chiave o superchiave**.

$X \subseteq T$ è **superchiave** di $R(T, F)$ se e solo se $X \rightarrow T \in F^+$, ovvero se e solo se $T \subseteq X^+$
 $X \subseteq T$ è **chiave** di $R(T, F)$ se e solo se $T \subseteq X^+$, e non esiste $Y \subset X$ tale che $T \subseteq Y^+$

Equivalenza

Due insiemi di **dipendenze funzionali** F e G sugli attributi T di una relazione $R(T)$ sono **equivalenti**, in simboli $F \equiv G$, se e solo se $F^+ = G^+$.

Se $F \equiv G$ allora F è una **copertura** di G e viceversa

La relazione di equivalenza permette di stabilire se due schemi di relazione rappresentano gli stessi fatti.

Basta che abbiano gli stessi attributi e dipendenze funzionali equivalenti.

Per **verificare l'equivalenza** è sufficiente

Esempio 4

Verificare se F e G sono equivalenti:

$$F = \{A \rightarrow C, AC \rightarrow D, E \rightarrow AD, E \rightarrow H\} \quad G = \{A \rightarrow CD, E \rightarrow AH\}$$

Verificare che tutte le dipendenze di G appartengano a F^+

$A \rightarrow C \Rightarrow A \rightarrow AC$ (*riflessività*), $AC \rightarrow D \Rightarrow A \rightarrow D$ (*transitività*) $\Rightarrow A \rightarrow CD$ (*unione*)

$E \rightarrow AD \Rightarrow E \rightarrow A$ (*decomposizione*), $E \rightarrow H \Rightarrow E \rightarrow AH$ (*unione*)

Verificare che tutte le dipendenze di F appartengano a G^+

$A \rightarrow CD \Rightarrow A \rightarrow C$

$A \rightarrow CD \Rightarrow AC \rightarrow CD \Rightarrow AC \rightarrow D$

$E \rightarrow AH \Rightarrow E \rightarrow H$

$E \rightarrow AH \Rightarrow E \rightarrow A \Rightarrow E \rightarrow AE$

$A \rightarrow CD \Rightarrow A \rightarrow D \Rightarrow A \rightarrow AD \Rightarrow AE \rightarrow ADE$

$E \rightarrow ADE \Rightarrow E \rightarrow AD$

Esempio 5

Verificare se F e G sono equivalenti:

$$F = \{A \rightarrow C, AC \rightarrow D, E \rightarrow AD, E \rightarrow H\} \quad G = \{A \rightarrow CD, E \rightarrow AH\}$$

Invece di verificare se $X \rightarrow Y \in F$ è anche in G^+ e viceversa, possiamo verificare se $Y \subseteq X^{+G}$ e, viceversa, per ogni dipendenza funzionale.

Verifichiamo F su X^{+G}

$$A \rightarrow C : A^{+G} = ACD, \text{ quindi } C \in A^{+G}$$

$$AC \rightarrow D : AC^{+G} = ACD, \text{ quindi } D \in AC^{+G}$$

$$E \rightarrow AD : E^{+G} = EADCH, \text{ quindi } AD \in E^{+G}$$

$$E \rightarrow H : E^{+G} = EADCH, \text{ quindi } H \in E^{+G}$$

$$\text{Verifichiamo } G \text{ su } X^{+F} \quad A \rightarrow CD : A^{+F} = ACD, \text{ quindi } CD \in A^{+F}$$

$$E \rightarrow AH : E^{+F} = EADHC, \text{ quindi } AH \in E^{+F}$$

Ridondanza

Sia F un insieme di dipendenze funzionali

- Data $X \rightarrow Y \in F$, X contiene un **attributo estraneo** se e solo se $(F - \{X \rightarrow Y\}) \cup (X - \{A\} \rightarrow Y) \equiv F$, o, in altre parole, se e solo se $X - \{A\} \rightarrow Y \in F^+$
- $X \rightarrow Y$ è una **dipendenza ridondante** se e solo se $(F - \{X \rightarrow Y\}) \equiv F$, o, in altre parole, se e solo se $X \rightarrow Y \in (F - \{X \rightarrow Y\})^+$

Le dipendenze che **non contengono attributi estranei** e la cui **parte destra è un unico attributo** sono dette **dipendenze elementari**.

Se tentiamo di eliminare la dipendenza ridondante prima di eliminare l'attributo estraneo, non ci riusciamo, quindi l'**ordine** delle due attività è **importante**.

Copertura minimale

Sia F un insieme di **dipendenze funzionali**.

Si dice che F è una **copertura minimale** se e solo se:

- ogni parte destra di una dipendenza ha un unico attributo
- le dipendenze non contengono attributi estranei
- non esistono dipendenze ridondanti

In alcuni testi una **copertura minimale** è chiamata

- Insieme minimale
- Copertura canonica

Calcolo della copertura minimale

Input: insieme di dipendenze funzionali F

Output: copertura minimale G di F

```
 $G \leftarrow F$ 
for each  $X \rightarrow Y \in G$  do
   $Z \leftarrow X$ 
  for each  $A \in X$  do
    if  $Y \in (Z - (A))_F^+$  then
       $Z \leftarrow Z - \{A\}$ 
     $G \leftarrow (G - \{X \rightarrow Y\}) \cup \{Z \rightarrow Y\}$ 
for each  $f \in G$  do
  if  $f \in (G - \{f\})^+$  then
     $G \leftarrow G - \{f\}$ 
return  $G$ 
```

Calcoliamo gli attributi estranei delle dipendenze

Eliminiamo gli attributi estranei delle dipendenze

Eliminiamo le dipendenze ridondanti

Copertura minimale

Il precedente algoritmo dimostra il seguente teorema.

Teorema

“Per ogni insieme di dipendenze funzionali F esiste una copertura minimale”

Si noti che il teorema nulla dice sull'**unicità** della copertura minimale.

Infatti, per $F = \{AB \rightarrow C, A \rightarrow B, B \rightarrow A\}$,

- $\{A \rightarrow C, A \rightarrow B, B \rightarrow A\}$ è una copertura minimale
- $\{B \rightarrow C, A \rightarrow B, B \rightarrow A\}$ è una copertura minimale

Lezione 08: Normalizzazione

Eliminare le anomalie

Abbiamo sviluppato la teoria delle dipendenze funzionali per **identificare le anomalie** in uno schema mal definito.

Adesso siamo in grado di affrontare il passaggio da **schemi “con anomalie”** a **schemi “ben fatti”**.

Per fare ciò definiremo un nuovo concetto, le **forme normali**, intese come proprietà che devono essere soddisfatte dalle dipendenze fra attributi di schemi “ben fatti”.

Vedremo solo la forma normale di Boyce-Codd(BCNF) e la terza forma normale (3NF).

Forma Normale di Boyce-Codd

Uno schema R (T, F) è in **forma normale di Boyce-Codd (BCNF)** se e solo se per ogni dipendenza funzionale non banale $X \rightarrow Y \in F^+$, X è una **superchiave** di R.

L’idea su cui si basa la **BCNF** è che una dipendenza funzionale $X \rightarrow A$, in cui X non contiene attributi estranei, indica che, nella realtà che si modella, esiste una collezione di entità omogenee che sono univocamente identificate da X.

Dalla definizione, il fatto che uno schema sia in **BCNF dipende dalla chiusura F^+ , non dalla specifica copertura F**.

Purtroppo, per **calcolare F^+** abbiamo solo algoritmi di **complessità esponenziale**, che costano troppo. Tuttavia, possiamo facilmente **stabilire** se uno schema è in BCNF con un algoritmo di **complessità polinomiale**.

Teorema su Forma Normale di Boyce-Codd

“Uno schema $R(T, F)$ è in BCNF se e solo se per ogni dipendenza funzionale non banale $X \rightarrow Y \in F^+$, X è una superchiave.”

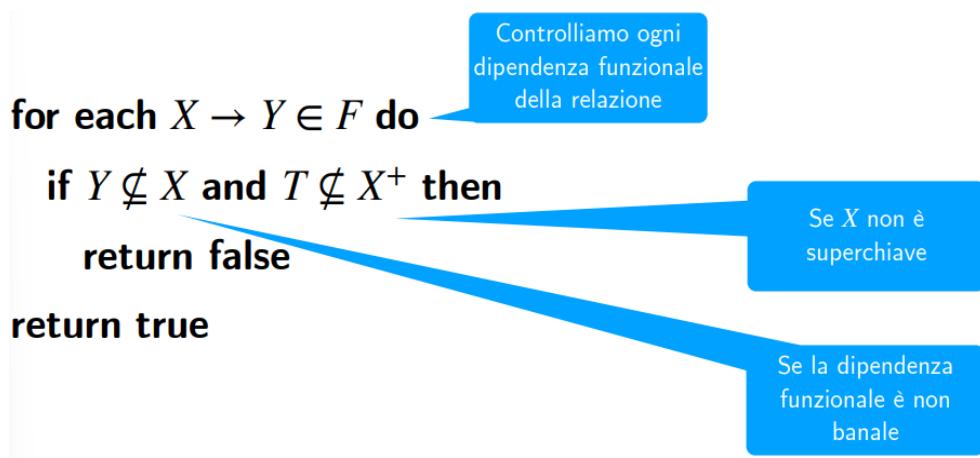
Corollario:

Uno schema $R(T, F)$ con F copertura minimale è in BCNF se e solo se per ogni **dipendenza funzionale elementare $X \rightarrow A \in F^+$, X è una superchiave**.

Verifica BCNF

Input: schema R(T, F)

Output: **true** se R è in BCNF, **false** altrimenti



Esempio:

Impiegato	Stipendio	Progetto	Bilancio	Funzione
Rossi	20	Marte	2	tecnico
Verdi	35	Giove	15	progettista
Verdi	35	Venere	15	progettista
Neri	55	Venere	15	direttore
Neri	55	Giove	15	consulente
Neri	55	Marte	2	consulente
Mori	48	Marte	2	direttore
Mori	48	Venere	15	progettista
Bianchi	48	Venere	15	progettista
Bianchi	48	Giove	15	direttore

Impiegato → Stipendio

Progetto → Bilancio

Impiegato, Progetto → Funzione

Spiegazione:

Proviamo a normalizzare il precedente schema in **BCNF** con una procedura intuitiva.

Questa procedura non è valida in generale, ma solo in alcuni “casi semplici”.

Per ogni dipendenza $X \rightarrow Y$ che viola la **BCNF**, definiamo una nuova relazione su XY ed eliminiamo Y dalla relazione originaria.

Impiegato	Stipendio
Rossi	20
Verdi	35
Neri	55
Mori	48
Bianchi	48

Progetto	Bilancio
Marte	2
Giove	15
Venere	15

Impiegato	Progetto	Funzione
Rossi	Marte	tecnico
Verdi	Giove	progettista
Verdi	Venere	progettista
Neri	Venere	direttore
Neri	Giove	consulente
Neri	Marte	consulente
Mori	Marte	direttore
Bianchi	Venere	progettista
Bianchi	Giove	direttore

ATTENZIONE: in alcuni casi ricostruendo la tabella si avranno delle *n-uple* in più!

Decomposizione di schemi

Dato uno schema $R(T)$, l'insieme di schemi $\rho = \{R_1(T_1), \dots, R_k(T_k)\}$ è una **decomposizione** di R se e solo se $\cup_i T_i = T$.

Si noti che la precedente definizione non richiede che gli schemi R_i siano disgiunti.
Come caratterizzare **l'equivalenza tra schema originario e sua decomposizione?**
In generale la decomposizione deve:

- **preservare i dati**
- **preservare le dipendenze**

Esempio di perdita di dati

R			$R_1 = \pi_{PT}(R)$		$R_2 = \pi_{PC}(R)$		$R_1 \bowtie R_2$		
P	T	C	P	T	P	C	P	T	C
p1	t1	c1	p1	t1	p1	c1	p1	t1	c1
p1	t2	c2	p1	t2	p1	c1	p1	t1	c2
p1	t3	c2	p1	t3	p1	c2	p1	t2	c1

Esempio di perdita di dipendenze

R

P	T	C
p1	t1	c1
p1	t2	c2
p1	t3	c2

$R_1 = \pi_{PT}(R)$

P	T
p1	t1
p1	t2
p1	t3

$R_2 = \pi_{TC}(R)$

T	C
t1	c1
t2	c2
t3	c2

$T \rightarrow C$

$C \rightarrow P$

questa decomposizione preserva i dati

questa decomposizione non preserva la dipendenza

$C \rightarrow P$

perché gli attributi sono in relazioni diverse

Teorema della perdita di dati

Se $\rho = \{R_1(T_1), \dots, R_k(T_k)\}$ è una decomposizione di $R(T, F)$ allora per ogni istanza r di $R(T)$ si ha

$$r \subseteq \pi_{T_1}(r) \bowtie \dots \bowtie \pi_{T_k}(r)$$

Questo teorema ci dice che perdiamo informazione quando, **ricostruendo una relazione, otteniamo più *n-uple* che nella relazione originaria.**

Decomposizione che preserva i dati

Dato uno schema $R(T, F)$ ed una decomposizione $\rho = \{R_1(T_1), \dots, R_k(T_k)\}$

ρ è una **decomposizione $R(T, F)$ che preserva i dati** se e solo se, per ogni relazione r che soddisfa $R(T, F)$ si ha:

$$r = \pi_{T_1}(r) \bowtie \dots \bowtie \pi_{T_k}(r)$$

Questa definizione ci dice che, per una decomposizione che preserva i dati, ogni istanza valida r della relazione di partenza **deve essere uguale al join naturale delle sue proiezioni** sui vari T_i .

Diciamo che r si decomponete senza perdita su X_1 e X_2 se il join delle due proiezioni è uguale a r stessa.

Teorema della preservazione dei dati

Sia $\rho = \{R_1(T_1), R_2(T_2)\}$ una decomposizione di $R(T, F)$

essa preserva i dati se e solo se $T_1 \cap T_2 \rightarrow T_1 \in F^+$ oppure $T_1 \cap T_2 \rightarrow T_2 \in F^+$

In altre parole, gli **attributi comuni** alle due relazioni **devono essere chiave in una delle due tabelle.**

Possiamo dire che r si decompone senza perdita su due relazioni se l'insieme degli attributi comuni alle due relazioni è chiave per almeno una delle relazioni decomposte.

Impiegato	Sede
Rossi	Roma
Verdi	Milano
Neri	Milano

Progetto	Sede
Marte	Roma
Giove	Milano
Saturno	Milano
Venere	Milano

Impiegato	Progetto	Sede
Rossi	Marte	Roma
Verdi	Giove	Milano
Verdi	Venere	Milano
Neri	Saturno	Milano
Neri	Venere	Milano
Verdi	Saturno	Milano
Neri	Giove	Milano

Nel nostro esempio, Sede è l'attributo a comune tra le due tabelle, ma non è chiave per nessuna delle due.

Non c'è nessuna dipendenza con Sede come parte sinistra.

Proiezioni di un insieme di dipendenze

Dato $R(T, F)$ e $T_i \subseteq T$, la proiezione dell'insieme di dipendenze F sull'insieme di attributi T_i è

$$\pi_{T_i}(F) = \{X \rightarrow Y \in F^+ \mid X, Y \subseteq T_i\}$$

NOTA BENE: la proiezione è costruita considerando le dipendenze in F^+ , non quelle in F .

Esempio:

$R(ABC, \{ A \rightarrow B, B \rightarrow C, C \rightarrow A \})$

$\pi_{AB}(F) = \{ A \rightarrow B, B \rightarrow A \}$

$\pi_{AC}(F) = \{ A \rightarrow C, C \rightarrow A \}$

Algoritmo per il calcolo di $\pi_{T_i}(F)$

Input: $R(T, F)$ e $T_i \subseteq T$

Output: $\pi_{T_i}(F)$ $Z \leftarrow \{ \}$

for each $Y \subset T_i$ **do**

$W \leftarrow Y^+ - Y$

$Z \leftarrow Z \cup \{ Y \rightarrow (W \cap T_i) \}$

return Z

Calcolo di $\pi_{T_i}(F)$

L'algoritmo precedente ha **complessità esponenziale** nel caso peggiore.

Consideriamo:

- $R(A_1, \dots, A_n, B_1, \dots, B_n, C_1, \dots, C_n, D)$
- $F = (\cup_i \{ A_i \rightarrow C_i, B_i \rightarrow C_i \}) \cup \{ C_1 \dots C_n \rightarrow D \}$

La proiezione di F su $A_1, \dots, A_n, B_1, \dots, B_n, D$ è *pari a* $\{X_1, \dots, X_n \rightarrow D\}$ dove $X_i = A_i$ oppure $X_i = B_i$

La sua dimensione è **esponenziale** rispetto al numero di attributi e di dipendenze funzionali.

Si può dimostrare che nessun altro insieme **equivalente** ha cardinalità inferiore.

Decomposizione che preserva le dipendenze

Dato uno schema $R(T, F)$ ed una decomposizione $\rho = \{R_1(T_1), \dots, R_k(T_k)\}$, ρ è una **decomposizione di $R(T, F)$ che preserva le dipendenze** se e solo se:

$$\cup_i \pi_{T_i}(F) \equiv F$$

Si noti il simbolo di equivalenza \equiv

Le decomposizioni di $R(T, F)$ in due relazioni con attributi X e Y è una decomposizione che preserva le dipendenze se $\pi_X(F) \cup \pi_Y(F) \equiv F$, cioè se

$$(\pi_X(F) \cup \pi_Y(F))^+ = F^+$$

Verificare una decomposizione

Per **verificare** se una decomposizione di $R(T, F)$ in due relazioni con attributi X e Y preserva le dipendenze bisogna verificare che $(\pi_X(F) \cup \pi_Y(F))^+ = F^+$. Per fare ciò:

- è **necessario** saper **calcolare la proiezione** di un insieme di dipendenze funzionali su un insieme di attributi.
- È **necessario** saper **determinare l'equivalenza** di due insiemi di dipendenze funzionali.

Quindi:

Per **calcolare la proiezione** di un insieme di dipendenze funzionali su un insieme di attributi abbiamo un algoritmo con **complessità esponenziale**.

Per **verificare l'equivalenza** di due insiemi di dipendenze funzionali F e G abbiamo bisogno di un algoritmo con **complessità polinomiale**.

- Per ogni $X \rightarrow Y \in F$, calcoliamo X^+_G e verifichiamo se $Y \in X^+_G$
- Per ogni $X \rightarrow Y \in G$, calcoliamo X^+_F e verifichiamo se $Y \in X^+_F$

Algoritmo per decomposizione in BCNF

Input: $R(T, F)$ (per semplicità gli elementi di F sono nella forma $X \rightarrow A$)

Output: ρ che preserva i dati

$\rho \leftarrow \{ R(T, F) \}$

while esiste $R_i(T_i, F_i) \in \rho$ che non è in BCNF **do**

for each $X \rightarrow A \in F_i$ **do**

if $A \notin X$ **and** $T_i \not\subseteq X^+$ **then**

$R_1 \leftarrow R_i(T_i - A, \pi_{T_i - A}(F_i))$

$R_2 \leftarrow R_i(X + A, \pi_{X+A}(F_i))$

$\rho \leftarrow \rho - \{ R_i \} \cup \{ R_1, R_2 \}$

break

return ρ

Teorema:

Qualunque sia la relazione, l'esecuzione dell'algoritmo per decomposizione in BCNF su tale relazione termina e produce una decomposizione della relazione tale che:

- la decomposizione prodotta è in **BCNF**
- la decomposizione prodotta preserva i dati

NON è garantito che la decomposizione generata preservi le dipendenze

Esempio

Sia $R = \text{Telefoni}$

Sia $T = \{\text{Prefisso}, \text{Numero}, \text{Località}\}$

Sia $F = \{\text{Prefisso}, \text{Numero} \rightarrow \text{Località}; \text{Località} \rightarrow \text{Prefisso}\}$

Inizialmente $\rho = \{\text{Telefoni}\}$

La dipendenza $\text{Località} \rightarrow \text{Prefisso}$ viola la **BCNF**

Rimpiazziamo $R = \text{Telefoni}$ in ρ con $R_1 (\{\text{Numero}, \text{Località}\}, \{ \})$ e $R_2 (\{\text{Località}, \text{Prefisso}\}, \{\text{Località} \rightarrow \text{Prefisso} \})$

La decomposizione $\rho = \{R_1, R_2\}$ è in **BCNF** e quindi l'algoritmo termina.

La decomposizione ρ preserva i dati, ma non preserva le dipendenze funzionali.

Prefisso, Numero → Località è perduta.

Qualità delle decomposizioni

Una decomposizione dovrebbe sempre garantire

- Essere in **BCNF**
- L'assenza di perdite sui dati, in modo da poter ricostruire le informazioni originarie tramite join naturali
- La conservazione delle dipendenze funzionali, in modo da mantenere i vincoli di integrità originari

Esempio

Ogni dirigente ha una sede, e un progetto può essere diretto da più persona, ma in sedi diverse.

Dirigente	Progetto	Sede
Rossi	Marte	Roma
Verdi	Giove	Milano
Verdi	Marte	Milano
Neri	Saturno	Milano
Neri	Venere	Milano

Dirigente → Sede

Progetto, Sede → Dirigente

Questa relazione è in **BCNF**? Applichiamo l'**algoritmo di verifica!**

Progetto, Sede → Dirigente **SI**

Dirigente → Sede **NO** (Dirigente non è superchiave)

Come decomponiamo la relazione?

La dipendenza Progetto, Sede → Dirigente coinvolge tutti gli attributi e quindi nessuna decomposizione potrà preservarla.

Possiamo calcolare una decomposizione in **BCNF**, ma non potrà preservare questa dipendenza. Quando non si può raggiungere una **BCNF** di buona qualità spesso si tratta di una cattiva progettazione, tuttavia possiamo abbandonare la **BCNF** e adottare una nuova forma normale meno restrittiva della **BCNF**.

Terza Forma Normale

Una relazione $R(T,F)$ è in **Terza Forma Normale (3NF)** se e solo se, per ogni **dipendenza funzionale non banale** $X \rightarrow A \in F^+$, è verificata almeno una delle seguenti condizioni:

- X è **una superchiave** di R ;
- A è **contenuto in almeno una chiave di R** (in questo caso si dice che A è un attributo primo)

Come si vede dalla definizione, se R è in **BCNF** allora R è in **3NF**.

$$\text{BCNF} \Rightarrow \text{3NF}$$

Esempio

Dirigente	Progetto	Sede
Rossi	Marte	Roma
Verdi	Giove	Milano
Verdi	Marte	Milano
Neri	Saturno	Milano
Neri	Venere	Milano

Progetto, Sede \rightarrow Dirigente

Dirigente \rightarrow Sede

L'attributo Sede è contenuto in una **chiave**, quindi la relazione è in **3NF**.

Tuttavia c'è da notare una ridondanza nella ripetizione della Sede del Dirigente per i vari progetti che dirige.

Verifica di 3NF

Il problema di **decidere** se uno schema di relazione è in **3NF** è **NP- Completo**.

Il miglior algoritmo deterministico noto ha **complessità esponenziale** nel caso peggiore, infatti:

- Per stabilire se uno schema è in **3NF** occorre conoscere gli attributi primi, cioè le chiavi
- L'algoritmo per calcolare le chiavi ha complessità esponenziale

Tuttavia **si può sempre ottenere una decomposizione in 3NF** che preserva dati e dipendenze funzionali.

Algoritmo per decomposizione in 3NF

Dato un insieme di attributi T ed una **copertura minimale** G, si divide G in gruppi G_i in modo che tutte le dipendenze funzionali di ogni gruppo G_i abbiano la **stessa parte sinistra**.

Da ogni gruppo G_i si definisce uno schema di relazione composto da tutti gli attributi che appaiono in G_i , la cui chiave, detta **chiave sintetizzata**, è la parte sinistra comune.

Input: $R(T, F)$

Output: ρ che preserva i dati e le dipendenze e con ogni elemento in **3NF**

1. Trovare una copertura minimale G di F e porre $\rho \leftarrow \{ \}$
2. Sostituire in G ogni insieme di dipendenze $\{X \rightarrow A_1, \dots, X \rightarrow A_h\}$ con la dipendenza $X \rightarrow A_1 \dots A_h$
3. Per ogni dipendenza $X \rightarrow Y \in G$ creare uno schema con attributi XY in ρ
4. Eliminare da ρ ogni schema che sia contenuto in un altro schema di ρ
5. Se ρ non contiene nessuno schema i cui attributi costituiscono una superchiave di R, aggiungere a ρ uno schema con attributi W, dove W è una chiave di R.

Teorema

Qualunque sia la relazione, l'esecuzione dell'algoritmo per decomposizione in **3NF** su tale relazione termina e produce una decomposizione della relazione tale che:

- la decomposizione prodotta è in **3NF**
- la decomposizione prodotta preserva i dati e le dipendenze funzionali

La **complessità** dell'algoritmo è **polinomiale**.

Esempio

Dato $R(ABCD, F)$ con $F = \{AB \rightarrow C, C \rightarrow D, D \rightarrow B\}$

F è una copertura minimale.

$AB \rightarrow C : R_1(ABC)$ con chiave sintetizzata AB

$C \rightarrow D : R_2(CD)$ con chiave sintetizzata C

$D \rightarrow B : R_3(BD)$ con chiave sintetizzata D

$\pi_{R2}(F) = \{C \rightarrow D\}$

$\pi_{R3}(F) = \{D \rightarrow B\}$

$\pi_{R1}(F) = \{AB \rightarrow C, C \rightarrow B\}$

Esempio

Dato $R(ABCDEFGH, F)$ con $F = \{ ABC \rightarrow DEG, BD \rightarrow ACE, C \rightarrow BH, H \rightarrow BDE \}$ Per prima cosa, calcoliamo la copertura minimale

$$F \equiv F_1 = \{$$

$ABC \rightarrow D, ABC \rightarrow E, ABC \rightarrow G, BD \rightarrow A, BD \rightarrow C, BD \rightarrow E, C \rightarrow B, C \rightarrow H, H \rightarrow B, H \rightarrow D, H \rightarrow E \}$

ABC contiene attributi estranei? $C^+ = CBHDEAG$, quindi A, B sono estranei in ABC

BD contiene attributi estranei? $B^+ = B, D^+ = D$ quindi non ci sono attributi estranei in BD.

$$F_2 \equiv F_1 = \{ C \rightarrow D, C \rightarrow E, C \rightarrow G, BD \rightarrow A, BD \rightarrow C, BD \rightarrow E, C \rightarrow B, C \rightarrow H, H \rightarrow B, H \rightarrow D, H \rightarrow E \}$$

F_2 contiene dipendenze ridondanti?

$C \rightarrow D$ perché $C \rightarrow H \rightarrow D$

$C \rightarrow E$ perché $C \rightarrow H \rightarrow E$

$BD \rightarrow E$ perché $BD \rightarrow C \rightarrow H \rightarrow E$

$C \rightarrow B$ perché $C \rightarrow H \rightarrow B$

$$G \equiv F_2 = \{ C \rightarrow G, BD \rightarrow A, BD \rightarrow C, C \rightarrow H, H \rightarrow B, H \rightarrow D, H \rightarrow E \}$$

Prima di eseguire le sostituzioni previste, controlliamo se le parti sinistre delle dipendenze in G sono superchiavi.

