

# Basi di dati

Giovanna Rosone

# Docente

- ▶ Giovanna Rosone
- ▶ Web: <https://pages.di.unipi.it/rosone/>
- ▶ Email: [giovanna.rosone@unipi.it](mailto:giovanna.rosone@unipi.it)
  
- ▶ Cosa inseguo (quest'anno)?
  - ▶ Teoria dell'Informazione (complementare)
  - ▶ Basi di Dati (obbligatorio)
  - ▶ Laboratorio di Basi di dati (complementare)
  
- ▶ Ricevimento:
  - ▶ remoto/presenza su appuntamento o nello slot predefinito

# Teoria dell'Informazione (complementare - I semestre)

- ▶ Codifica dell'Informazione
  - ▶ Trasmissione dell'informazione (con rumore e senza)
- ▶ Compressione Dati
  - ▶ Tecniche basate sui dizionari
    - ▶ esempio gZip
    - ▶ <https://www.gzip.org/>
  - ▶ Tecniche basate sulla permutazione dei simboli
    - ▶ Esempio bZip
    - ▶ <http://www.bzip.org/>

# Basi di Dati (questo corso)

- ▶ Modelli dei dati, linguaggi e sistemi per lo sviluppo di applicazioni che prevedono l'uso di grandi quantità di dati permanenti organizzati in basi di dati.
- ▶ Argomenti:
  - ▶ Funzionalità dei sistemi per basi di dati-DBMS
  - ▶ La modellazione a oggetti
  - ▶ I sistemi relazionali e il linguaggio SQL
  - ▶ Cenni alla teoria delle basi di dati relazionali
  - ▶ Architettura dei DBMS

# Laboratorio di Basi di dati (complementare)

Argomenti:

- ▶ Approfondire alcune nozioni introdotte a "BASE DI DATI"
- ▶ Progettazione di un'applicazione WEB (**no PHP**) basata sulle basi di dati
- ▶ Uso e amministrazione di un DBMS (in particolare Oracle)
- ▶ Programmazione in un linguaggio per le basi di dati (PL/SQL)
- ▶ Realizzazione di applicazioni con interfaccia grafica
- ▶ Acquisire esperienza su alcuni strumenti, come ORACLE PL/SQL e ORACLE web server

Svolgimento:

- ▶ Lavoro di gruppo in aula e a casa

Esami:

- ▶ Progetto + Discussione

# Laboratorio di Basi di dati (complementare)

## ► Esempio di progetto

The screenshot shows a web application interface for entering company information. At the top, there is a blue header bar with a back arrow, the text 'Call Center', and five dropdown menus labeled 'Gruppo 1' through 'Gruppo 5'. On the right side of the header, it says 'mario.rossi' and has a 'logout' link.

The main content area has a title 'Inserisci la tua Azienda' with a checked checkbox icon. Below it, a red error box displays the message 'Impossibile inviare il modulo!' with the sub-instruction 'Non sono stati compilati tutti i campi richiesti'.

The form fields include:

- Ragione sociale\*: A text input field containing 'Ches' with a red border, indicating it is required.
- Partita Iva: A text input field containing '0123' with a blue border, indicating it is required.
- Indirizzo: A text input field containing 'via d' with a red border, indicating it is required. A tooltip message 'Rispetta il formato richiesto. Inserire un numero di caratteri uguale o superiore a 11' is displayed next to it.
- Telefono: A text input field containing '3 44' with a red border, indicating it is required.
- Email: An empty text input field.
- Website: An empty text input field.

At the bottom of the form are two buttons: 'Conferma' (Confirm) and 'Reset'.

# Basi di dati

# E-learning

- ▶ Password iscrizione
- ▶ Comunicazioni
- ▶ Materiale didattico
- ▶ Esercitazioni
- ▶ Consegna progetto per l'esame
- ▶ Slot esami orali

# TESTI DI RIFERIMENTO

- ▶ A. Albano, G. Ghelli e R. Orsini, Fondamenti di basi di dati, Zanichelli, Bologna, 2005

Liberamente scaricabile da:

<http://fondamentidibasididati.it/>

- ▶ Paolo Atzeni, Stefano Ceri, Piero Fraternali, Stefano Paraboschi, Riccardo Torlone, BASI DI DATI - MODELLI E LINGUAGGI DI INTERROGAZIONE. McGraw-Hill Education

# MODALITÀ DI ESAME

- ▶ Progetto:
  - ▶ Realizzazione di un progetto in 5 giorni (consegna il giorno dell'appello di esame)
  - ▶ Il progetto cambia ad ogni appello di esame e «non viene conservato»
  - ▶ Gruppi di al più 3 studenti
- ▶ Orale (individuale anche se progetto svolto in gruppo):
  - ▶ Obbligatorio
  - ▶ Discussione del progetto e «tutto il resto»

# Principali skills

- ▶ Comprensione e ragionamento
- ▶ Problem solving
- ▶ Pensare alle «conseguenze» delle scelte
- ▶ Memoria: non sufficiente!!!

Suggerimenti:

- ▶ esercitazioni in aula e a casa (possibilmente in gruppo)!!!!
- ▶ «teoria»

# BASI DI DATI, DATA WAREHOUSE, DATA LAKE, DATA SCIENCE...

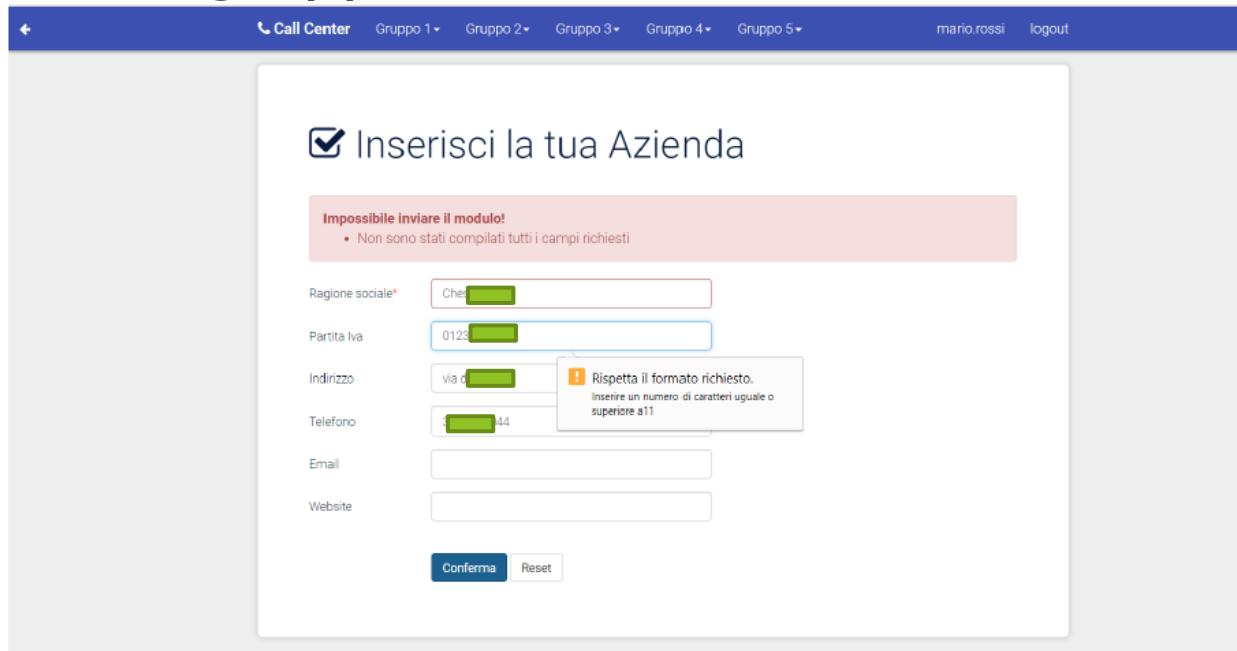
- ▶ Base di dati: tecnologia di base, gestione delle attività quotidiane dell'organizzazione, il tema di questo corso
- ▶ Data Warehouse, Data Lake, Big Data, Data Science: termini che hanno a che vedere con l'analisi dei dati, e non rientrano tra i temi di questo corso

# ASPETTI TRATTATI IN ALTRI CORSI - Triennale

## ► Sperimentazione:

- Laboratorio di Basi di dati: progettazione e implementazione di una base di dati in Oracle PL/SQL  
<http://pages.di.unipi.it/rosone/LBD.html>

## ► Lavoro di gruppo



The screenshot shows a web application interface for entering company information. At the top, there's a blue header bar with navigation links for 'Call Center', 'Gruppo 1', 'Gruppo 2', 'Gruppo 3', 'Gruppo 4', 'Gruppo 5', a user name 'mario.rossi', and a 'logout' link. Below the header is a white form area with a title 'Inserisci la tua Azienda' and a checked checkbox. A red error message box contains the text 'Impossibile inviare il modulo!' and a bullet point 'Non sono stati compilati tutti i campi richiesti'. The form fields include:

- Ragione sociale\*: A field containing 'Che' with a red border.
- Partita Iva: A field containing '0123' with a blue border.
- Indirizzo: A field containing 'via d' with a yellow warning icon and the text 'Rispetta il formato richiesto. Inserire un numero di caratteri uguale o superiore a 11'.
- Telefono: A field containing '44' with a red border.
- Email: An empty field.
- Website: An empty field.

At the bottom of the form are two buttons: 'Conferma' (Confirm) and 'Reset'.

# ASPETTI TRATTATI IN ALTRI CORSI - Magistrali

- ▶ Approfondimento di architetture e algoritmi per SGBD
- ▶ Sistemi per il supporto alle decisioni, business intelligence, data warehousing, data mining, big data
- ▶ Information retrieval

---

# Basi di dati

## Che cosa è una Base di dati

---

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

# ESEMPIO DI BASE DI DATI

## Materie

<b>Titolo</b>	<b>Codice</b>	<b>Syllabus</b>
Basi di Dati	AA024	Progettazione e interrogazione...
Reti di Calc.	AA019	Realizzazione e uso di reti - protocollo TCP

## Corsi

<b>Materia</b>	<b>AA</b>	<b>Sem</b>	<b>Titolare</b>
AA024	2014	1	Albano
AA024	2017	1	Ghelli
AA019	2017	1	Rossi
AA024	2019	2	Bianchi



# COSTRUZIONE DI UNA BASE DI DATI: FIGURE COINVOLTE

---

- Committente
  - Dirigente
  - Operatore
- Fornitore
  - Direttore del progetto
  - Analista
  - Progettista di BD
  - Programmatore di applicazioni che usano BD
- Manutenzione e messa a punto della BD - Gestione del DBMS
  - Amministratore del DBMS

# SISTEMI INFORMATIVI

---

- Le basi di dati hanno quindi un ruolo nel **sistema informativo e nell'organizzazione di un'azienda**.
- **Definizione** Un **sistema informativo** di un'organizzazione è una combinazione di risorse, umane e materiali, e di procedure organizzate per:
  - la raccolta,
  - l'archiviazione,
  - l'elaborazione e
  - lo scambio

delle informazioni necessarie alle attività:

- operative (informazioni di servizio),
- di programmazione e controllo (informazioni di gestione), e
- di pianificazione strategica (informazioni di governo).

# ESEMPI DI SISTEMI INFORMATIVI

---

- Azienda manifatturiera
  - Gestione degli ordini dei clienti
  - Gestione degli ordini e dei pagamenti di materiali ai fornitori
  - Gestione del magazzino
  - Pianificazione della produzione e controllo dei costi

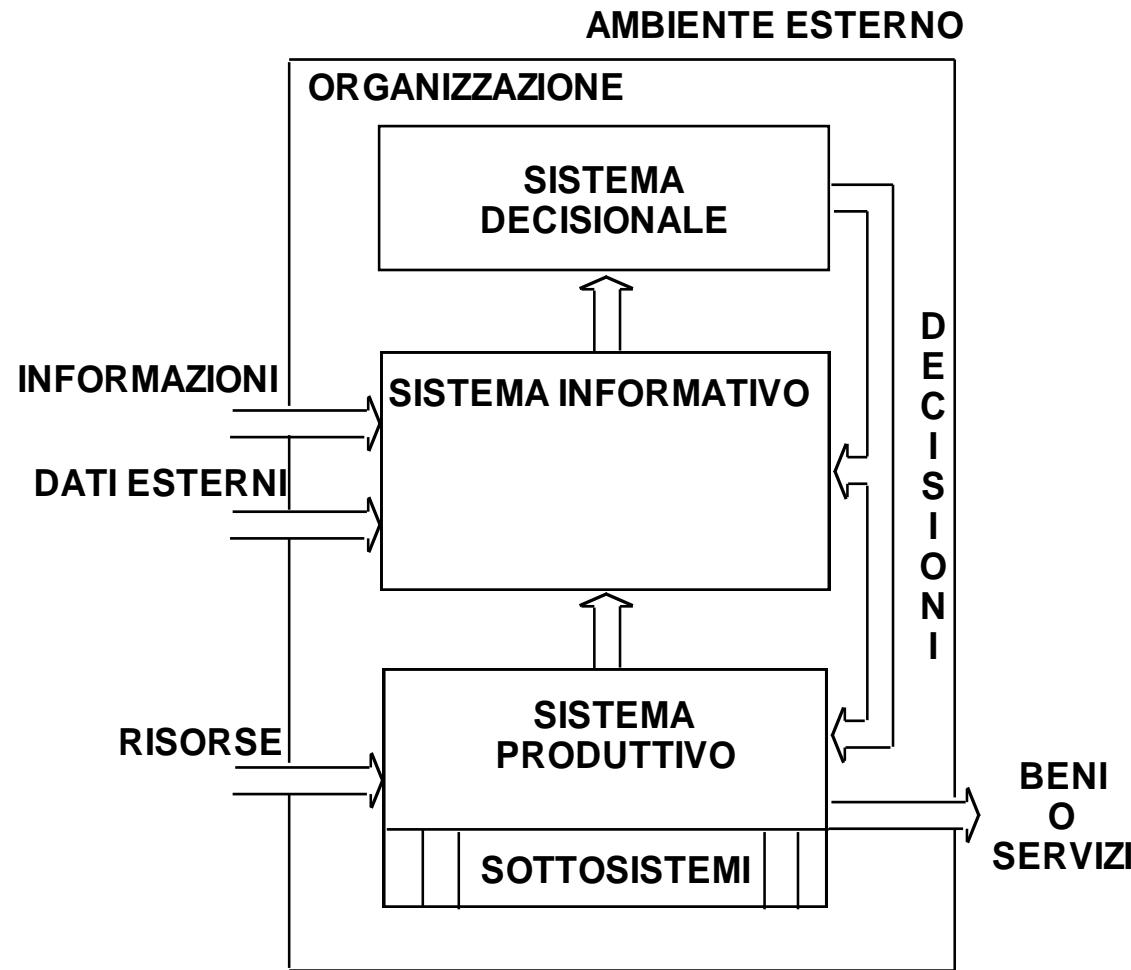
# ESEMPI DI SISTEMI INFORMATIVI

---

- Un Comune
  - Gestione dei servizi demografici (anagrafe, stato civile, servizio elettorale e vaccinale) e della rete viaria.
  - Gestione dell'attività finanziaria secondo la normativa vigente.
  - Gestione del personale per il calcolo della retribuzione in base al tipo di normativa contrattuale.
  - Gestione dei servizi amministrativi e sanitari delle Unità Sanitarie Locali.
  - Gestione della cartografia generale e tematica del territorio.

# SISTEMA INFORMATIVO NELLE ORGANIZZAZIONI

---



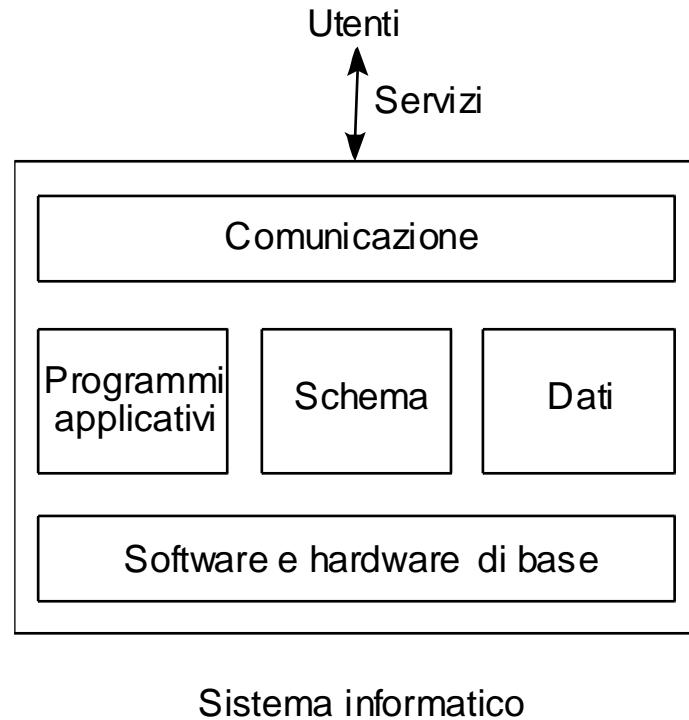
# SISTEMI INFORMATICI

- Il **sistema informativo automatizzato** è quella parte del sistema informativo in cui le informazioni sono raccolte, elaborate, archiviate e scambiate usando un **sistema informatico**.
- Il **sistema informatico** è l'insieme delle tecnologie informatiche e della comunicazione (Information and Communication Technologies, ICT) a supporto delle attività di un'organizzazione.
- Terminologia
  - sistema informativo ≈ sistema informativo automatizzato
  - sistema informativo automatizzato ≈ sistema informatico



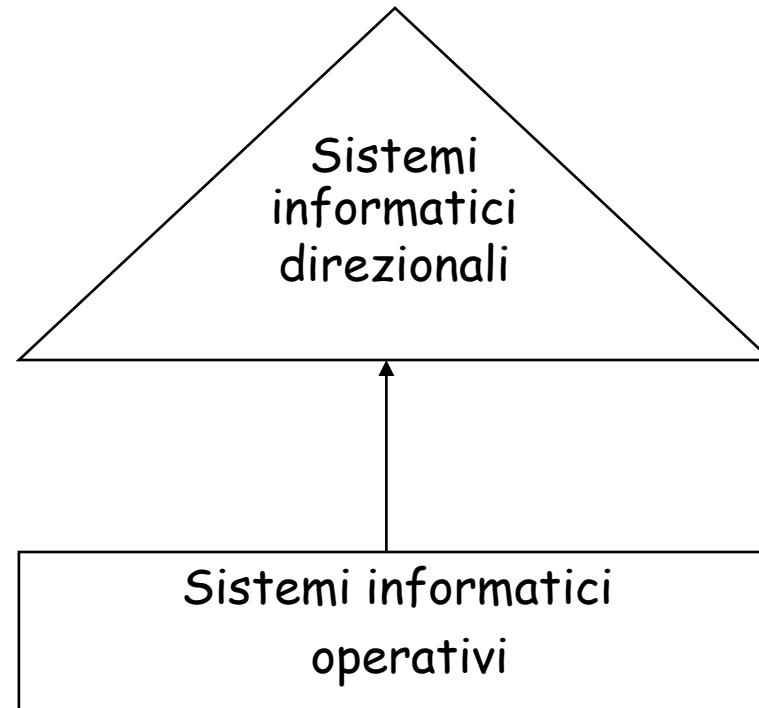
# COMPONENTI DI UN SISTEMA INFORMATICO

---



# CLASSIFICAZIONE DEI SISTEMI INFORMATICI

---



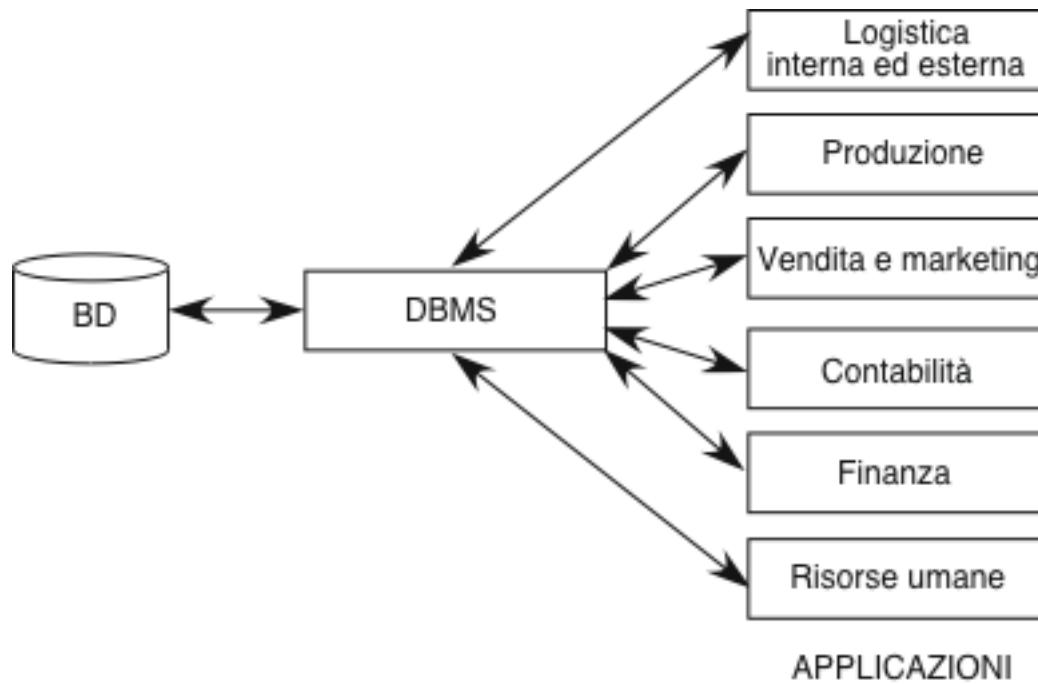
# SISTEMI INFORMATICI OPERATIVI

---

- I dati sono organizzati in BD
- Le applicazioni si usano per svolgere le classiche **attività strutturate e ripetitive dell'azienda** nelle aree amministrativa e finanziaria, vendite, produzione, risorse umane ecc.
- Alcune sigle
  - Data processing (DP), Electronic Data processing (EDP)
  - Transaction Processing Systems (TPS)

## SISTEMA INFORMATICO OPERATIVO (cont.)

---



Le caratteristiche delle basi di dati sono garantite da un sistema per la gestione di basi di dati (**Data Base Management System, DBMS**), che ha il controllo dei dati e li rende accessibili agli utenti autorizzati.

# ELABORAZIONI SU BD: OLTP (On-Line Transaction Processing)

## Sistemi transazionali

---

- Uso principale dei DBMS
- Tradizionale elaborazione di transazioni, che realizzano i processi operativi per il funzionamento di organizzazioni:
  - Operazioni predefinite e relativamente semplici
  - Ogni operazione coinvolge "pochi" dati
  - Dati di dettaglio, aggiornati

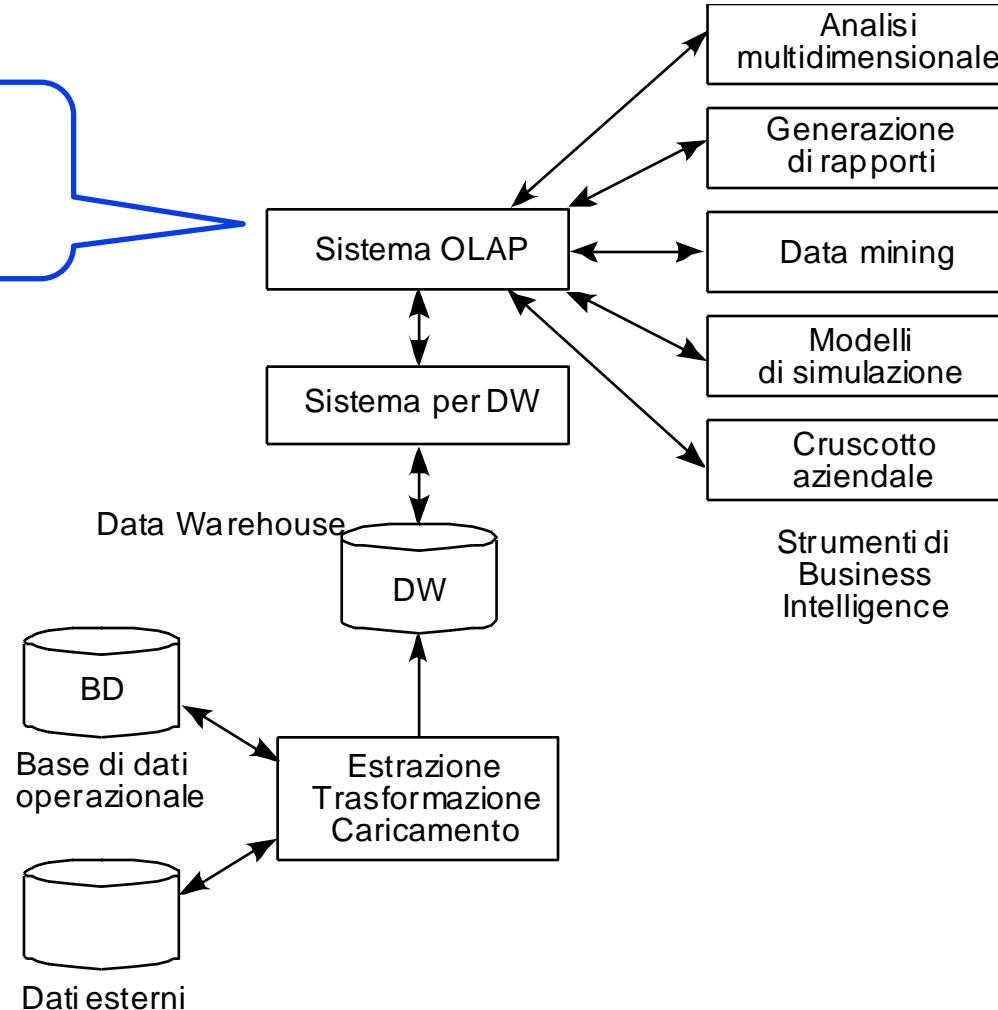
# SISTEMI INFORMATICI DIREZIONALI

---

- I dati sono organizzati in **Data Warehouse** (DW) e gestiti da un opportuno sistema
- Le applicazioni, dette di **Business intelligence**, sono strumenti di supporto ai processi di controllo delle prestazioni aziendali e di decisione manageriale
- Terminologia anglosassone:
  - Management Information Systems (MIS)
  - Decision support systems (DSS), data or model based
  - Executive Information System (EIS)

# SISTEMA INFORMATICO DIREZIONALE

**OLAP**  
(On-Line Analytical  
Processing)



## ELABORAZIONI SU DW: OLAP (On-Line Analytical Processing)

---

- Uso principale dei **data warehouse**
- Analisi dei dati di supporto alle decisioni
  - Operazioni complesse e non-ripetitive
  - Ogni operazione può coinvolgere molti dati
  - Dati aggregati, storici, anche non attualissimi

# DIFFERENZE TRA OLTP E OLAP

---

	OLTP	OLAP
<b>Scopi</b>	Supporto operatività	Supporto decisioni
<b>Utenti</b>	Molti, esecutivi	Pochi, dirigenti e analisti
<b>Dati</b>	Analitici, relazionali	Sintetici, multidimensionali
<b>Usi</b>	Noti a priori	Poco prevedibili
<b>Quantità di dati per attività</b>	Bassa (decine)	Alta (milioni)
<b>Orientamento</b>	Applicazione	Soggetto
<b>Aggiornamenti</b>	Frequenti	Rari
<b>Visione dei dati</b>	Corrente	Storica
<b>Ottimizzati per</b>	Transazioni	Analisi dei dati

## ANALISI DEI DATI: REQUISITI

---

- Dati **aggregati**: non interessa un dato, ma la somma, la media, il minimo, il massimo di una misura.
- Presentazione **multidimensionale**: interessa incrociare le informazioni, per analizzarle da punti di vista diversi e valutare i risultati del business per intervenire sui problemi critici o per cogliere nuove opportunità.
- Analisi a **diversi livelli di dettaglio**: per es. una volta scoperto un calo delle vendite in un determinato periodo in una regione specifica, si passa ad un'analisi dettagliata nell'area di interesse per cercare di scoprire le cause (dimensioni con gerarchie).

# BIG DATA

---

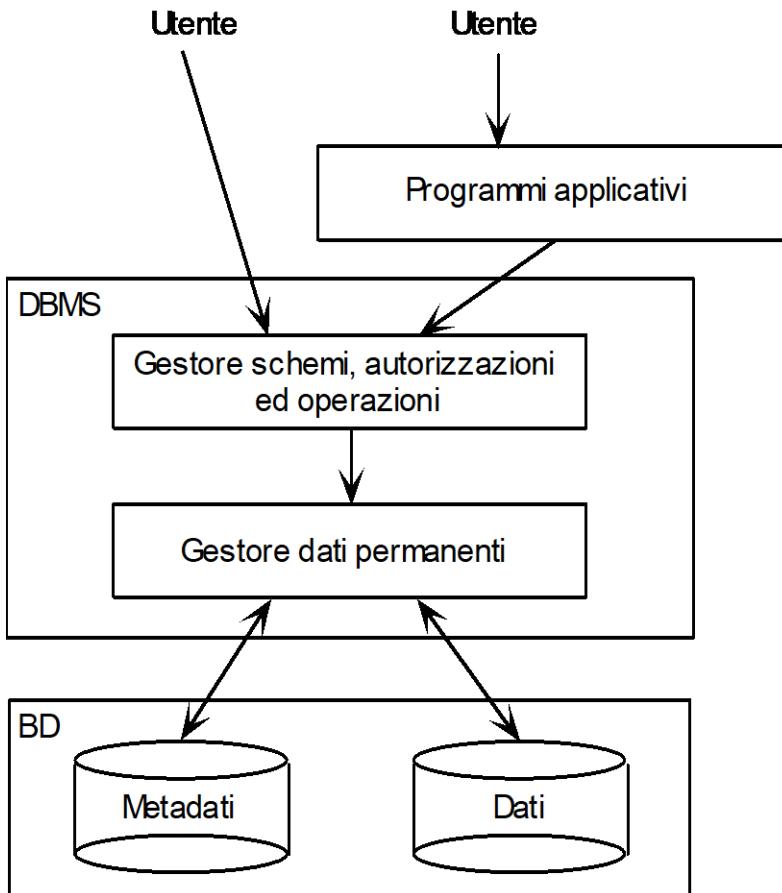
- Big Data è un termine ampio, riferito a situazioni in cui l'approccio '**schema-first**' tipico di DB o DW risulta troppo **restrittivo** o troppo **lento**
- Le tre V:
  - Volume
  - Varietà
  - Velocità
- I Big Data sono in genere associati a
  - Sistemi NoSQL
  - Machine learning
  - Approccio Data Lake

# SISTEMI PER BASI DI DATI (DATA BASE MANAGEMENT SYSTEMS - DBMS)

---

- **Definizione:** Un **DBMS** è un sistema centralizzato o distribuito che offre opportuni linguaggi per:
  - definire lo **schema** di una basi di dati (lo schema va definito prima di creare dati),
  - scegliere le **strutture dati** per la memorizzazione dei dati,
  - memorizzare i dati rispettando i **vincoli** definiti nello schema;
  - recuperare e modificare i dati interattivamente (**linguaggio di interrogazione** o **query language**) o da programmi.

# ARCHITETTURA DEI DBMS CENTRALIZZATI



- Una base di dati è una raccolta di dati permanenti suddivisi in due categorie:
  - i **metadati**: descrivono fatti sullo schema dei dati, utenti autorizzati, applicazioni, parametri quantitativi sui dati, ecc.
    - I metadati sono descritti da uno schema usando il modello dei dati adottato dal DBMS e sono interrogabili con le stesse modalità previste per i dati;
  - i **dati**: le rappresentazioni di certi fatti conformi alle definizioni dello schema, con le seguenti caratteristiche.



## CARATTERISTICHE DEI DATI GESTITI DAI DBMS

---

- Sono organizzati in **insiemi strutturati e omogenei**, fra i quali sono definite delle **relazioni**. La struttura dei dati e le relazioni sono descritte nello schema usando i meccanismi di astrazione del modello dei dati del DBMS;

## CARATTERISTICHE DEI DATI GESTITI DAI DBMS

---

- Sono organizzati in **insiemi strutturati e omogenei**, fra i quali sono definite delle **relazioni**. La struttura dei dati e le relazioni sono descritte nello schema usando i meccanismi di astrazione del modello dei dati del DBMS;
- Sono **molti**, in assoluto e rispetto ai metadati, e non possono essere gestiti in memoria temporanea;
- Sono accessibili mediante **transazioni**, unità di lavoro atomiche che non possono avere effetti parziali;
- Sono **protetti** sia da accesso da parte di utenti non autorizzati, sia da corruzione dovuta a malfunzionamenti hardware e software;
- Sono **utilizzabili contemporaneamente** da utenti diversi.

# UN ESEMPIO DI SESSIONE CON UN DBMS RELAZIONALE

---

- Il **modello relazionale** dei dati è il più diffuso fra i DBMS commerciali.
- Il meccanismo di astrazione fondamentale è la **relazione (tabella)**, sostanzialmente un insieme di record con campi elementari;
- Lo schema di una relazione ne definisce il nome e descrive la struttura dei possibili elementi della relazione (insieme di attributi con il loro tipo).
- Definizione base di dati:
  - **create database** EsempioEsami
- Definizione schema:
  - **create table** Esami (Materia char(5), Candidato char(8), Voto int, Lode char(1), Data char(6))

## UN ESEMPIO DI SESSIONE (cont)

---

- Inserzione dati:

```
➤insert into Esami values ('BDSI1','080709',30, 'S',  
070900)
```

- Interrogazione:

```
➤select Candidato  
➤from Esami  
➤where Materia = "BDSI1" and Voto = 30
```

- Candidato
  - 080709
- Inoltre: accesso ai dati da programma, interrogazioni con interfaccia grafica, direttive per definire le strutture fisiche...

# FUNZIONALITÀ DEI DBMS

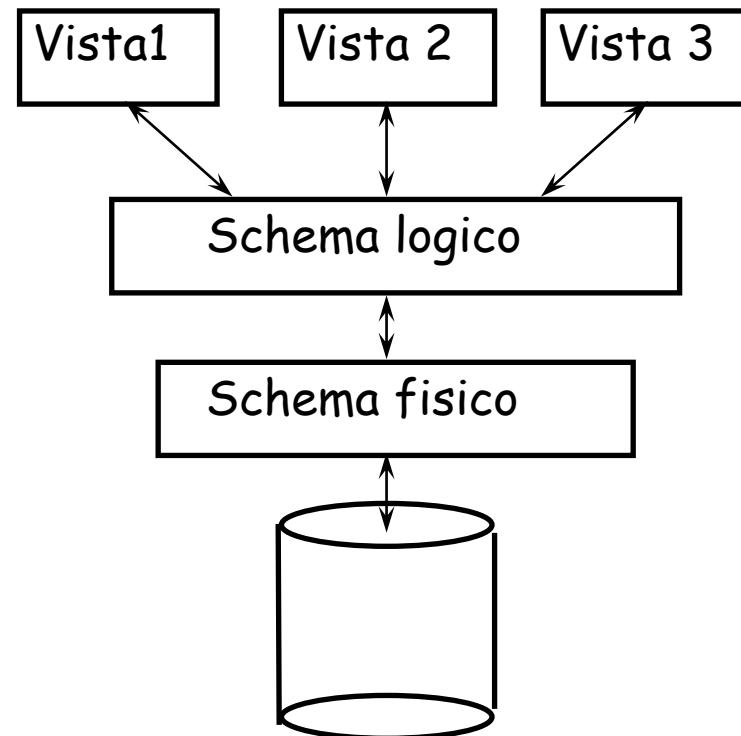
---

- Linguaggio per la definizione della base di dati;
- Linguaggi per l'uso dei dati;
- Meccanismi per il controllo dei dati;
- Strumenti per il responsabile della base di dati;
- Strumenti per lo sviluppo delle applicazioni

# LINGUAGGIO PER LA DEFINIZIONE DELLA BASE DI DATI (DDL)

---

- È utile distinguere tre diversi livelli di descrizione dei dati (**schemi**):
  - il livello di vista logica,
  - il livello logico,
  - il livello fisico.



## LIVELLO LOGICO

---

- **Livello logico:** descrive la struttura degli insiemi di dati e delle relazioni fra loro, secondo un certo modello dei dati, senza nessun riferimento alla loro organizzazione fisica nella memoria permanente (Schema logico).
- Esempio di schema logico:

Studenti(Matricola char(8), Nome char(20), login char(8),  
AnnoNascita int, Reddito real)

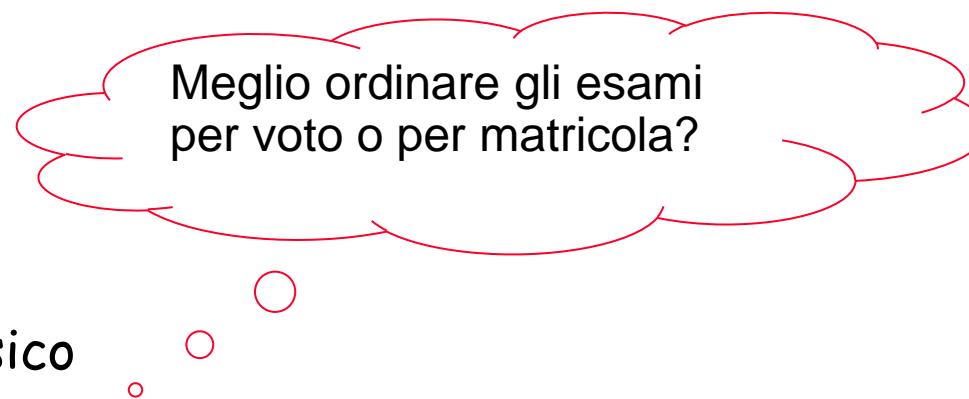
Corsi(IdeC char(8), Titolo char(20), Credito int)

Esami(Matricola char(8), IdeC char(8), Voto int)

## LIVELLO FISICO

---

- **Livello fisico:** descrive come vanno organizzati **fisicamente** i dati nelle memorie permanenti e quali strutture dati ausiliarie prevedere per facilitarne l'uso (Schema fisico o interno).



- Esempio di schema fisico
  - Relazioni Studenti e Esami organizzate in modo seriale, Corsi organizzata sequenziale con indice
  - Indice su Matricola, (Matricola, IdeC)

## LIVELLO VISTA LOGICA

---

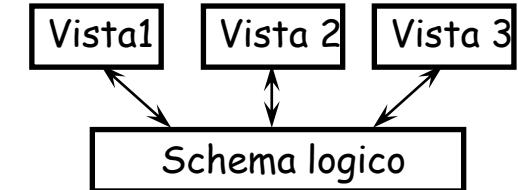
- **Vista logica:** descrive come deve apparire la struttura della base di dati ad una certa applicazione (**Schema esterno o vista**).

- Esempio di schema esterno

- InfCorsi (IdeC char(8), Titolo char(20), NumEsami int)

- Esempio:

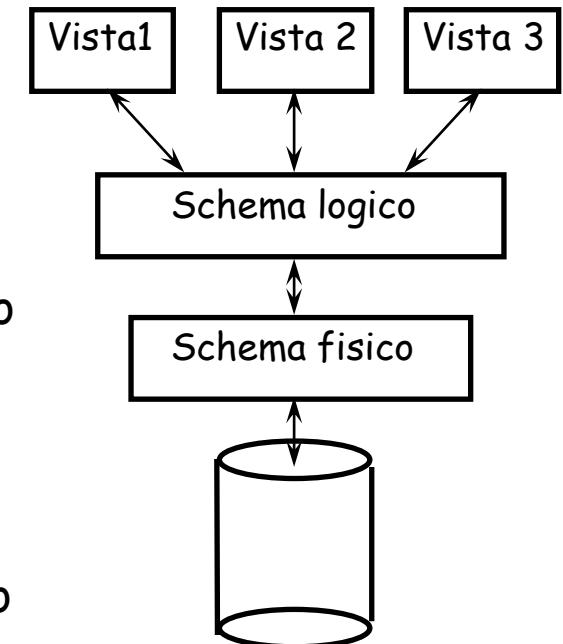
- Nell'organizzazione di una banca, lo **schema logico** conterrà tutte le tabelle e i dati relativi ai conti correnti, ma anche al personale. Lo schema logico conserva tutte le informazioni della banca.
  - Nello **schema esterno**, ogni correntista potrà accedere solo ad alcune informazioni che sono per lui di interesse: quelle relative al **proprio** conto corrente.



## LINGUAGGIO PER LA DEFINIZIONE DELLA BD (cont.)

---

- L'approccio con tre livelli di descrizione dei dati è stato proposto come un modo per garantire le **proprietà di indipendenza logica e fisica dei dati**, che sono fra gli obiettivi più importanti dei DBMS.



- **Indipendenza logica:** i programmi applicativi non devono essere modificati in seguito a *modifiche dello schema logico*.
- **Indipendenza fisica:** i programmi applicativi non devono essere modificati in seguito a *modifiche dell'organizzazione fisica dei dati*.

# FUNZIONALITÀ DEI DBMS: LINGUAGGI PER L'USO DEI DATI

---

- Un **DBMS** deve prevedere **più modalità d'uso** per soddisfare le esigenze delle diverse categorie di utenti che possono accedere alla base di dati (dati e catalogo):
  - Un'interfaccia grafica per accedere ai dati
  - Un linguaggio di interrogazione per gli utenti non programmati
  - Un linguaggio di programmazione per gli utenti che sviluppano applicazioni:
    - integrazione DDL e DML nel linguaggio ospite: procedure predefinite, estensione del compilatore, precompilazione
    - comunicazione tra linguaggio e DBMS
  - Un linguaggio per lo sviluppo di interfacce per le applicazioni

## Una distinzione terminologica (separazione fra dati e programmi)

---

Nei linguaggi di interrogazione di basi di dati  
distinguiamo tra

### Data Manipulation Language (DML)

per l'interrogazione e l'aggiornamento di  
(*istanze* di) basi di dati

```
Select insegnamento  
From Orario  
Where docente="Mario Rossi"
```

### Data Definition Language (DDL)

per la definizione di *schemi* (logici, esterni,  
fisici) e altre operazioni generali

```
CREATE TABLE orario (  
    insegnamento CHAR(20) ,  
    docente      CHAR(20) ,  
    aula         CHAR(4) ,  
    ora          CHAR(5)  
)
```

## **FUNZIONALITÀ DEI DBMS (cont.)**

---

- **Strumenti per l'amministratore** della base di dati
  - Un linguaggio per la definizione e la modifica degli schemi logico, interno ed esterno.
  - Strumenti per il controllo e messa a punto del funzionamento del sistema.
  - Strumenti per stabilire i diritti di accesso ai dati e per ripristinare la base di dati in caso di malfunzionamenti.
- **Strumenti per lo sviluppo** delle applicazioni
  - Produzione di rapporti, grafici, fogli elettronici
  - Sviluppo di menu, forme, componenti grafici

## Linguaggi per le basi di dati

---

- Disponibilità di vari linguaggi e interfacce diverse
  - linguaggi testuali interattivi (es. **SQL**)
  - comandi (come quelli del linguaggio interattivo) immersi in un linguaggio **ospite** (es. C, Cobol, etc.)
  - comandi (come quelli del linguaggio interattivo) immersi in un linguaggio **ad hoc** (es. PL/SQL), con anche altre funzionalità (p.es. per grafici o stampe strutturate), anche con l'ausilio di strumenti di sviluppo (p. es. per la gestione di maschere)
  - con interfacce amichevoli (senza linguaggio testuale)

# SQL immerso in linguaggio ad hoc (Oracle PL/SQL)

---

```
declare Stip number;
begin
    select Stipendio into Stip from Impiegato where Matricola = '575488'
    for update of Stipendio;
    if Stip > 30 then
        update Impiegato set Stipendio = Stipendio * 1.1 where Matricola = '575488';
    else
        update Impiegato set Stipendio = Stipendio * 1.15 where Matricola = '575488';
    end if;
    commit;
exception
    when no_data_found then
        insert into Errori values('Non esiste la matricola specificata',sysdate);
end;
```

# SQL, un linguaggio interattivo

DS3 dove si trova?

## Corsi

Corso	Docente	Aula
Basi di dati	Rossi	DS3
Sistemi	Neri	N3
Reti	Bruni	N3
Controlli	Bruni	G

## Aule

NomeAula	Edificio	Piano
DS1	OMI	Terra
N3	OMI	Terra
G	Pincherle	Primo

SELECT Corso, Aula, Piano

FROM Aule, Corsi

WHERE NomeAula = Aula

AND Piano="Terra"

Corso	Aula	Piano
Sistemi	N3	Terra
Reti	N3	Terra

# FUNZIONALITÀ DEI DBMS: MECCANISMI PER IL CONTROLLO DEI DATI

---

- Una caratteristica molto importante dei DBMS è il tipo di meccanismi offerti per garantire le seguenti proprietà di una base di dati:
  - **Integrità**: mantenimento delle proprietà specificate in modo dichiarativo nello schema (vincoli d'integrità)
  - **Sicurezza**: protezione dei dati da usi non autorizzati
  - **Affidabilità**: protezione dei dati da malfunzionamenti hardware o software (fallimenti di transazione, di sistema e disastri) e da *interferenze indesiderate* dovute all'accesso concorrente ai dati da parte di più utenti.

# CONTROLLO DEI DATI: LE TRANSAZIONI

---

- **Definizione** Una **transazione** è una sequenza di azioni di lettura e scrittura in memoria permanente e di elaborazioni di dati in memoria temporanea, con le seguenti proprietà:
  - **Atomicità**: Le transazioni che terminano prematuramente (aborted transactions) sono trattate dal sistema come se non fossero mai iniziate; pertanto eventuali loro effetti sulla base di dati sono annullati.
  - **Persistenza**: Le modifiche sulla base di dati di una transazione terminata normalmente sono permanenti, cioè non sono alterabili da eventuali malfunzionamenti.
  - **Serializzabilità**: Nel caso di esecuzioni concorrenti di più transazioni, l'effetto complessivo è quello di una esecuzione seriale.

# SISTEMI PER BASI DI DATI (DATA BASE MANAGEMENT SYSTEMS - DBMS)

---

Un DataBase Management System (DBMS) è un sistema (prodotto software) in grado di gestire collezioni di dati che siano (anche):

- **Grandi**
- **Persistenti** (con un periodo di vita indipendente dalle singole esecuzioni dei programmi che le utilizzano)
- **Condivise** (utilizzate da applicazioni diverse)

garantendo **affidabilità** (resistenza a malfunzionamenti hardware e software-recovery) e **privatezza** (con una disciplina e un controllo degli accessi). Come ogni prodotto informatico, un DBMS deve essere **efficiente** (utilizzando al meglio le risorse di spazio e tempo del sistema) ed **efficace** (rendendo produttive le attività dei suoi utilizzatori).