$C^+ = CBHDEAG$, quindi C è chiave

$BD^+ = BDACGHE$, quindi BD è superchiave

$H^+ = HBDEACG$, quindi H è chiave

Possiamo concludere che il nostro schema è in **BCNF**, e quindi in **3NF**, e non va decomposto.

Altro esempio

- Dato $R(ABCDEFGH, F)$ con

$$F = \{AB \rightarrow CDE, CE \rightarrow AB, A \rightarrow G, G \rightarrow BD\}$$

- Per prima cosa, calcoliamo la copertura minimale

$$F \equiv F_1 = \{AB \rightarrow C, AB \rightarrow D, AB \rightarrow E, CE \rightarrow A,$$

- $CE \rightarrow B, A \rightarrow G, G \rightarrow B, G \rightarrow D\}$

• AB contiene attributi estranei?

- $A^+ = AGBDCE$, quindi B è estraneo in AB

• CE contiene attributi estranei?

- $C^+ = C, E^+ = E$ quindi non ci sono attributi estranei in CE

$$F_2 \equiv F_1 = \{A \rightarrow C, A \rightarrow D, A \rightarrow E, CE \rightarrow A,$$

- $CE \rightarrow B, A \rightarrow G, G \rightarrow B, G \rightarrow D\}$

$$F_2 \equiv F_1 = \{A \rightarrow C, A \rightarrow D, A \rightarrow E, CE \rightarrow A,$$

- $CE \rightarrow B, A \rightarrow G, G \rightarrow B, G \rightarrow D\}$

• F_2 contiene dipendenze ridondanti?

- $A \rightarrow D$ perché $A \rightarrow G \rightarrow D$

- $CE \rightarrow B$ perché $CE \rightarrow A \rightarrow G \rightarrow B$

$$G \equiv F_2 = \{A \rightarrow C, A \rightarrow E, CE \rightarrow A,$$

- $A \rightarrow G, G \rightarrow B, G \rightarrow D\}$

Controllo superchiavi

In G nessuna

dipendenza funzionale

include H , quindi

nessuna delle parti

sinistre delle

dipendenze in G sono

superchiavi.

$$G \equiv F_2 = \{A \rightarrow C, A \rightarrow E, CE \rightarrow A,$$

- $A \rightarrow G, G \rightarrow B, G \rightarrow D\}$

• Decomponiamo!

- $A \rightarrow C, A \rightarrow E, A \rightarrow G$, quindi creiamo $R_1(ACEG)$

- $CE \rightarrow A$, quindi creiamo $R_2(CEA)$

- $G \rightarrow B, G \rightarrow D$, quindi creiamo $R_3(GBD)$

• Eliminiamo!

- $R_2(CEA)$ è contenuta in $R_1(ACEG)$, quindi la eliminiamo

• Controllo superchiave!

- Né $R_1(ACEG)$ né $R_3(GBD)$ contengono H

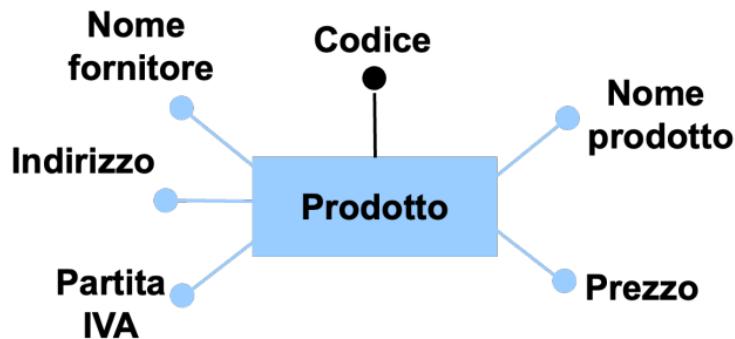
- Siccome AH è chiave, aggiungiamo $R_0(AH)$ alla decomposizione

- $\rho = \{R_1(ACEG), R_3(GBD), R_0(AH)\}$ è in 3NF

Progettazione e Normalizzazione

La teoria della normalizzazione serve per verificare la qualità dello schema logico. Ma si può usare anche durante la progettazione concettuale per ottenere uno schema di buona qualità (verifica ridondanze, partizionamento di entità/relazioni)

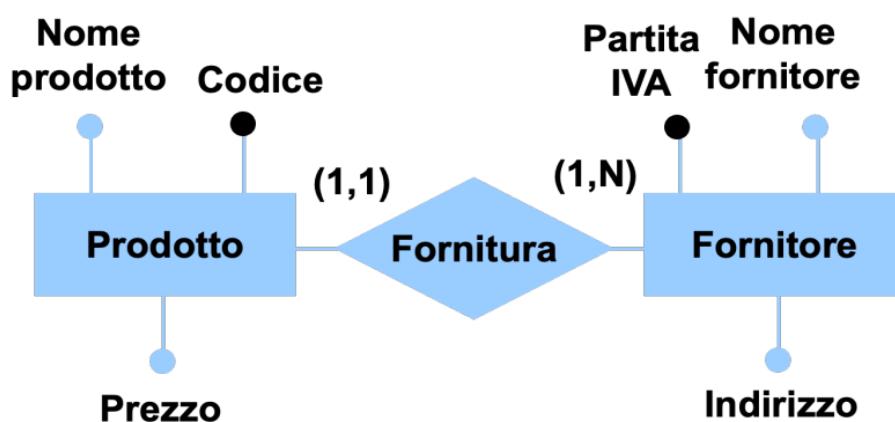
Verifica di normalizzazione su entità



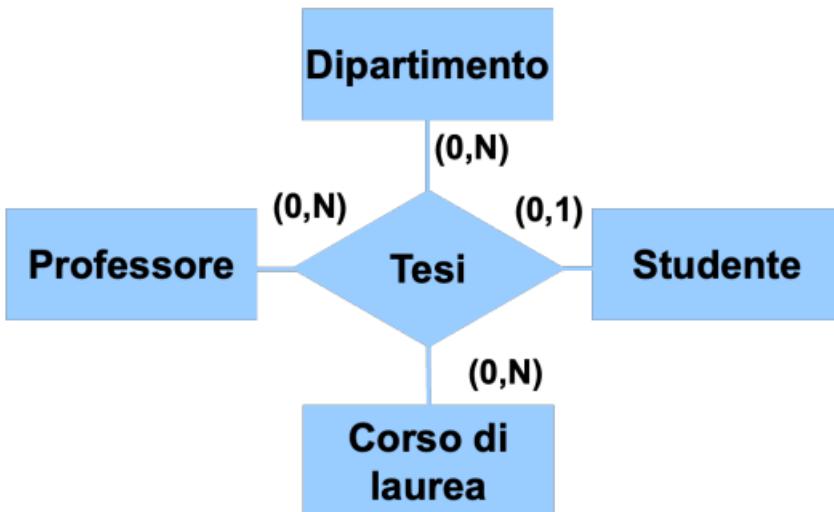
Abbiamo la dipendenza funzionale Partita Iva → Nome fornitore, Indirizzo

- Codice è chiave
- Partita IVA → Nome fornitore, Indirizzo
 - Partita IVA non è superchiave
 - Nome fornitore e Indirizzo non fanno parte di una chiave
- L'entità **viola** la terza forma normale

Verifica di normalizzazione su entità

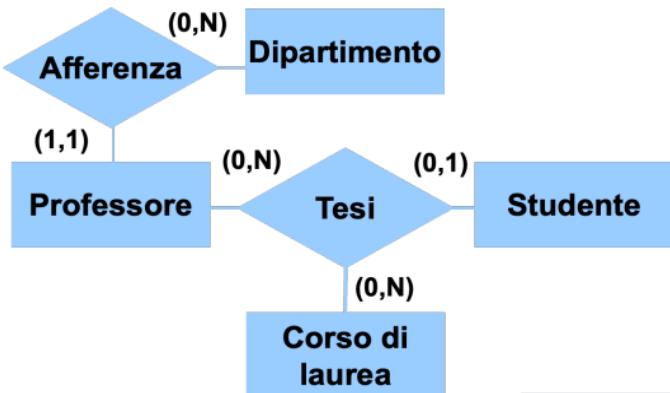


Verifica di normalizzazione su relationship



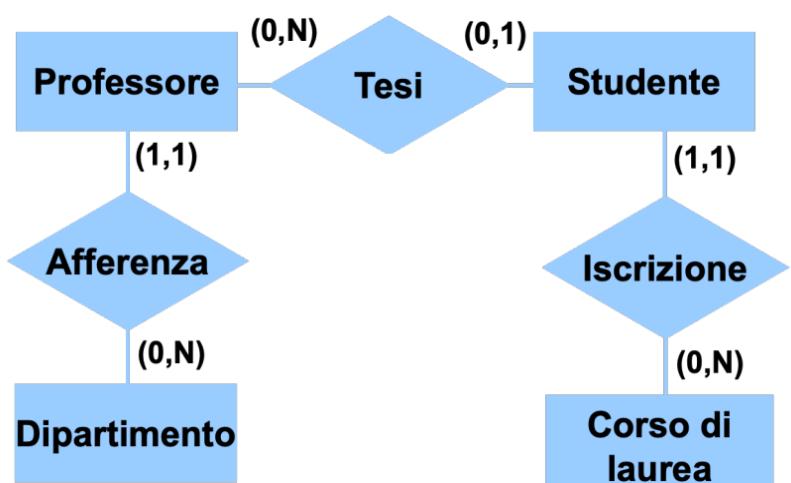
- Studente → Corso di laurea NON VIOLA la **3NF**
- Studente → Professore NON VIOLA la **3NF**
- Professore → Dipartimento VIOLA la **3NF**
- Studente è **chiave**.

Verifica di normalizzazione su relationship

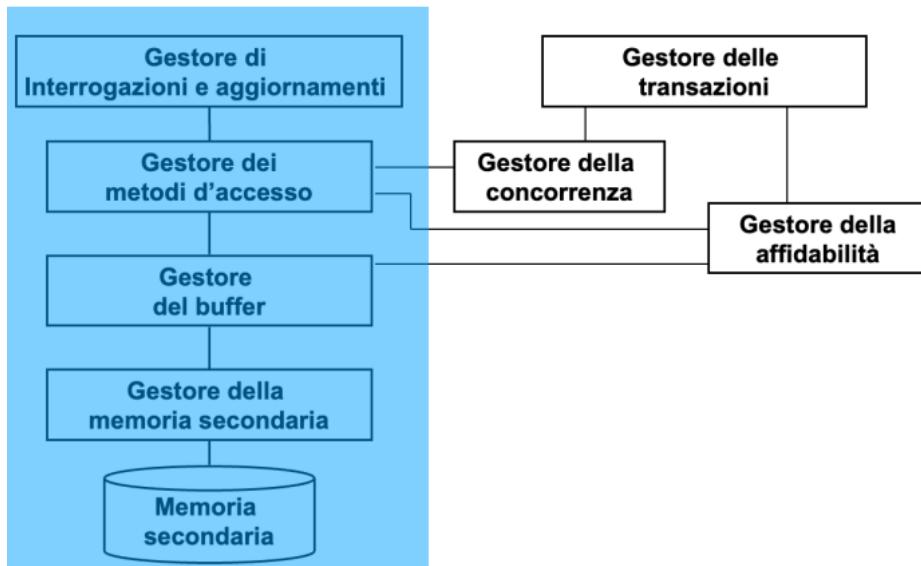


Le due relationship Afferenza e Tesi sono in **3NF** (ed in **BCNF**).

Tesi lo è in virtù delle dipendenze Studente → Corso di laurea e Studente → Professore



Lezione 09: Gestione del buffer e della memoria secondaria



Memoria Principale⁶ e Secondaria⁷

I programmi possono fare riferimento solo ai **dati in memoria principale**.

Le basi di dati devono essere in **memoria secondaria** per due motivi:

- dimensioni
- persistenza

I dati in **memoria secondaria** possono essere utilizzati solo se prima trasferiti in **memoria principale**. Infatti, questo spiega i termini *principale* e *secondaria*.

I dispositivi di **memoria secondaria** sono organizzati in **blocchi di lunghezza fissa** (di solito).

Le **uniche operazioni** sui dispositivi sono la **lettura** e la **scrittura** di una **pagina**, ovvero dei **dati di un blocco** (Esempio: una stringa di byte).

Per comodità consideriamo *blocco* e *pagina* sinonimi.

Il **filesystem** è il componente del **sistema operativo** che **gestisce la memoria secondaria**.

I DBMS ne utilizzano le funzionalità per **creare ed eliminare file** e per **leggere e scrivere singoli blocchi o sequenze di blocchi contigui**.

⁶ **Memoria Principale**: sarebbe la RAM

⁷ **Memoria Secondaria**: sarebbe la ROM

Il DBMS gestisce i **file allocati** come se fossero un **unico grande spazio di memoria secondaria** e costruisce, in tale spazio, le **strutture fisiche** con cui implementa le relazioni.

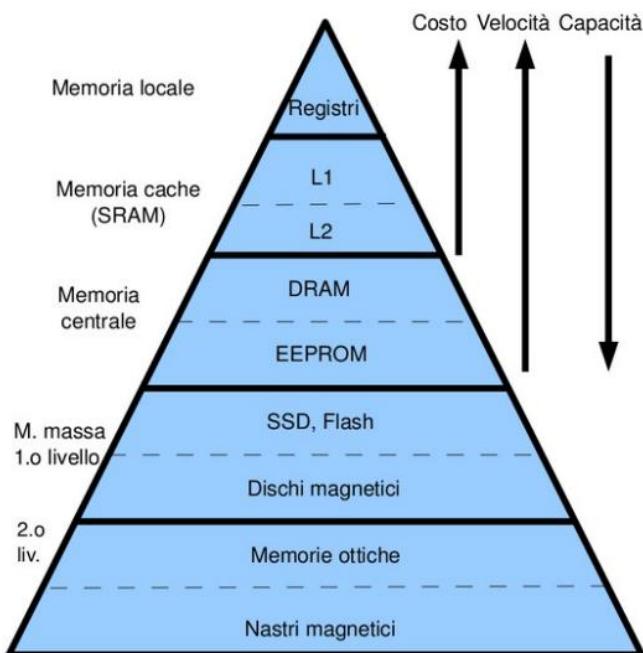
L'**organizzazione dei file**, sia in termini di distribuzione dei record nei blocchi sia relativamente alla struttura all'interno dei singoli blocchi, è **gestita direttamente dal DBMS**.

Il DBMS crea **file di grandi dimensioni** che utilizza per **memorizzare diverse relazioni** (o al limite l'intero database).

Quindi è possibile che un file contenga i dati di più relazioni e che le varie *n-uple* di una relazione siano in file diversi.

Spesso, ma non sempre, ogni blocco è dedicato a *n-uple* di un'unica relazione

Gerarchia di memoria



Memoria Secondaria

Dato un **indirizzo di accesso**, le **prestazioni** di memoria secondaria si misurano in termini della **somma** tra:

- il **tempo** che la testina impiega per raggiungere la **traccia** di interesse (un disco è organizzato in tracce concentriche ed esiste una sola testina che legge tutte le tracce)
- la **latenza** ovvero il tempo per accedere al primo byte del blocco di interesse
- il **tempo di trasferimento** ovvero il tempo necessario a muovere tutti i dati del blocco.

Il costo di un accesso a memoria secondaria è **quattro o più ordini di grandezza maggiore** di quello per operazioni in memoria centrale.

Perciò nelle applicazioni **I/O bound**, cioè con molti accessi a memoria secondaria e relativamente poche operazioni) il **costo** dipende esclusivamente dal **numero di accessi** a memoria secondaria.

Inoltre, gli accessi a **blocchi vicini costano meno** (contiguità).

Gestione del buffer

Il buffer è un **area di memoria centrale**, gestita dal DBMS (preallocata) e **condivisa** fra le transazioni.

È **organizzato in pagine** di dimensioni pari o multiple di quelle dei **blocchi di memoria secondaria** (1KB-100KB).

Se assumiamo che **coincidano pagina e blocco**, il caricamento di **una pagina** del buffer richiede **una lettura** in memoria secondaria, mentre salvare una pagina corrisponde ad **una scrittura**.

Scopo della Gestione del buffer

Lo scopo della gestione del buffer è di **ridurre** il numero di accessi alla memoria secondaria:

- In **caso di lettura**, se la pagina è già presente nel buffer, non è necessario accedere alla memoria secondaria.
- In **caso di scrittura**, il gestore del buffer può decidere di differire la scrittura fisica (ammesso che ciò sia compatibile con la gestione dell'affidabilità)

La gestione dei buffer e la differenza di costi fra memoria principale e secondaria possono suggerire **algoritmi innovativi**.

Dati gestiti dal buffer manager

I dati gestiti dal buffer manager sono:

- il **buffer** stesso;
- una **directory** che **per ogni pagina** mantiene per esempio:
il file fisico ed il numero del blocco

due variabili di stato:

- un **contatore** che indica quanti programmi utilizzano la pagina
- un **bit** che indica se la pagina è stata modificata

Funzioni del buffer manager

Le sue funzioni sono:

- ricevere **richieste di lettura e scrittura** (di pagine)
- le **esegue** accedendo alla **memoria secondaria** solo quando **indispensabile** ed utilizzando invece il **buffer** quando **possibile**.
- esegue le **primitive fix, unfix, setDirty, force**.

Le **politiche** sono simili a quelle relative alla gestione della memoria da parte dei sistemi operativi:

- **località dei dati:** è alta la probabilità di dover riutilizzare i dati attualmente in uso
- **legge 80/20:** l'80% delle operazioni utilizza sempre lo stesso 20% dei dati

Blocchi e *n-uple*

I **file** sono logicamente organizzati in **record** e quest'ultimi sono mappati nei blocchi di memoria secondaria. Le ***n-uple*** di una relazione (record di file) stanno **in blocchi contigui**. A volte in un blocco ci sono -uple di relazioni diverse ma correlate (i join sono favoriti).

I **blocchi** (componenti “**fisici**” di un file) e le -uple o record (componenti “**logici**” di una relazione) hanno dimensioni in generale diverse:

- la **dimensione del blocco** dipende dal file system
- la **dimensione del record** dipende dalle esigenze dell'applicazione, e può anche variare nell'ambito di un file.

Fattore di blocco

Numero di record in un blocco:

- L_R : **dimensione** di un record, per semplicità costante nel file "record a lunghezza fissa"
- L_B : **dimensione** di un **blocco**

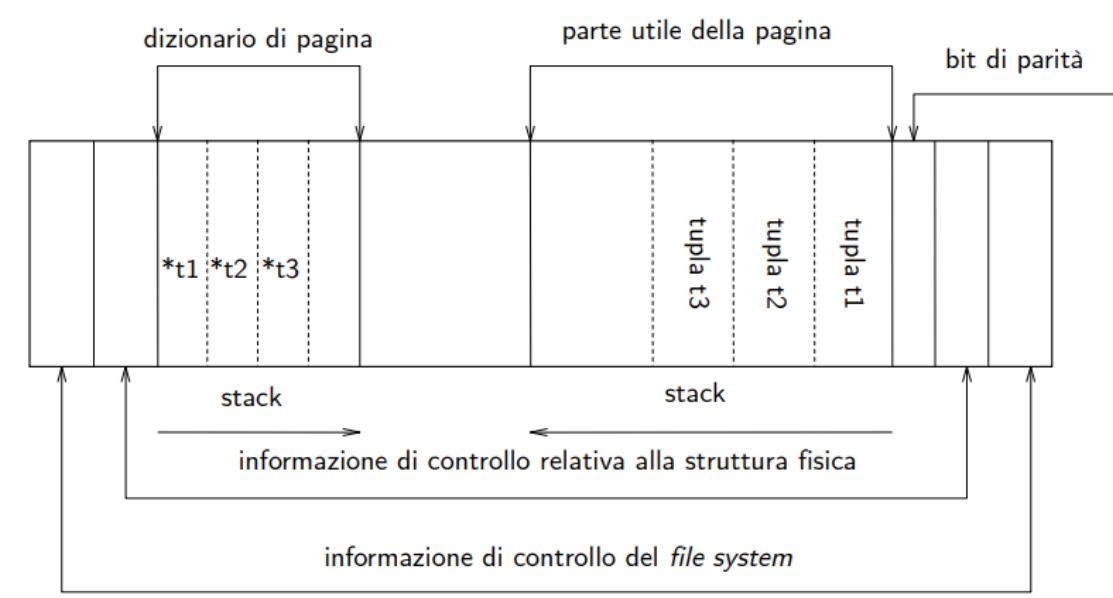
Se $L_B > L_R$ allora possiamo avere più record in un blocco:

$$\lfloor L_B / L_R \rfloor$$

Lo **spazio residuo** può essere:

- **utilizzato** (record “**spanned**”)
- **non utilizzato** (record “**unspanned**”)

Organizzazione delle *n-uple* nelle pagine



Strutture sequenziali

Esiste un ordinamento fra le *n-uple*, che può essere rilevante ai fini della gestione:

- **seriale**: ordinamento fisico ma non logico
- **ordinata**: l'ordinamento delle tuple coerente con quello di un campo
- **con accesso calcolato**: posizioni individuate attraverso indici

Struttura seriale

viene chiamata anche:

- *entry-sequenced*
- file heap
- file disordinato

È molto diffusa nelle **basi di dati relazionali**.

Gli inserimenti vengono effettuati:

- in coda (con riorganizzazioni periodiche)
- al posto di record cancellati

La sequenza delle *n-uple* è indotta dall'ordine di immissione.

Struttura ordinata

Permettono ricerche ordinarie, ma solo fino ad un certo punto (ad esempio, come troviamo la "metà del file")?

Il problema è mantenere l'ordinamento.

Struttura con accesso calcolato

I file hash permettono un **accesso calcolato** molto efficiente.

Infatti la tecnica si basa su quella utilizzata per le **tavole hash** in memoria centrale.

- **Tavola Hash**

L'obiettivo della **Tavola Hash** è l'accesso diretto ad un insieme di **record** sulla base del **valore di un campo** (detto **chiave**, che per semplicità supponiamo identificante, ma non è necessario).

Se i **possibili valori** della chiave sono in **numero paragonabile al numero di record** (e corrispondono ad un "*tipo indice*") allora usiamo un array.

Esempio: Università con 1000 studenti e numeri di matricola compresi fra 1 e 1000 o poco più e file con tutti gli studenti.

- Se i **possibili valori** della chiave sono **molti di più** di quelli **effettivamente utilizzati**, non possiamo usare l'array (spreco);

Esempio: 40 studenti e numero di matricola di 6 cifre (un milione di possibili chiavi).

Volendo continuare ad usare qualcosa di simile ad un array, ma **senza sprecare spazio**, possiamo pensare di trasformare i valori della chiave in possibili indici di un array.

In parole povere la **funzione hash**:

- associa ad ogni **valore** della chiave un "**indirizzo**", in uno spazio di dimensione paragonabile (leggermente superiore) rispetto a quello strettamente necessario
- poiché il **numero di possibili chiavi** è molto maggiore del **numero di possibili indirizzi** ("lo spazio delle chiavi è più grande dello spazio degli indirizzi"), la funzione **non può essere iniettiva** e quindi **esiste la possibilità di collisioni** (chiavi diverse che corrispondono allo stesso indirizzo)
- le **buone funzioni hash** distribuiscono in modo **casuale** e **uniforme**, **riducendo le probabilità di collisione** (che si riduce aumentando lo spazio ridondante)

Esempio:

- 40 record
- tavola hash con 50 posizioni:
 - 1 collisione a 4
 - 2 collisioni a 3
 - 5 collisioni a 2
- numero medio di accessi:
 - 32 record x 1 accesso +
 - 5 record x 2 accessi +
 - 2 record x 3 accessi +
 - 1 record x 4 accessi
 - 52 accessi / 40 record = 1.3 accessi in media

M	M mod 50	M	M mod 50
60600	0	200268	18
66301	1	205619	19
205751	1	210522	22
205802	2	205724	24
200902	2	205977	27
116202	2	205478	28
200604	4	200430	30
66005	5	210533	33
116455	5	205887	37
200205	5	200138	38
201159	9	102338	38
205610	10	102690	40
201260	10	115541	41
102360	10	206092	42
205460	10	205693	43
205912	12	205845	45
205762	12	200296	46
200464	14	205796	46
205617	17	200498	48
205667	17	206049	49

Gestione delle collisioni:

Ci sono varie tecniche:

- **posizioni successive** disponibili
- tabella di **overflow** (gestita in forma collegata)
- funzioni **hash alternative**.

Notare che:

- le collisioni ci sono (quasi) sempre
- le collisioni multiple hanno probabilità che decresce al crescere della molteplicità
- la molteplicità media delle collisioni è molto bassa.

File hash

L'idea è la stessa, ma si **sfrutta l'organizzazione in blocchi** ed il fatto che l'**accesso è al blocco**.

In questo modo si **ammortizzano le probabilità di collisione**.

Esempio:

- 40 record
- tavola hash con 50 posizioni:
 - 1 collisione a 4
 - 2 collisioni a 3
 - 5 collisioni a 2
- numero medio di accessi:
 - 32 record x 1 accesso +
 - 5 record x 2 accessi +
 - 2 record x 3 accessi +
 - 1 record x 4 accessi
 - 52 accessi / 40 record = 1.3 accessi in media

M	M mod 50	M	M mod 50
60600	0	200268	18
66301	1	205619	19
205751	1	210522	22
205802	2	205724	24
200902	2	205977	27
116202	2	205478	28
200604	4	200430	30
66005	5	210533	33
116455	5	205887	37
200205	5	200138	38
201159	9	102338	38
205610	10	102690	40
201260	10	115541	41
102360	10	206092	42
205460	10	205693	43
205912	12	205845	45
205762	12	200296	46
200464	14	205796	46
205617	17	200498	48
205667	17	206049	49

Esempio:

60600	66301	205802	200268	200604
66005	205751	200902	205478	201159
116455	115541	116202	210533	200464
200205	200296	205912	200138	205619
205610	205796	205762	102338	205724
201260		205617	205693	206092
102360		205667	200498	
205460		210522		
200430		205977		
102690		205887		
205845		206092		

- tavola hash con 50 posizioni:
 - 1 collisione a 4
 - 2 collisioni a 3
 - 5 collisioni a 2
- numero medio di accessi: 1.3
- file hash con fattore di blocco 10
 - 5 blocchi con 10 posizioni
 - 2 soli overflow
 - numero medio di accessi:
 - $(38 + 4) / 40 = 1.05$

Osservazioni

È l'organizzazione **più efficiente per l'accesso diretto basato su valori della chiave con condizioni di uguaglianza** (accesso puntuale).

Il **costo medio di poco superiore all'unità** (il caso peggiore è molto costoso ma talmente improbabile da poter essere ignorato).

Le **collisioni** (overflow) sono di solito gestite con **blocchi collegati**

Non è efficiente per ricerche basate **su intervalli** (né per ricerche basate su altri attributi).

I file hash “degenerano” se si riduce lo spazio sovrabbondante: funzionano solo con file la cui dimensione non varia molto nel tempo.

Indice di file

L'**indice di file** è una struttura ausiliaria per l'accesso (*efficiente*) ai record di un file sulla base dei **valori di un campo** (o di una *concatenazione di campi*) detto **chiave** (o meglio *pseudo-chiave* perché non è necessariamente identificante).

L'idea fondamentale è come se fosse **un indice analitico di un libro**.

Quindi possiamo dire che è una **lista di coppie ordinate**, alfabeticamente sui termini, posta in fondo al libro separabile da esso.

Un indice I di un file f è un altro file, con record a **due campi: chiave ed indirizzo** (dei record di f o dei relativi blocchi), ordinato secondo i valori della chiave.

Tipi di indice

Esistono diversi tipi di indice:

- **Indice primario:** che si trova su un **campo** sul cui ordinamento è **basata la memorizzazione**. Vengono anche chiamati **indici di cluster**, anche se talvolta si chiamano primari quelli su una chiave identificante e di cluster quelli su una chiave identificante
- **Indice secondario:** che si trova su un **campo con** ordinamento **diverso da quello di memorizzazione**.
- **Indice denso:** che contiene un record per ciascun valore del campo chiave.
- **Indice sparso:** che contiene un numero di record inferiore rispetto al numero di valori diversi del campo chiave

Caratteristiche degli indici

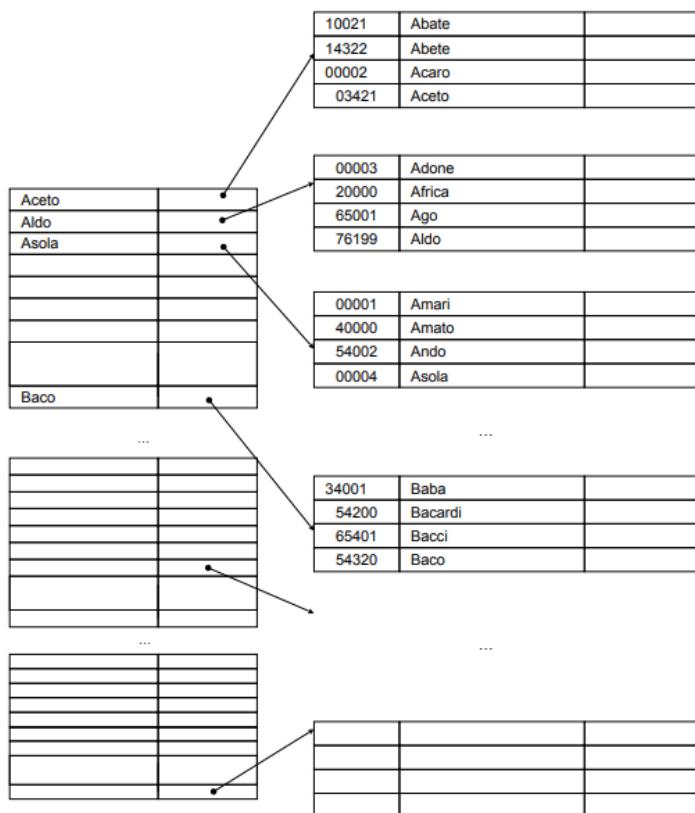
Le caratteristiche importanti sono:

- **Accesso diretto** (sulla chiave) **efficiente**, sia **puntale** sia per **intervalli**.
- **Scansione sequenziale ordinata efficiente**
- **Modifiche** della chiave, **inserimenti**, **eliminazioni inefficienti** (come nei file ordinati)
- **Tecniche per alleviare i problemi:**
 - marcatura per le eliminazioni
 - riempimento parziale
 - blocchi collegati (non contigui)
 - riorganizzazioni periodiche

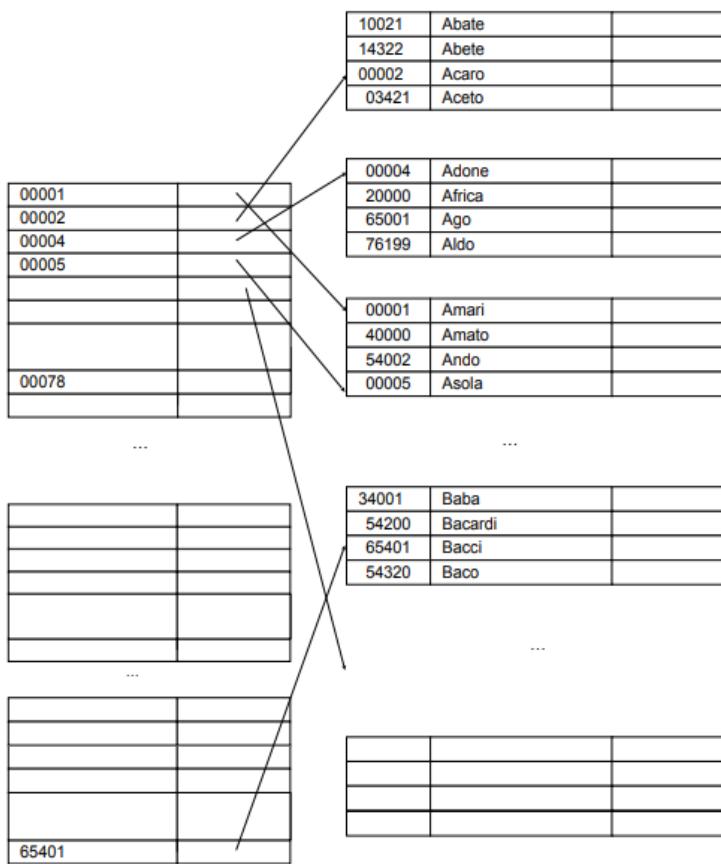
Commenti

- Un indice **primario può essere sparso**:
 - **Non tutti i valori** della chiave compaiono nell'indice;
Esempio, sempre rispetto ad un libro: indice generale
- Un **indice secondario deve essere denso**
 - **Tutti i valori** della chiave secondaria devono essere raggiungibili
Esempio, sempre rispetto ad un libro: indice analitico
- Ogni file può avere **al più un indice primario ed un numero qualunque di indici secondari** (su campi diversi)
Esempio: una guida turistica può avere l'indice dei luoghi e quello degli artisti.
- Un **file hash non può avere un indice primario**.

Indice primario



Indice secondario



Dimensione dell'indice

- T= numero di record nel file
- B= dimensione dei blocchi
- R= lunghezza dei record (fissa)
- K= lunghezza del campo chiave
- P= lunghezza degli indirizzi (ai blocchi)
- Numero di blocchi per il file (circa)

$$N_B = T \times R/B$$

- Numero di blocchi per un indice denso:

$$N_D = T \times (K+P)/B$$

- Numero di blocchi per un indice sparso:

$$N_S = N_B \times (K + P)/B$$

Indici Secondari

Si possono usare **puntatori ai blocchi** oppure **puntatori ai record**

- I puntatori ai **blocchi** sono **più compatti**
- I puntatori ai **record** permettono di effettuare alcune operazioni (effettuate solo sull'indice, senza accedere al file se non quando indispensabile)

Indici Multilivello

Gli indici sono file essi stessi e quindi ha senso costruire **indici sugli indici**, per evitare di fare ricerche fra blocchi diversi

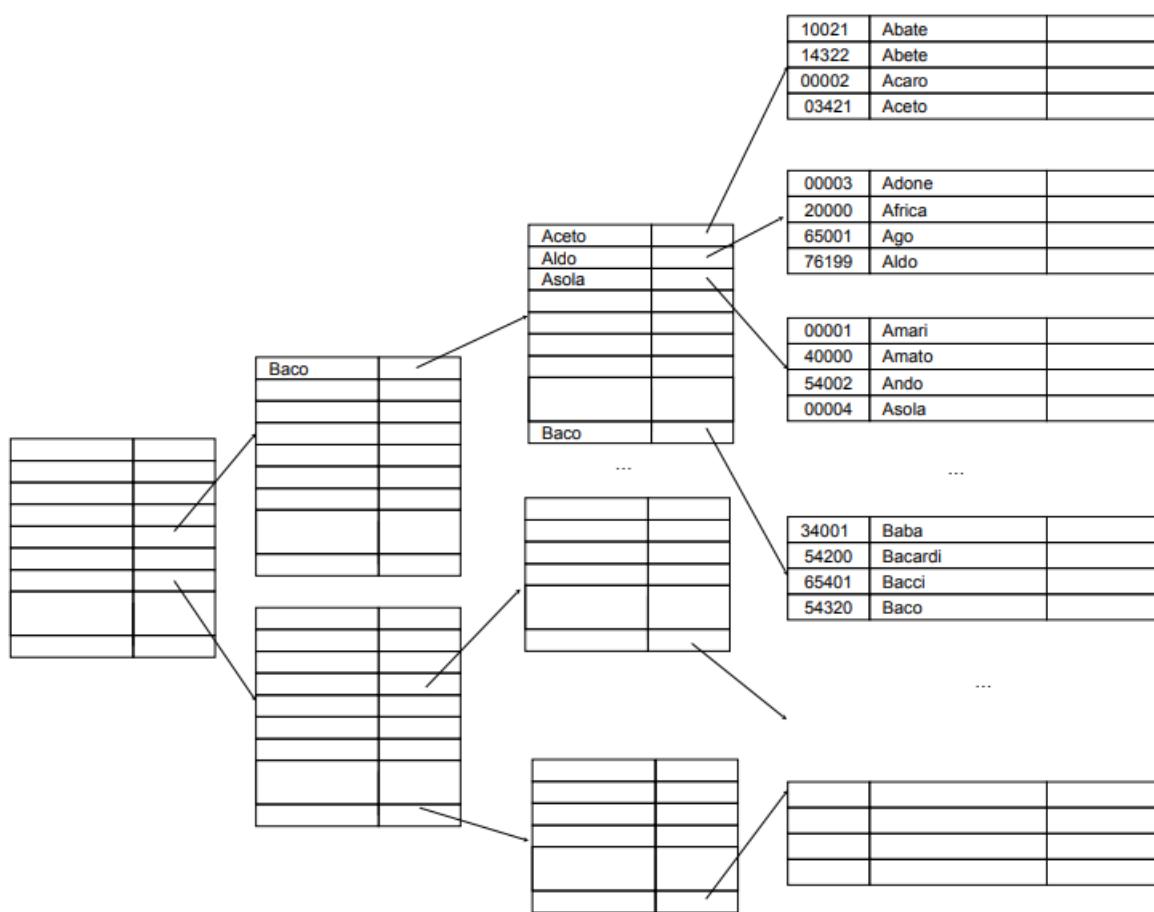
Possono esistere più livelli fino ad avere il livello più alto con un solo blocco; i **livelli** sono di solito **abbastanza pochi**, perché:

- l'indice è ordinato, quindi l'**indice sull'indice è sparso**
- i record dell'indice sono piccoli

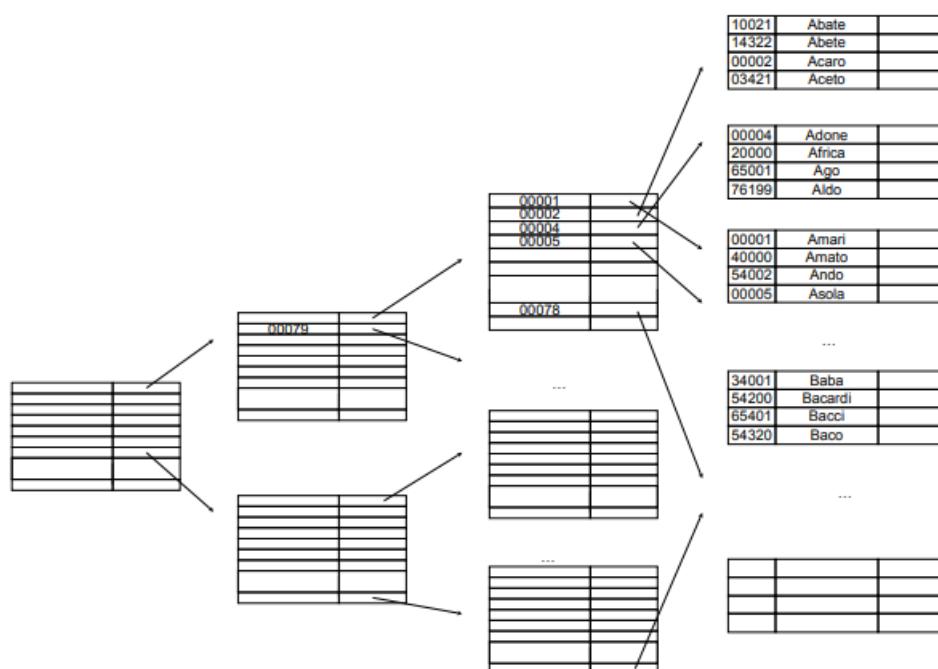
Numero di blocchi al livello j dell'indice (circa):

$$N_j = N_{j-1} \times (K+P) / B$$

Indice Primario Multilivello



Indice Secondario Multilivello



Problemi degli indici

Le strutture di indice basate su strutture ordinate sono poco flessibili in presenza di elevata dinamicità.

Gli indici usati dai DBMS sono in generale **indici dinamici multilivello** efficienti anche in caso di aggiornamenti

Vengono memorizzati e gestiti come *Binary Tree* (alberi di ricerca bilanciati)

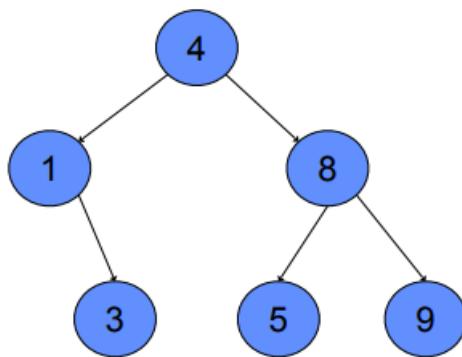
- Alberi binari di ricerca
- Alberi *n-ari* di ricerca
- Alberi *n-ari* di ricerca bilanciati

Albero Binario di ricerca

È un albero binario **etichettato** in cui per ogni nodo il **sottoalbero sinistro** contiene solo **etichette minori** di quella del nodo ed il **sottoalbero destro** solo **etichette maggiori o uguali** di quella del nodo

Tempo di ricerca (*inserimento*) pari alla profondità

- **Logaritmico** nel caso medio (assumendo un ordine di inserimento casuale)



Albero di ricerca di ordine P

Ogni nodo ha fino a P figli e fino a P etichette, ordinate.

Nell'*i-esimo* sottoalbero abbiamo tutte etichette maggiori o uguali della *(i-1)-esima* etichetta e minori della *i-esima*.

Ogni ricerca o modifica comporta la visita di un **cammino radice-foglia**.

In strutture fisiche, un **nodo** può corrispondere ad un **blocco**.

Il **legame** tra nodi è dato da **puntatori** che collegano le **pagine**

Ogni nodo ha un numero di discendenti abbastanza grande per cui gli alberi hanno un **numero limitato di livelli**.

La maggior parte delle pagine è nei nodi foglia.

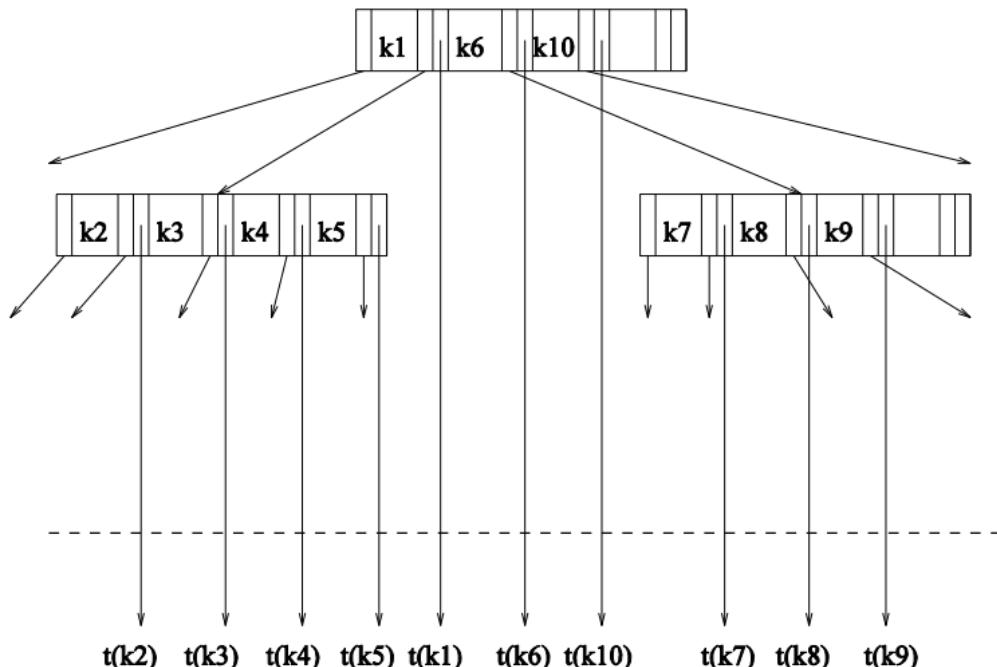
Gli **alberi** sono **bilanciati**.

Binary Tree (B-Tree)

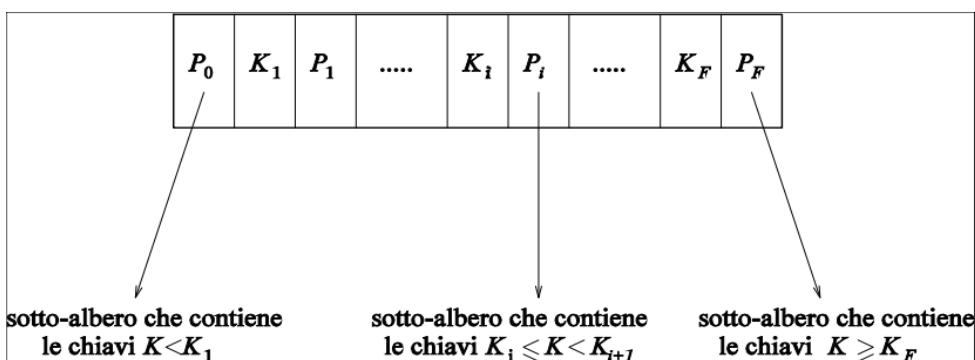
Un **Binary Tree** è un albero di ricerca che viene mantenuto bilanciato grazie a:

- Riempimento parziale (mediamente 70%)
- Riorganizzazioni (locali) in caso di sbilanciamento

Esempio di B-Tree



Organizzazione dei nodi nel B-Tree



- Sequenza di F valori ordinati di chiave
- Ogni etichetta K_i seguita da un puntatore P_i
- F dipende dall'ampiezza della pagina e dalla dimensione occupata dai valori di chiave e di puntatore

Ricerca nel B-Tree

Dato un valore V, si seguono i puntatori partendo dalla radice.

Ad ogni nodo intermedio:

- Se $V < K_1$ si segue il puntatore P_0
- Se $V \geq K_F$ si segue il puntatore P_F

Altrimenti si segue il puntatore P_j tale che $K_j \leq V < K_{j+1}$

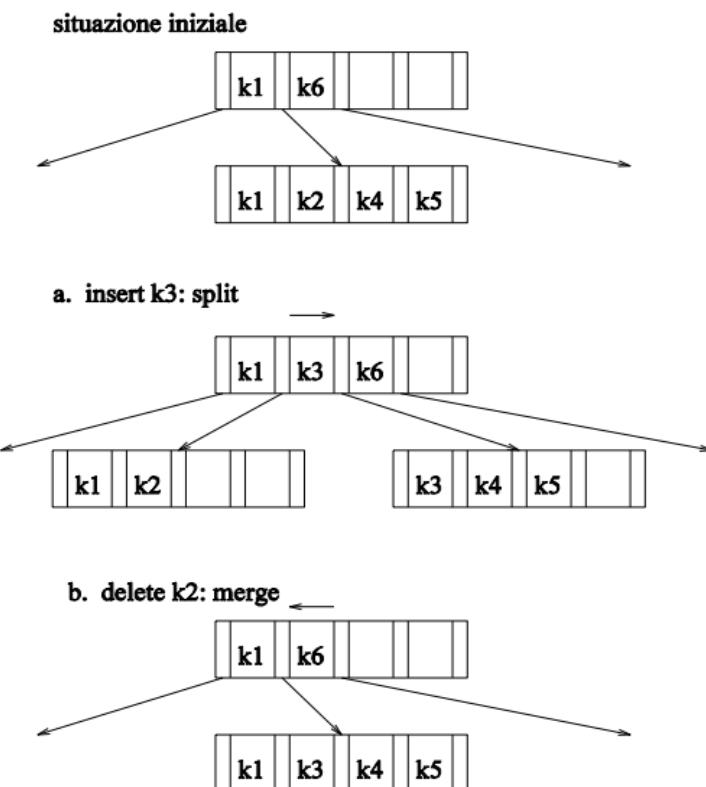
La ricerca prosegue fino ai nodi foglia dell'albero.

Inserimento ed eliminazione nel B-Tree

Inserimenti ed eliminazioni provocano aggiornamenti degli indici e sono precedute da una ricerca fino ad una foglia.

- Per gli inserimenti, se c'è posto nella foglia, ok, altrimenti il nodo va suddiviso (*split*), con necessità di un puntatore in più per il nodo genitore.
Se non c'è posto, si sale ancora, eventualmente fino alla radice. Il riempimento rimane sempre superiore al 50%.
- Per le eliminazioni, è possibile avere una riduzione di nodi (*merge*)
- Modifiche del campo chiave vanno trattate come eliminazioni seguite da inserimenti

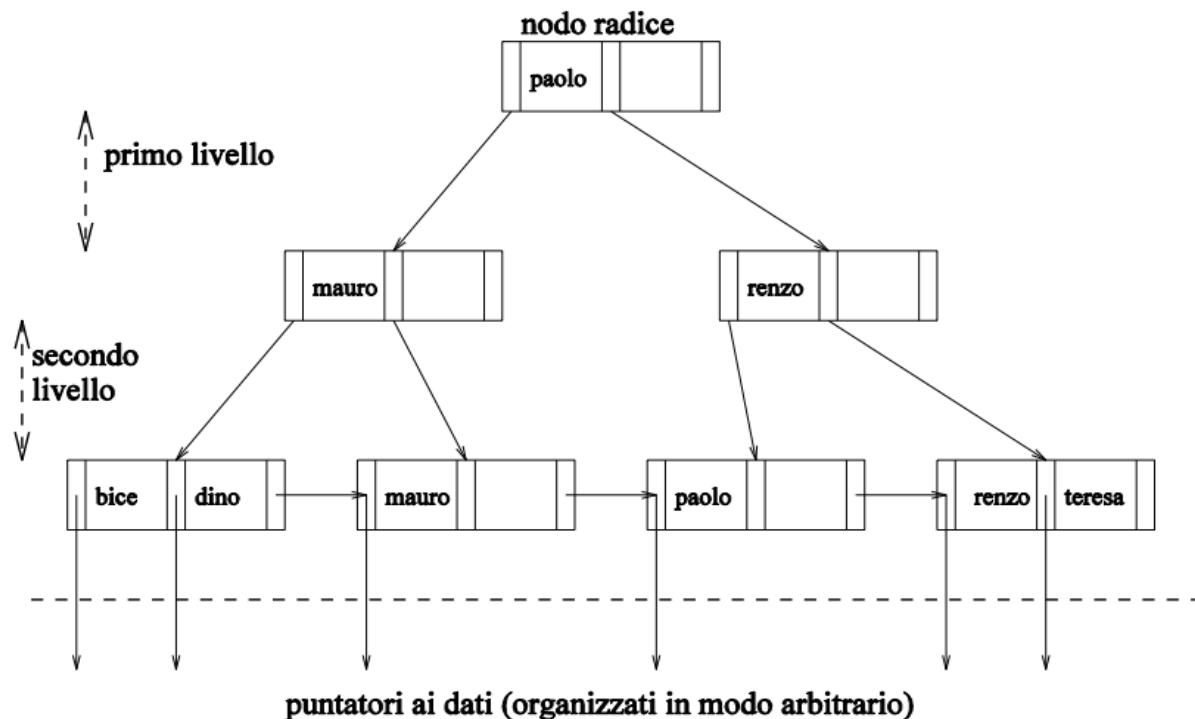
Esempio:



B Tree e B+ Tree (differenze)

- **B+ Tree:**
 - le foglie sono collegate in una lista
 - sono ottimi per le ricerche su intervalli
 - sono usati molto nei DBMS
- **B Tree:**
 - I nodi intermedi possono avere puntatori direttamente ai dati

Esempio di B+ Tree



Lezione 10: Ottimizzazione delle Interrogazioni

Ottimizzazione delle Interrogazioni

Query processor (od ottimizzatore) è un modulo del DBMS.

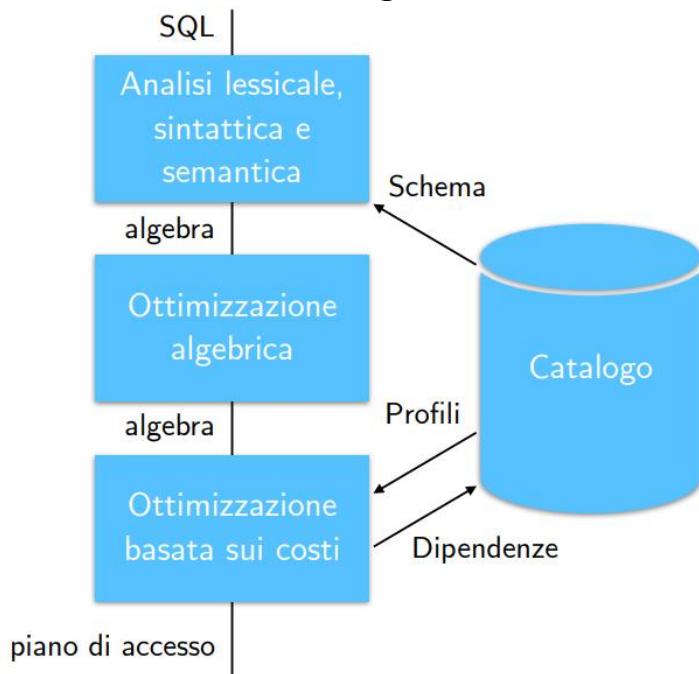
È più importante nei sistemi attuali che in quelli “vecchi” (gerarchici e reticolari):

le interrogazioni sono espresse **ad alto livello** (ricorda il concetto di indipendenza dei dati):

- insiemi di *n-uple*
- poca proceduralità

L’ottimizzazione sceglie la **strategia realizzativa** (di solito fra diverse alternative) a partire dall’istruzione SQL.

Esecuzione delle Interrogazioni



Profili delle Relazioni

Informazioni quantitative:

- **cardinalità** di ciascuna relazione
- **dimensione** delle *n-uple*
- **dimensione** dei valori
- **numero** di valori distinti degli attributi
- valore **minimo** e **massimo** di ciascun attributo.

Sono **memorizzate** nel *catalogo* ed **aggiornate** con comandi del tipo update statistics.

Sono utilizzate nella **fase finale** dell'ottimizzazione, per **stimare le dimensioni** dei **risultati intermedi**.

Ottimizzazione Algebrica

In realtà il termine **ottimizzazione** è **improprio** (anche se efficace) perché il processo utilizza **euristiche**. Si basa sulla nozione di **equivalenza**, quindi:

Due espressioni sono equivalenti se producono lo stesso risultato qualunque sia l'istanza attuale della base di dati.

I DBMS cercano sempre di eseguire espressioni equivalenti a quelle date, ma **meno costose**.

Euristica fondamentale:

- selezioni e proiezioni il più presto possibile (per **ridurre le dimensioni dei risultati intermedi**):
 - Push selections down
 - Push projections down

Esecuzione delle operazioni

I DBMS implementano gli operatori dell'algebra relazionale (le loro combinazioni) per mezzo di operazioni di livello abbastanza basso.