## RIEPILOGO DEI VANTAGGI DEI DBMS

---

- Indipendenza dei dati
- Recupero efficiente dei dati
- Integrità e sicurezza dei dati
- Accessi interattivi, concorrenti e protetti dai malfunzionamenti
- Amministrazione dei dati
- Riduzione dei tempi di sviluppo delle applicazioni
- La riduzione dei costi della tecnologia e i possibili tipi di DBMS disponibili sul mercato facilitano la loro diffusione.

## SVANTAGGI DEI DBMS

---

- Prima di caricare i dati è necessario definire uno schema
- Possono gestire solo dati strutturati e omogenei
- Possono essere costosi e complessi da installare e mantenere in esercizi
- Sono ottimizzati per le applicazioni OLTP, non per quelle OLAP

## COSTRUZIONE DI UNA BASE DI DATI: FIGURE COINVOLTE

- Committente
  - Dirigente
  - Operatore
- Fornitore
  - Direttore del progetto
  - Analista
  - Progettista di BD
  - Programmatore di applicazioni che usano BD
- Manutenzione e messa a punto della BD - Gestione del DBMS
  - Amministratore del DBMS

# LA PROGETTAZIONE DI BASI DI DATI

---

- Progettare una basi di dati vuole dire progettare la struttura dei dati e le applicazioni
- La progettazione dei dati è l'attività più importante
- Per progettare i dati al meglio è necessario che i dati siano un modello fedele del dominio del discorso
- Per questo studiamo ora la **MODELLAZIONE**

# MODELLI INFORMATICI

---

- **Definizione:** Un modello **astratto** è la rappresentazione formale di idee e conoscenze relative a un fenomeno.
- Aspetti di un modello:
  - il modello è la rappresentazione di certi fatti;
  - la rappresentazione è data con un linguaggio formale;
  - il modello è il risultato di un processo di interpretazione, guidato dalle idee e conoscenze possedute dal soggetto che interpreta.
- La stessa realtà può utilmente essere modellata in modi diversi, e a diversi livelli di astrazione

# LA PROGETTAZIONE DELLE BASI DI DATI

---



- Ciascuna di queste fasi è centrata sulla modellazione
- La modellazione verrà discussa quindi con riferimento alla problematica della progettazione delle basi di dati

# Metodologia in più fasi



## Fasi



## Modello dei dati

---

- insieme di costrutti utilizzati per organizzare i dati di interesse e descriverne la dinamica
- componente fondamentale: **meccanismi 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
- ad esempio, il **modello relazionale** prevede il costruttore **relazione**, che permette di definire insiemi di record omogenei

## ASPETTI DEL PROBLEMA

---

- Quale conoscenza del dominio del discorso si rappresenta?
  - aspetto **ontologico** (studio di ciò che esiste): ciò che si suppone esistere nell'universo del discorso e quindi sia da modellare.
- Con quali meccanismi di astrazione si modella?
  - aspetto **logico**
- Con quale linguaggio formale si definisce il modello?
  - aspetto **linguistico**
- Come si procede per costruire un modello?
  - aspetto **pragmatico**: metodologia (insieme di regole finalizzate alla costruzione del modello informatico) da seguire nel processo di modellazione



## ASPETTO ONTOLOGICO: COSA SI MODELLA

---

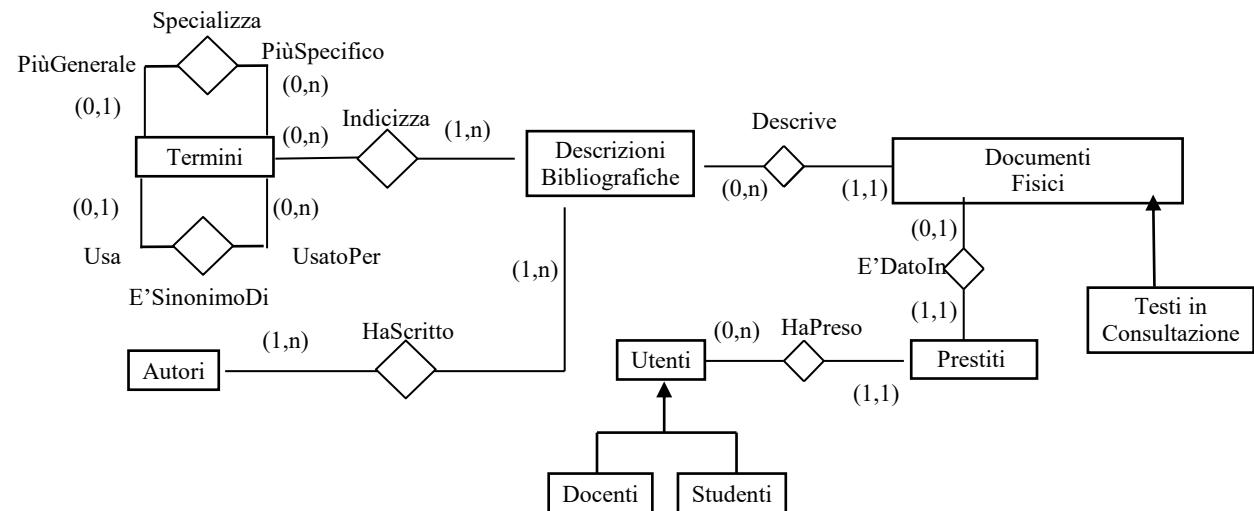
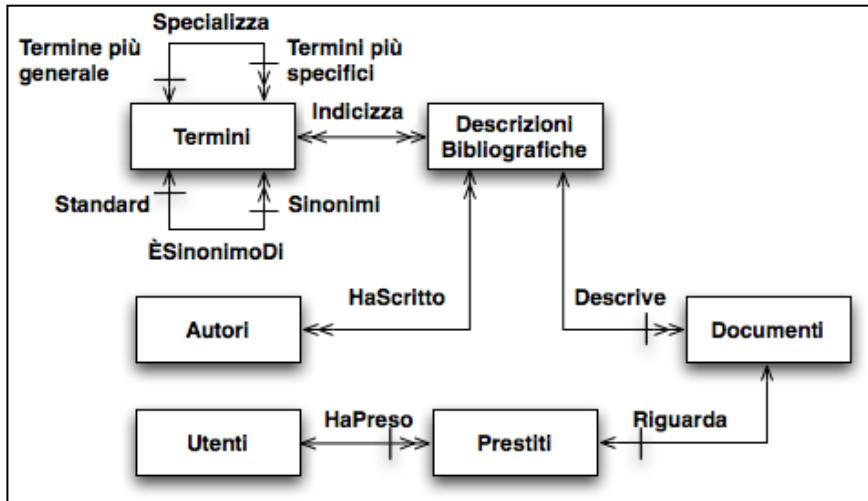
- Conoscenza **concreta**:
  - I **fatti**
- Conoscenza **astratta**
  - **Struttura e vincoli** sulla conoscenza concreta
- Conoscenza **procedurale**, comunicazioni:
  - Le operazioni di base, le operazioni degli utenti
  - Come si comunicherà con il sistema informatico
- Nel seguito l'attenzione sarà sulla conoscenza **concreta e astratta**.

## ASPETTO LOGICO: IL MODELLO DEI DATI A OGGETTI

---

- Un **modello dei dati** è un insieme di meccanismi di astrazione per descrivere la struttura della conoscenza concreta.
- **Schema**: la descrizione della struttura della conoscenza concreta e dei vincoli di integrità usando un particolare *modello dei dati*.
- Useremo come notazione grafica una variante dei cosiddetti diagrammi a oggetti o diagrammi ER (Entità-Relazione)
- Nozioni fondamentali:
  - Oggetto, Tipo di oggetto, Classe
  - Ereditarietà, Gerarchia fra tipi, Gerarchia fra classi

# Esempi di modellazioni



## COSA SI MODELLA: LA CONOSCENZA CONCRETA

---

- La **conoscenza concreta** riguarda i fatti specifici che si vogliono rappresentare.
- Fatti specifici che si vogliono rappresentare:
  - le *entità* con le loro proprietà,
  - le *collezioni* di entità omogenee e
  - le *associazioni* fra entità.

## CONOSCENZA CONCRETA: ENTITÀ E PROPRIETÀ

---

- Le **entità** sono ciò di cui interessa rappresentare alcuni fatti (o proprietà) (es.: una descrizione bibliografica di un libro, un libro o documento fisico, un prestito, un utente della biblioteca).
- Le **proprietà** sono fatti che interessano solo in quanto descrivono caratteristiche di determinate entità (es.: un indirizzo interessa perché è l'indirizzo di un utente).

## Esempio di entità e proprietà

---

Una catena di negozi vende elettrodomestici e mantiene informazioni sui relativi centri di assistenza. Per ogni modello di elettrodomestico interessa il tipo, il peso, il prezzo di listino, e la marca.

Domanda:

Identificare entità e proprietà

## CONOSCENZA CONCRETA: COLLEZIONI DI ENTITÀ

---

- Una **proprietà** è una coppia (Attributo, valore di un certo tipo).
- **Tipi di entità**: ogni entità appartiene ad un tipo che ne specifica la natura.
  - Ad es. Antonio ha tipo Persona con proprietà (Nome: string) e (Indirizzo:string).
- **Collezione**: un insieme variabile nel tempo di entità omogenee (dello stesso tipo).
  - Ad es. la collezione di tutte le persone nel dominio del discorso.

## Caratteristiche delle proprietà

---

- Ogni proprietà ha associato un dominio, ovvero l'insieme dei possibili valori che tale proprietà può assumere:
  - proprietà **atomica** (o primitiva), se il suo valore non è scomponibile; altrimenti è detta **strutturata**;
  - proprietà **univoca**, se il suo valore è unico; altrimenti è detta **multivaleore**;
  - proprietà **totale** (obbligatoria), se ogni entità dell'universo del discorso ha per essa un valore specificato, altrimenti è detta **parziale** (opzionale)
  - Proprietà **costante**, altrimenti è detta **variabile**
  - Proprietà **calcolata**, altrimenti è detta **non calcolata**

## CONOSCENZA ASTRATTA: ESEMPI DI TIPI DI ENTITÀ

---

Tipo Entità	Attributi
Studente	Nome, AnnoNascita, Matricola, E-mail, ...
Esame	Materia, Candidato, Voto, Lode, Data, ...
Auto	Modello, Colore, Cilindrata, Targa, ...
Descrizione bibliografica	Autori, Titolo, Editore, LuogoEdizione, ...

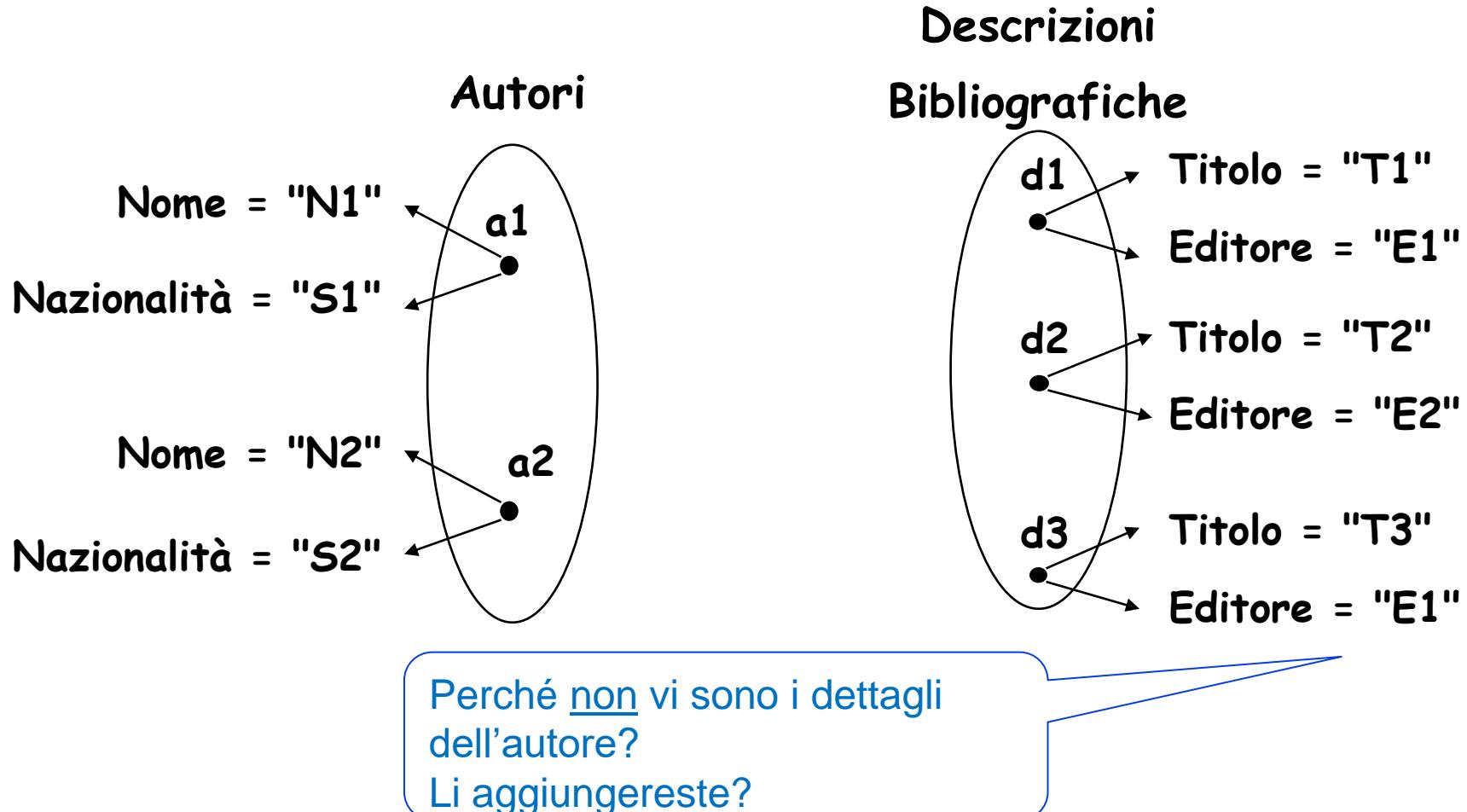
## Esempio Biblioteca

---

- Si vuole gestire una biblioteca, che deve consentire il prestito di libri da parte di utenti.
- Riflettere sui:
  - fatti con le relative proprietà
  - legami fra i fatti

## CONOSCENZA CONCRETA: COLLEZIONI

Una **collezione** è un insieme variabile nel tempo di entità omogenee interessanti dell'universo del discorso.



## Esercizio: individuare entità e proprietà

---

Una catena di negozi vende elettrodomestici e mantiene informazioni sui relativi centri di assistenza. Per ogni modello di elettrodomestico interessa il tipo, il peso, il prezzo di listino, e la marca.

Per ogni marca di elettrodomestico interessa il nome e interessa conoscere i relativi centri di assistenza.

Per ogni marca ci possono essere numerosi modelli di elettrodomestici. Di un centro d'assistenza interessano il nome, l'indirizzo, un insieme di recapiti telefonici, e l'insieme di marche per le quali offre assistenza.

La catena tiene inoltre traccia di eventuali reclami ricevuti.

Per ogni reclamo interessano la data e il nome indicato dal segnalante. Se il reclamo riguarda un negozio, interessa inoltre conoscere il negozio in questione.

Per ogni negozio, la catena tiene traccia di un nome, un indirizzo, e l'insieme dei reclami ricevuti. Se invece riguarda un'operazione di assistenza, interessa tenere traccia del modello di elettrodomestico in questione e del centro di assistenza che ha gestito la riparazione.

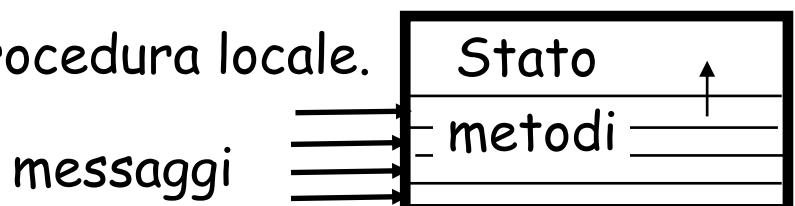
Il centro di assistenza deve essere uno di quelli che offrono assistenza per la marca del modello di elettrodomestico.

---

## MODELLAZIONE A OGGETTI: GLI OGGETTI

---

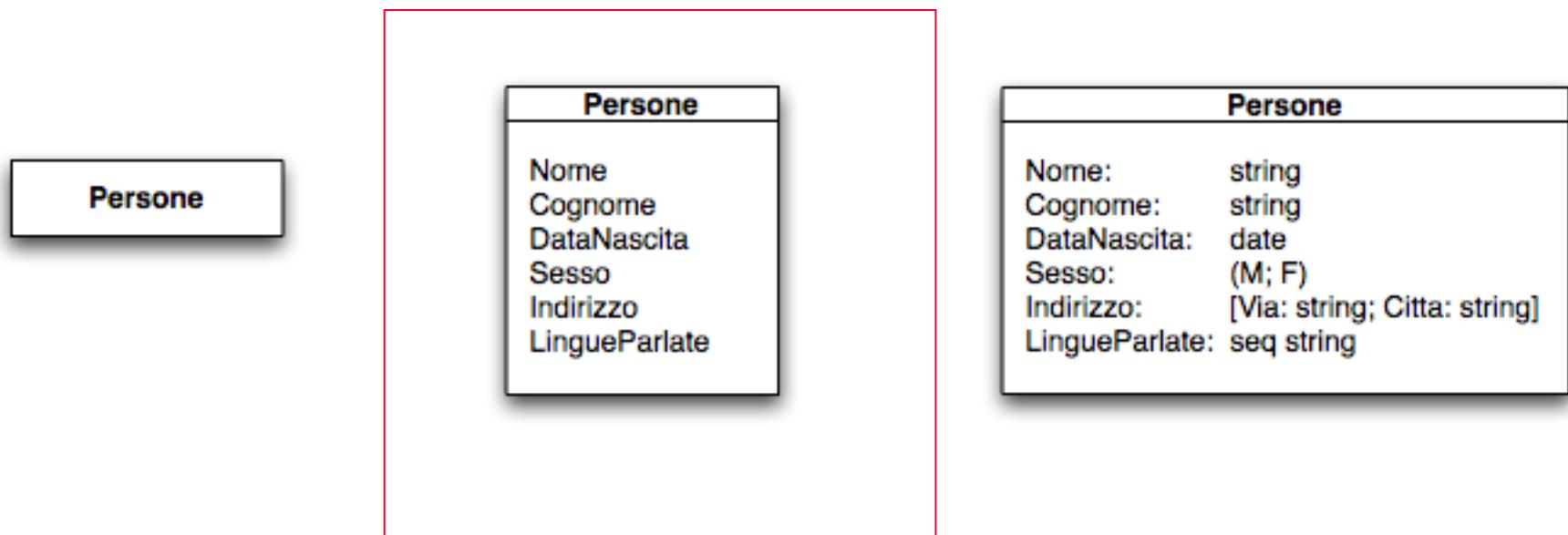
- Ad ogni entità del dominio corrisponde un oggetto del modello.
- **Oggetto**: un'entità software con **stato**, **comportamento** e **identità**, che modella un'entità dell'universo:
  - Lo **stato** è modellato da un insieme di costanti o variabili con valori di qualsiasi complessità.
  - **Comportamento**: un insieme di procedure locali chiamate **metodi**, che modellano le operazioni di base che riguardano l'oggetto e le proprietà derivabili da altre.
- Un **oggetto** può rispondere a dei **messaggi**, restituendo valori memorizzati nello stato o calcolati con una procedura locale.



# MODELLO A OGGETTI: LE CLASSI

---

- Una **classe** è un insieme di oggetti dello stesso tipo, modificabile con operatori per includere o estrarre elementi dall'insieme.



Una classe Persone a diversi livelli di specifica

## Esempio

---

- Rappresentiamo le classi che rappresentano la situazione degli esami degli studenti, dove per esame si intende l'evento dell'esame superato (quindi registrato).