Gli operatori fondamentali sono:

- **accesso diretto**
- **scansione**

A livello più alto invece:

- **join**, che è l'operazione più costosa.

Accesso Diretto

L'accesso diretto può essere eseguito solo se le strutture fisiche lo permettono:

- **indici**
- **strutture hash**

Accessi diretto basato su indice

È efficace per le interrogazioni, sulla “chiave dell’indice” che possono essere:

- **puntuali** ($A_i = v$)
- **su intervallo** ($v_1 \leq A_i \leq v_2$)
- **Per predicati congiuntivi:** si sceglie il più selettivo per l’accesso diretto e si verifica poi sugli altri dopo la lettura (e quindi i memoria centrale)
- **Per i predicati disgiuntivi:** servono indici su tutti, ma conviene usarli se molto selettivi e facendo attenzione ai duplicati

Accesso diretto basato su hash

È efficace per interrogazioni (sulla “chiave dell’indice”)

- **puntuali** ($A_i = v$)
- **NON su intervallo** ($v_1 \leq A_i \leq v_2$)

Per **predicati congiuntivi e disgiuntivi**, vale lo stesso discorso fatto per gli indici.

Join

Il join è l’operazione più costosa e viene effettuata su vari metodi, tra cui quelli più noti:

- **nested-loop**
- **merge-scan**
- **hash-based**

Nested Loop

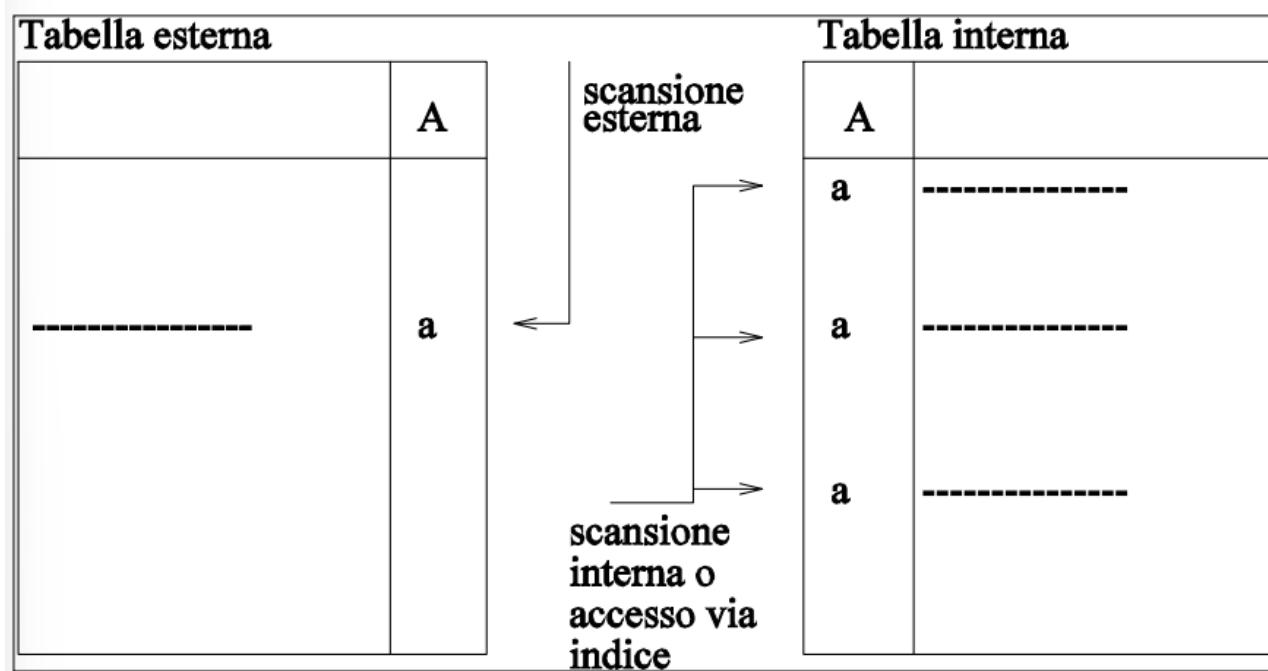
Per ogni *n-upla* nella tabella esterna si esaminano tutte le *n-uple* di quella interna per verificare la condizione di join.

Date R ed S, si hanno due possibilità:

- **R esterna**
- **R interna**

Il costo, in termini di trasferimenti in memoria, dipende dal numero di accessi e dal fatto che la tabella interna è piccola

NB: Il nested loop può sopportare il left outer join e l’inner join.

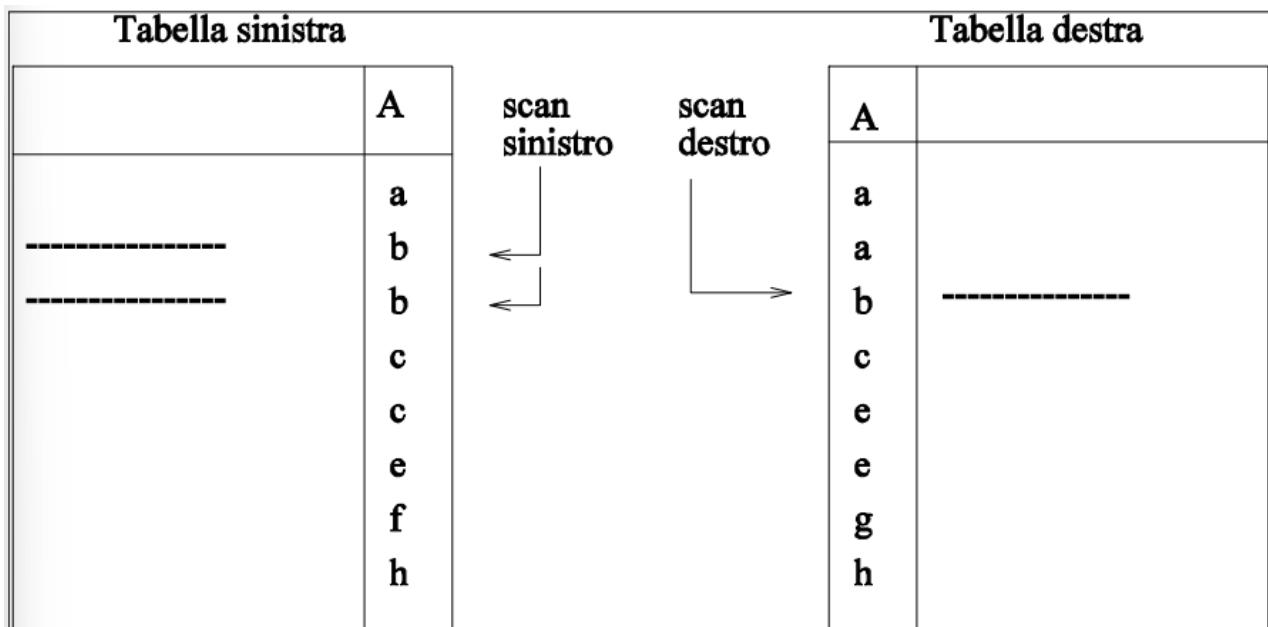


Merge Scan

Il merge Scan funziona così:

- Si ordinano le tabelle in base agli attributi di Join.
- Si trova l'elemento della seconda tabella da cui partire rispetto al primo elemento della prima tabella e poi si continua da lì.

Il costo è l'ordinamento.



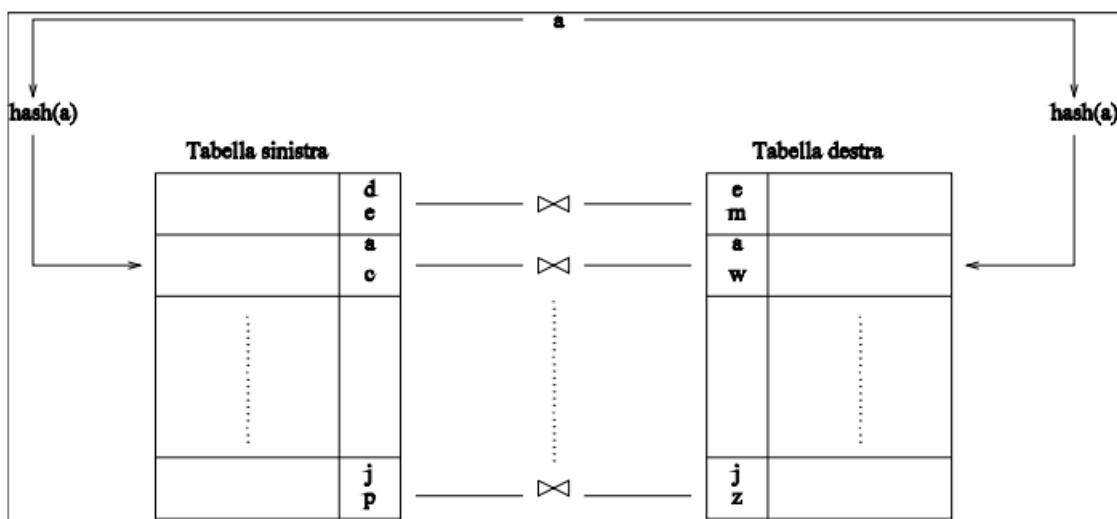
Hash Join

L'hash join è utile solo per **join naturale ed equi-join**.

La funzione hash h sull'attributo di Join è usata per **partizionare le n-uple** di entrambe le relazioni.

Le partizioni sono inserite in tabelle aggiuntive, h produce partizioni di S tali che ognuna sta in memoria, R è partizionata di conseguenza.

Serve memoria ed è migliore nel **nested loop** nel caso di *equi-join*



Ottimizzazione basata sui costi

Quindi un problema si dice articolato con scelte relative a:

- **operazioni da eseguire**, Esempio scansione o accesso diretto
- **ordine delle operazioni**, Esempio: join di tre relazioni
- **dettagli del metodo**, Esempio: quale metodo di join

Le architetture parallele e distribuite aprono ulteriori **gradi di libertà**.

Misura del costo di una query

Ci sono molti fattori che contribuiscono al costo di una query, ovvero il tempo necessario per avere la risposta, tra questi sono:

- Gli **accessi al disco**, il tempo di CPU o il tempo di rete
L'accesso dal disco è il tempo predominante ed è anche facilmente calcolabile tenendo conto di:
 - numero scansioni
 - numero di letture
 - numero di scritture
- Oltre al **numero di trasferimenti**, bisogna considerare anche la memoria che serve per memorizzare i risultati intermedi.
Esempio: l'hash join necessita di tabelle intermedie
- **Operazioni a più operandi** che devono essere eseguite un passo alla volta.

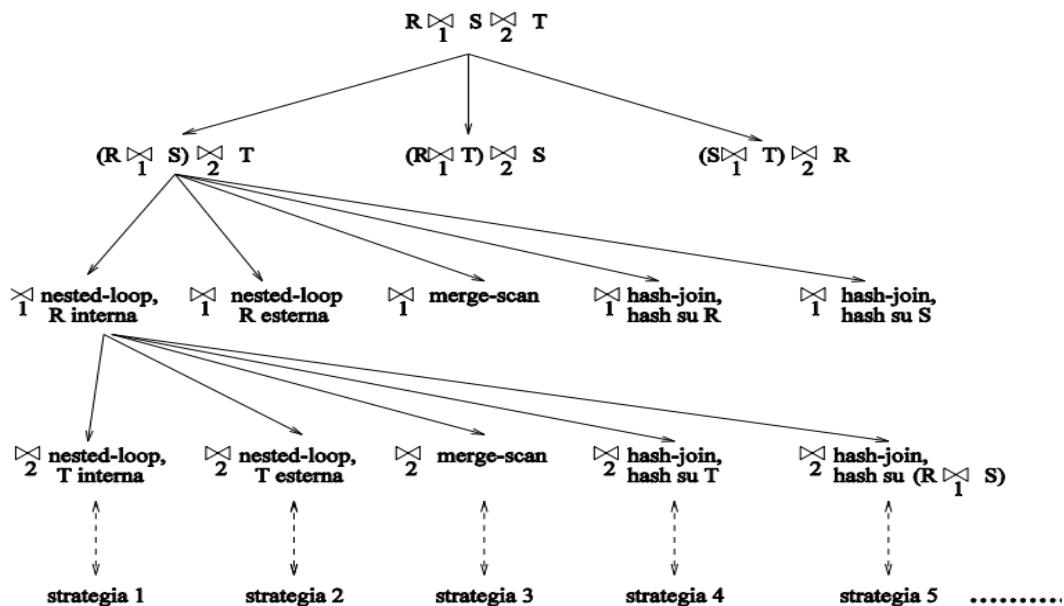
Processo di ottimizzazione

Il processo di ottimizzazione avviene nel seguente modo:

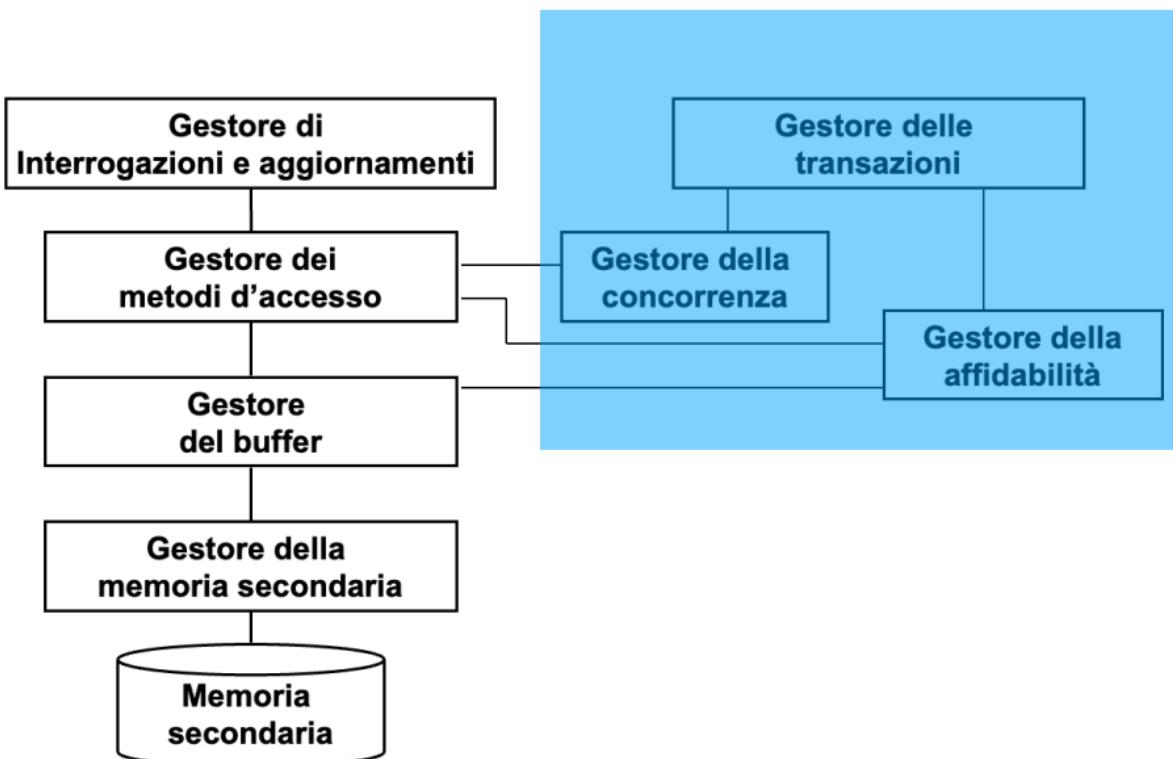
- si costruisce un albero di decisione con le varie alternative, *piani di esecuzione*.
- si valuta il costo di ciascun piano
- si sceglie il piano di costo minore

Considera che l'**ottimizzatore** trova di solito una *buona soluzione* **non** necessariamente la *soluzione ottima*.

Esempio:



Lezione 11: Gestione Transazioni



Sistemi Multiutente

Un criterio per classificare un sistema di basi di dati è il numero degli utenti che possono fruirne simultaneamente.

Un DBMS è **monoutente** se il sistema può essere usato al massimo da un utente alla volta.

Un DBMS è **multiutente** se invece può essere usato contemporaneamente da più utenti, generando accessi concorrenti alla base di dati.

La maggior parte dei DBMS è multiutente.

Multitasking

L'accesso alla base di dati e più in generale l'utilizzo dei sistemi di elaborazione da parte di più utenti contemporaneamente è possibile grazie al concetto di multitasking.

Il multitasking consente al calcolatore di eseguire più programmi (processi) nello stesso tempo.

Se esiste una sola CPU questa è in realtà in grado di eseguire un processo alla volta, tuttavia i sistemi operativi multitasking eseguono alcuni comandi di un processo, poi lo sospendono per eseguirne altri, e così via.

L'esecuzione di un processo **viene ripresa nel punto in cui è stata sospesa ogniqualvolta che il processo torna ad usare la CPU**.

L'esecuzione concorrente **dei processi risulta quindi alternata**.

Se il sistema è dotato di più CPU è possibile realizzare una qualche forma di elaborazione parallela di più processi.

Transazioni

Una **transazione** identifica una unità elementare di lavoro svolta da un'applicazione, cui si vogliono associare particolari caratteristiche di correttezza, robustezza ed isolamento.

Vediamo un'altra definizione, più carina:

Una **transazione** è una successione di comandi/operazioni in esecuzione che forma un'unità di elaborazione sulla base di dati.

Quindi una transazione comprende **una o più operazioni di accesso** alla base di dati:

- inserimenti, cancellazioni, modifiche **o** interrogazioni
- quindi una sequenza di letture (read) **e** scritture (write).

Un sistema che mette a disposizione un meccanismo per la definizione e l'esecuzione di transazioni è detto **sistema transazionale**.

Un **sistema transazionale** è in grado di definire ed eseguire transazioni per conto di applicazioni concorrenti.

Esempio:

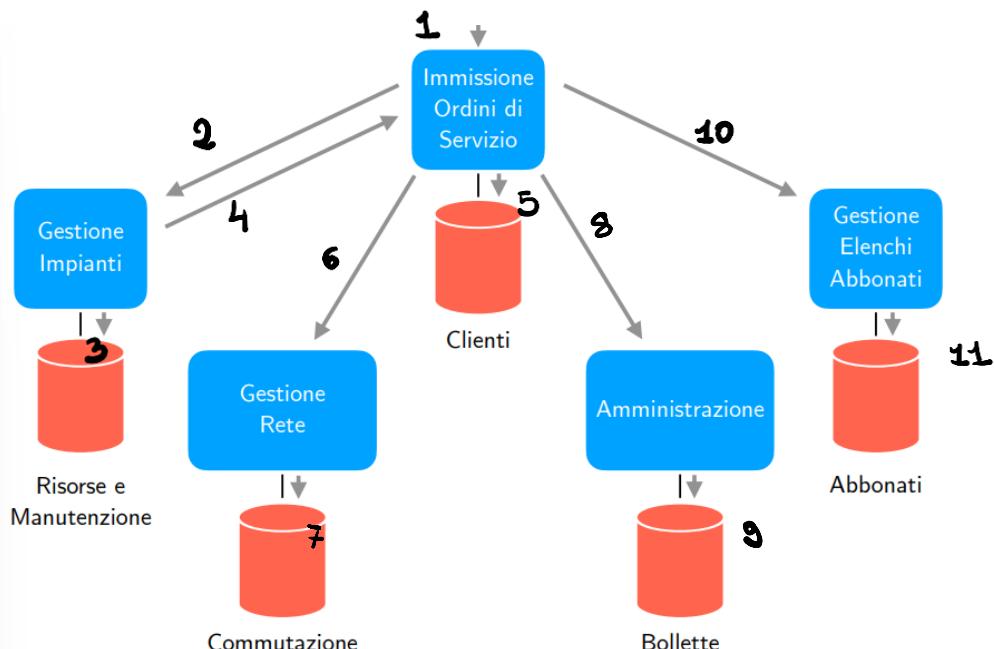


Figura 45: Esempio di sistema transazionale

Definizione di Transazione

Una **transazione** è parte di programma caratterizzata da un inizio, una fine ed al cui interno deve essere eseguito una ed una sola volta una dei seguenti comandi:

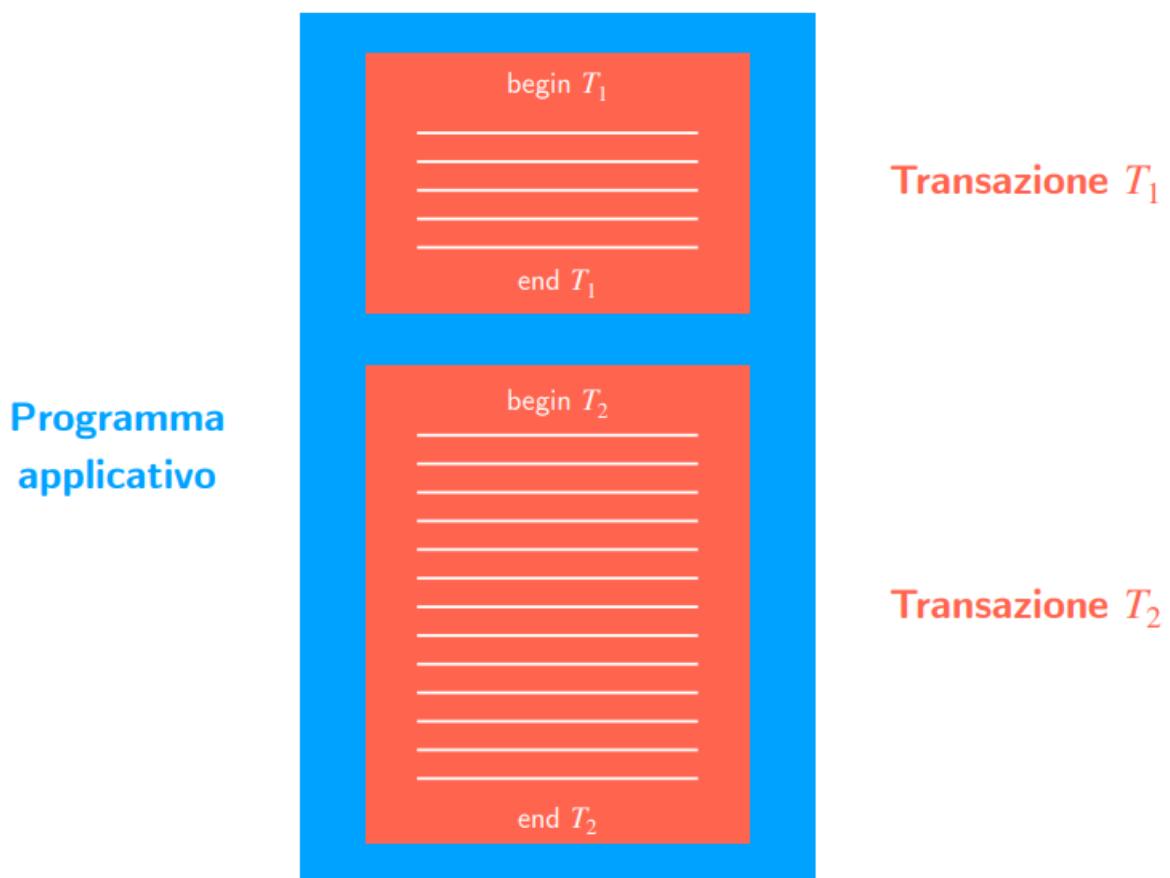
- **commit** per terminare correttamente
- **rollback/abort** per abortire la transazione

In SQL:

- **inizio:** begin-transaction, start transaction
- **fine:** end-transaction, non esplicitata

Un sistema transazionale (online transaction processing **OLTP**) è in grado di definire ed eseguire transazioni per conto di un certo numero di applicazioni concorrenti.

Differenza tra Programma e Transazione



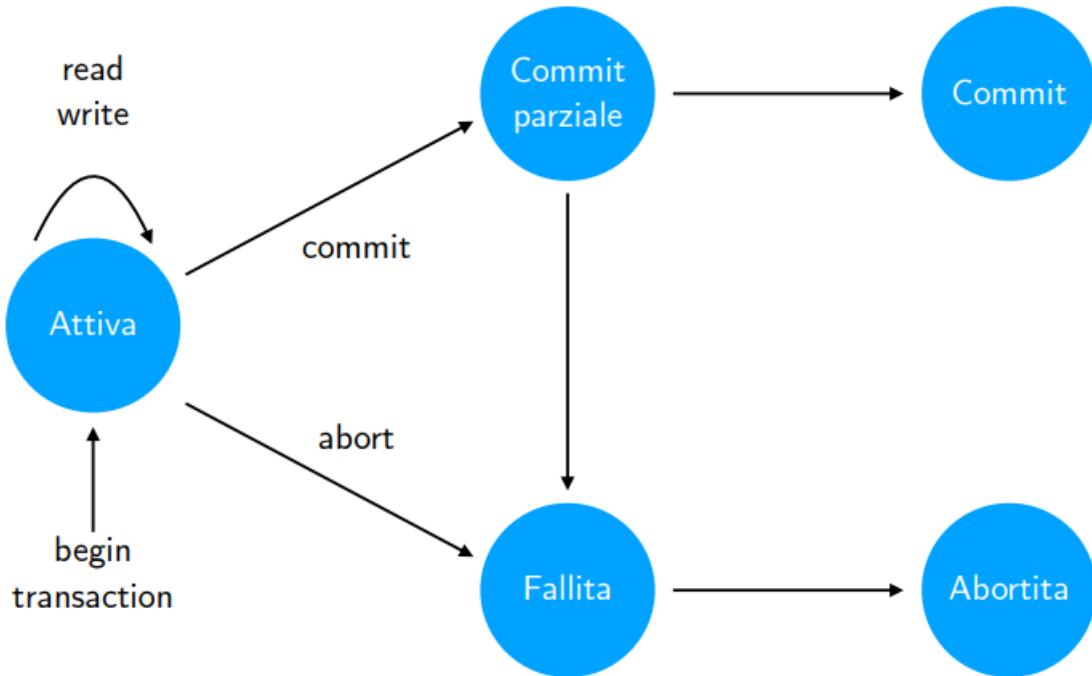
Esempio di transazione:

```
start transaction;  
update ContoCorrente  
    set Saldo = Saldo + 10  
        where NumConto = 12202;  
update ContoCorrente  
    set Saldo = Saldo - 10  
        where NumConto = 42177;  
commit work;
```

Vediamo un altro esempio di transazione:

```
start transaction;  
update ContoCorrente  
    set Saldo = Saldo + 10 where NumConto = 12202;  
update ContoCorrente  
    set Saldo = Saldo - 10 where NumConto = 42177;  
select Saldo as A  
    from ContoCorrente  
    where NumConto = 42177;  
if (A >= 0) then commit work  
    else rollback work;
```

Stato di una Transazione



Una transazione entra nello **stato attivo** subito dopo essere iniziata, dove rimane per tutta la sua esecuzione.

Una transazione si sposta nello **stato commit parziale** quando termina.

Una transazione si sposta nello **stato commit** se è stata eseguita con successo e tutti i suoi aggiornamenti alla base di dati sono permanenti.

Una transazione si sposta nello **stato fallito** quando non può passare da **commit parziale a commit** o se è stata interrotta a seguito ad un fallimento mentre era nello **stato attivo**.

Una transazione si sposta nello **stato abortito** se è stata interrotta e tutti i suoi aggiornamenti alla base di dati sono stati annullati.

Proprietà delle Transazioni

- **Atomicità**
- **Consistenza**
- **Isolamento**
- **Durabilità (o persistenza)**

Atomicità

Una transazione è una **unità atomica di elaborazione**, quindi non può lasciare la base di dati in uno stato intermedio:

- un guasto o un errore prima del **commit** debbono causare l'annullamento (UNDO) delle operazioni svolte;
- un guasto o errore dopo il **commit** non deve avere conseguenze;
- se necessario vanno ripetute (REDO) le operazioni.

Vediamo l'esito della transazione:

- **Commit**: caso normale e più frequente (99%)
- **Rollback (Abort)**: richiesto dall'applicazione e dal sistema e si entra nel caso di violazione dei vincoli, concorrenza, incertezza in caso di fallimento.

Consistenza

La transazione rispetta i vincoli di integrità se lo **stato iniziale** è corretto allora anche lo **stato finale** è corretto.

Conseguenza:

- all'**inizio** ed alla **fine** della transazione il sistema è in uno **stato consistente**;
- durante l'esecuzione della transazione stessa il sistema può temporaneamente essere in uno **stato inconsistente**.

Isolamento

La transazione non risente degli effetti delle altre transazione concorrenti.

L'esecuzione concorrente di una collezione di transazioni deve produrre un risultato che si potrebbe ottenere con una esecuzione sequenziale.

Conseguenza:

- una transazione non espone i suoi **stati intermedi**, si evita quindi un effetto domino.

Durabilità (o persistenza)

Gli effetti di una transazione andata in commit non vanno perduti ("durano per sempre"). Conseguenza:

- i guasti non hanno effetto sullo stato del database, infatti uno **stato consistente** sarà sempre recuperato.

Gestore della Affidabilità

Gestisce l'esecuzione dei comandi transazionali che sono:

- **Start transaction** (B, begin)
- **Commit work** (C)
- **Rollback work** (A, abort)

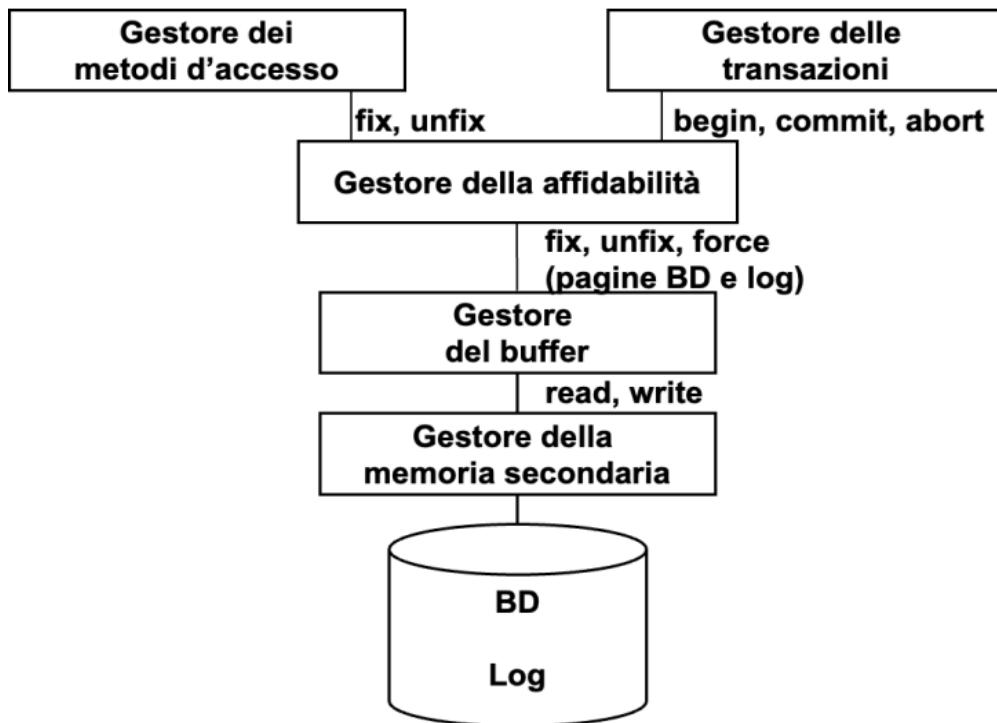
Gestisce anche le operazioni di ripristino(recovery) dopo i guasti:

- **ripresa a caldo** (warm restart)
- **ripresa a freddo** (cold restart)

Assicura **atomicità e durabilità**.

Infine usa il **log** che è un archivio permanente che registra le operazioni svolte.

Architettura del gestore dell'affidabilità



Persistenza delle memorie

- **Memoria centrale**: non è persistente, l'informazione viene distrutta da qualunque guasto del sistema.
- **Memoria di massa**: è persistente, sopravvive ai guasti di sistema, ma può danneggiarsi l'unità di memorizzazione.
- **Memoria stabile**: memoria che non può danneggiarsi (è una astrazione) ed è perseguita attraverso la ridondanza: dischi replicati, nastri... con probabilità di fallimento indipendenti.

Log

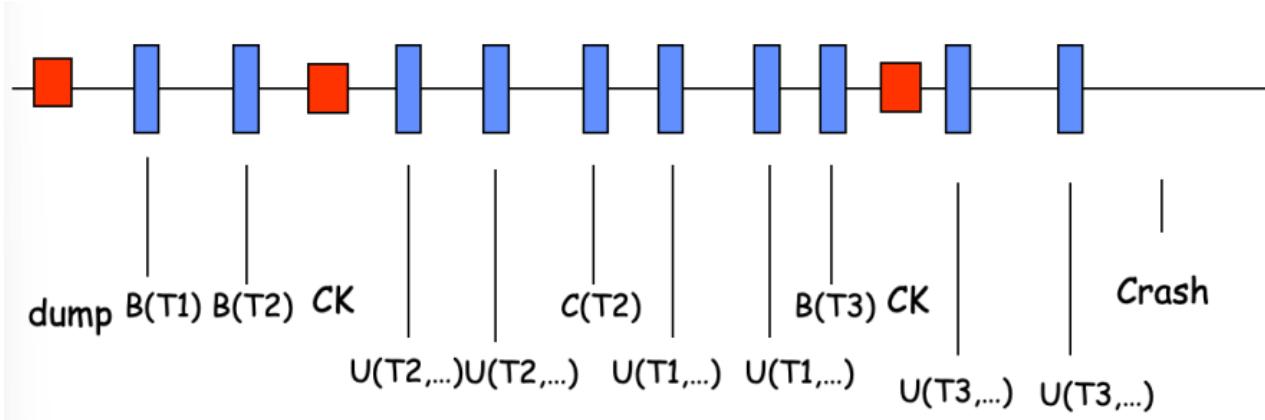
Il **log** è un file sequenziale gestito dal gestore dell'affidabilità, scritto **in memoria stabile**. Possiamo dire che è un "Diario di bordo" dove riporta tutte le operazioni in **ordine**. Pertanto il log ha un blocco corrente (top), l'ultimo a essere stato allocato al log stesso. Esistono due tipi di record del log:

- **Record di Transazione:** registrano le attività svolte da ciascuna transazione nell'ordine in cui esse vengono effettuate.
 - $\text{begin}, B(T)$
 - $\text{insert}, I(T, O, AS)$
 - nel record O , T memorizza per la prima volta l'**after state** AS
 - $\text{delete}, D(T, O, BS)$
 - nel record O , T elimina per sempre il **before state** BS
 - $\text{update}, U(T, O, BS, AS)$
 - nel record O , T elimina il **before state** BS e memorizza l'**after state** AS
 - $\text{commit}, C(T)$
 - $\text{abort}, A(T)$
- **Record di sistema:** indicano l'effettuazione di operazioni specifiche del controllore dell'affidabilità
 - dump (abbastanza rara)
 - checkpoint (frequente)

Esempio di log

- begin transaction $B(T_1)$
- $w[A]$ $A = 50$ $I(T_1, A, 50)$
- $w[A]$ $A = 20$ $U(T_1, A, 50, 20)$
- $r[A]$ $A = 20$ nessuna modifica
- $r[B]$ $B = 50$ nessuna modifica
- $w[B]$ $B = 80$ $U(T_1, B, 50, 80)$
- commit $C(T_1)$

Struttura del Log



Checkpoint e Dump

Il log serve a “**ricostruire**” le operazioni.

Checkpoint e **Dump** servono ad evitare che la ricostruzione debba partire dall'inizio dei tempi, infatti, si usano con riferimento a tipi di guasti diversi.

Checkpoint

L'operazione di **checkpoint** serve a "fare il punto" della situazione, semplificando le successive operazioni di ripristino. Ha lo scopo di **registrare** quali transazioni sono attive in un certo istante, cioè le transazioni “*a metà strada*” e di **confermare** che le altre o non sono **iniziate** o sono **finite**. Per tutte le transazioni che hanno **effettuato** il **commit** i **dati** possono essere **trasferiti** in **memoria di massa**.

Ci sono **varie modalità**, vediamo la più semplice:

- Si sospende **l'accettazione** delle operazioni di **commit** o **abort** da parte delle transazioni.
- Si forza (**force**) la **scrittura in memoria di massa** delle pagine in memoria modificate da **transazioni** che hanno già fatto **commit**.
- Si forza (**force**) la **scrittura nel log** di un record contenente gli identificatori delle **transazioni attive**
- Si riprendono **ad accettare** tutte le operazioni da parte delle transazioni.

Con questo funzionamento si **garantisce la persistenza** delle transazioni che hanno eseguito il **commit**.

Dump

Copia completa ("di riserva", backup) della base di dati

- Solitamente prodotta mentre **il sistema non è operativo**
- Salvato in **memoria stabile**, come backup
- Un **record di dump** nel log indica il momento in cui il log è stato effettuato

Esito di una transazione

L'esito di una transazione è **determinato irrevocabilmente** quando viene scritto il **record di commit** nel log in modo **sincrono**, con una force.

- Un **guasto prima** di tale istante **porta ad un UNDO** di tutte le azioni, per ricostruire lo stato originario della base di dati.
- Un **guasto successivo** non deve avere conseguenze: lo **stato finale** della base di dati deve essere ricostruito con **REDO** se necessario.

I **record di abort** possono essere scritti in modo **asincrono**.

Regole di modifica del Log

- **Regola Write-Ahead-Log (WAL)** letteralmente scrivi il log per primo:
 - Si scrive la **parte BS** dei record del log prima di effettuare la corrispondente operazione sul database.
 - Consente di **disfare le azioni** già memorizzate (UNDO) di transazioni senza commit avendo in memoria stabile un valore corretto.
- **Regola Commit-Precedenza:**
 - si scrive la **parte AS** dei record di log prima del commit
 - consente di **rifare le azioni** (REDO) di una transazione che ha effettuato il commit ma le cui pagine modificate non sono ancora state trascritte in **memoria di massa**.

Operazione UNDO e REDO

- **Undo di una azione su un oggetto O:**
 - update, delete: copiare il valore del Before State (BS) nell'oggetto O
 - insert: eliminare O
- **Redo di una azione su un oggetto O:**
 - insert, update: copiare il valore del After state (AS) nell'oggetto O
 - delete: eliminare O
- **Idempotenza** di undo e redo:
 - $\text{undo}(\text{undo}(A)) = \text{undo}(A)$
 - $\text{redo}(\text{redo}(A)) = \text{redo}(A)$

E la base di dati?

Quando scriviamo nella base di dati? Ci sono diverse modalità.

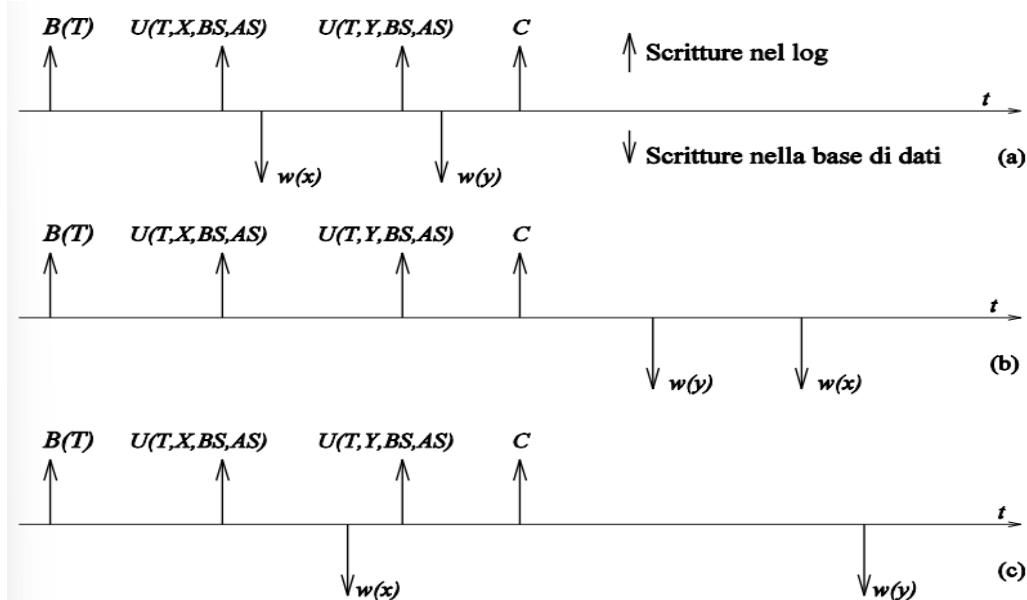
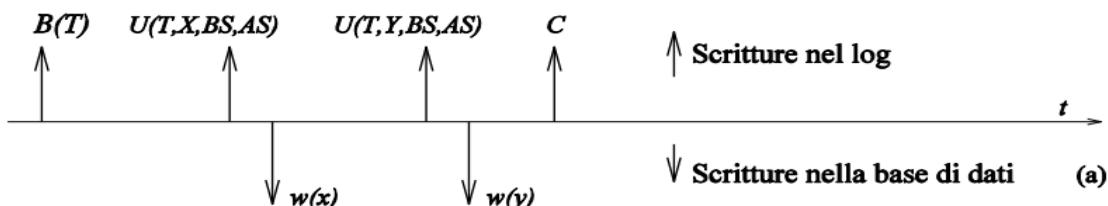


Figura 46: Esempi Scrittura nel log e nella base di dati

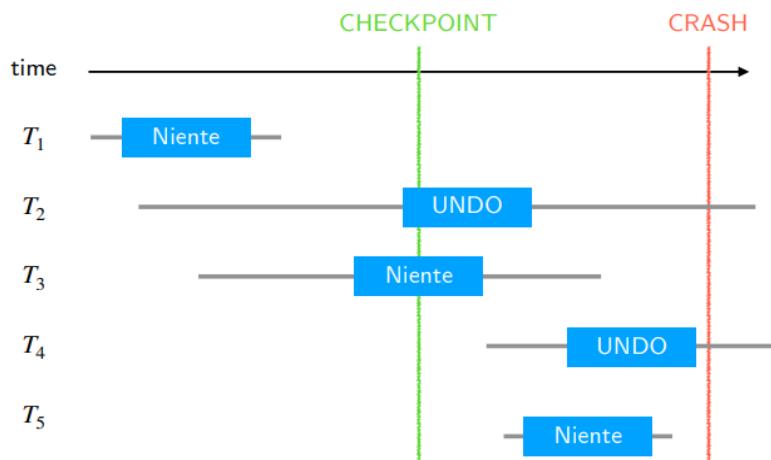
1. Modalità Immediata

La base di dati contiene i valori **AS** provenienti da transazioni uncommitted.

- Richiede **UNDO** dalle operazioni di transazioni uncommitted al momento del guasto.
- Non richiede **REDO**



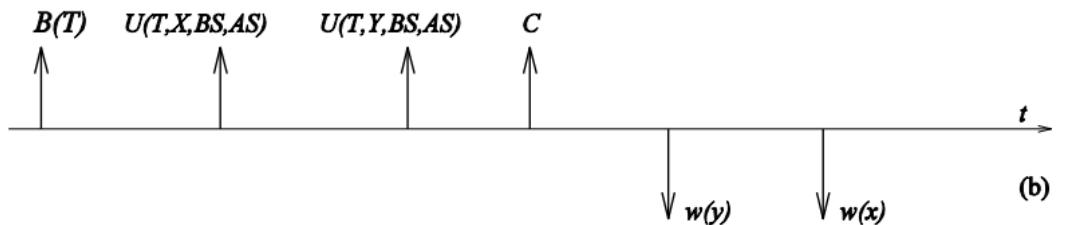
Esempio:



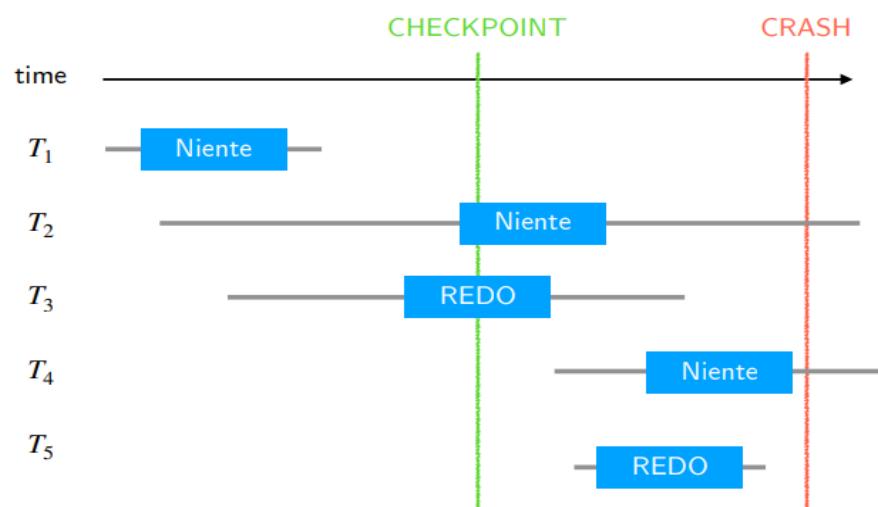
2. Modalità Differita

La base di dati non contiene valori AS provenienti da transazioni uncommitted.
In caso di abort, non occorre fare niente.

Rende superflua la procedura di UNDO, non ci sono scritture prima del commit.
Richiede REDO.



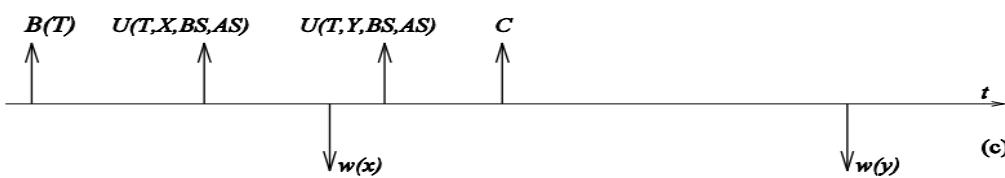
Esempio:



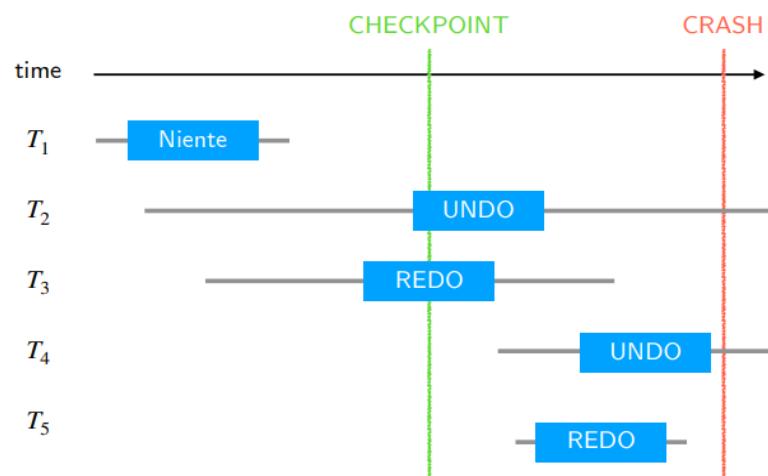
3. Modalità Mista

La scrittura può avvenire in modalità sia **immediata** che **differita**.

Infatti richiede sia UNDO che REDO.



Esempio:



Scrittura nel log e nella base di dati

La **modalità differita** di scrittura, pur permettendo una procedura di recupero più semplice ed efficiente, **non viene utilizzata in pratica**. Perché?

Questa modalità è **più efficiente** nel **recovery**, ma è complessivamente **meno efficiente** di una in cui il gestore può decidere liberamente quando **scrivere in memoria secondaria**. Quindi?

È preferibile una gestione ordinaria più efficiente rispetto ad una gestione più semplice dei guasti, poiché **si assume che i guasti siano abbastanza rari**.

Guasti

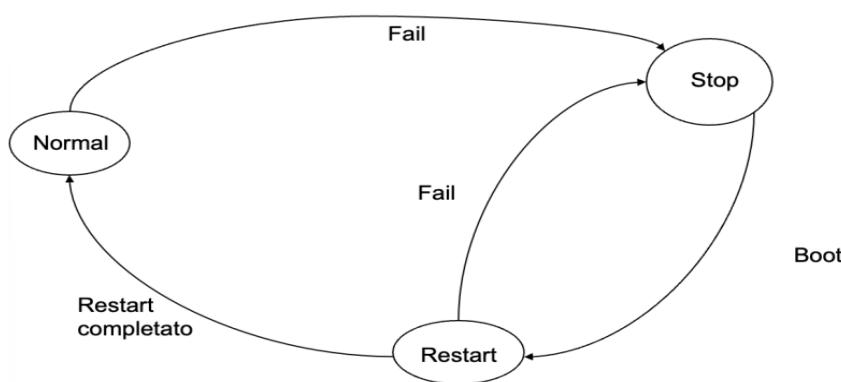
- **Guasti “soft”**: errori di programma, crash di sistema, caduta di tensione
 - **si perde la memoria centrale**
 - **non si perde la memoria secondaria**, cioè la base di dati
 - **non si perde la memoria stabile**, cioè il log
 - **warm restart, ripresa a caldo**
- **Guasti “hard”**: dei dispositivi di memoria secondaria
 - **si perde anche la memoria secondaria**, cioè parte della base di dati
 - **non si perde la memoria stabile**, cioè il **log**
 - **cold restart, ripresa a freddo**

La **perdita del log** è considerato un **evento catastrofico** e quindi non è definita alcuna strategia di recupero.

Modello di funzionamento Fail-Stop

L'individuazione di un **guasto** forza l'**arresto completo** delle transazioni. Avviene la seguente procedura:

- Il sistema operativo viene **riavviato**.
- Viene avviata una procedura di **restart**.
- Al termine del **restart** il **buffer** è vuoto, ma le transazioni possono ripartire.



Processo di restart

L'obiettivo di questo processo è di **classificare le transazioni** in:

- **completate** (tutti i dati in memoria stabile)
- in **commit** ma **non necessariamente completate** (può servire REDO)
- **senza commit** (vanno annullate, UNDO)

Protocollo del gestore dell'affidabilità

Il gestore dell'affidabilità al restart del sistema:

1. **Legge su un file** di RESTART (sempre contenuto nel log) l'indirizzo dell'ultimo checkpoint.
2. **Prepara due file: UNDO list** con gli identificatori delle transazioni attive, **REDO list** vuoto.
3. Nessun utente **è attivo** durante il RESTART.

Ripresa a Caldo

La **ripresa a caldo** ha quattro fasi:

1. **trovare l'ultimo checkpoint** (ripercorrendo il log a ritroso)
2. **costruire** gli insiemi UNDO (transazioni attive ma non committed prima del guasto, da disfare) e REDO (transazioni committed tra CK ed il guasto, da rifare)
3. **ripercorrere il log all'indietro (rollback)**, fino alla più vecchia azione delle transazioni in UNDO e REDO, **disfacendo** tutte le azioni delle **transazioni in UNDO**.
4. **ripercorrere il log in avanti (rollforward)**, **rifacendo** tutte le azioni delle **transazioni in REDO**.

Esempio ripreso dal libro:

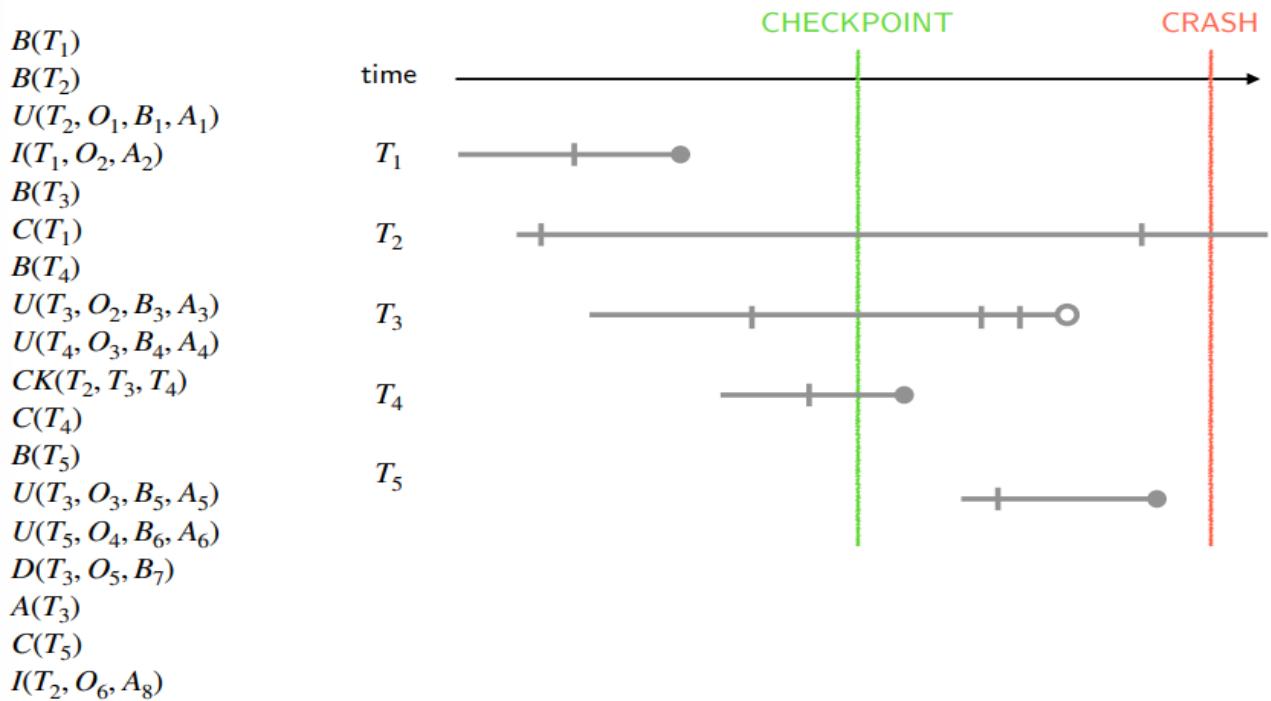
B(T1), B(T2), U(T2, O1, B1, A1), I(T1,O2,A2), B(T3), C(T1), B(T4), U(T3, O2, B3, A3), U(T4, O3,B4, A4), CK(T2,T3,T4), C(T4), B(T5), U(T3,O3,B5, A5), U(T5,04,B6,A6), D(T3,O5,B7), A(T3), C(T5), I(T2, O6, A8).