STUDENTI
Nome
Cognome
Matricola

ESAMI
Materia
Data
Voto



## TIPO OGGETTO

---

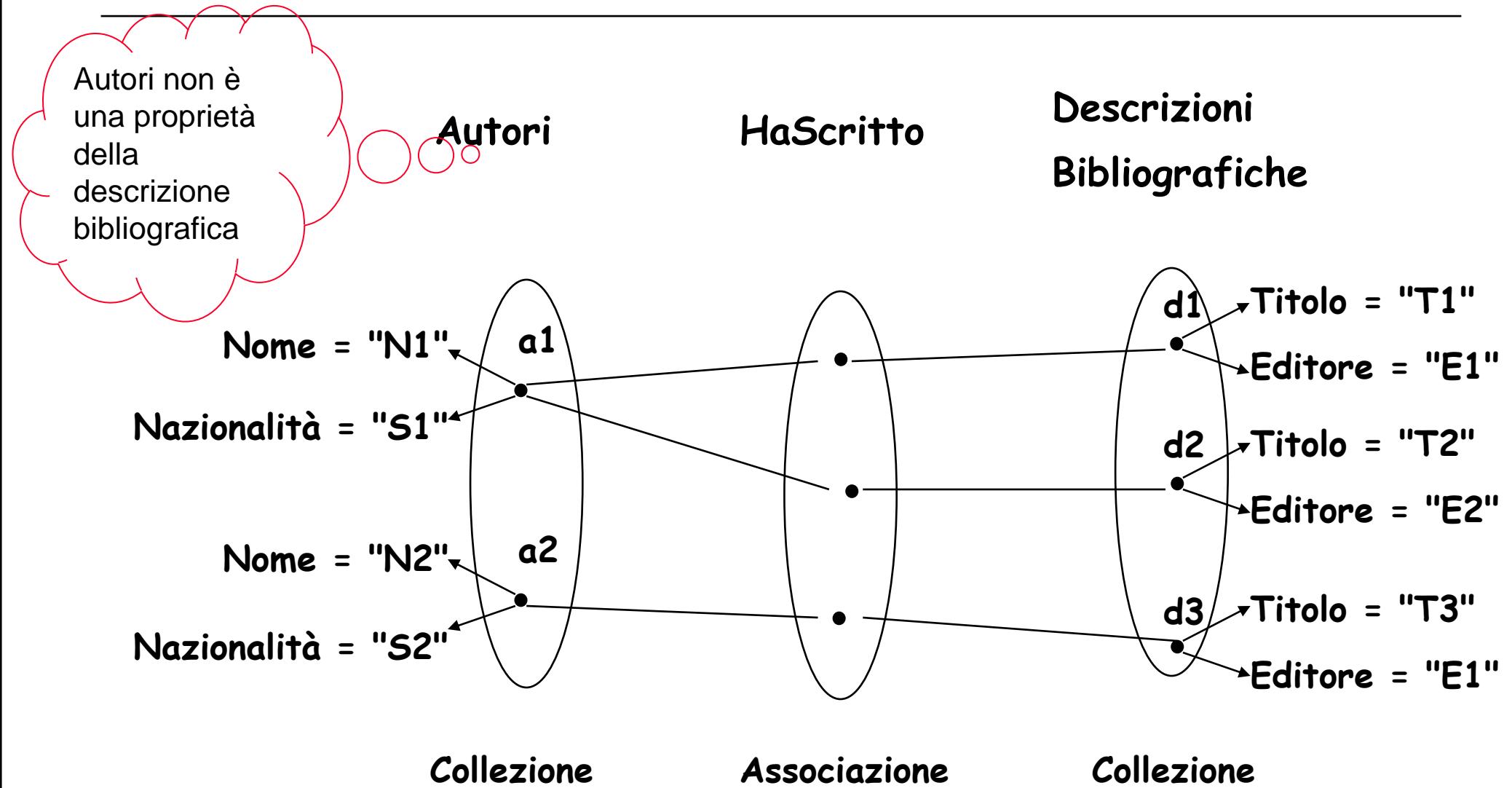
- Il primo passo nella costruzione di un modello consiste nella **classificazione delle entità del dominio con la definizione dei tipi degli oggetti che le rappresentano.**
- Un **tipo oggetto** definisce l'insieme dei messaggi (interfaccia) a cui può rispondere un insieme di possibili oggetti.
- I **nomi dei messaggi** sono detti anche attributi degli oggetti.
- IL TIPO OGGETTO NEI DIAGRAMMI ER
  - I tipi oggetti non si rappresentano nei diagrammi, dove invece l'attenzione è sulle collezioni e sulle associazioni
  - Tuttavia, la rappresentazione grafica di una collezione indica anche gli attributi del tipo oggetto associato

## CONOSCENZA CONCRETA: LE ASSOCIAZIONI

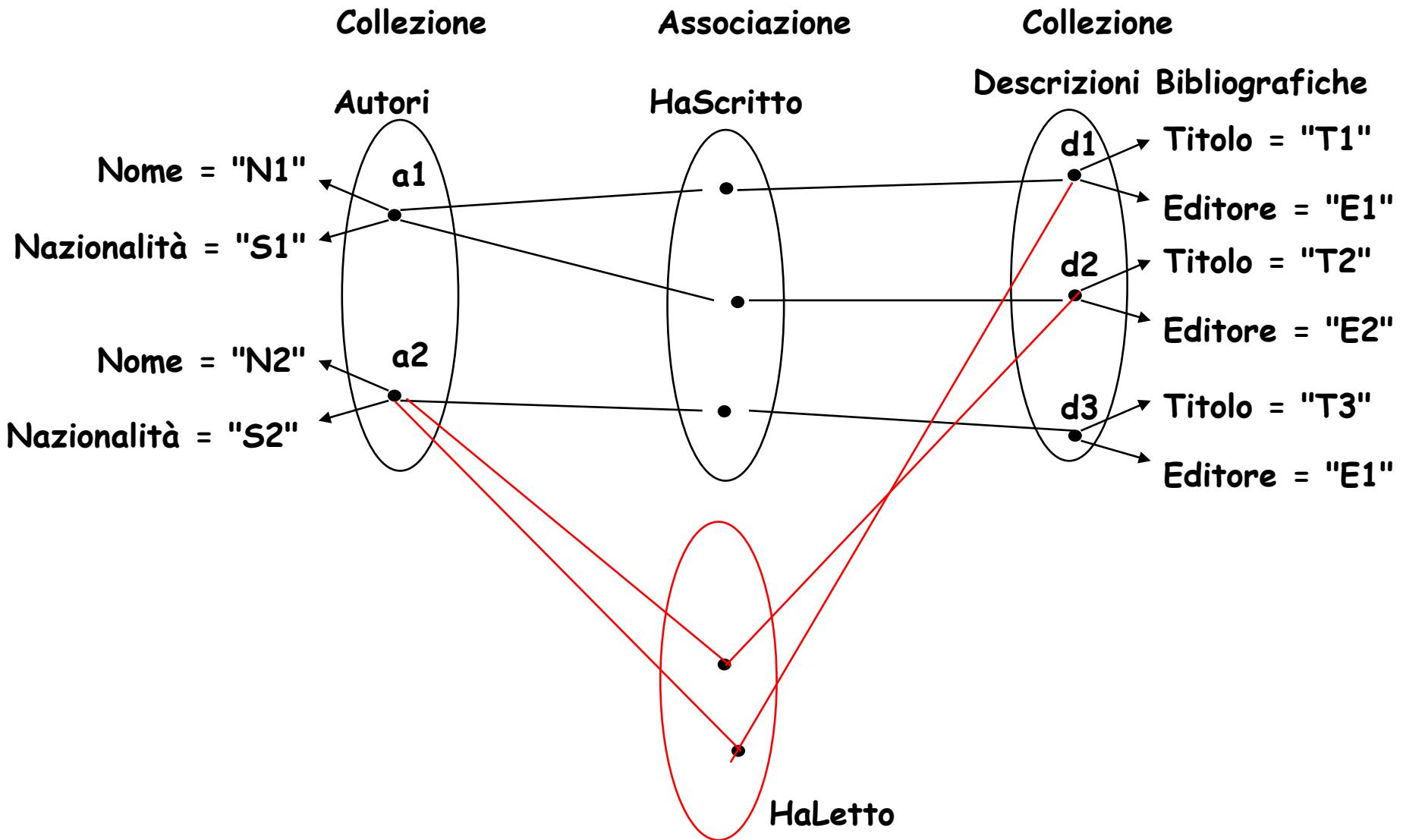
---

- Un'**istanza di associazione** è un fatto che correla due o più entità, stabilendo un legame logico tra di loro.
  - la descrizione bibliografica con titolo "Basi di Dati" riguarda il documento fisico con collocazione "d3-55-2"
  - l'utente Tizio ha in prestito una copia della Divina Commedia
- Un'**associazione**  $R(X, Y)$  fra due collezioni di entità  $X$  e  $Y$  è un insieme di istanze di associazione tra elementi di  $X$  e  $Y$ , che varia in generale nel tempo.
- Il **prodotto cartesiano**  $(X \times Y)$  è detto **dominio dell'associazione**.

# ASSOCIAZIONI: ESEMPIO



# ASSOCIAZIONI: ESEMPIO



## TIPI DI ASSOCIAZIONE

---

Un'associazione è caratterizzata dalle seguenti proprietà strutturali:  
**molteplicità e totalità.**

**Definizione:** (Vincolo di univocità) Un'associazione  $R(X, Y)$  è univoca rispetto a  $X$ :

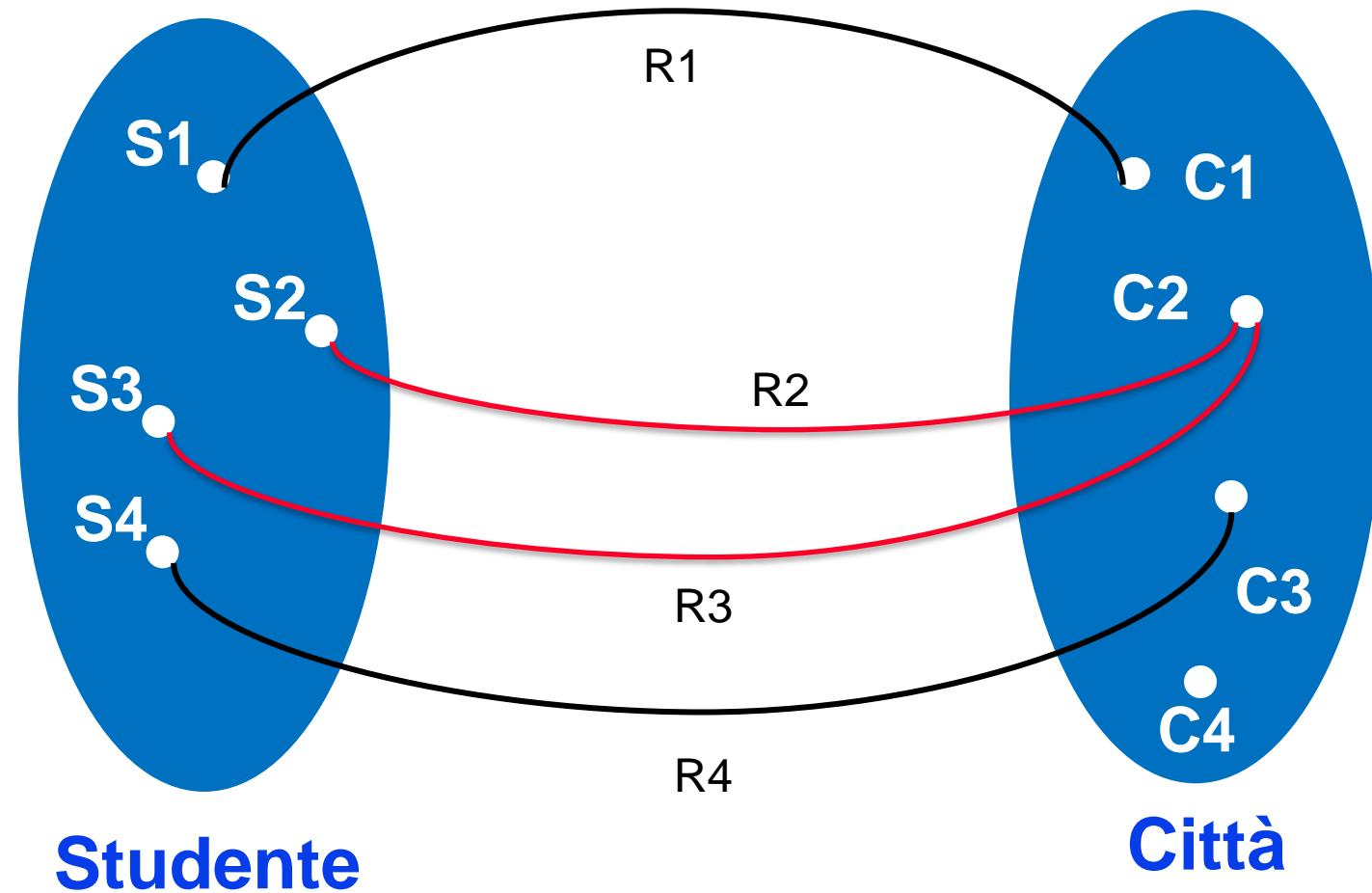
- se per ogni elemento  $x$  di  $X$  esiste al più un elemento di  $Y$  che è associato a  $x$ ;
- se non vale questo vincolo, l'associazione è multivalue rispetto ad  $X$ .

Cardinalità dell'associazione:

- $R(X, Y)$  è (1:N) se essa è multivalue su  $X$  ed univoca su  $Y$
- $R(X, Y)$  è (N:1) se essa è univoca su  $X$  e multivalue su  $Y$
- $R(X, Y)$  è (N:M) se essa è multivalue su  $X$  e multivalue su  $Y$
- $R(X, Y)$  è (1:1): se essa è univoca su  $X$  e univoca su  $Y$

## Occorrenze di Residenza

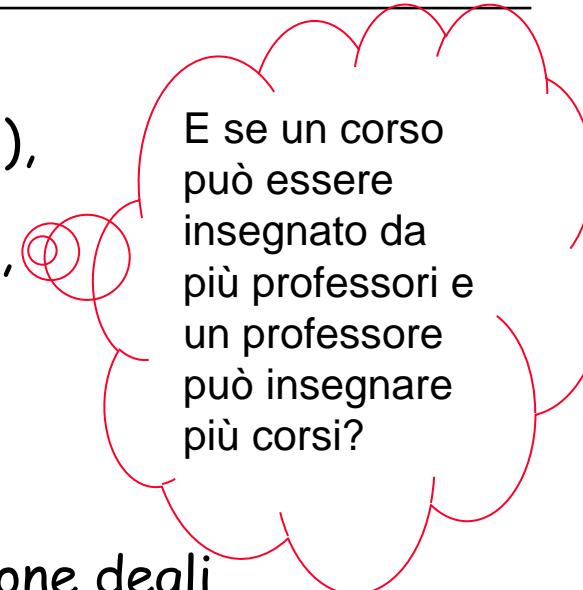
---



## TIPI DI ASSOCIAZIONE: ESEMPI

---

- Frequentta(Studenti, Corsi) ha cardinalità (M:N),
  - Insegna(Professori, Corsi) ha cardinalità (1:N),
  - Dirige(Professori, Dipartimenti) ha cardinalità (1:1).
- 
- Rappresentiamo le classi che rappresentano la situazione degli esami degli studenti, dove per esame si intende l'evento dell'esame superato (quindi registrato).
    - SuperatoDa(Esami, Studenti) ha cardinalità (N:1),



E se un corso può essere insegnato da più professori e un professore può insegnare più corsi?

## TIPI DI ASSOCIAZIONE: VINCOLI

---

Definizione (Vincolo di totalità):

Un'associazione  $R(X, Y)$  è

- **totale** (o surgettiva) su  $X$  se per ogni elemento  $x$  di  $X$  esiste almeno un elemento di  $Y$  che è associato ad  $x$ ;
- se non vale questo vincolo, l'associazione è **parziale** rispetto ad  $X$ .
- Ad esempio:
  - Insegna(Professori, Corsi) è totale su Corsi in quanto non può esistere un corso del piano di studi senza il corrispondente docente che lo tiene.

## Rappresentazione delle associazioni - Associazione binaria senza proprietà

---

- Un'associazione fra due collezioni  $C_1$  e  $C_2$  si rappresenta con una linea che collega le classi che rappresentano le due collezioni.
- La linea è etichettata con il nome dell'associazione che di solito viene scelto utilizzando un predicato ("soggetto predicato complemento")
- L'univocità di un'associazione, rispetto ad una classe  $C_1$ , si rappresenta disegnando una freccia singola sulla linea che esce dalla classe  $C_1$  ed entra nella classe  $C_2$ ; l'assenza di tale vincolo è indicata da una freccia doppia.
- Similmente, la parzialità è rappresentata con un taglio sulla linea vicino alla freccia, mentre il vincolo di totalità è rappresentato dall'assenza di tale taglio.

## TIPI DI ASSOCIAZIONE: ESEMPI

---

Tipi di associazioni fra Persone e Città:

NataA(Persone, Città)

ha cardinalità (N:1), totale su Persone e parziale su Città

HaVisitato(Persone, Città)

ha cardinalità (N:M), parziale su Persone e Città

E'SindacoDi(Persone, Città)

ha cardinalità (1:1), totale su Persone e Città

E'VissutaA(Persone, Città)

ha cardinalità (N:M), parziale su Persone e Città

## Esercizio: individuare entità, proprietà e associazioni

---

Una catena di negozi vende elettrodomestici e mantiene informazioni sui relativi centri di assistenza. Per ogni modello di elettrodomestico interessa il tipo, il peso, il prezzo di listino, e la marca.

Per ogni marca di elettrodomestico interessa il nome e interessa conoscere i relativi centri di assistenza.

Per ogni marca ci possono essere numerosi modelli di elettrodomestici. Di un centro d'assistenza interessano il nome, l'indirizzo, un insieme di recapiti telefonici, e l'insieme di marche per le quali offre assistenza.

La catena tiene inoltre traccia di eventuali reclami ricevuti.

Per ogni reclamo interessano la data e il nome indicato dal segnalante. Se il reclamo riguarda un negozio, interessa inoltre conoscere il negozio in questione.

Per ogni negozio, la catena tiene traccia di un nome, un indirizzo, e l'insieme dei reclami ricevuti. Se invece riguarda un'operazione di assistenza, interessa tenere traccia del modello di elettrodomestico in questione e del centro di assistenza che ha gestito la riparazione.

Il centro di assistenza deve essere uno di quelli che offrono assistenza per la marca del modello di elettrodomestico.

---

# QUESTIONI TERMINOLOGICHE

---

*Dominio del discorso*

entità

tipo entità

collezione

associazione

*Modello Informatico*

oggetto (entity instance)

tipo oggetto (entity type)

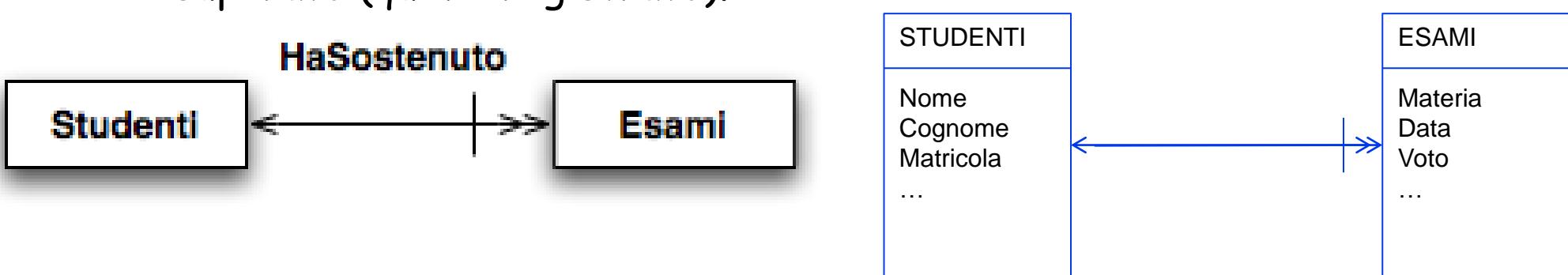
classe (entity)

associazione (relationship)

## Esempio: associazione fra studenti ed esami (eventi di esaminazione)

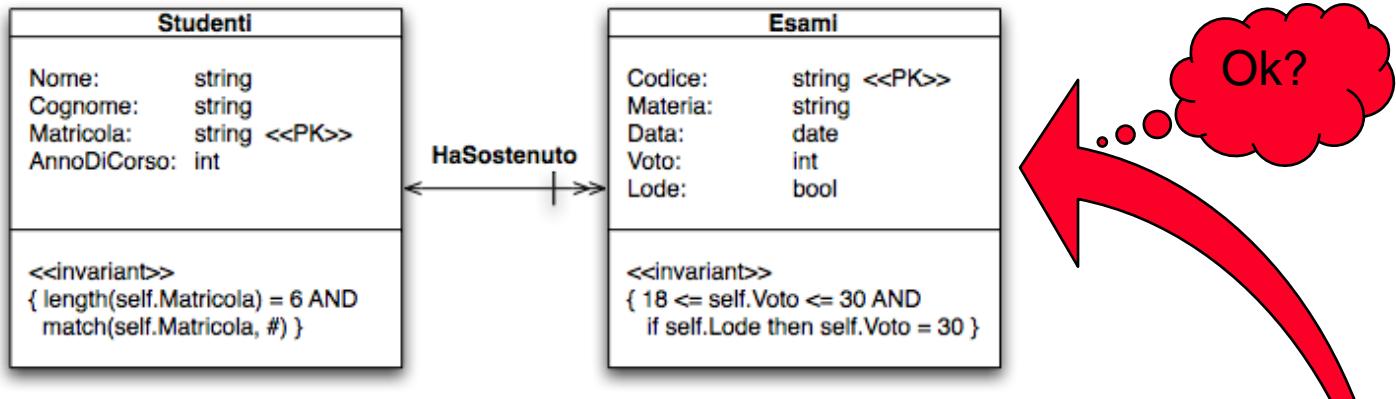
---

- Rappresentiamo le classi che rappresentano la situazione degli esami degli studenti, dove per esame si intende l'evento dell'esame superato (quindi registrato).



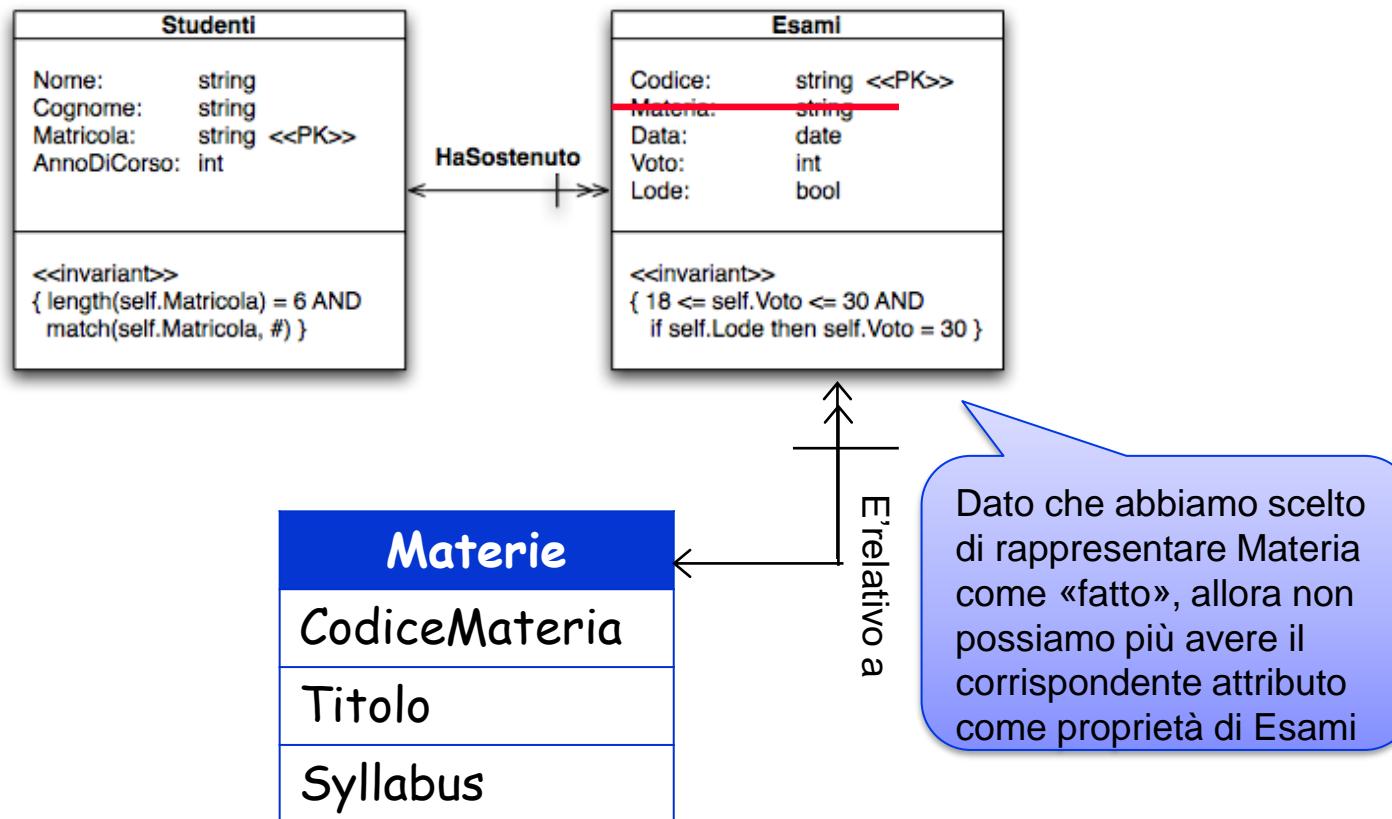
- Ogni esame riguarda uno ed un solo studente:
  - univocità (freccia singola) e totalità (assenza del taglio).
  - Vincolo di parzialità (taglio) e assenza di univocità sugli esami (freccia doppia) superati da uno studente.