Successivamente si verifica un guasto. Il protocollo opera come segue:

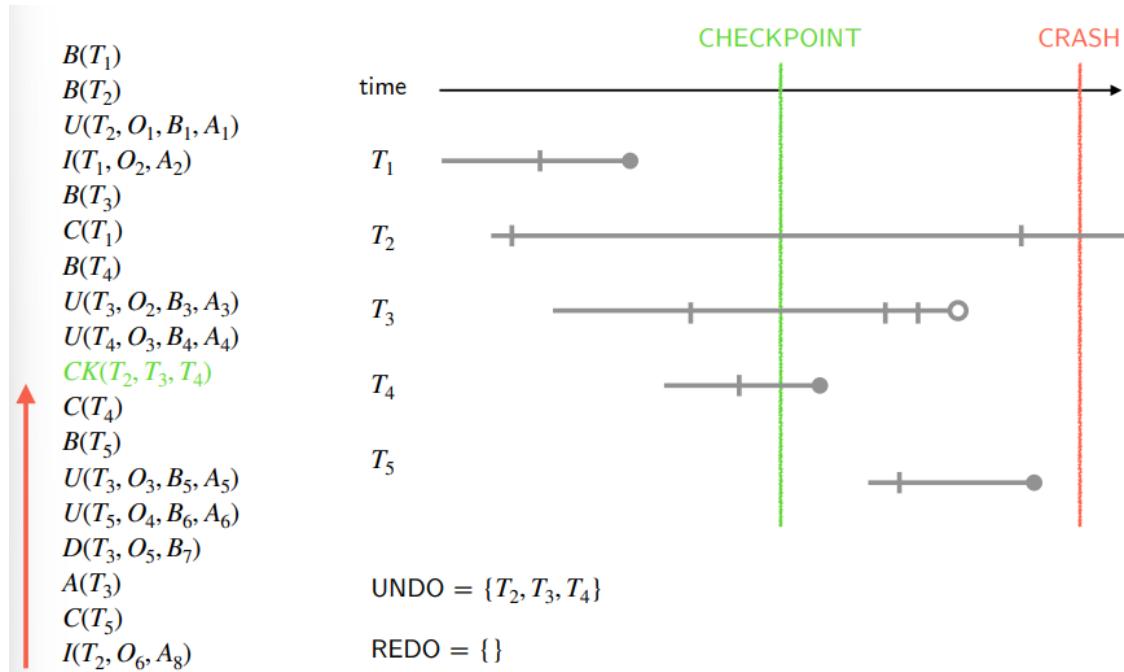
1. Si accede al record di checkpoint; UNDO={T2,T3,T4}, REDO={}.
2. Successivamente si percorre in avanti il record di log, e si aggiornano gli insiemi di UNDO e REDO:
 - a. C(T4): UNDO={T2,T3}, REDO={T4}
 - b. B(T5): UNDO={T2,T3,T5}, REDO={T4}
 - c. C(T5): UNDO={T2,T3}, REDO={T4,T5}

3. Successivamente si ripercorre indietro il log fino all'azione $U(T_2, O_1, B_1, A_1)$, eseguendo le seguenti azioni di UNDO:
- Delete(O_6)
 - Re-Insert($O_5=B_7$)
 - $O_3=B_5$
 - $O_2=B_3$
 - $O_1=B_1$
4. Infine, vengono svolte le azioni di REDO:
- $O_3=A_4$ (nota: $A_4=B_5!$)
 - $O_4=A_6$

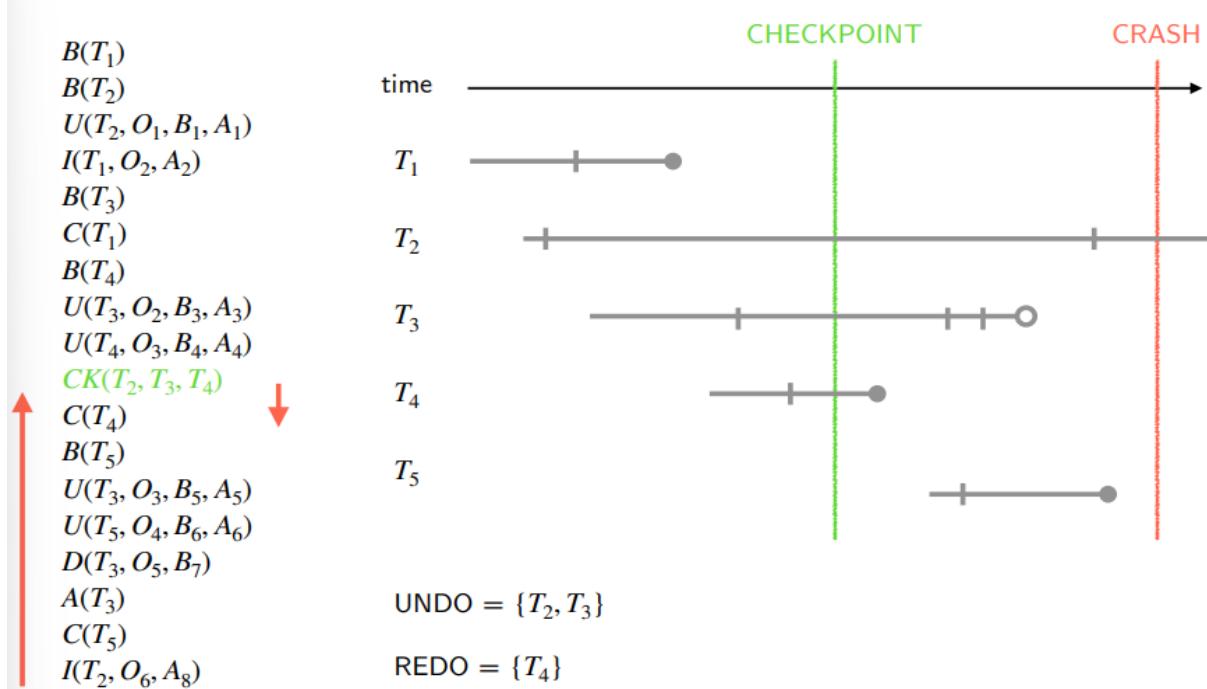
Vediamo un altro esempio:



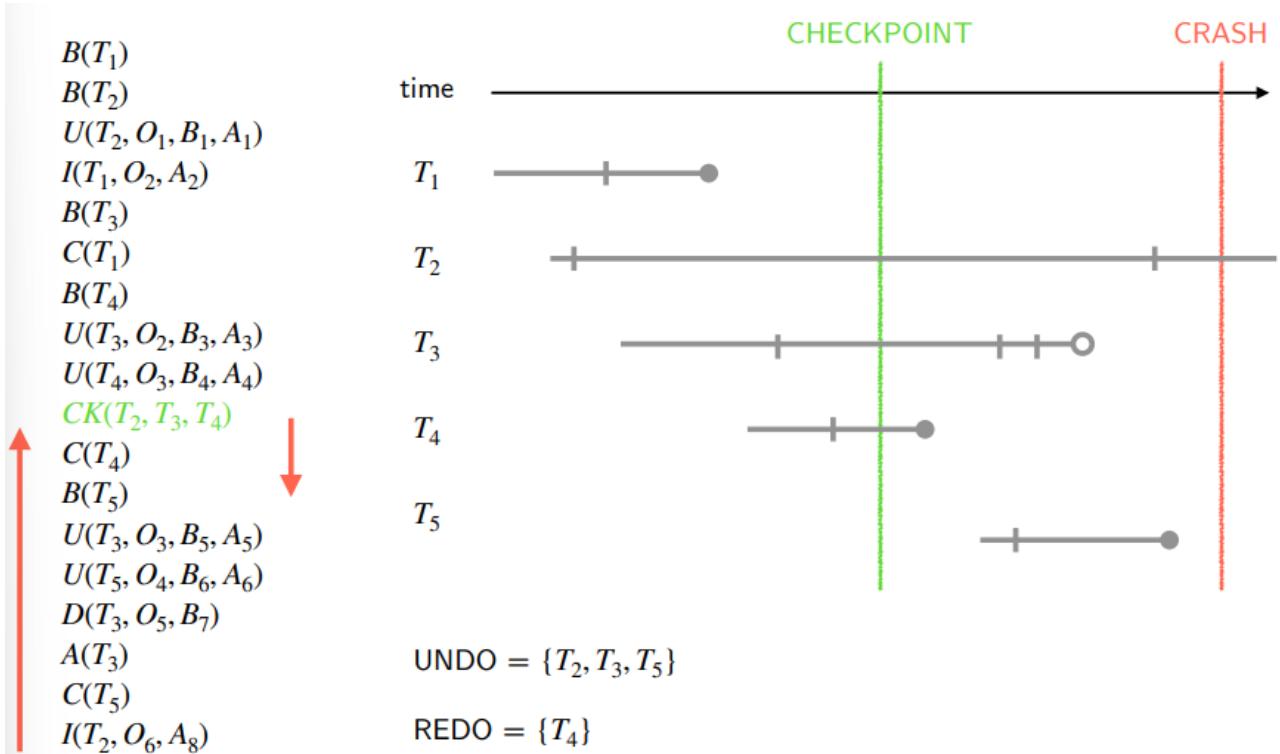
Trovare l'ultimo checkpoint



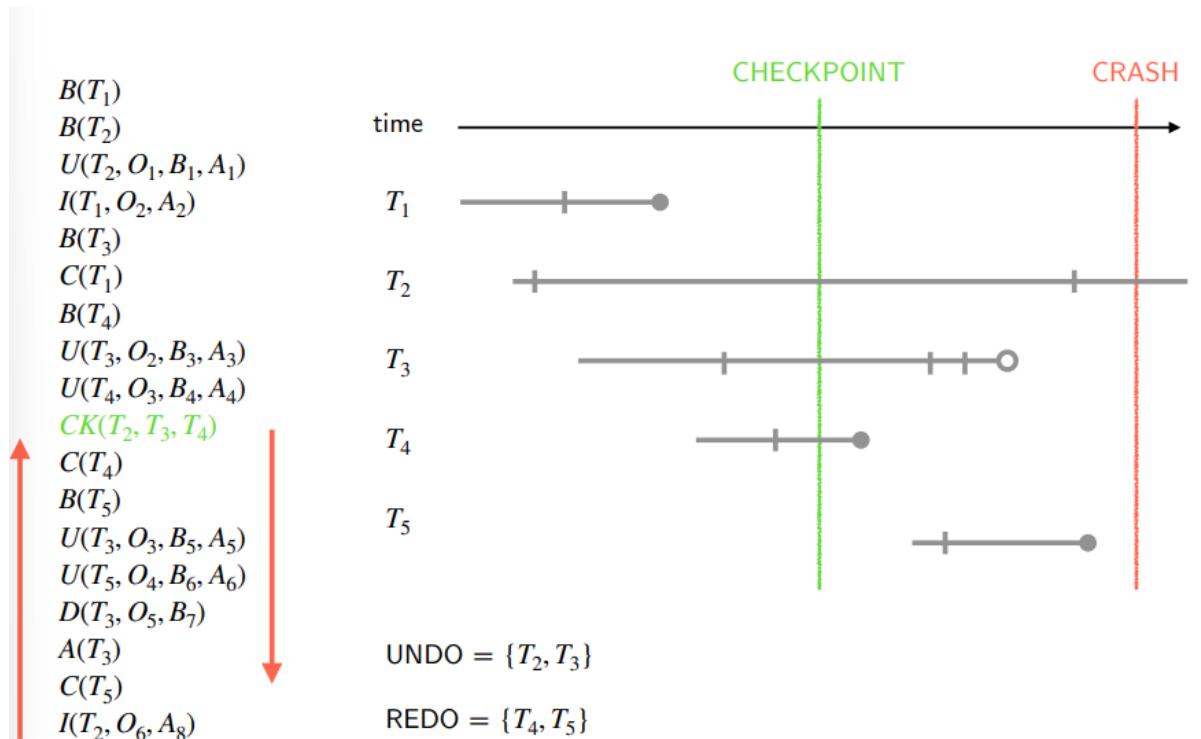
Costruire UNDO e REDO (1)



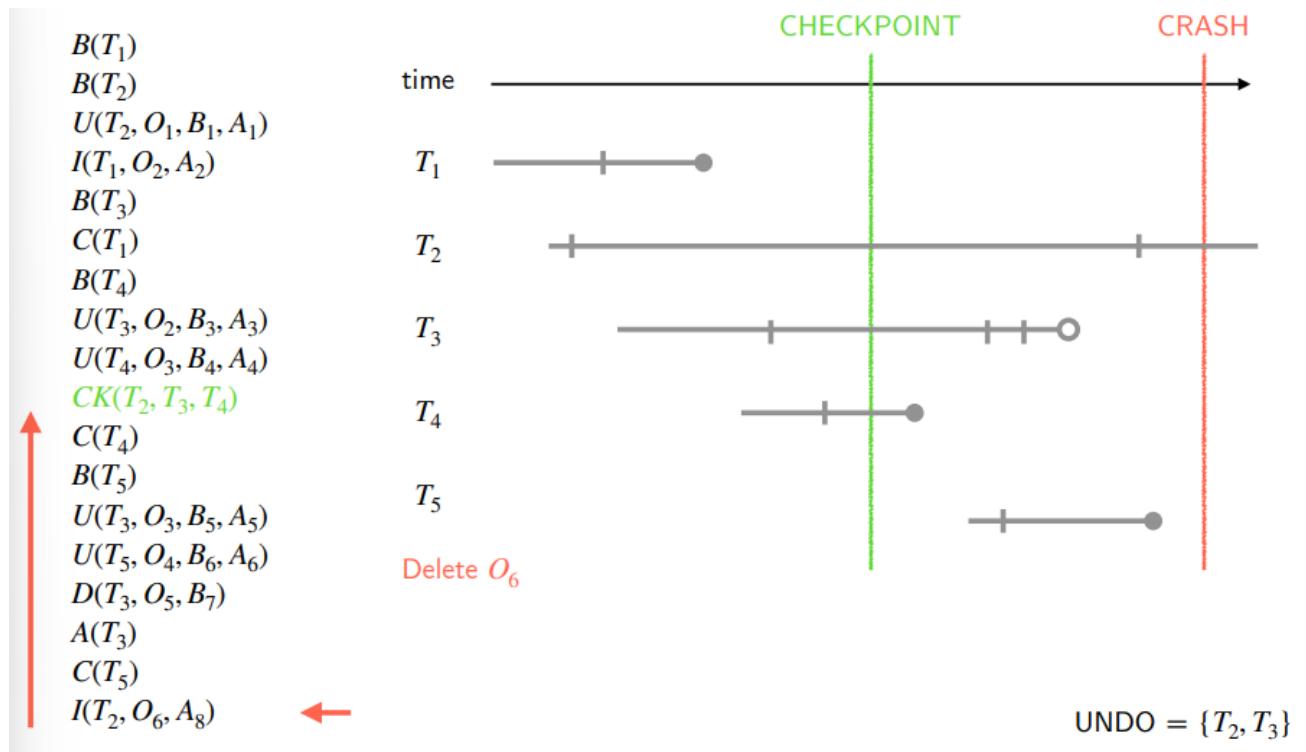
Costruire UNDO e REDO (2)



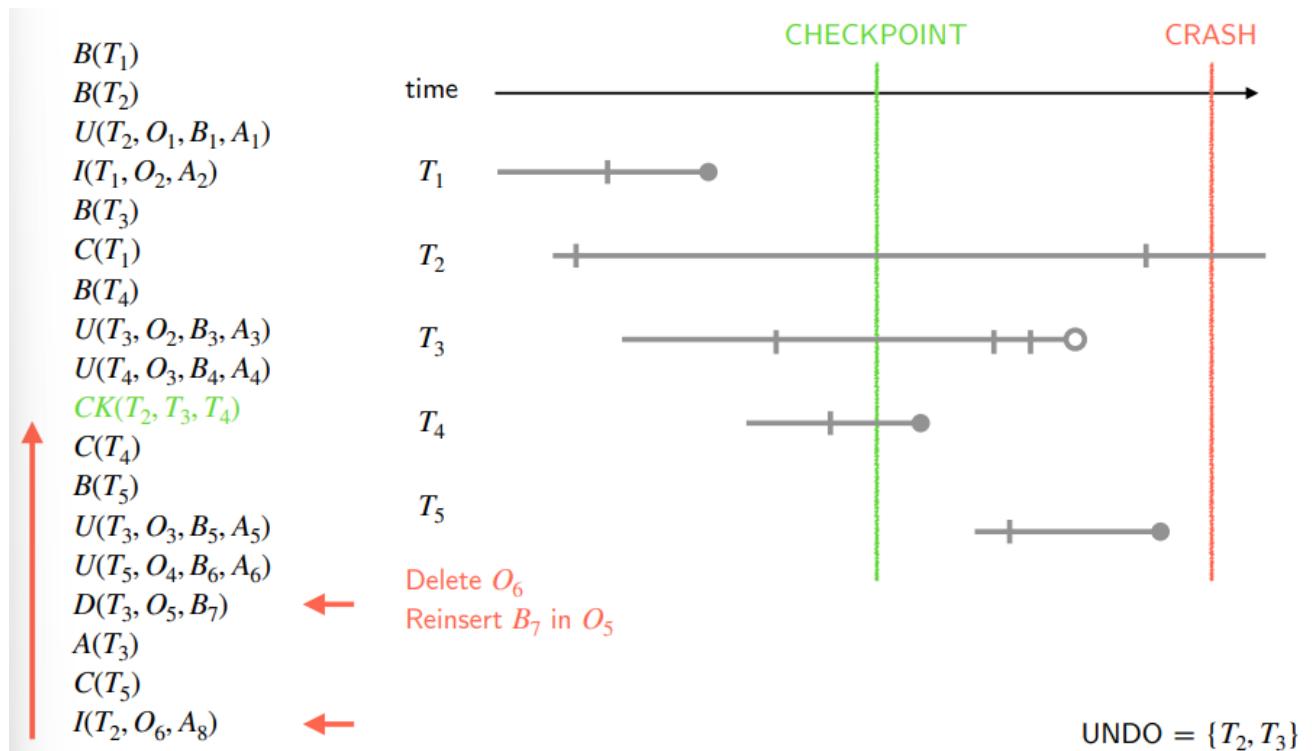
Costruire UNDO e REDO (3)



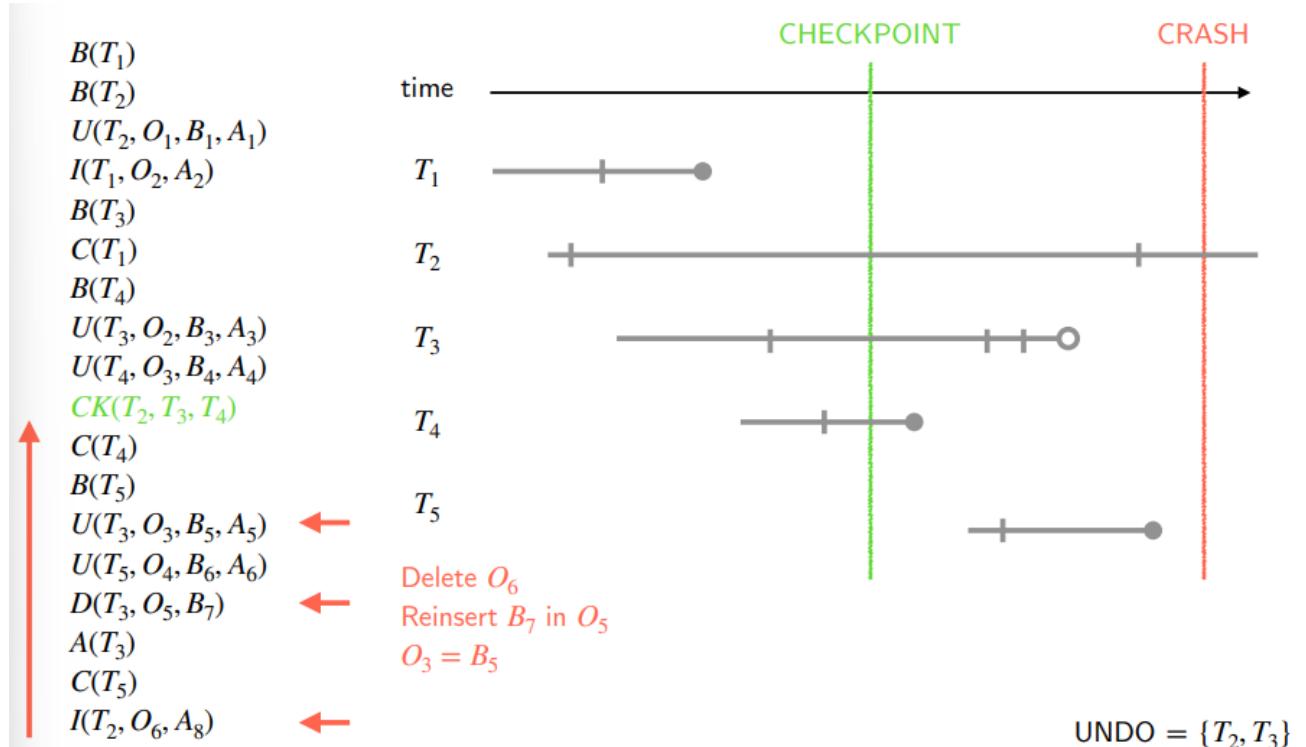
Disfare gli UNDO a ritroso (1)



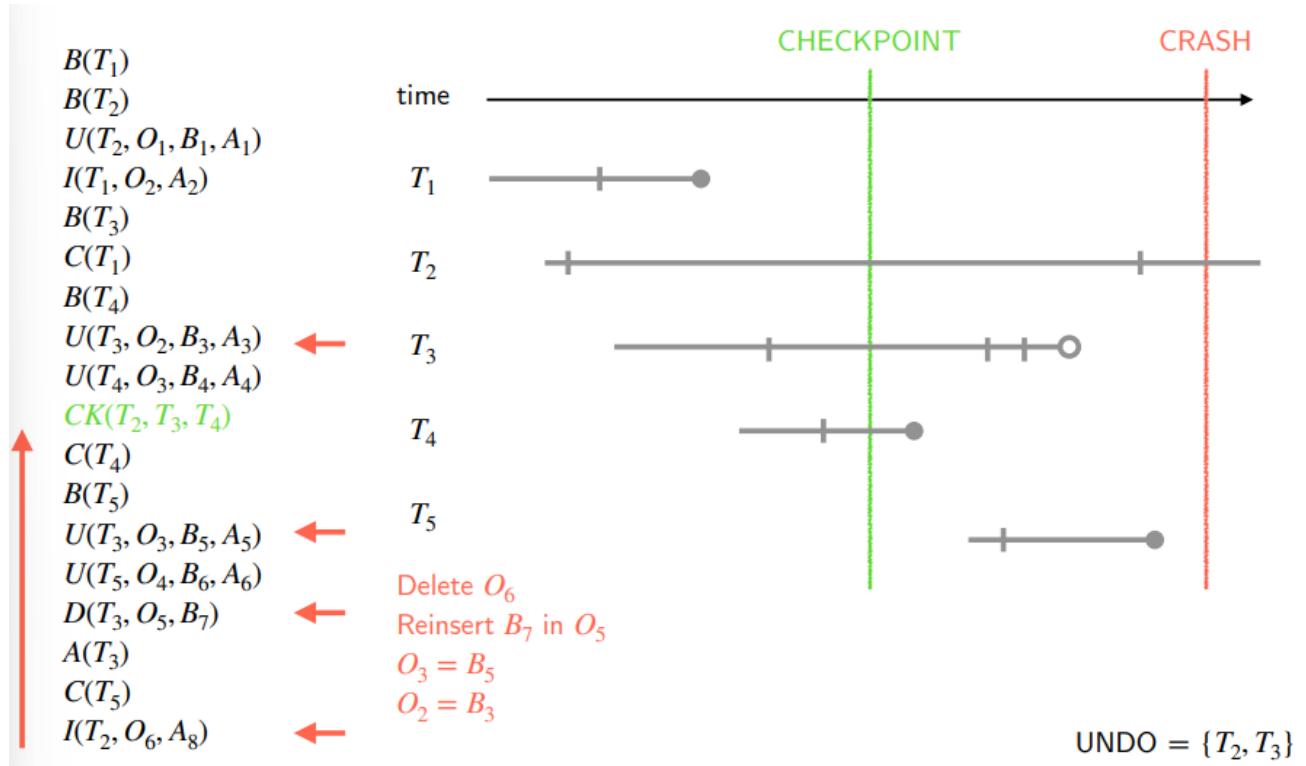
Disfare gli UNDO a ritroso (2)



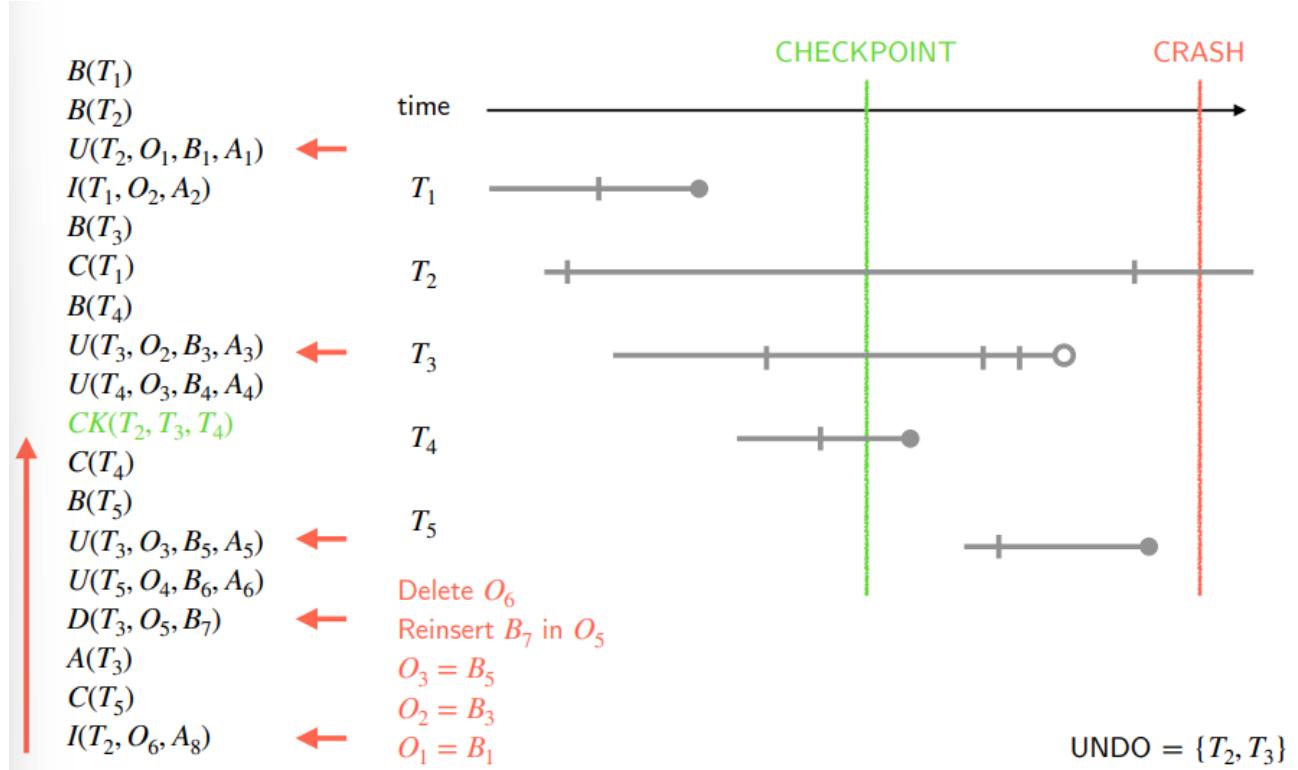
Disfare gli UNDO a ritroso (3)