## Esempio - Parte I

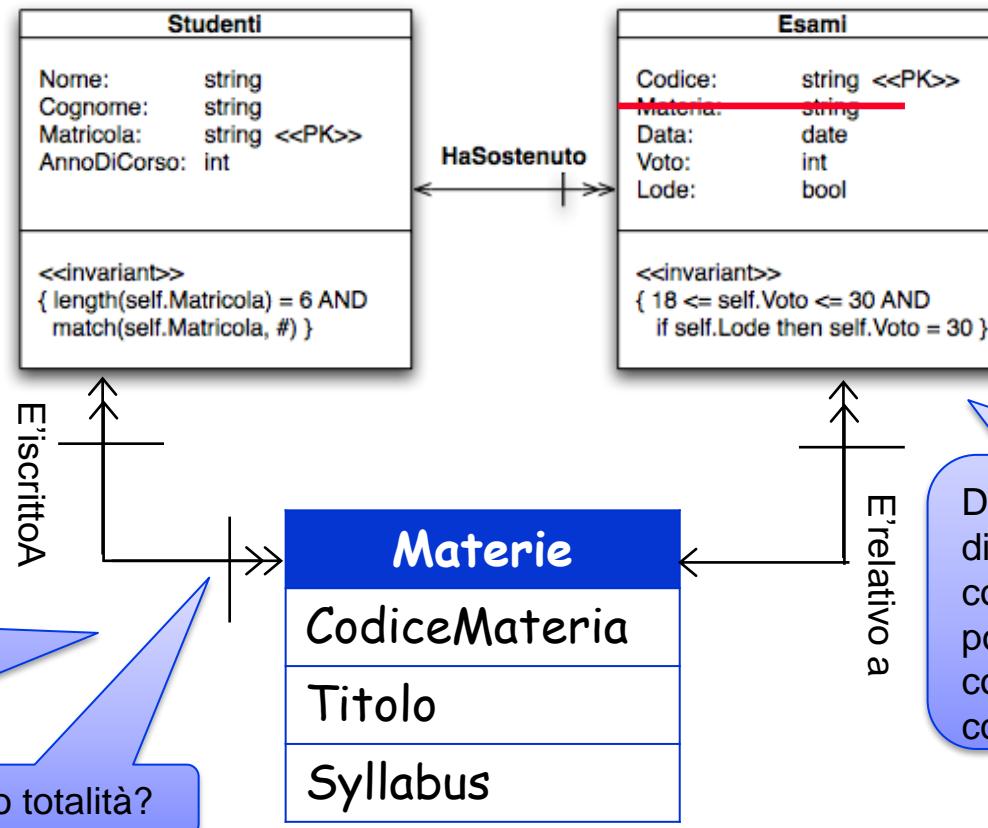


- Esami (eventi degli esami):
  - uno studente può sostenere zero o più esami
  - Un esame (evento, «istanza di verbalizzazione») riguarda esclusivamente uno studente
- **Errore comune**: mettere i dettagli (matricola, nome, cognome, ....) dello studente come proprietà di Esami (materie, data, voto, ...).
- Nota che materia è un attributo di Esami (in questo esempio)

## Esempio - Parte II



## Esempio - Parte II

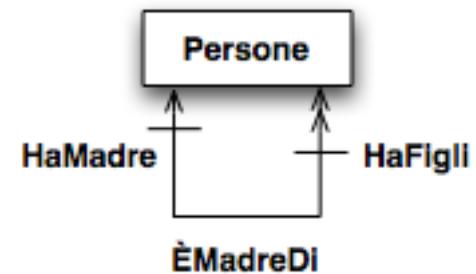
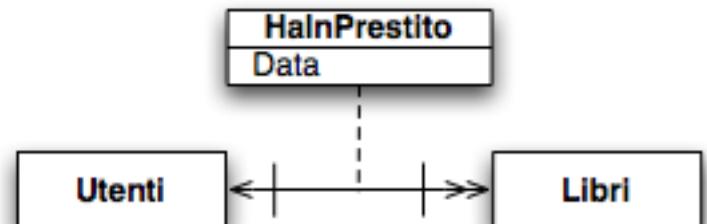
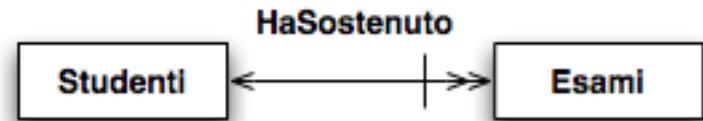


- Per dare l'informazione sulla cardinalità occorre dare 4 valori:
  - Minimo e massimo per ciascuna delle due direzioni

# MODELLO A OGGETTI: LE ASSOCIAZIONI

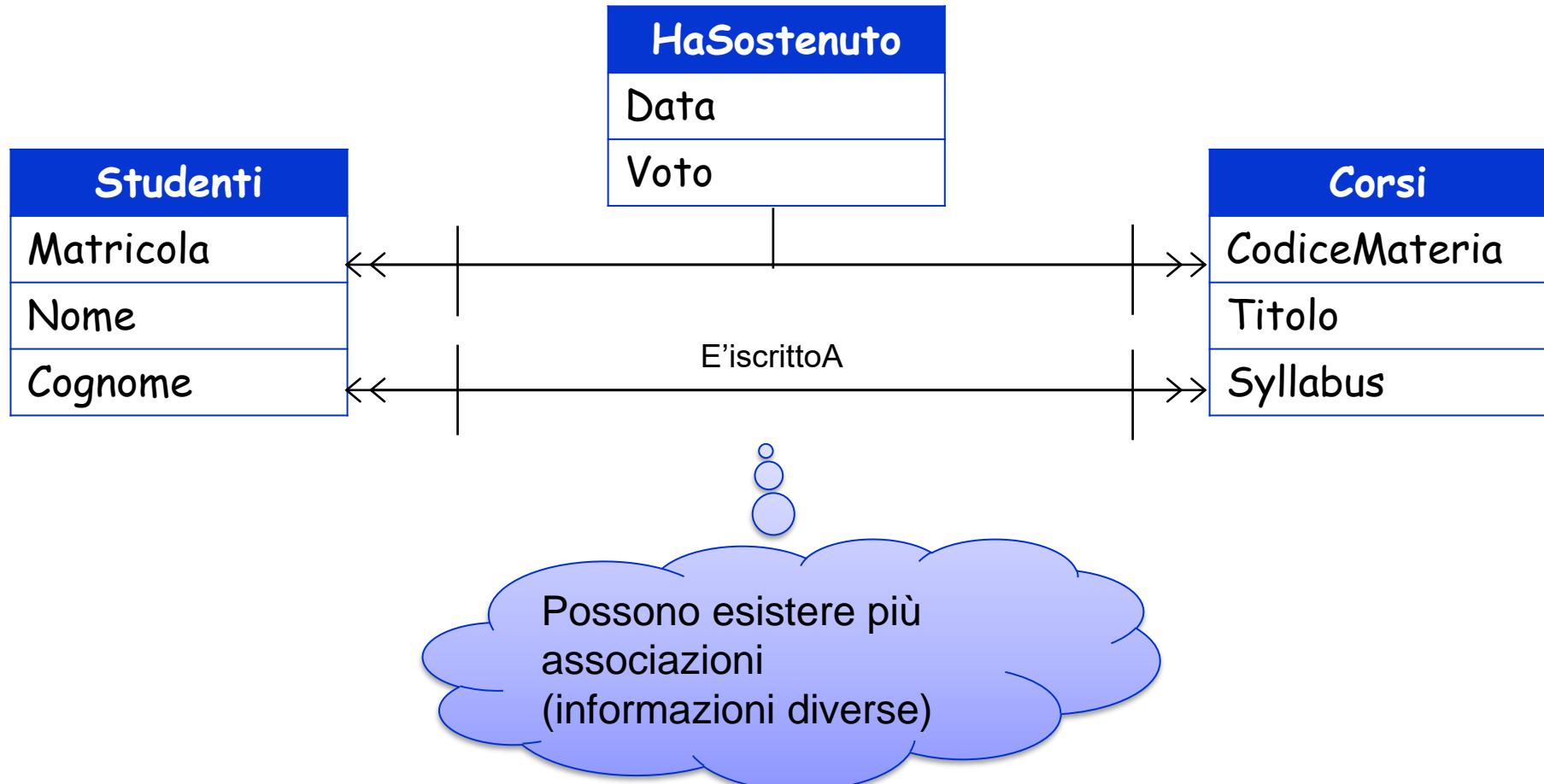
---

- Le associazioni si modellano con un costrutto apposito
- Le associazioni possono avere delle proprietà
- Le associazioni possono essere ricorsive



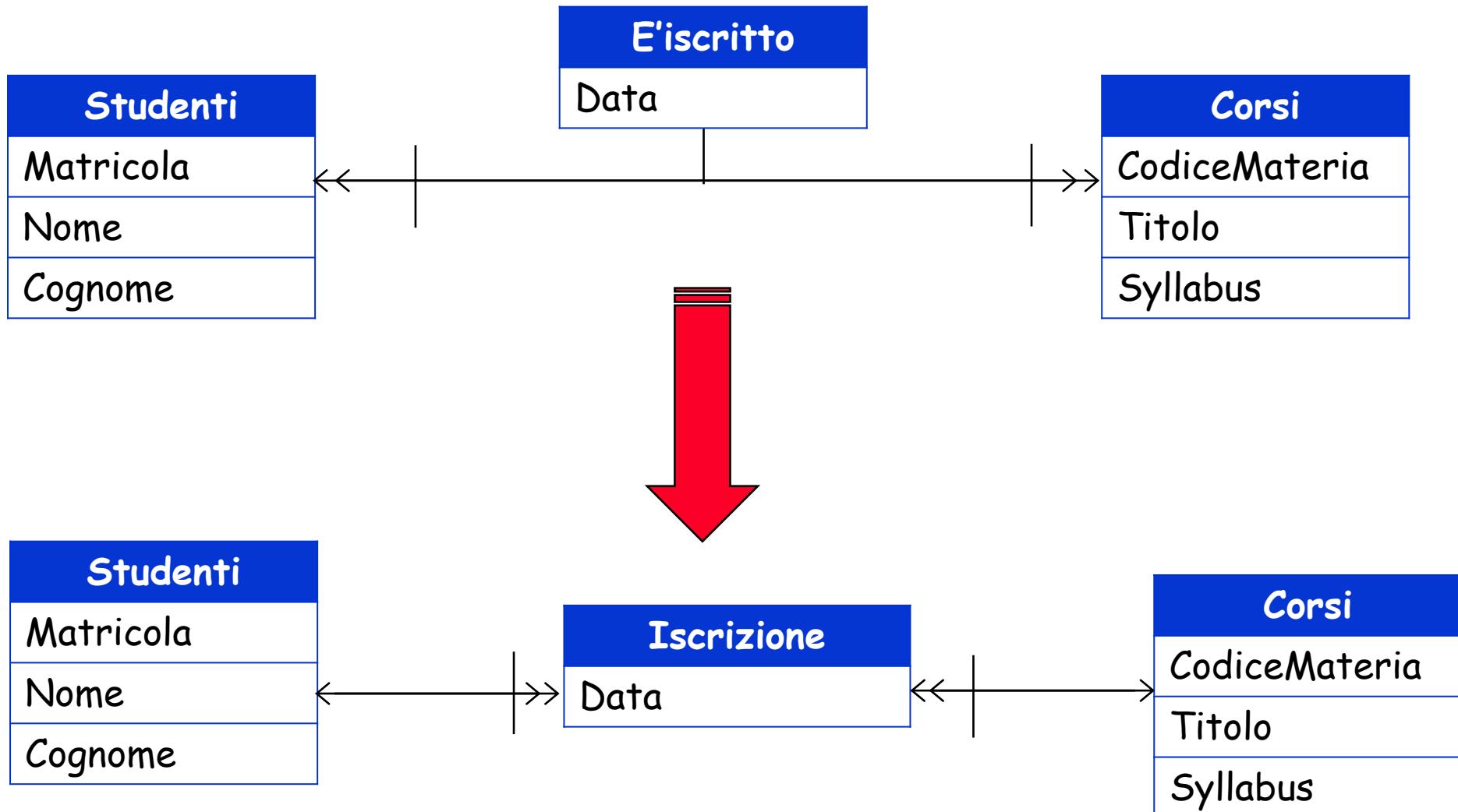
## Esempio - Parte III

---



## Esempio - reificazione delle associazioni molti a molti

---

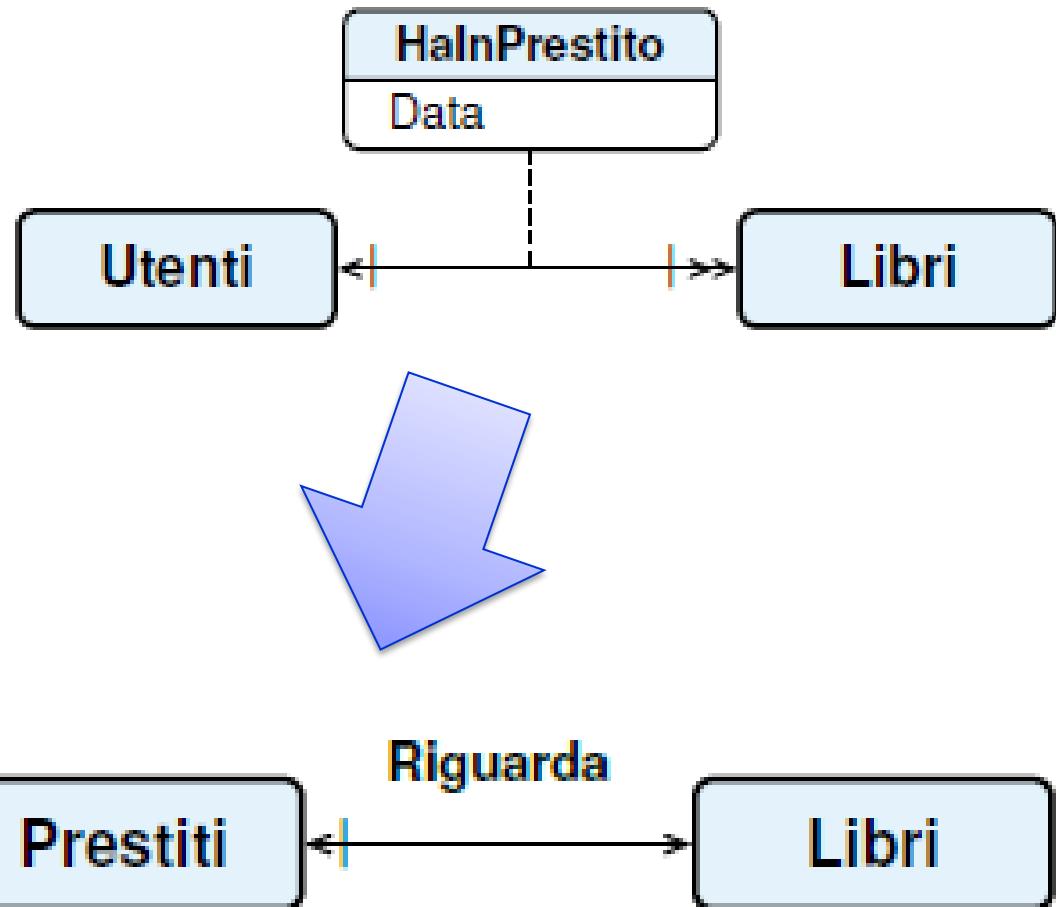


## Associazioni con proprietà

---

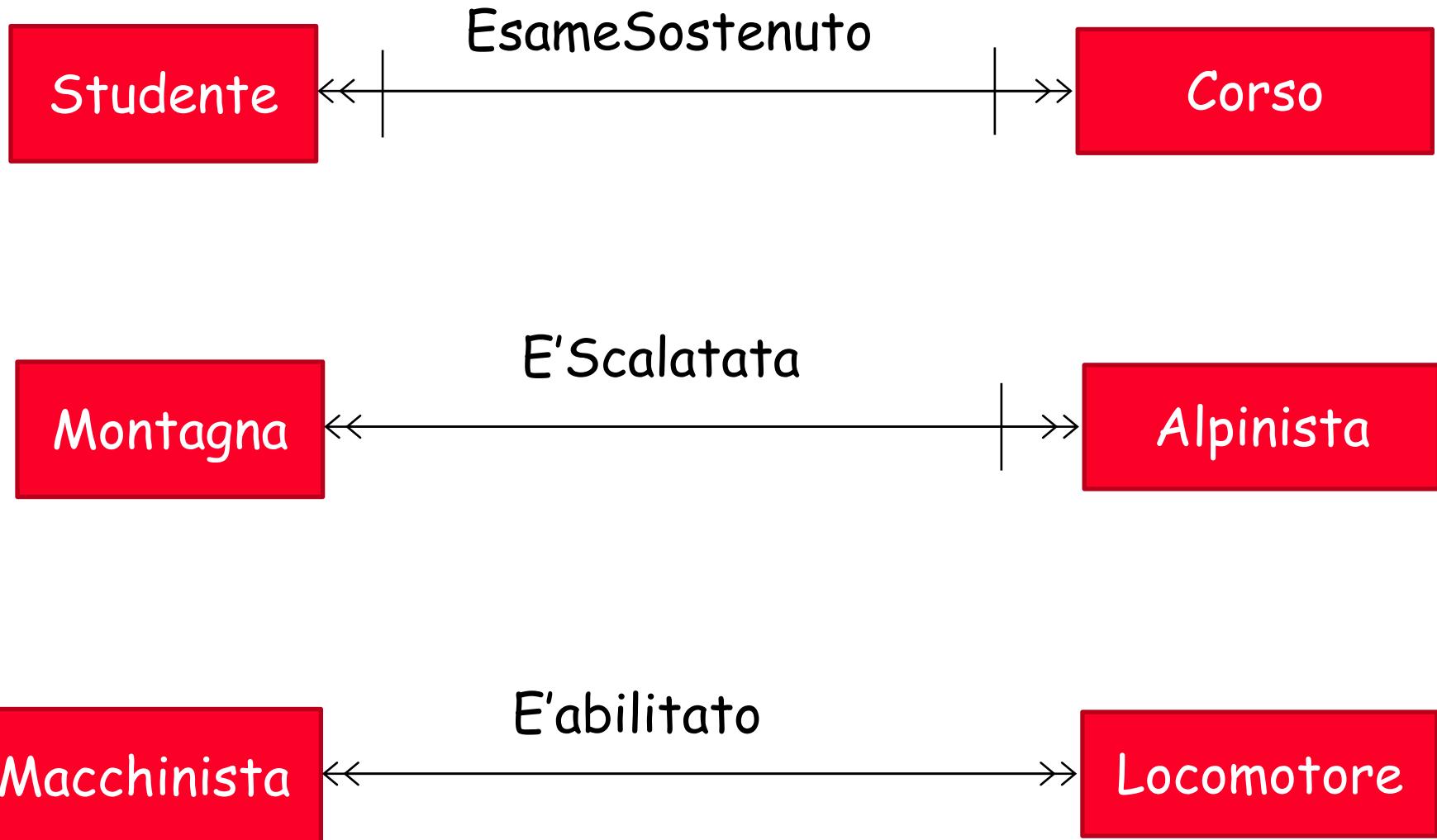
Associazione 1 e/a molti

Classe PRESTITI associata in modo (1 : 1) ai LIBRI e in modo (N : 1) agli UTENTI,  
E' possibile aggiungere un attributo DATA alla classe PRESTITI stessa



## Esempi: molti a molti

---



## Esempi: uno a uno

---



◦◦◦ Ci potrebbero essere cattedre scoperte



## Esercizio

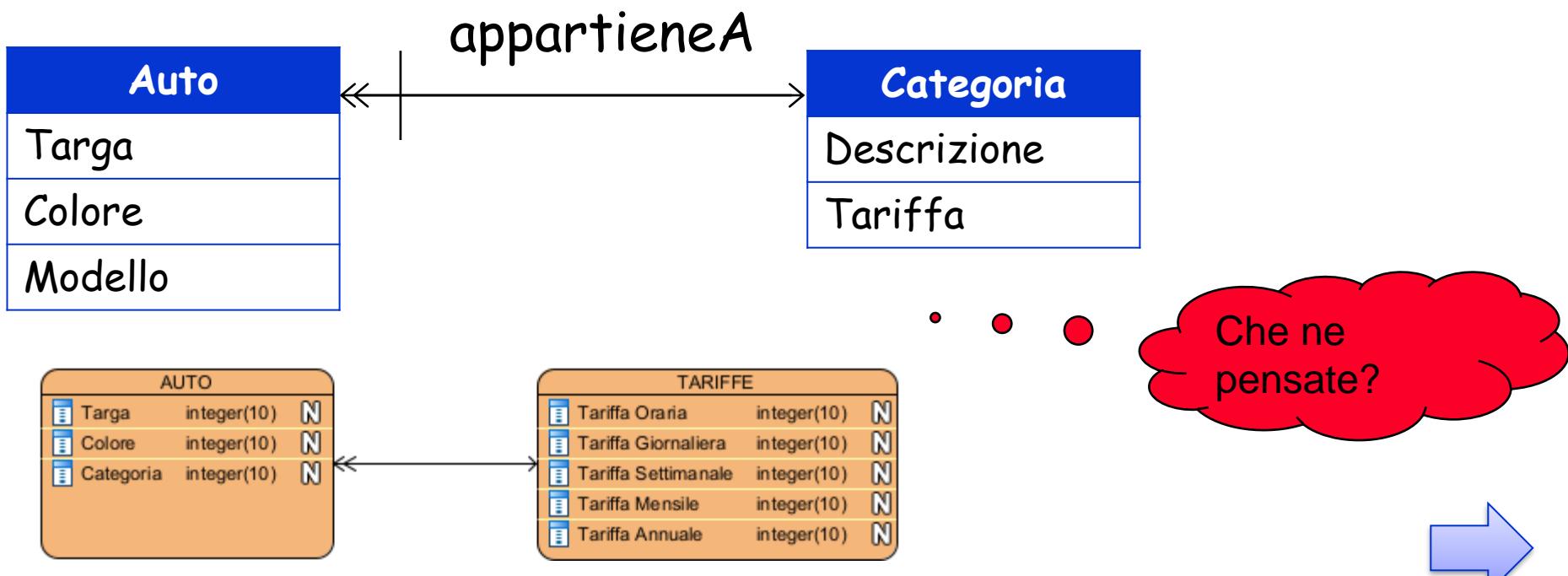
---

- Gestire un parco macchine (**immatricolate**) dove per una macchina ha una targa, colore, cilindrata e può avere diversi proprietari.
- Di ogni proprietario vogliamo sapere il nome, cognome, CF, la data di nascita



## Esercizio

Una agenzia di noleggio di autovetture ha un parco macchine ognuna delle quali ha una targa, un colore e fa parte di una categoria; per ogni categoria c'è una tariffa di noleggio

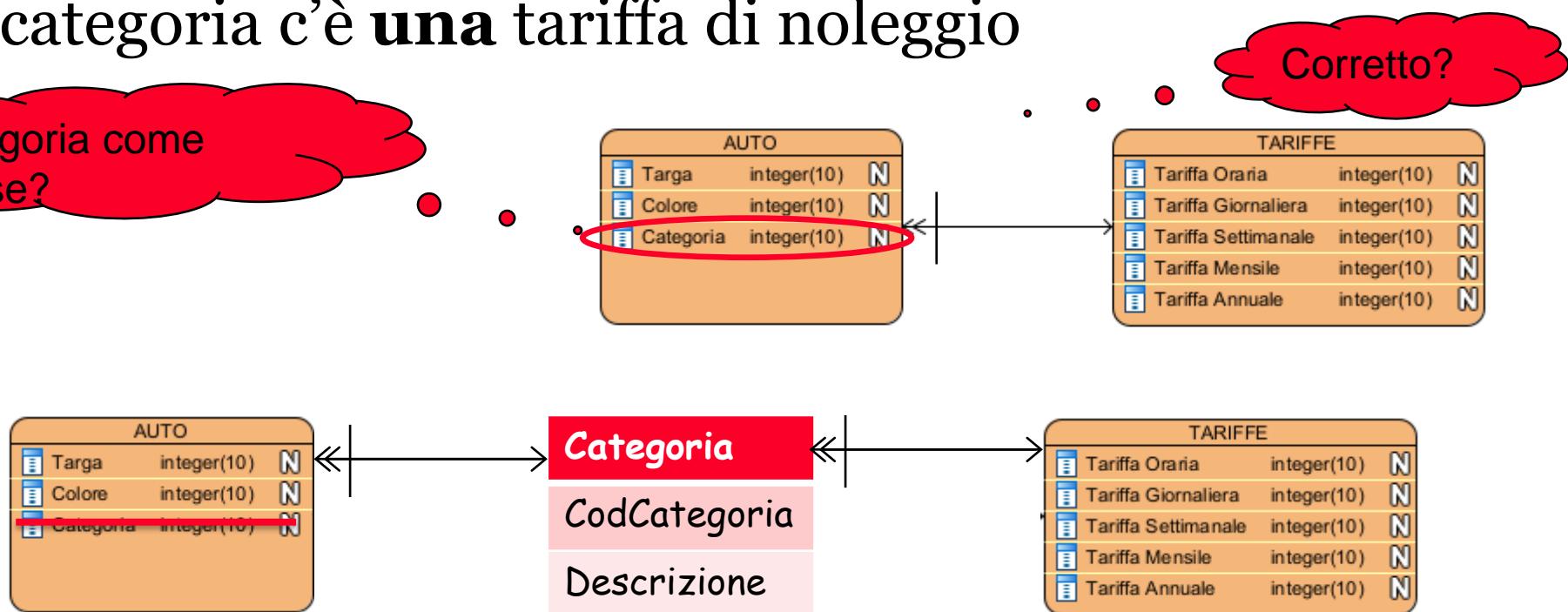


## Esercizio

Una agenzia di noleggio di autovetture ha un parco macchine ognuna delle quali ha una targa, un colore e fa parte di una categoria; per ogni categoria c'è **una** tariffa di noleggio

Categoria come classe?

Corretto?



## Esercizio

---

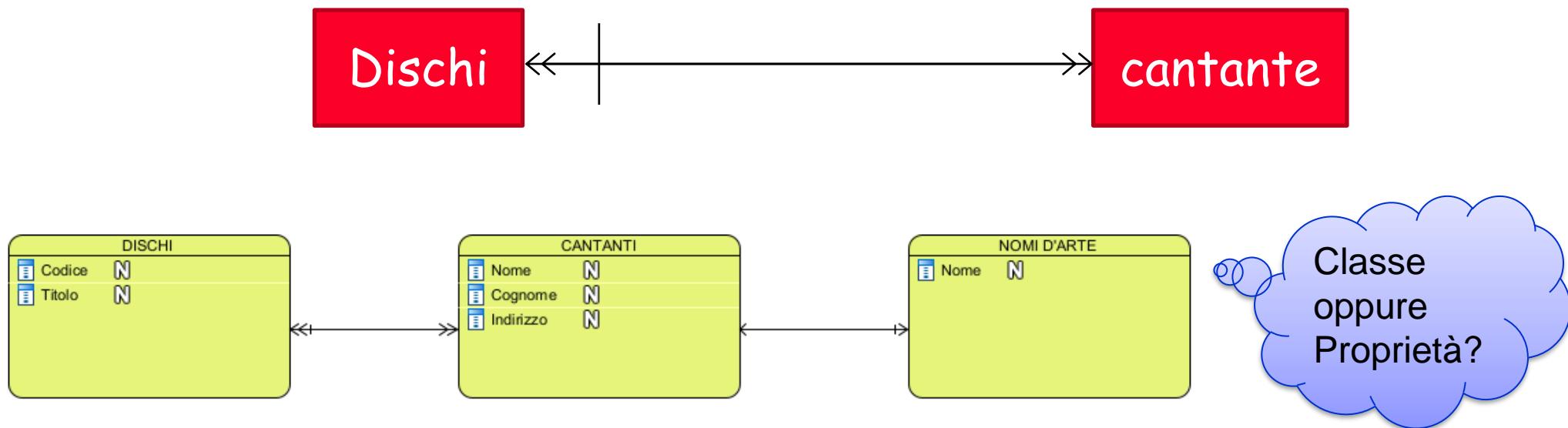
In un giardino zoologico ci sono degli animali appartenenti a una specie e aventi una certa età; ogni specie è localizzata in un settore (avente un nome) dello zoo.



## Esercizio

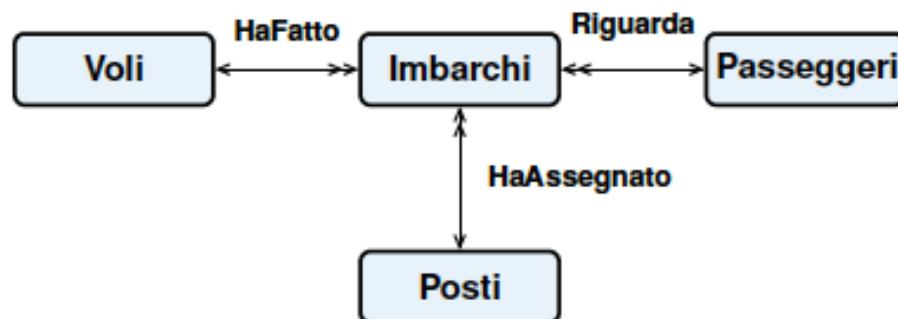
---

Una casa discografica produce dischi aventi un codice e un titolo; ogni disco è inciso da uno o più cantanti, ognuno dei quali ha un nome, un indirizzo e, qualcuno, un nome d'arte



## Associazione non binarie

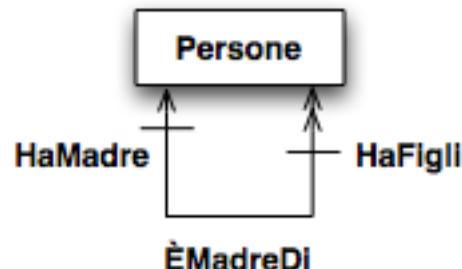
- Per semplicità non daremo una notazione grafica per rappresentare associazioni **non binarie**.
- Trasformazione di associazione ternaria in binarie (associazione fra Voli, Passeggeri e Posti)
- L'associazione è multivaleore rispetto ad ogni collezione, poiché:
  - **ogni singolo volo, passeggero, e posto**, partecipa in generale a **diverse istanze di associazione**, ma con i vincoli che, in un singolo volo,
  - **ogni posto** è associato **al più ad un passeggero** ed
  - **ogni passeggero** può occupare **al più un posto**



## Associazione ricorsive

---

- Le **associazioni ricorsive** sono relazioni binarie fra gli elementi di una stessa collezione.

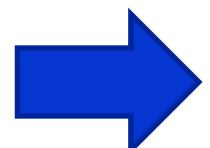


- Una persona può essere madre di più persone della collezione e può avere una madre nella collezione.
- Occorre etichettare la linea non solo con il nome dell'associazione, ma anche con dei nomi per specificare il ruolo che hanno le due componenti in un'istanza di associazione

## DESCRIZIONE DI UN CASO: biblioteca

---

- Si vogliono modellare alcuni fatti riguardanti una biblioteca universitaria:
  - le descrizioni bibliografiche dei libri, opere con un solo volume,
  - i termini del thesaurus (parole chiave),
  - le copie dei libri disponibili che corrispondono ad una descrizione bibliografica,
  - gli autori dei libri,
  - gli utenti della biblioteca
  - i prestiti in corso.



## DESCRIZIONE DI UN CASO : biblioteca (cont.)

---

- Il thesaurus (parole chiavi) è un insieme di termini, e di associazioni fra di loro, che costituiscono il lessico specialistico da usare per descrivere il contenuto dei libri. Di ogni termine interessa anche una descrizione.
- Fra i termini del thesaurus interessano le seguenti relazioni:
  - Preferenza. Per esempio:
    - Elaboratore Standard (vedi) Calcolatore;
    - Calcolatore Sinonimi (UsatoPer) Elaboratore, Calcolatrice, Stazione di lavoro.
  - Gerarchia. Per esempio:
    - Felino → Più Specifico: Gatto Leone Tigre;
    - Gatto → Più Generale: Felino;

## CONOSCENZA CONCRETA: SCELTA DELLE ENTITÀ E DELLE PROPRIETÀ

---

- Certi fatti possono essere interpretati come proprietà in certi contesti e come entità in altri.
- Ad esempio:
  - Descrizione bibliografica con attributi Autori, Titolo, Editore, LuogoEdizione, Anno
- oppure
  - Descrizione bibliografica con attributi ...
  - Autore con attributi Nome, Nazionalità, AnnoNascita ...
  - Editore con attributi Nome, Indirizzo, e-mail, ...

# DESCRIZIONE DI UN CASO: biblioteca

---

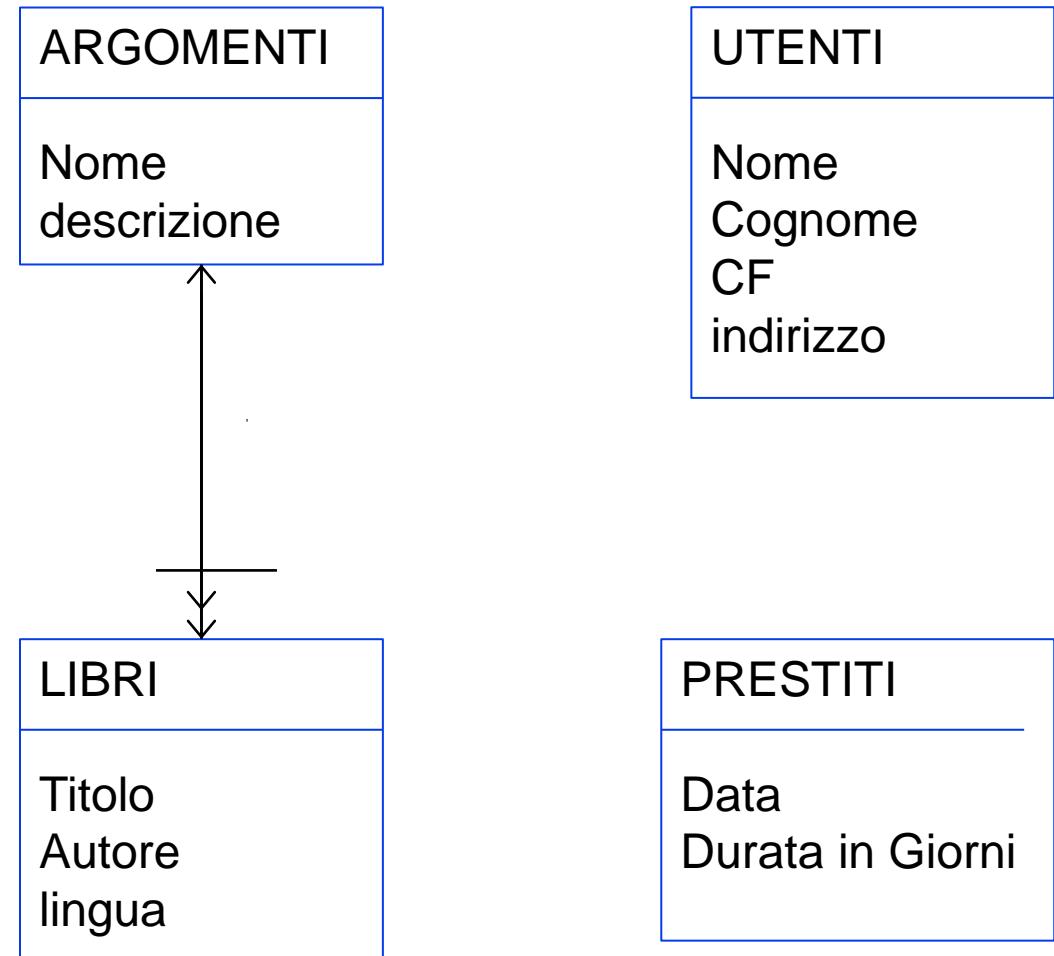
- Libri:
  - titolo
  - autori
  - lingua
- gli utenti
  - Nome, Cognome
  - Codice fiscale
  - Libri in prestito
- i prestiti in corso:
  - Data
  - Durata in giorni
  - Libro
  - utente



# Prima Modellazione: biblioteca

---

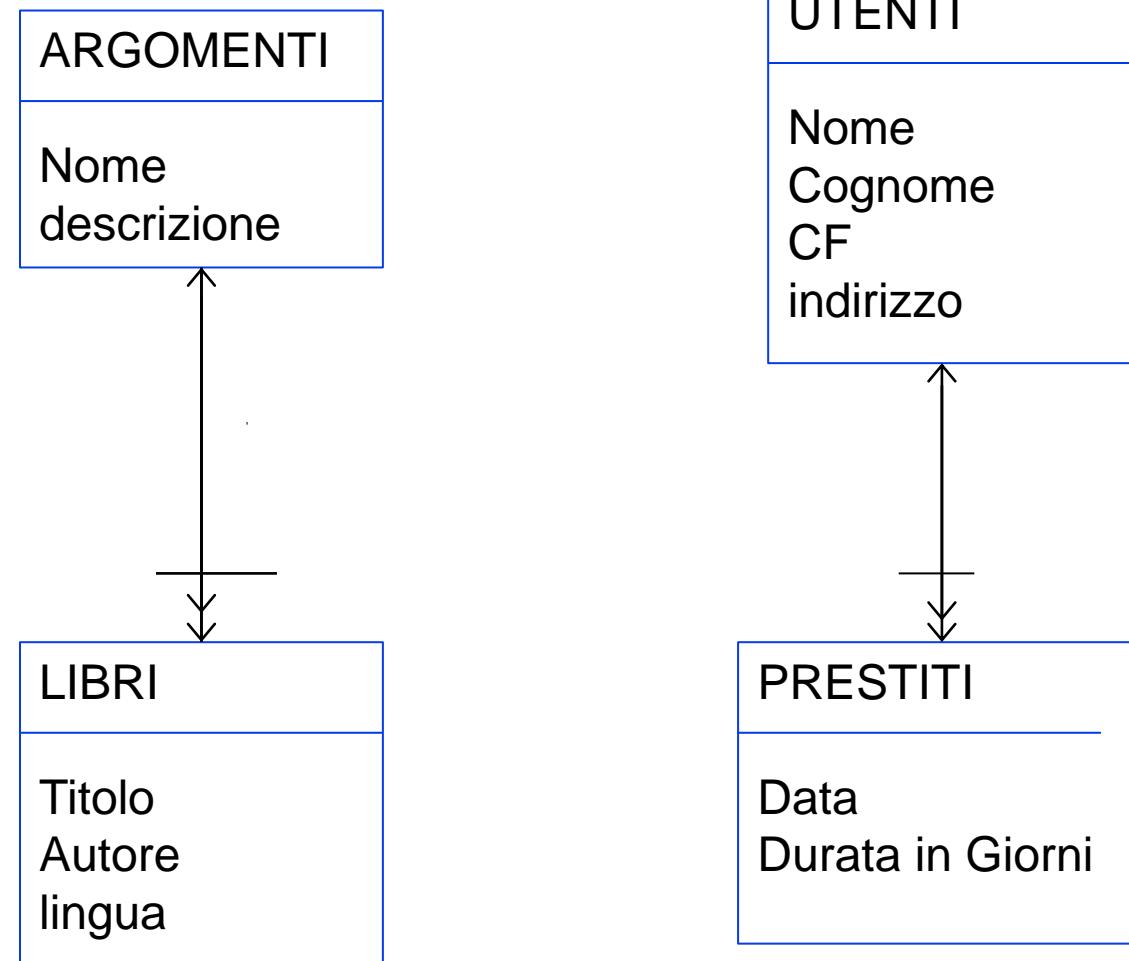
- Libri:
  - titolo
  - autori
  - lingua
- gli utenti
  - Nome, Cognome
  - Codice fiscale
  - Libri in prestito
- Argomenti:
  - nome
  - descrizione
- i prestiti in corso:
  - Data
  - Durata in giorni
  - Libro
  - utente



# Prima Modellazione: biblioteca

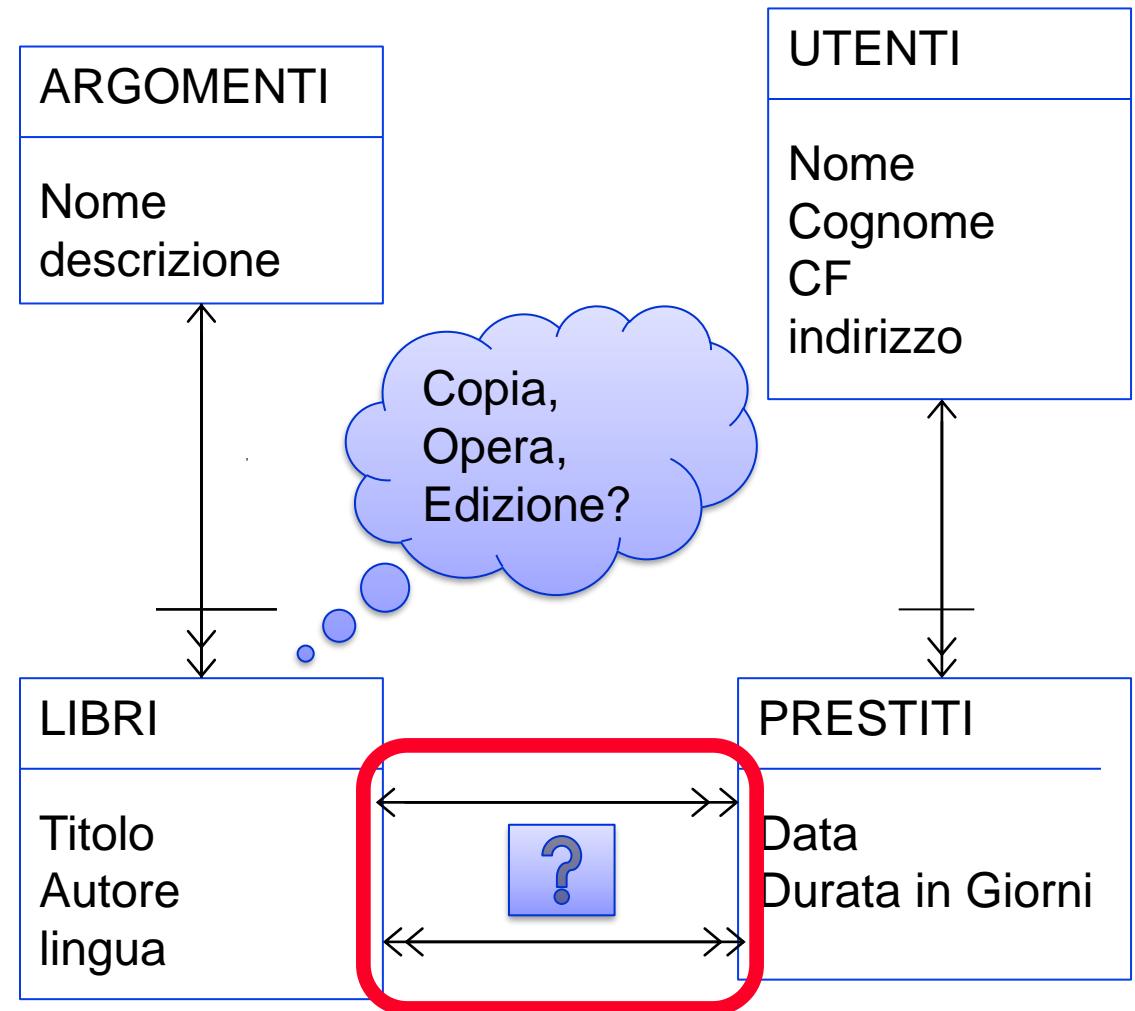
---

- Libri:
  - titolo
  - autori
  - lingua
- gli utenti
  - Nome, Cognome
  - Codice fiscale
  - Libri in prestito
- Argomenti:
  - nome
  - descrizione
- i prestiti in corso:
  - Data
  - Durata in giorni
  - Libro
  - utente



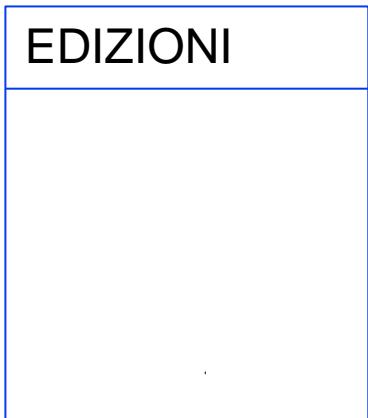
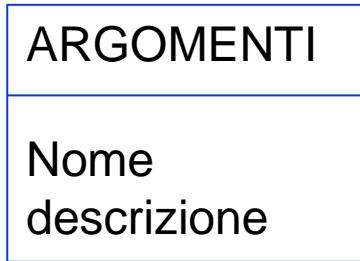
# Prima Modellazione: biblioteca

- Libri:
  - titolo
  - autori
  - lingua
- gli utenti
  - Nome, Cognome
  - Codice fiscale
  - Libri in prestito
- Argomenti:
  - nome
- i prestiti in corso:
  - Data
  - Durata in giorni
  - Libro
  - utente

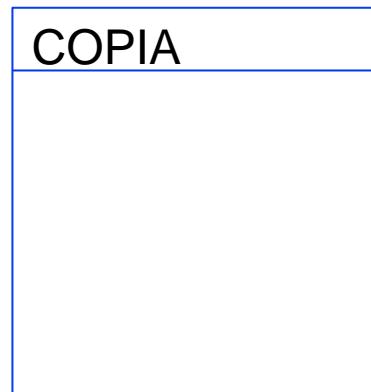
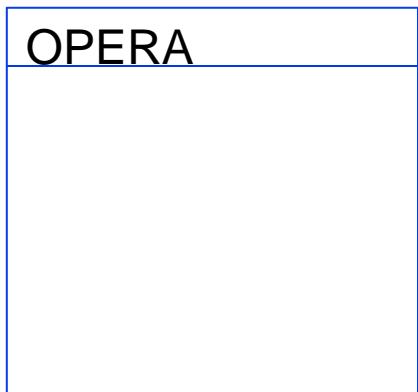


# Prima Modellazione: biblioteca

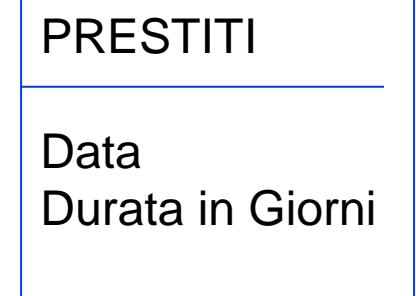
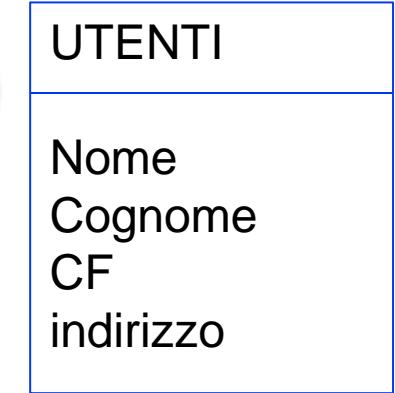
---



Quali sono gli attributi che vanno in opere e quali in copie e quali in edizioni?