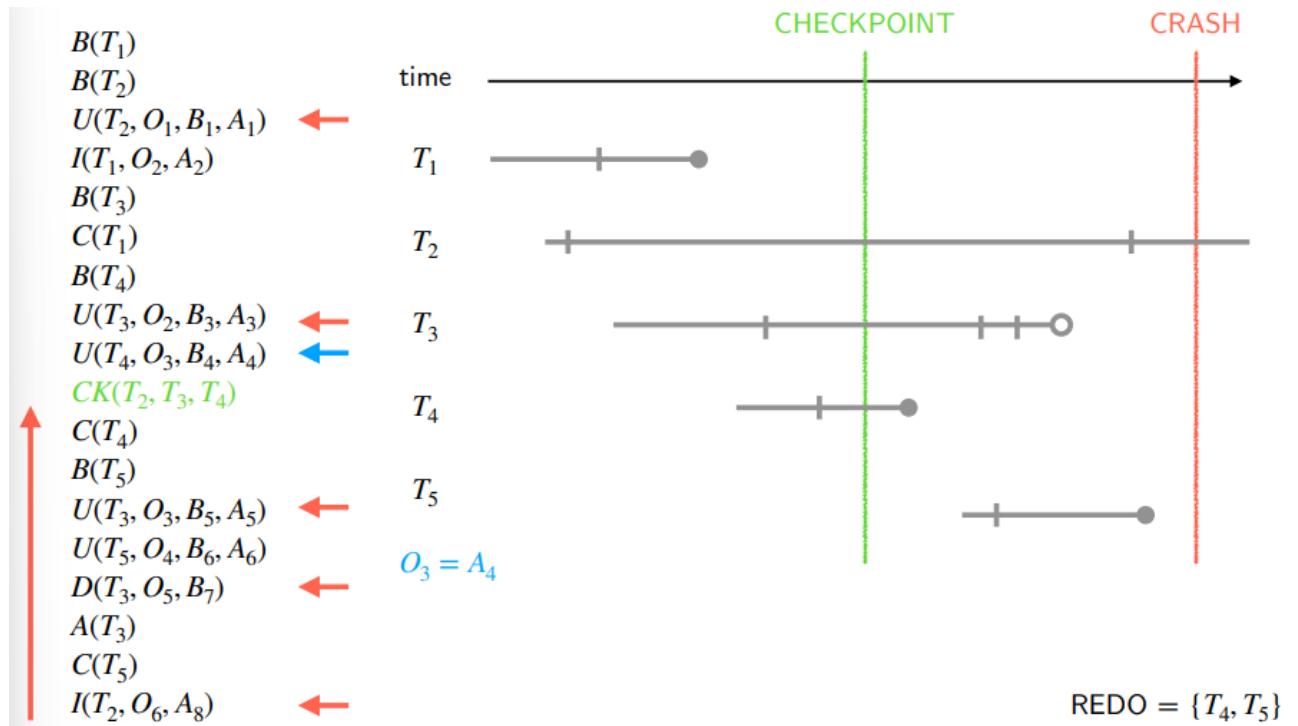
Disfare gli UNDO a ritroso (4)



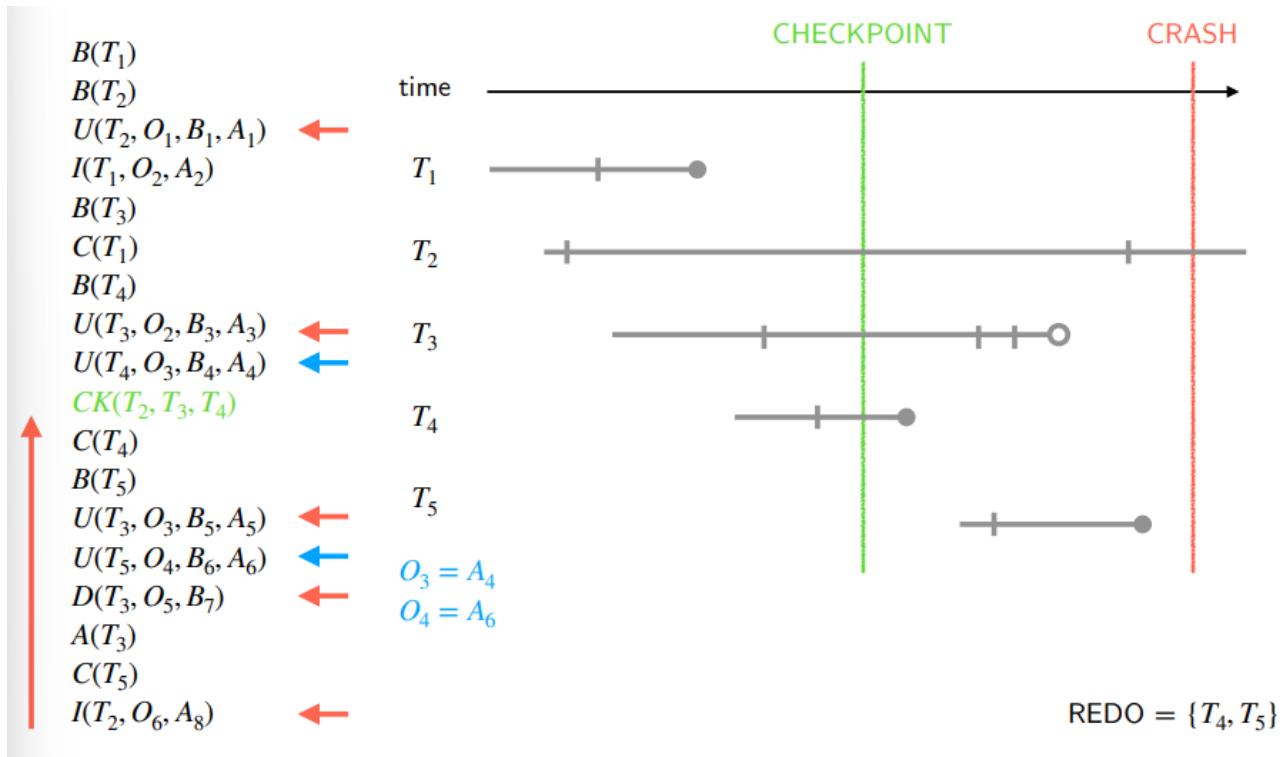
Disfare gli UNDO a ritroso (5)



Rifare i REDO in avanti



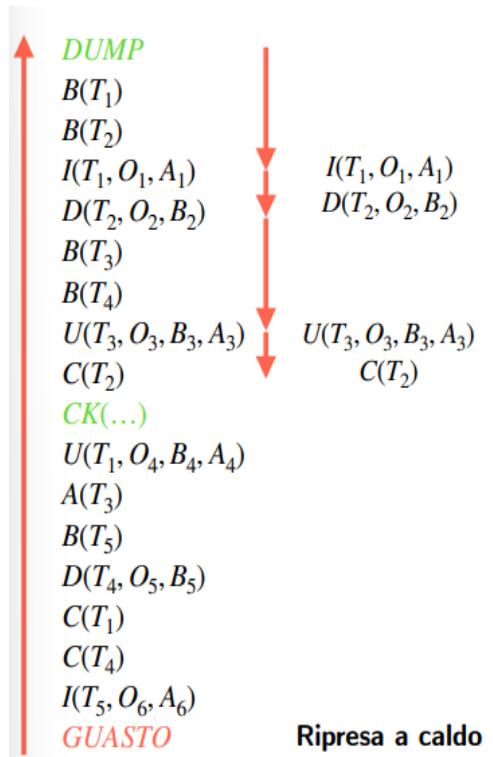
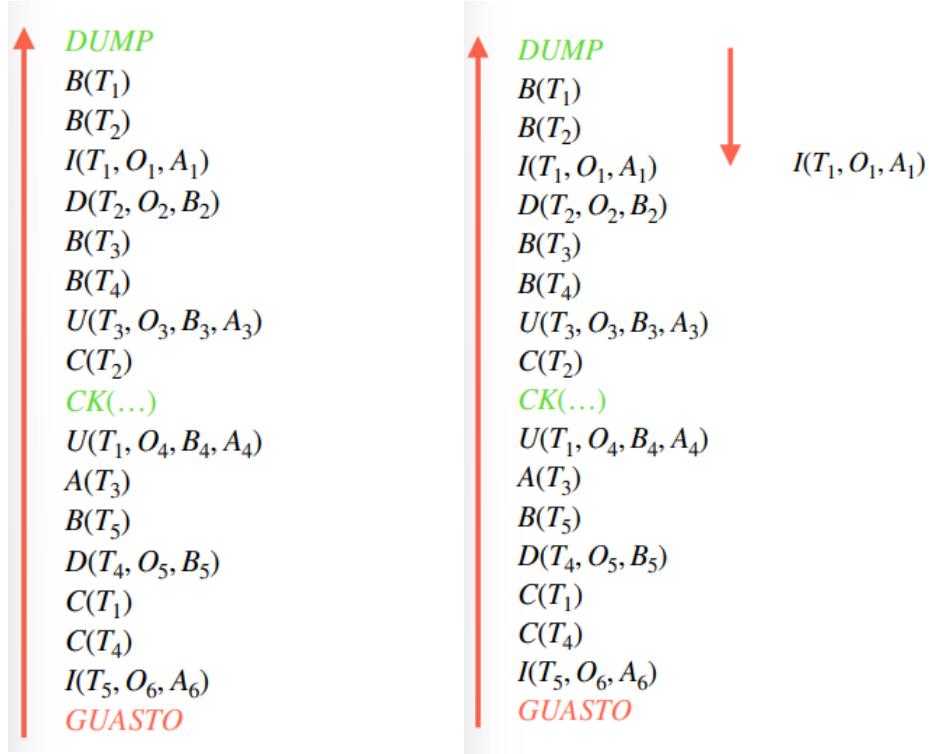
Rifare i REDO in avanti



Ripresa a Freddo

- Ci si riporta al **record di dump** più recente nel **log** e si ripristina la parte di dati deteriorata.
- Si eseguono le operazioni registrate sul log sulla parte deteriorata fino all'istante del guasto
- Si esegue una ripresa a caldo

Esempio



Lezione 12: Gestione della Concorrenza

Controllo della concorrenza

La **concorrenza è fondamentale**: infatti decine e migliaia di transazioni al secondo non possono essere seriali.

Esempio: banche e prenotazioni aeree...

Modello di riferimento: vengono svolte **operazioni** di input/output su **oggetti astratti** x, y, z .

Problema: ci sono **anomalie** causate dall'**esecuzione concorrente**, che quindi va governata.

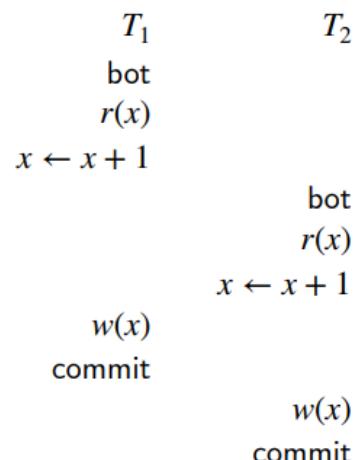
Perdita di aggiornamento:

Siano due transazioni identiche:

- T1: $r(x)$, $x \leftarrow x + 1$, $w(x)$
- T2 : $r(x)$, $x \leftarrow x + 1$, $w(x)$

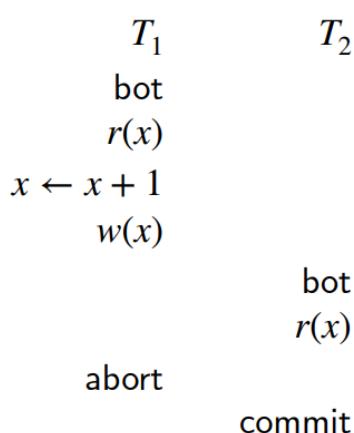
Inizialmente $x = 2$, dopo un'esecuzione seriale $x = 4$

Con un'esecuzione concorrente



un aggiornamento viene perso: $x = 3$

Lettura sporca



T_2 ha letto uno **stato intermedio (sporco)** e le può comunicare all'esterno.

Letture inconsistenti

T_1	T_2	
bot		
$r(x)$		
	bot	T_1 legge due valori diversi per x
	$r(x)$	
$x \leftarrow x + 1$		
	$w(x)$	
	commit	
$r(x)$		
commit		

Aggiornamento fantasma

Assumiamo di avere il vincolo $y+z = 1000$:

T_1	T_2	
bot		
$r(y)$		$s=1100$: il vincolo sembra non soddisfatto , T_1 vede un aggiornamento non coerente.
	bot	
	$r(y)$	
$y \leftarrow y - 100$		
	$r(z)$	
$z \leftarrow z + 100$		
	$w(y)$	
	$w(z)$	
	commit	
$r(z)$		
$s \leftarrow y + z$		
commit		

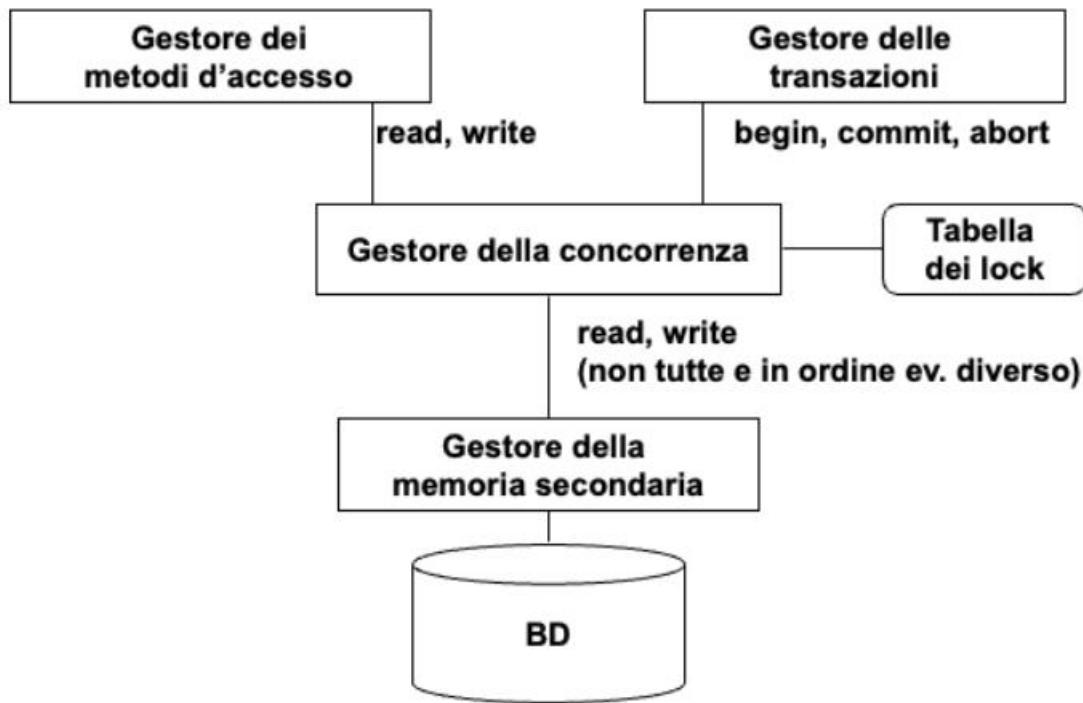
Inserimento fantasma

T_1	T_2	
bot		
legge gli stipendi degli impiegati del dip. A e calcola la media		
	bot	
	inserisce un impiegato in A	
	commit	
legge gli stipendi degli impiegati del dip. A e calcola la media		
commit		

Anomalie

- Perdita di aggiornamento: W-W
- Lettura sporca: R-W (o W-W) con abort
- Letture inconsistenti: R-W
- Aggiornamento fantasma: R-W
- Inserimento fantasma: R-W su dato nuovo

Gestione della concorrenza



Schedule

Uno **schedule S** è una sequenza di operazioni di lettura/scrittura di transazioni concorrenti.

Esempio:

$S : r_1(x), r_2(z), w_1(x), w_2(z)$

Dove:

- $r_1(x)$ rappresenta la lettura dell'oggetto x da parte della transazione T_1 .
- $w_2(z)$ rappresenta la scrittura dell'oggetto z da parte della transazione T_2 .

Le operazioni compaiono nello schedule nell'ordine temporale di esecuzione sulla base di dati.

Controllo di concorrenza

Il **controllo della concorrenza** è eseguito dallo **scheduler**, che tiene traccia di tutte le operazioni eseguite sulla base di dati dalle transazioni e decide se accettare o rifiutare le operazioni che vengono via via richieste.

L'obiettivo è di **eliminare le anomalie**.

Per il momento, assumiamo che l'**esito** (commit/abort) delle transazioni sia **noto a priori** (ipotesi **commit-proiezione**).

- In questo modo possiamo **rimuovere** dallo schedule tutte le **transazioni abortite**.
- Si noti che tale assunzione **non consente** di trattare **alcune anomalie** (**lettura sporca**).

Schedule seriale

Uno **schedule** di un insieme di transazioni $T = \{T_1, \dots, T_n\}$ è detto **seriale** se, per ogni coppia di transazioni $T_i, T_j \in T$, tutte le operazioni di T_i sono eseguite prima di qualsiasi operazione di T_j , o viceversa.

Esempio:

$$T = \{T_0, T_1, T_2\}$$

$$\begin{aligned} S = & r_0(x) r_0(y) w_0(x) r_1(y) r_1(x) \\ & w_1(y) r_2(x) r_2(y) w_2(z) w_2(z) \end{aligned}$$

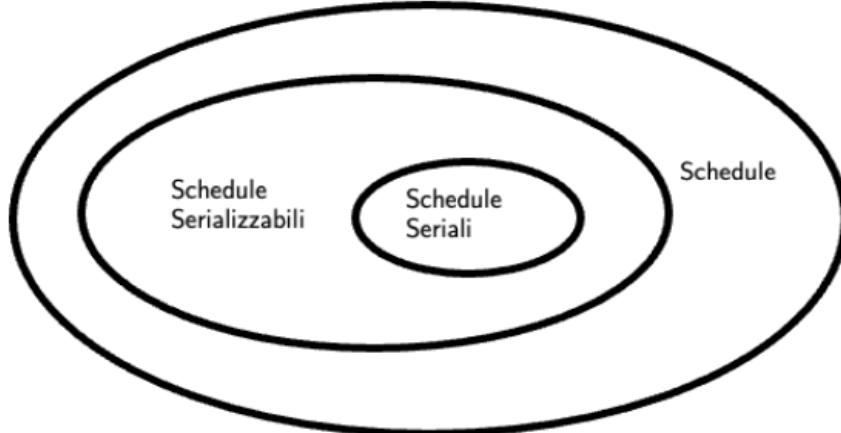
Schedule serializzabile

Uno **schedule** di un insieme di transazioni è **serializzabile** se la sua esecuzione produce lo stesso risultato di uno schedule seriale sulle stesse transazioni.

Richiede una nozione di **equivalenza fra schedule**.

Idea Base

Individuare **classi** di **schedule serializzabili** che siano **sottoclassi** degli **schedule** possibili, siano **serializzabili** e la cui proprietà di serializzabilità sia **verificabile a costo basso**.



View-Seriazabilità

Diciamo che esiste la relazione **legge-da** tra le operazioni $r_i(x)$ e $w_j(x)$ presenti in uno schedule S se $w_j(x)$ precede $r_i(x)$ in S e non c'è nessun $w_k(x)$, con $k \neq j$ tra $r_i(x)$ e $w_j(x)$ in S .

La scrittura in S è detta **scrittura finale** su x se è l'ultima scrittura su x in S .

Due schedule S_i e S_j sono detti **view-equivalenti**, $S_i \approx_v S_j$ se hanno la **stessa relazione legge-da** e le **stesse scritture finali su ogni oggetto**.

Uno schedule S è **view-serializzabile** se è *view-equivalente* ad un qualche *schedule seriale*.

L'insieme degli schedule **view-serializzabili** è indicato con **VSR**.

Esempio:

- Consideriamo i seguenti schedule:
 - $S_3 = w_0(x) r_2(x) r_1(x) w_2(x) w_2(z)$
 - $S_4 = w_0(x) r_1(x) r_2(x) w_2(x) w_2(z)$ (schedule seriale)
- S_3 è view-equivalente allo schedule seriale S_4 ?
 - $\text{legge-da}(S_3) = w_0(x)r_2(x), w_0(x)r_1(x)$
 - $\text{finale}(S_3) = w_2(x), w_2(z)$
 - $\text{legge-da}(S_4) = w_0(x)r_1(x), w_0(x)r_2(x)$
 - $\text{finale}(S_4) = w_2(x), w_2(z)$
 - Si, S_3 è view-equivalente allo schedule seriale S_4
 - Quindi è view-serializzabile

Esempio:

- Consideriamo i seguenti schedule:
 - $S_4 = w_0(x) r_1(x) r_2(x) w_2(x) w_2(z)$ (schedule seriale)
 - $S_5 = w_0(x) r_2(x) w_2(x) r_1(x) w_2(z)$
- S_5 è view-equivalente allo schedule seriale S_4 ?
 - $\text{legge-da}(S_4) = w_0(x)r_1(x), w_0(x)r_2(x)$
 - $\text{finale}(S_4) = w_2(x), w_2(z)$
 - $\text{legge-da}(S_5) = w_0(x)r_2(x), w_2(x)r_1(x)$
 - $\text{finale}(S_5) = w_2(x), w_2(z)$
 - No, S_5 non è view-equivalente allo schedule seriale S_4
 - Non vuol dire che non sia view-serializzabile, proviamo un altro schedule seriale

Esempio:

- Consideriamo i seguenti schedule:
 - $S_5 = w_0(x) r_2(x) w_2(x) r_1(x) w_2(z)$
 - $S_6 = w_0(x) r_2(x) w_2(x) w_2(z) r_1(x)$ (schedule seriale)
- S_5 è view-equivalente allo schedule seriale S_6 ?
 - $\text{legge-da}(S_5) = w_0(x)r_2(x), w_2(x)r_1(x)$
 - $\text{finale}(S_5) = w_2(x), w_2(z)$
 - $\text{legge-da}(S_6) = w_0(x)r_2(x), w_2(x)r_1(x)$
 - $\text{finale}(S_6) = w_2(x), w_2(z)$
 - Si, S_5 è view-equivalente allo schedule seriale S_6
 - Quindi è view-serializzabile

Esempio:

- Consideriamo i seguenti schedule:
 - $S_7 = r_1(x) r_2(x) w_1(x) w_2(x)$
 - nessuno schedule seriale view-equivalente
 - è una perdita di aggiornamento
 - $S_8 = r_1(x) r_2(x) w_2(x) r_1(x)$
 - nessuno schedule seriale view-equivalente
 - è una lettura inconsistente
 - $S_9 = r_1(x) r_1(y) r_2(z) r_2(y) w_2(y) w_2(z) r_1(z)$
 - nessuno schedule seriale view-equivalente
 - è un aggiornamento fantasma
- S_7, S_8 e S_9 non sono view serializzabili
 - Non sono view-equivalenti a nessuno schedule seriale

Uso della View-Seriazabilità

Complessità:

- la **verifica** della view-equivalenza di due dati schedule ha complessità **polinomiale**.
- Il **decidere** sulla view-serializzabilità di uno schedule è un problema **NP-completo**.
- È necessario confrontare lo schedule con tutti i possibili schedule seriali.

Non è utilizzabile in pratica: definiamo una **condizione di equivalenza più ristretta**, che non copra tutti i casi di equivalenza tra schedule coperti dalla view-equivalenza, ma che sia **utilizzabile nella pratica** (la procedura di verifica abbia cioè una complessità inferiore).

Conflict-Serializzabilità

Un operazione a_i è in **conflitto con un'altra operazione** a_j , con $i \neq j$, se operano sullo stesso oggetto e almeno una di esse è una scrittura.

NB: $a_i(x) a_j(x) \neq a_j(x) a_i(x)$, cioè nei conflitti conta l'ordine.

Esistono due casi:

- **conflitto read-write** (R-W o W-R)
- **conflitto write-write** (W-W)

Due schedule S_i e S_j sono detti **conflict-equivalenti**, $S_i \approx_c S_j$ se hanno le stesse operazioni ed ogni coppia di operazioni in conflitto compare nello stesso ordine in entrambi.

Uno schedule S è **conflict-serializzabile** se è *conflict-equivalente* ad un qualche schedule seriale.

L'insieme degli schedule *conflict-serializzabili* è indicato con **CSR**.

VSR e CSR (Teorema)

“Ogni schedule conflict-serializzabile è view-serializzabile, ma non necessariamente viceversa.”

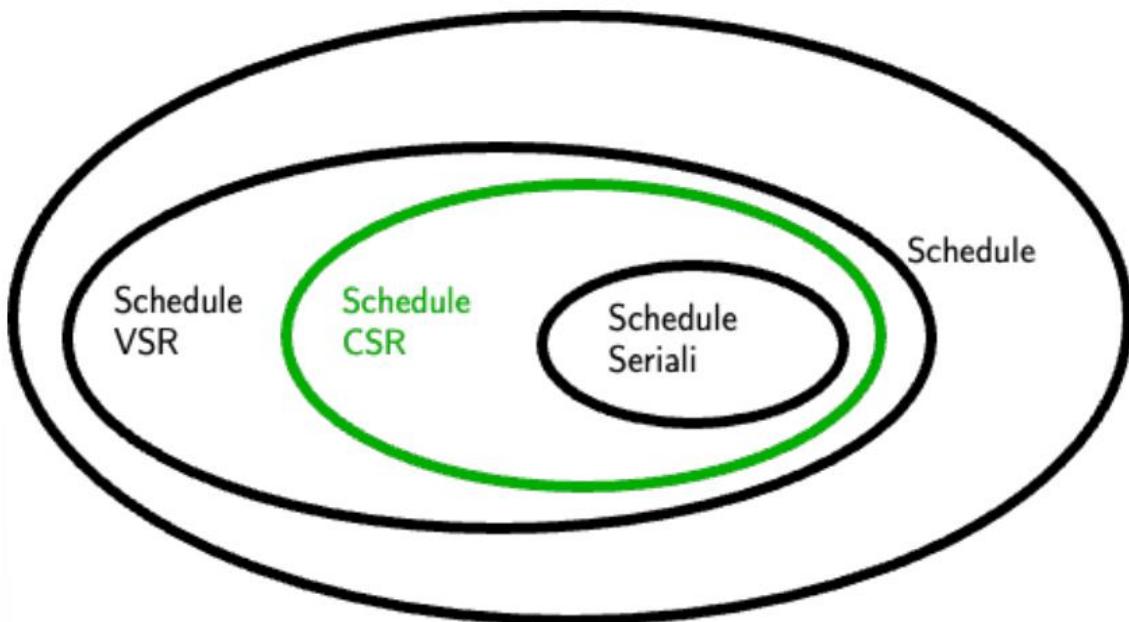
- Contro-esempio per la non necessità:
 - $r_1(x) w_2(x) w_1(x) w_3(x)$
- view-serializzabile: view-equivalente a
 - $r_1(x) w_1(x) w_2(x) w_3(x)$
- conflict-serializzabile:
 - No

Per dimostrare che **CSR** implica **VSR** è sufficiente dimostrare che la conflict-equivalenza \approx_c implica la view-equivalenza \approx_v , cioè che se due schedule sono \approx_c allora sono \approx_v .