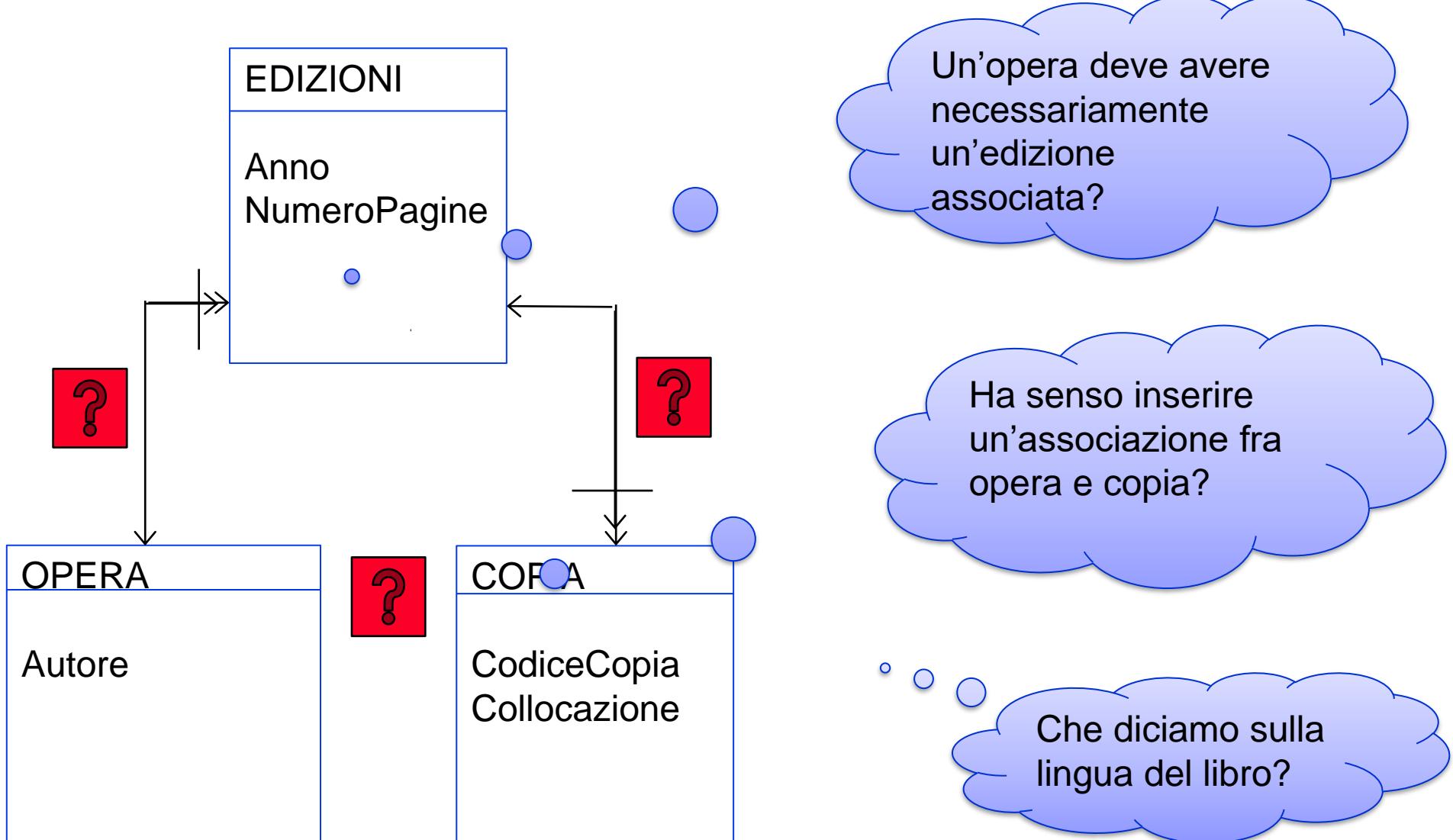


Quali associazioni?



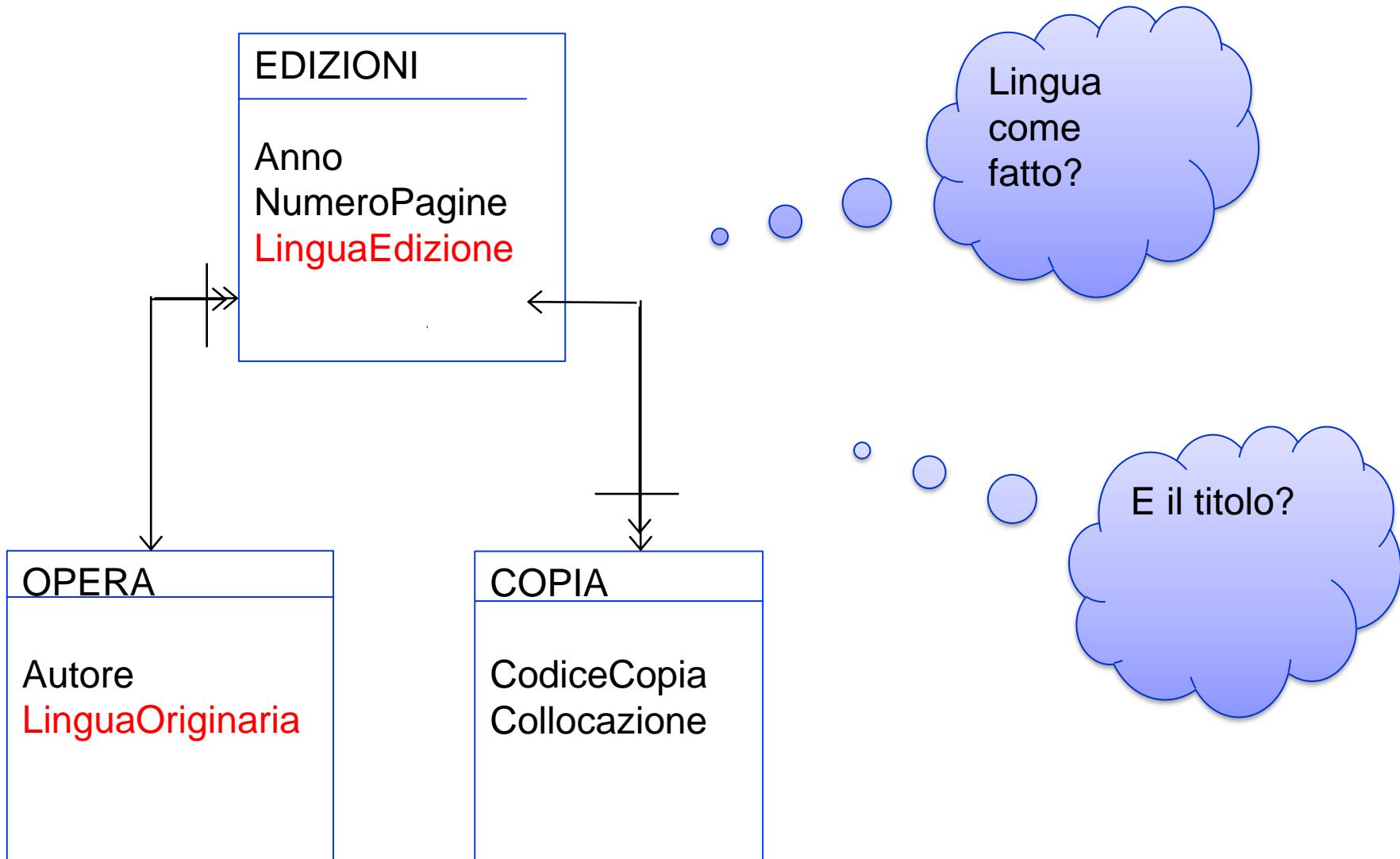
## Prima Modellazione: biblioteca - libri

---



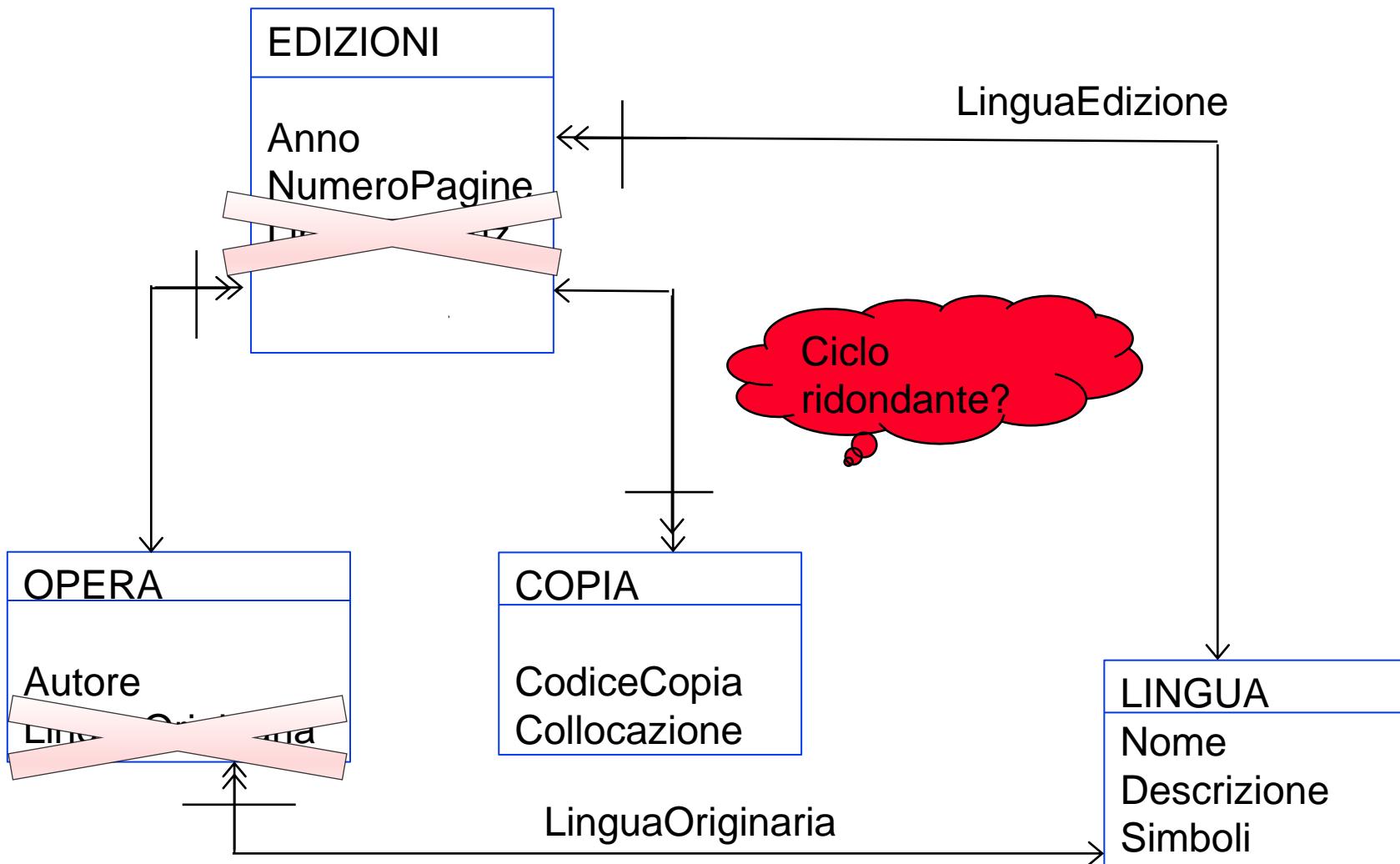
# Prima Modellazione: biblioteca - lingua

---



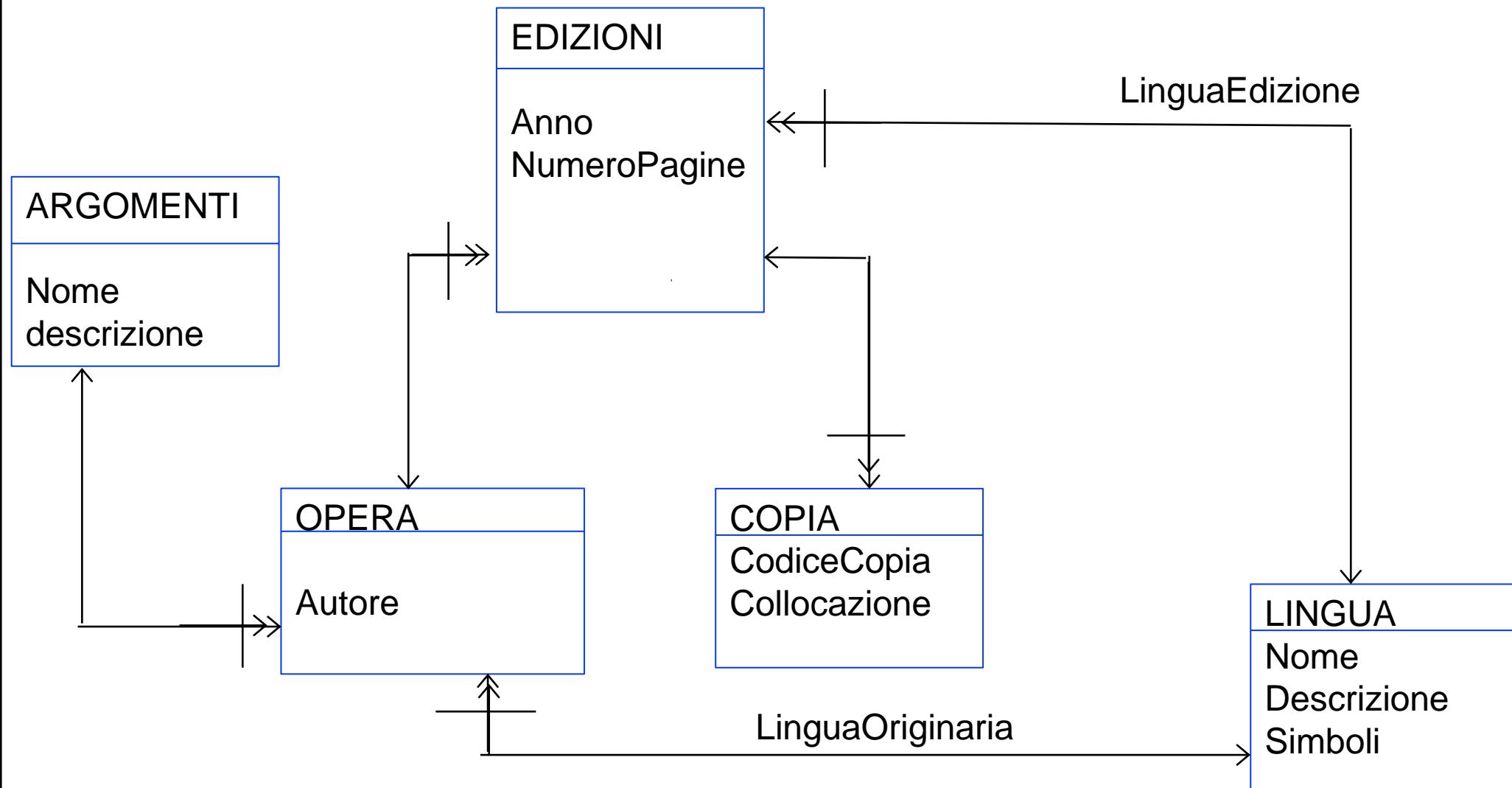
# Prima Modellazione: biblioteca - lingua

---



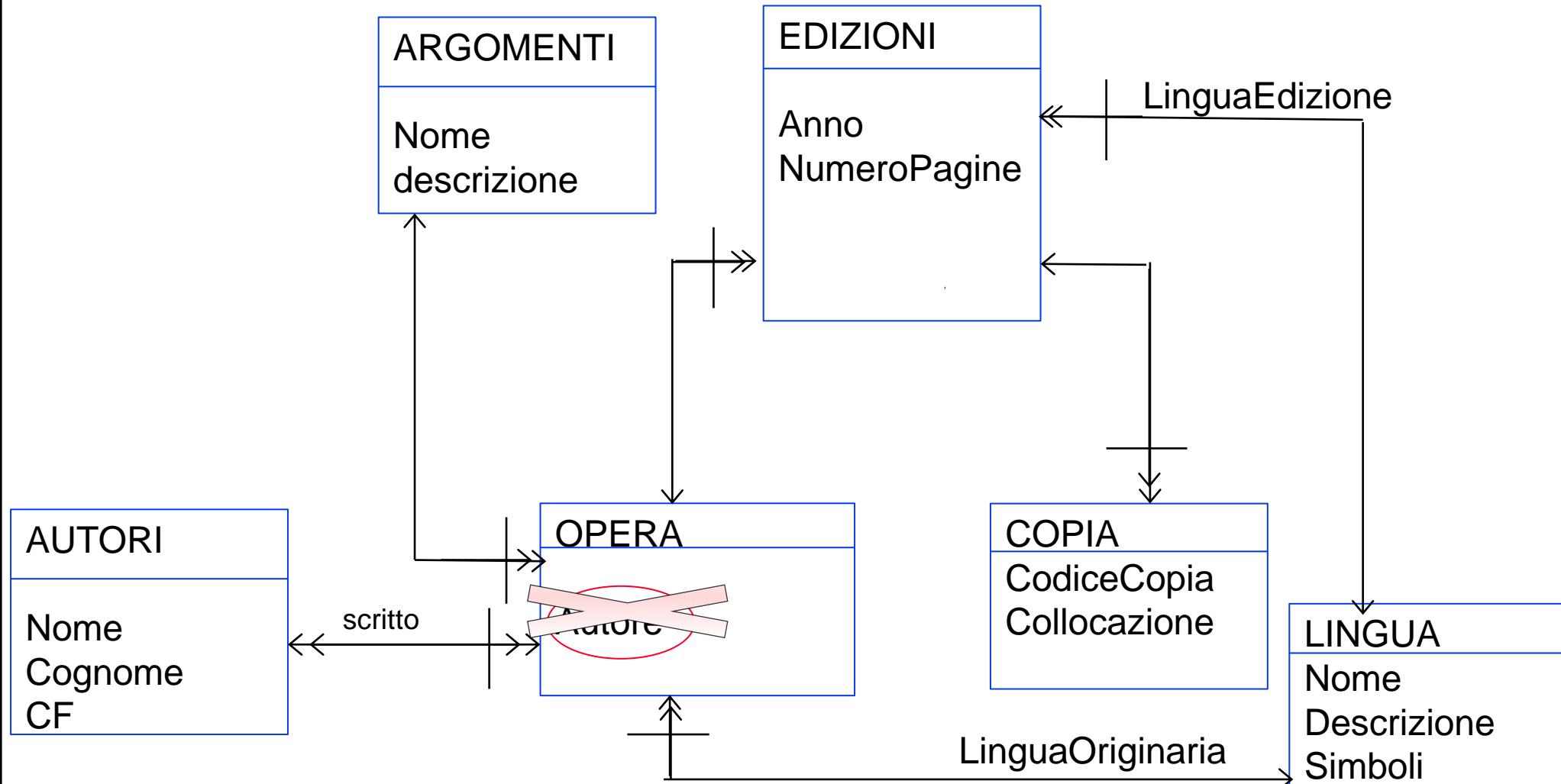
# Prima Modellazione: biblioteca - Argomento principale

---



# Prima Modellazione: biblioteca - Autori

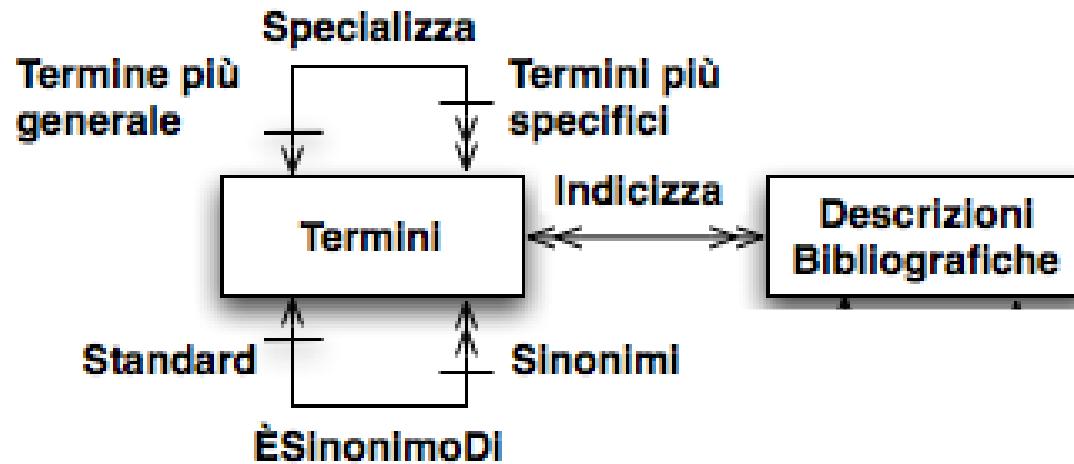
---



## Prima Modellazione: biblioteca - Parole chiavi

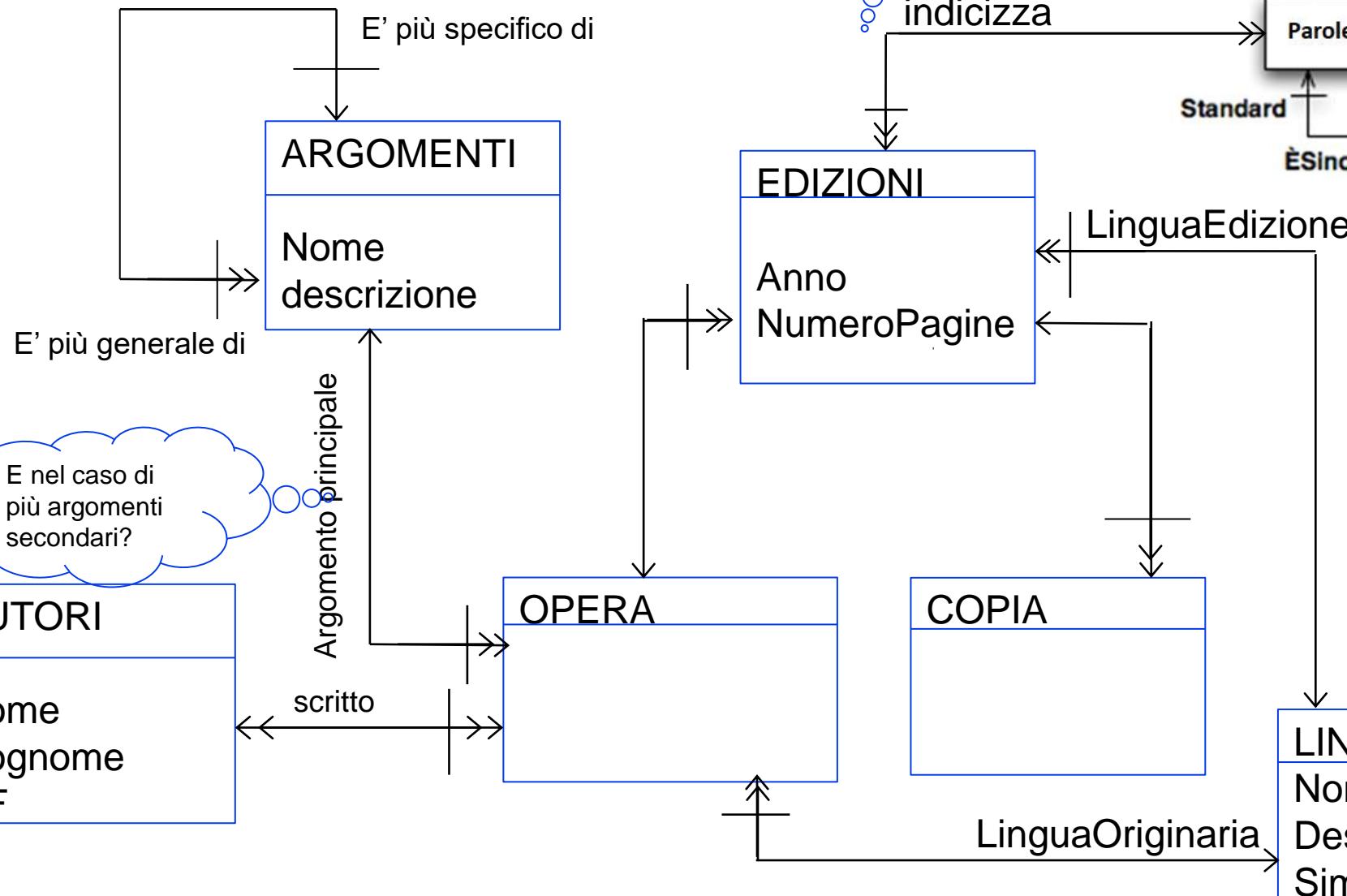
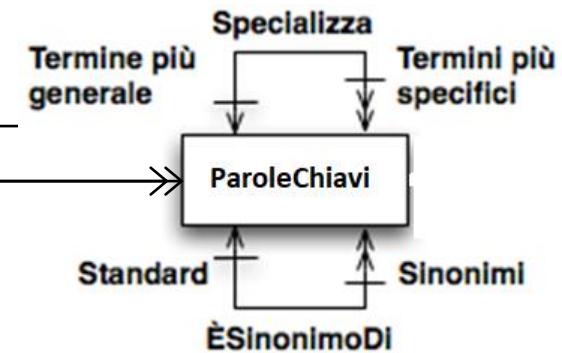
---

- i termini del thesaurus (parole chiave),



# Prima Modellazione: biblioteca - argomenti e parole chiavi

Concordate (edizioni oppure opera)?