Quindi, supponiamo $S_1 \approx_c S_2$ e dimostriamo che $S_1 \approx_v S_2$. I due schedule hanno:

- **stesse scritture finali:** se così non fosse, ci sarebbero almeno due scritture in ordine diverso e poiché due scritture sono in conflitto i due schedule non sarebbero \approx_c .
- **stessa relazione “legge-da”:** se così non fosse, ci sarebbero scritture in ordine diverso o coppie lettura-scrittura in ordine diverso e quindi, come sopra sarebbe violata la \approx_c .

VSR e CSR



Verifica della Conflict-Serializzabilità

Per mezzo del **grafo dei conflitti**:

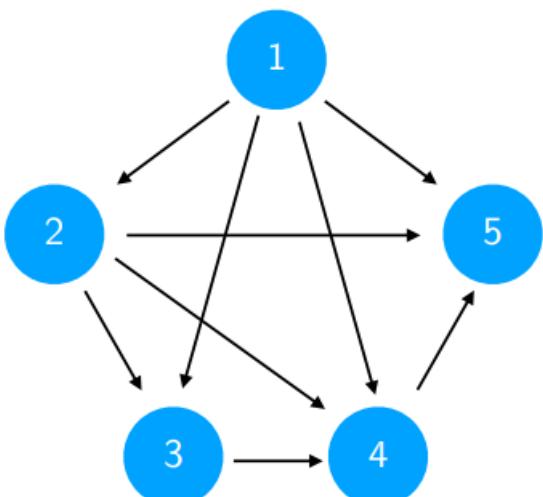
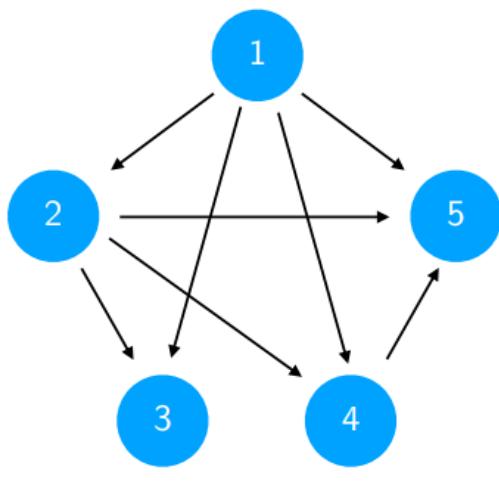
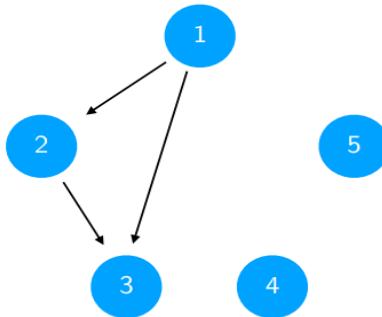
- un **nodo** per ogni **transazione** T_i
- un **arco (orientato)** da T_i a T_j se c'è almeno un **conflitto** fra un'azione a_i ed un'azione a_j tale che a_i **precede** a_j .

Teorema:

“Uno schedule è in CSR se e solo se il grafo è aciclico.”

Esempio:

- $S = w_1(x)w_2(x)r_3(x)r_1(y)w_2(y)r_1(z)w_3(z)r_4(z)w_4(y)w_5(y)$
- $x : w_1 \ w_2 \ r_3$
- $y : r_1 \ w_2 \ w_4 \ w_5$
- $z : r_1 \ w_3 \ r_4$



Il grafo è **aciclico (da un qualsiasi nodo non si può ritornare in sé stesso)**:

S è allora **CSR**, quindi anche **VSR**.

Cerchiamo uno schedule seriale conflict-equivalente

- T_1 ha conflitti con tutte le transazioni, quindi la mettiamo per prima
- T_2 ha conflitti con tutte le transazioni rimanenti, quindi la mettiamo per seconda
- T_3 deve venire prima di T_4
- T_4 deve venire prima di T_5
- $w_1(x) \ r_1(y) \ r_1(z) \ w_2(x) \ w_2(y) \ r_3(x) \ w_3(z) \ r_4(z) \ w_4(y) \ w_5(y)$

Esempio:

- $S = r_1(y)w_3(z)r_1(z)r_2(z)w_3(x)w_1(x)w_2(x)r_3(y)$

- $x : w_3 \ w_1 \ w_2$

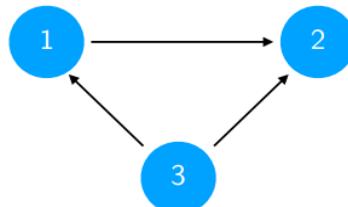
- $y : r_1 \ r_3$

- $z : w_3 \ r_1 \ r_2$

- Il grafo è aciclico

- S è *CSR*, quindi anche *VSR*

- $w_3(z)w_3(x)r_3(y)r_1(y)r_1(z)w_1(x)r_2(z)w_2(x)$



- $S' = r_1(y)\textcolor{red}{r_1(z)}w_3(z)r_1(z)r_2(z)w_3(x)w_1(x)w_2(x)r_3(y)$

- $x : w_3 \ w_1 \ w_2$

- $y : r_1 \ r_3$

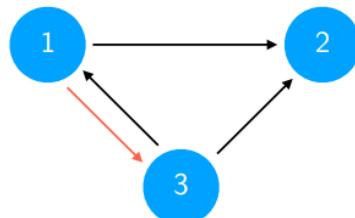
- $z : \textcolor{red}{r_1} \ w_3 \ r_1 \ r_2$

- Il grafo non è aciclico

- S non è *CSR*

- Ma S è *VSR*:

- $r_1(y)r_1(z)w_1(x)w_3(z)w_3(x)r_3(y)r_2(z)w_2(x)$



Verifica della Conflict-Serializzabilità

Anche la **conflict-serializzabilità**, per più rapidamente verificabile (l'algoritmo, con opportune strutture dati richiede **tempo lineare**), è **inutilizzabile in pratica**.

La tecnica sarebbe efficiente se potessimo **conoscere il grafo dall'inizio**, ma così non è, poiché uno scheduler deve operare **incrementalmente**, cioè ad ogni richiesta di operazione decidere se eseguirla subito oppure fare qualcos'altro; non è praticabile mantenere il grafo, aggiornarlo e verificarne l'aciclicità ad ogni richiesta di approvazione. Inoltre, la tecnica **si basa** sull'ipotesi di **commit-proiezione**.

In pratica, si utilizzano tecniche che:

- garantiscono la conflict-serializzabilità **senza dover costruire il grafo**.
- **non richiedono** l'ipotesi della **commit-proiezione**.

Lock

Principio:

- Tutte le **lettura** sono **precedute** da lock e **seguite** da unlock.
- Tutte le **scrittura** sono **precedute** lock da e **seguite** da unlock.

Il **lock manager** riceve queste richieste dalle transazioni e le **accoglie o rifiuta**.

Lock condiviso ed esclusivo

Per aumentare la **concorrenza** è possibile avere lock di tipo diverso, **condiviso o esclusivo**, usati in momenti diversi sulla stessa risorsa.

Principio:

- Tutte le **lettura** sono **precedute** da **r_lock** (lock condiviso) e **seguite** da **unlock**.
- Tutte le **scrittura** sono **precedute** da **w_lock** (lock esclusivo)

Quando una transazione **prima legge e poi scrive un oggetto**, può

- richiedere **subito** un lock esclusivo
- chiedere **prima** un lock condiviso e **poi** un lock esclusivo (lock escalation)

Il **lock manager** riceve queste richieste dalle transazioni e le **accoglie o rifiuta**, sulla base della **tavola dei conflitti**.

Comportamento dello scheduler

La politica dello scheduler è basata sulla **tavola dei conflitti**.

Il lock manager riceve richieste di lock dalle transazioni e **concede/rifiuta le richieste** sulla base dei lock **precedentemente concessi** ad altre transazioni.

- Quando viene concesso il lock su una risorsa ad una transazione, si dice che la **risorsa è acquisita** dalla transazione.
- Nel momento dell'unlock, la **risorsa** viene **rilasciata**.

Tavola dei conflitti

Permette di realizzare la politica per la gestione dei conflitti

Richiesta	Stato della risorsa		
	free	r_locked	w_locked
r_lock	OK → r_locked	OK → r_locked	NO - (w_locked)
w_lock	OK → w_locked	NO (r_locked)	NO - (r_locked)
unlock	ERROR	OK - dipende*	OK → free

Un contatore tiene il conto del numero dei lettori.

La risorsa è rilasciata solo quando il contatore scendere a zero.

Se la risorsa non è concessa, la **transazione** richiedente è **posta in attesa** (eventualmente in coda), fino a quando la risorsa non diventa disponibile.

Il lock manager gestisce una **tabella dei lock**, per ricordare la situazione.

Locking a due fasi

Un **algoritmo di scheduling** viene usato da quasi tutti i sistemi commerciali.

È basato su **due regole**:

- Se una transazione vuole leggere (scrivere) un dato, **prima deve acquisire un lock** condiviso (esclusivo) sul dato.
- Se la transazione **entra in conflitto** su un lock, **si pone in attesa**.

Una transazione, **dopo aver rilasciato** un lock, **non può acquisirne altri**.

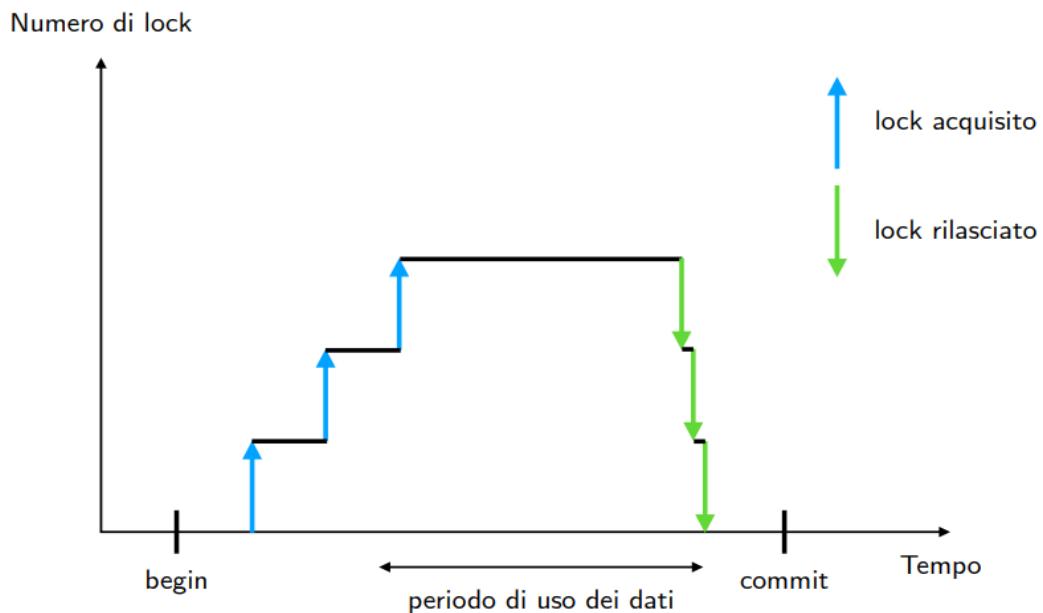
In altre parole:

- Una transazione attraversa una prima fase di acquisizione di ciò che le serve.
- Poi comincia a rilasciare e non può acquisire altro.

Esempio:

<i>begin(T₁)</i>	<i>begin(T₁)</i>
<i>WL₁(B)</i>	<i>WL₁(B)</i>
<i>r₁(B)</i>	<i>r₁(B)</i>
<i>B ← B – 50</i>	<i>B ← B – 50</i>
<i>w₁(B)</i>	<i>w₁(B)</i>
<i>WL₁(A)</i>	<i>UL₁(B)</i>
<i>r₁(A)</i>	<i>WL₁(A)</i>
<i>UL₁(B)</i>	<i>r₁(A)</i>
<i>A ← A + 50</i>	<i>A ← A + 50</i>
<i>w₁(A)</i>	<i>w₁(A)</i>
<i>UL₁(A)</i>	<i>UL₁(A)</i>
<i>commit(T₁)</i>	<i>commit(T₁)</i>

Rappresentazione grafica del 2PL



2PL e CSR (Teorema)

“Ogni schedule 2PL è anche conflict-serializzabile, ma non necessariamente viceversa.”

Contro-esempio per la non-necessità: $r_1(x)w_1(x)r_2(x)w_2(x)r_3(y)w_1(y)$ viola il **2PL** ed è **conflict-serializzabile**

Dimostrazione:

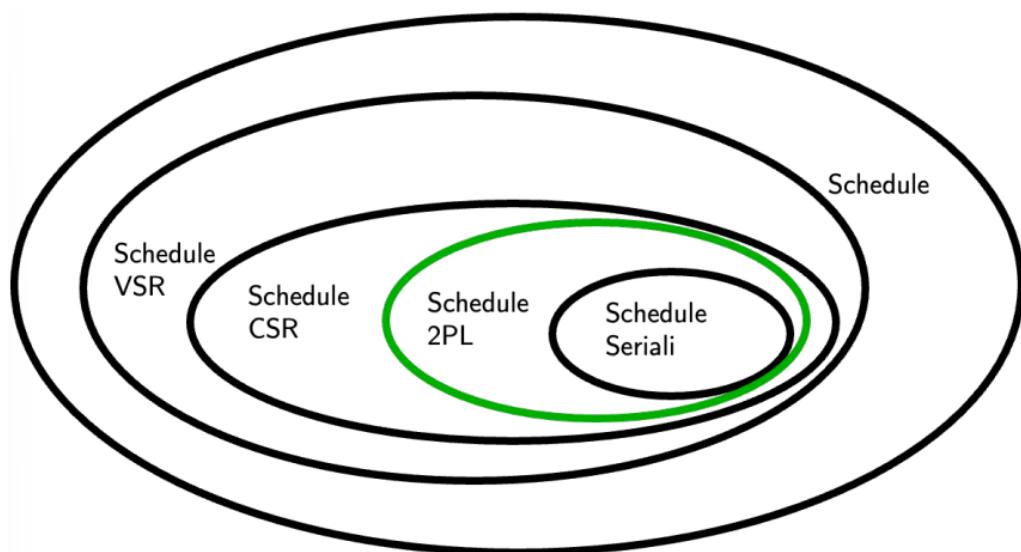
Sia uno schedule **2PL**:

- Consideriamo per ciascuna transazione nell'istante in cui ha tutte le risorse e sta per rilasciare la prima
- Ordiniamo le transazioni in accordo con questo valore temporale e consideriamo lo schedule seriale corrispondente
- Vogliamo dimostrare che tale schedule è equivalente ad S:
 - Consideriamo un conflitto fra un'azione di t_i e t_j un'azione di con $i < j$; è possibile che compaiano in ordine invertito in S?
 - No, perché in tal caso t_j dovrebbe aver rilasciato la risorsa in questione prima della sua acquisizione da parte di t_i .

Dimostrazione alternativa:

- Assumiamo, per assurdo, che esista uno schedule S tale che $S \in 2PL$ e $S \notin CSR$.
- Da $S \notin CSR$ segue che il grafo dei conflitti per contiene un ciclo $t_1, t_2, \dots, t_k, t_1$.
- Se esiste un arco (conflitto) tra t_1 e t_2 , significa che esiste una risorsa x su cui si verifica il conflitto:
 - t_2 può procedere solo se t_1 rilascia il lock su x così che t_2 lo può acquisire.
- Così avanti fino al conflitto tra t_k e t_1 :
 - t_1 deve acquisire il lock rilasciato da t_k , ma t_1 ha già rilasciato un lock per farlo acquisire da t_2 e quindi t_1 non rispetta il 2PL

VSR, CSR, 2PL



2PL e anomalie

È facile vedere che **2PL risolve** le anomalie di **perdita di aggiornamento**, di **aggiornamento fantasma** e di **lettura inconsistenti**.

Però **2PL presenta altre anomalie**:

- **Cascading rollback**: il fallimento di una transazione che ha scritto una risorsa deve causare il fallimento di tutte le transazioni che hanno letto il valore scritto
- **Deadlock (*attese incrociate o stallo*)**: due transazioni detengono ciascuna una risorsa e aspettano la risorsa detenuta dall'altra.

In generale, la **probabilità di deadlock è bassa, ma non nulla**.

Esempio di *cascading* rollback

<i>begin</i> (T_1)		
$WL_1(A)$		
$r_1(A)$		Quando T_1 fallisce, i
$RL_1(B)$		fallimento si deve
$r_1(B)$		trasmettere a T_2 e T_3 .
$w_1(A)$		
$UL_1(A)$		
<i>abort</i> (T_1)	<i>begin</i> (T_2)	
	$WL_2(A)$	
	$r_2(A)$	
	$w_2(A)$	
	$UL_2(A)$	
	...	
	<i>begin</i> (T_3)	
	$RL_3(A)$	
	$r_3(A)$	
	...	

Esempio di deadlock

<i>begin</i> (T_1)	
$WL_1(B)$	
$r_1(B)$	
$B \leftarrow B - 50$	
$w_1(B)$	<i>begin</i> (T_2)
	$RL_2(A)$
	$r_2(A)$
	$RL_2(B)$
	<i>wait</i> T_1
$WL_1(A)$	
<i>wait</i> T_2	
$r_1(B)$	
$UL_1(B)$	

Locking a 2 fasi stretto

- Condizione aggiuntiva: i lock possono essere **rilasciati solo dopo il commit**.
- Elimina il rischio di letture sporche e quindi di rollback in cascata.
- Supera la necessità dell'ipotesi di **commit-proiezione**.

Controllo di concorrenza basato su timestamp

Una tecnica alternativa al 2PL è il **Timestamp** che è un identificatore che definisce un ordinamento totale sugli eventi di un sistema.

- Ogni transazione ha un timestamp che rappresenta l'**istante di inizio della transazione**.
- Uno schedule è accettato solo se riflette l'**ordinamento seriale** delle transazioni **indotto dai timestamp**.

Dettagli

Lo scheduler ha due contatori **WTM(x)** e **RTM(x)** per ogni oggetto x che indicano rispettivamente i timestamp della transazione che ha eseguito l'ultima scrittura e della transazione con ts più grande che ha letto x.

Lo scheduler riceve richieste di letture e scritture (con indicato il timestamp della transazione):

- **read(x, ts)**: se $ts < WTM(x)$ allora la richiesta è respinta e la transazione viene uccisa, altrimenti la richiesta viene accolta e $RTM(x)$ è posto uguale al maggiore fra $RTM(x)$ e ts
- **write(x, ts)**: se $ts < WTM(x)$ o $ts < RTM(x)$ allora la richiesta è respinta e la transazione viene uccisa, altrimenti la richiesta viene accolta e $WTM(x)$ è posto uguale a ts

Vengono uccise molte transazioni.

Esempio:

$$RTM(x) = 0$$

$$WTM(x) = 0$$

Richiesta	Risposta	Nuovo valore
$read(x, t_1)$	OK	$RTM(x) = 1$
$write(x, t_1)$	OK	$WTM(x) = 1$
$read(x, t_2)$	OK	$RTM(x) = 2$
$read(x, t_1)$	OK	
$write(x, t_1)$	<i>No, t_1 aborted</i>	
$read(x, t_2)$	OK	
$write(x, t_2)$	OK	$WTM(x) = 2$

Esempio:

$$RTM(x) = 7$$

$$WTM(x) = 4$$

Richiesta	Risposta	Nuovo valore
$read(x, t_6)$	OK	
$read(x, t_8)$	OK	$RTM(x) = 8$
$read(x, t_9)$	OK	$RTM(x) = 9$
$write(x, t_8)$	$NO, t_8 \text{ aborted}$	
$write(x, t_{11})$	OK	$WTM(x) = 11$
$read(x, t_{10})$	$NO, t_{10} \text{ aborted}$	

Risoluzione del deadlock

L'ordine seriale delle transazioni è fissato prima che le operazioni vengano richieste, tutti gli altri ordinamenti non sono accettati.

- Quando T_1 comincia prima di T_2 , potrebbe essere abilitato uno schedule 2PL o CSR equivalente ad uno seriale T_2T_1 ; col TS non è possibile, al limite T_1 viene abortita e poi fatta ripartire dopo T_2
- In 2PL le transazioni sono poste in attesa quando non è possibile acquisire un lock, in TS uccise e rilanciate. Le ripartenze sono di solito più costose delle attese, infatti conviene il 2PL
- 2PL può causare deadlock, TS no; mediamente si uccide una transazione ogni due conflitti, ma la probabilità di insorgenza di deadlock è molto minore della probabilità di un conflitto, infatti conviene il 2P.

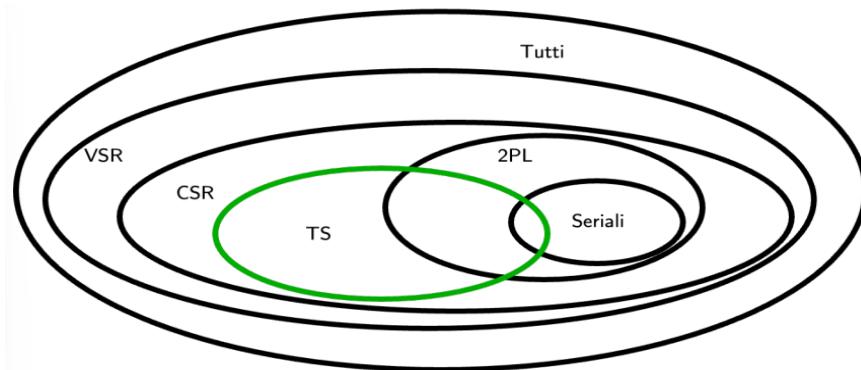
2PL vs TS

Gli schedule TS sono automaticamente CSR, corrispondono ad una esecuzione seriale (quella in cui le transazione sono eseguite nell'ordine in cui sono iniziate).

Ma **2PL e TS sono incomparabili**:

- Schedule in TS ma non in 2PL:
 $r_1(x)w_1(x)r_2(x)w_2(x)r_0(y)w_1(y)$
- Schedule in 2PL ma non in TS:
 $r_2(x)w_2(x)r_1(x)w_1(x)$
- Schedule in TS ed in 2PL:
 $r_1(x)r_2(y)w_2(y)w_1(x)r_2(x)w_2(x)$

VSR, CSR, 2PL, TS



Grafo delle attese

Se uno scheduler, come 2PL, permette dei deadlock, esso ha bisogno di **meccanismi per rilevare i deadlock**.

In genere si usa il **grafo delle attese**

- I nodi sono le **transazioni attive**
- Un **arco** (T_i, T_j) indica che T_i **attende** che T_j **rilasci** un lock di cui ha bisogno

Un **ciclo** in questo grafo corrisponde a un **deadlock**.

Tecniche di risoluzione dei deadlock

Tre tecniche di risoluzione:

1. Timeout

- a. Le transazioni rimangono in **attesa** di una risorsa **per un tempo prefissato**
- b. Se, trascorso tale tempo, la **risorsa non** è ancora stata **concessa**, alla richiesta di lock viene data **risposta negativa**
- c. In tal modo una transazione in potenziale stato di deadlock viene tolta dallo stato di attesa e di norma **abortita**
- d. **Tecnica molto semplice**, usata dalla gran parte dei sistemi commerciali
- e. **Problema**: scelta dell'intervallo

2. Rilevamento dello stallo

- a. **Ricerca di cicli** nel grafo delle attese

3. Prevenzione dello stallo

- a. **Uccisione di transazioni "sospetta"**

Scelta del Timeout

- Un **valore troppo elevato** tende a **risolvere tardi** i blocchi critici, dopo che le transazioni coinvolte hanno **trascorso diverso tempo in attesa**.
- Un **valore troppo basso** rischia di interpretare come blocchi anche situazioni in cui una transazione sta attendendo la disponibilità di una risorsa destinata a liberarsi, **uccidendo la transazione e sprecando il lavoro già svolto**.

Scelta della transazione da abortire

- **Politiche interrompenti:** un conflitto può essere risolto uccidendo la **transazione che possiede la risorsa** (in tal modo, essa rilascia la risorsa che può essere concessa ad un'altra transazione). Criterio aggiuntivo: uccidere le transazioni che hanno **svolto meno lavoro** (si spreca meno)
- **Politiche non interrompenti:** una transazione può essere uccisa **solo nel momento** in cui effettua una **nuova richiesta**.

Starvation

Una transazione, all'inizio della propria elaborazione, accede ad un oggetto richiesto da molte altre transazioni, così è **sempre in conflitto** con altre transazioni e, essendo all'inizio del suo lavoro, **viene ripetutamente uccisa. Non c'è deadlock, ma starvation.**

Possibile soluzione: mantenere invariato il tempo di partenza delle transazioni abortite e fatte ripartire, dando in questo modo priorità alle **transazioni più "anziane"**.

Esempio:

$S = r_1(y) w_3(z) r_1(z) r_2(z) w_3(x) w_1(x) w_2(x) r_3(y)$

Mostrare l'esecuzione delle operazioni in S quando:

- Si applica il 2PL stretto
- Si applica il protocollo basato su timestamp

Esempio:

$S = r_1(y)w_3(z)r_1(z)r_2(z)w_3(x)w_1(x)w_2(x)r_3(y)$	
$r_1(y)$	$\rightarrow RL_1(y)$
$w_3(z)$	$\rightarrow WL_3(z)$
$r_1(z)$	$\rightarrow T_1$ waiting for T_3
$r_2(z)$	$\rightarrow T_2$ waiting for T_3
$w_3(x)$	$\rightarrow WL_3(x)$
$r_3(y)$	$\rightarrow RL_3(y)$
$commit(T_3)$	$\rightarrow UL(y), UL(z), UL(x)$, release T_1 and T_2
$r_1(z)$	$\rightarrow RL_1(z)$
$r_2(z)$	$\rightarrow RL_2(z)$
$w_1(x)$	$\rightarrow WL_1(x)$
$commit(T_1)$	$\rightarrow UL(z), UL(x)$
$w_2(x)$	$\rightarrow WL_2(x)$
$commit(T_2)$	$\rightarrow UL(z), UL(x)$

Schedule eseguito: $r_1(y)w_3(z)r_3(y)r_1(z)r_2(z)w_1(x)w_2(x)$

Esempio:

$$S = r_1(y)w_3(z)r_1(z)r_2(z)w_3(x)w_1(x)w_2(x)r_3(y)$$

$r_1(y)$	$\rightarrow RTM(y) = 1$
$w_3(z)$	$\rightarrow WTM(z) = 3$
$r_1(z)$	\rightarrow abort T_1 , restart now as T_4
$r_4(y)$	$\rightarrow RTM(y) = 4$
$r_4(z)$	$\rightarrow RTM(z) = 4$
$r_2(z)$	\rightarrow abort T_2 , restart now as T_5
$r_5(z)$	$\rightarrow RTM(z) = 5$
$w_3(x)$	$\rightarrow WTM(x) = 3$
$w_4(x)$	$\rightarrow WTM(x) = 4$
$w_5(x)$	$\rightarrow WTM(x) = 5$
$r_3(y)$	$\rightarrow RTM(y) = 4$