---

# Gerarchie



## CONOSCENZA CONCRETA: GERARCHIE DI CLASSI

---

- Spesso le classi di entità sono organizzate in una gerarchia di **specializzazione/generalizzazione**. Una classe della gerarchia minore di altre viene detta **sottoclasse** (le altre sono superclassi ).
- Due importanti caratteristiche delle gerarchie:
  - **ereditarietà** delle proprietà
  - gli elementi di una sottoclasse sono un sottoinsieme degli elementi della **superclasse**

## GERARCHIE DI CLASSI: SCELTA DELLE SOTTOCLASSI

---

- La classe degli *studenti universitari* è una generalizzazione delle classi:
  - *matricole e dei laureandi.*
  - *studenti in corso e degli studenti fuori corso.*
  - *studenti pisani e degli studenti fuori sede.*

La classe *persone* è una generalizzazione delle classi:

- Patentati (in cui viene specificato il tipo di patente) e non patentati
- Studenti e non studenti

## MODELLO A OGGETTI: GERARCHIA TRA TIPI OGGETTO

---

- Fra i tipi oggetto è definita una relazione di **sottotipo**, con le seguenti proprietà:
  - È **asimmetrica, riflessiva e transitiva** (relazione di ordine parziale)
  - Se  $T$  è **sottotipo** di  $T'$ , allora gli elementi di  $T$  possono essere usati in ogni contesto in cui possano apparire valori di tipo  $T'$  (**sostitutività**). In particolare:
    - gli elementi di  $T$  hanno tutte le proprietà degli elementi di  $T'$
    - per ogni proprietà  $p$  in  $T'$ , il suo tipo in  $T$  è un sottotipo del suo tipo in  $T'$ .
  - La gerarchia può essere **semplice o multipla**

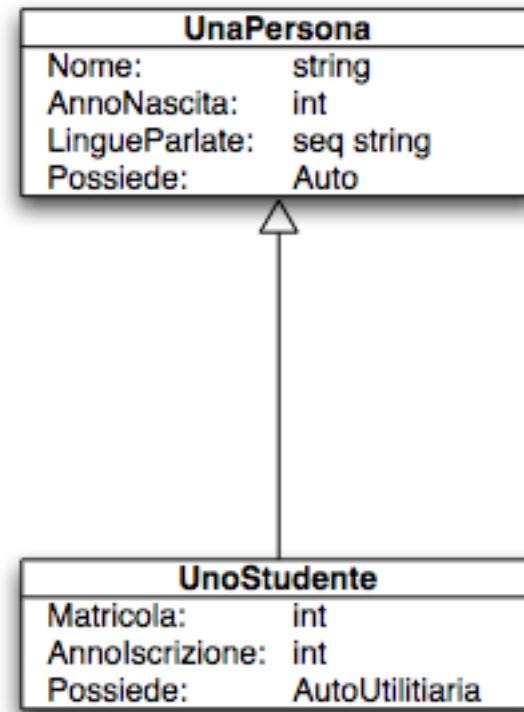
## EREDITARIETÀ

---

- L'ereditarietà (inheritance) permette di definire:
  - un tipo oggetto a partire da un altro
  - l'implementazione di un tipo oggetto a partire da un'altra implementazione
- Normalmente l'eredità tra tipi si usa solo per definire sottotipi, e l'ereditarietà tra implementazioni per definire implementazioni di sottotipi (ereditarietà stretta); in questo caso:
  - gli attributi possono essere solo aggiunti
  - gli attributi possono essere ridefiniti solo specializzandone il tipo

# TIPI DEFINITI PER EREDITARIETA'

---



## GERARCHIA TRA CLASSI

---

- Fra le classi può essere definita una relazione di sottoclasse, detta anche **Sottoinsieme**, con le seguenti proprietà:
  - È asimmetrica, riflessiva e transitiva.
  - Se ***C*** è sottoclasse di ***C'***, allora il **tipo** degli elementi di ***C*** è **sottotipo** del tipo degli elementi di ***C'*** (vincolo **intensionale**)
  - Se ***C*** è sottoclasse di ***C'***, allora gli **elementi** di ***C*** sono un **sottoinsieme** degli elementi di ***C'*** (vincolo **estensionale**).

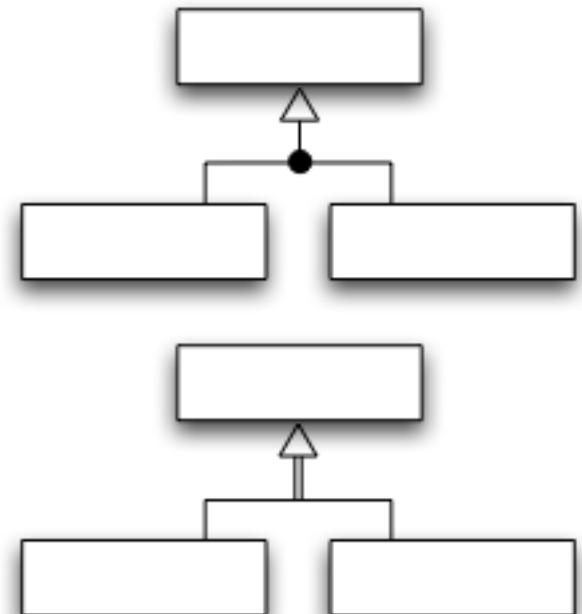
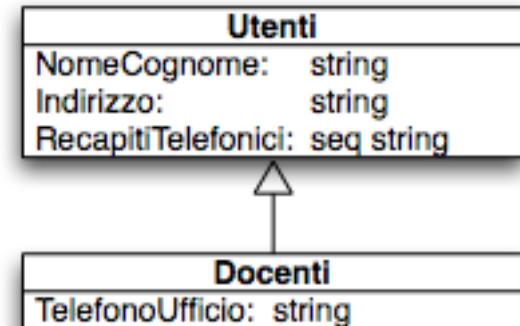
# ESEMPI

---

- **Inclusione**

- **Vincoli su insiemi di sottoclassi:**

- **Disgiunzione**: ogni coppia di sottoclassi in questo insieme è disgiunta, ovvero è priva di elementi comuni (**sottoclassi disgiunte**);
- **Copertura**: l'unione degli elementi delle sottoclassi coincide con l'insieme degli elementi della superclasse (**sottoclassi copertura**).



## 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.

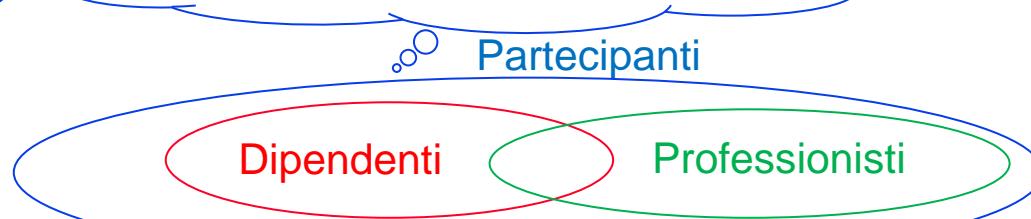
Non disgiunte e copertura?



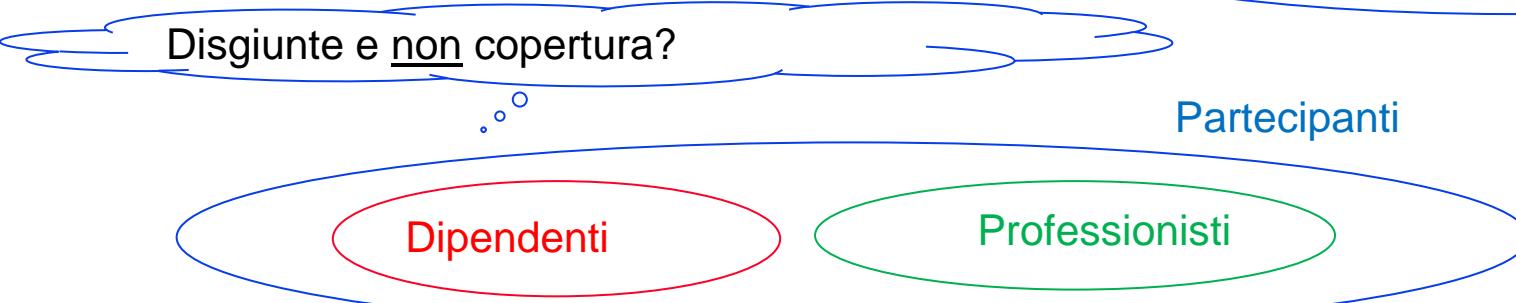
Partizione (Disgiunte e copertura)?



Non disgiunte e non copertura?

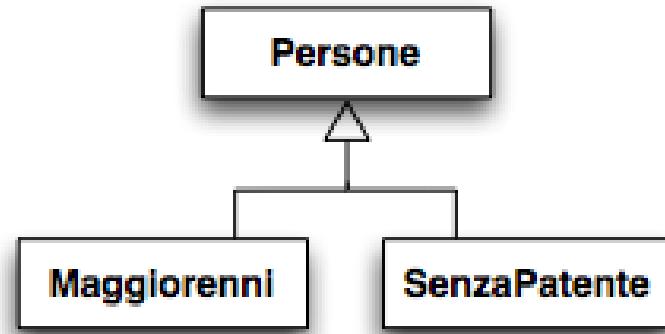


Disgiunte e non copertura?

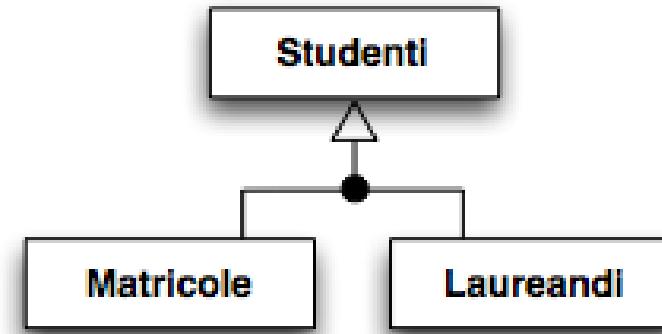


# RELAZIONE TRA SOTTOINSIEMI

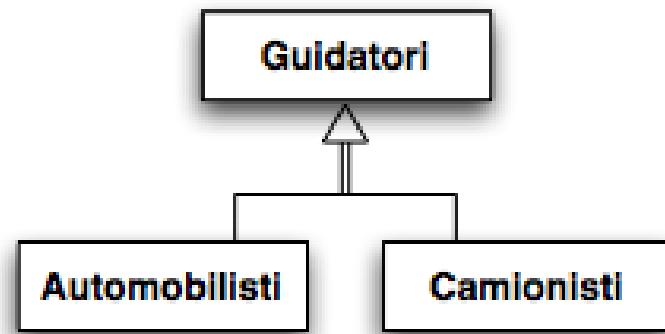
---



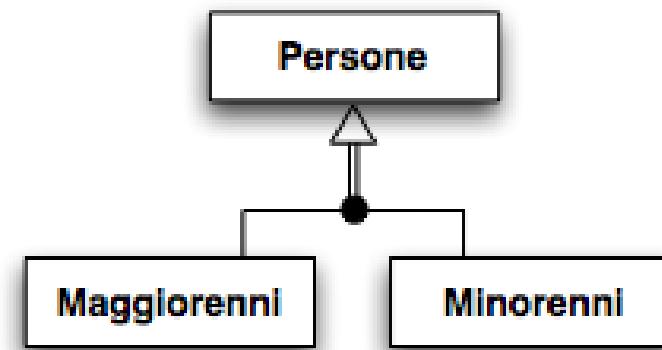
*Sottoclassi scorrelate*



*Sottoclassi disgiunte*



*Sottoclassi copertura*



*Sottoclassi partizione*

## Sottosclassi scorrelate

---

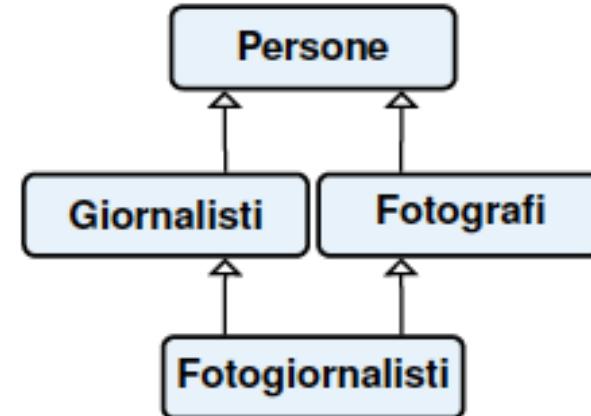
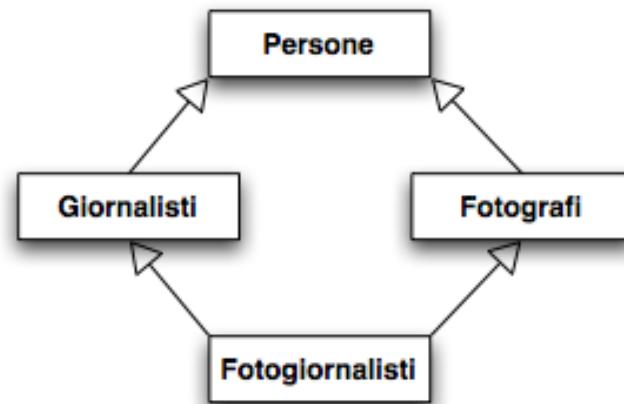
- Sottoclassi scorrelate, non richiedendo né il vincolo di copertura né quello di disgiunzione, si possono rappresentare anche nel seguente modo:



## GERARCHIA MULTIPLA

---

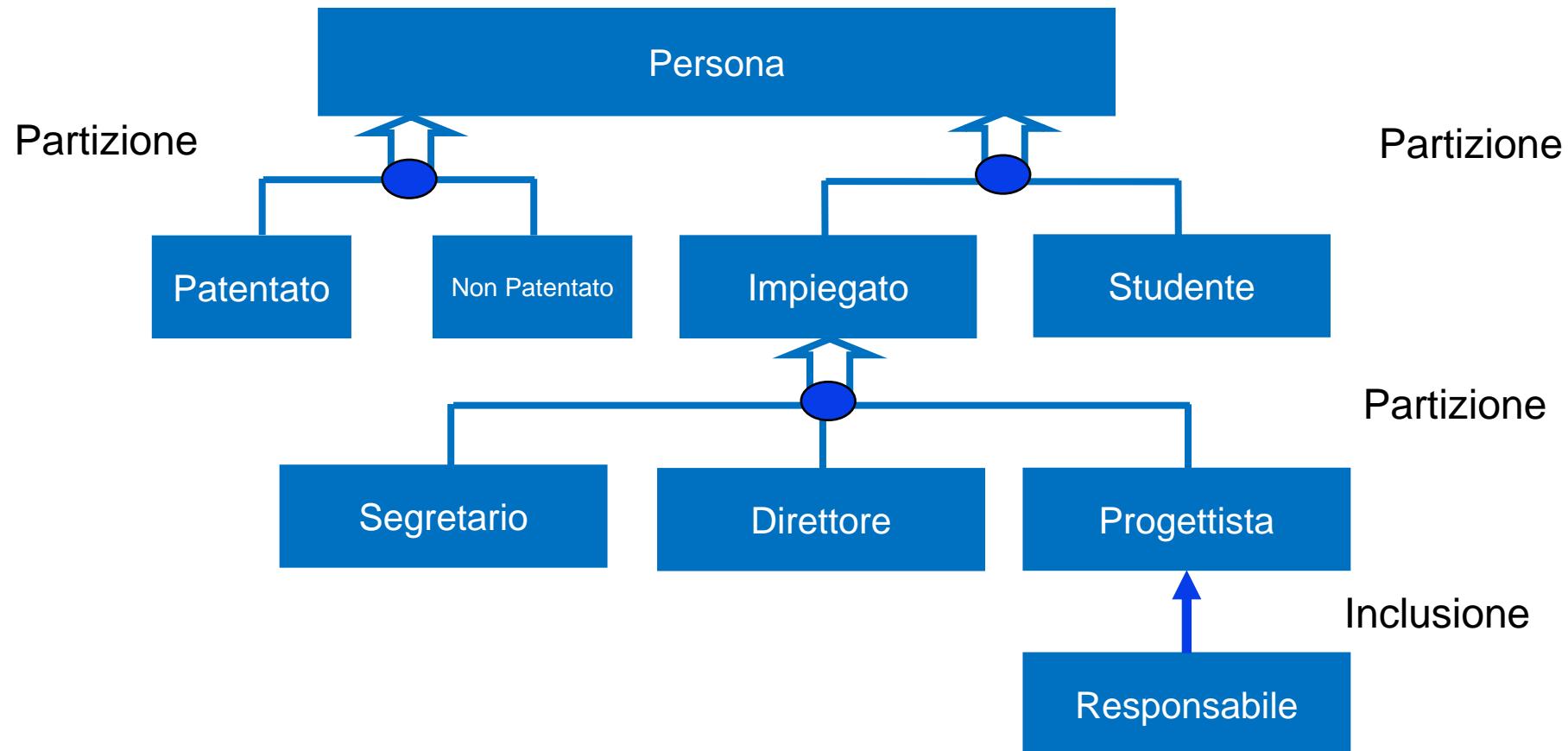
- un tipo può essere definito per ereditarietà a partire da un unico supertipo (ereditarietà singola) o da più supertipi (ereditarietà multipla).
- **Ereditarietà multipla:** è molto utile ma può creare alcuni problemi quando lo stesso attributo viene ereditato, con tipi diversi, da più tipi antenati.



# Esercizio

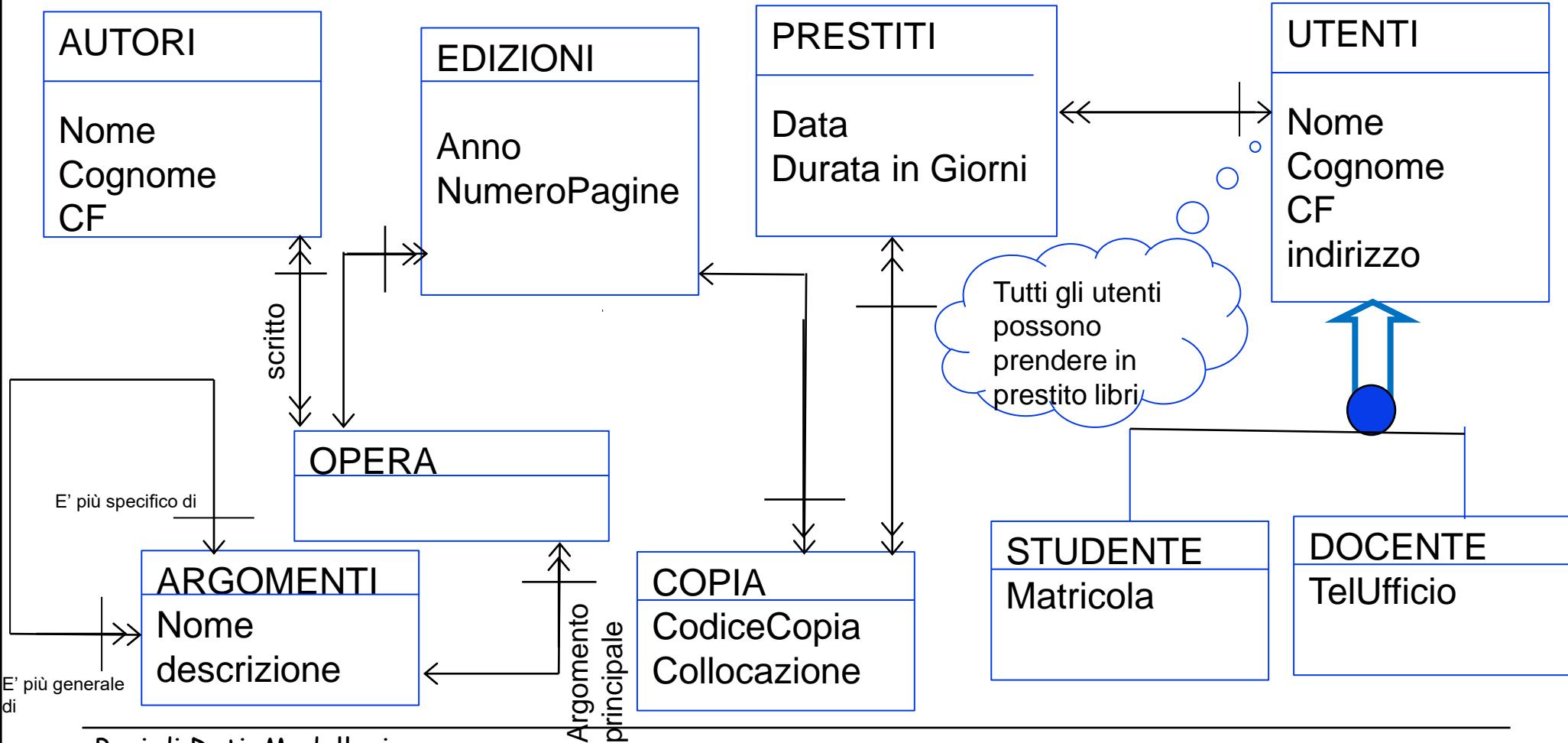
---

- Specificare il tipo di gerarchie



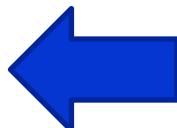
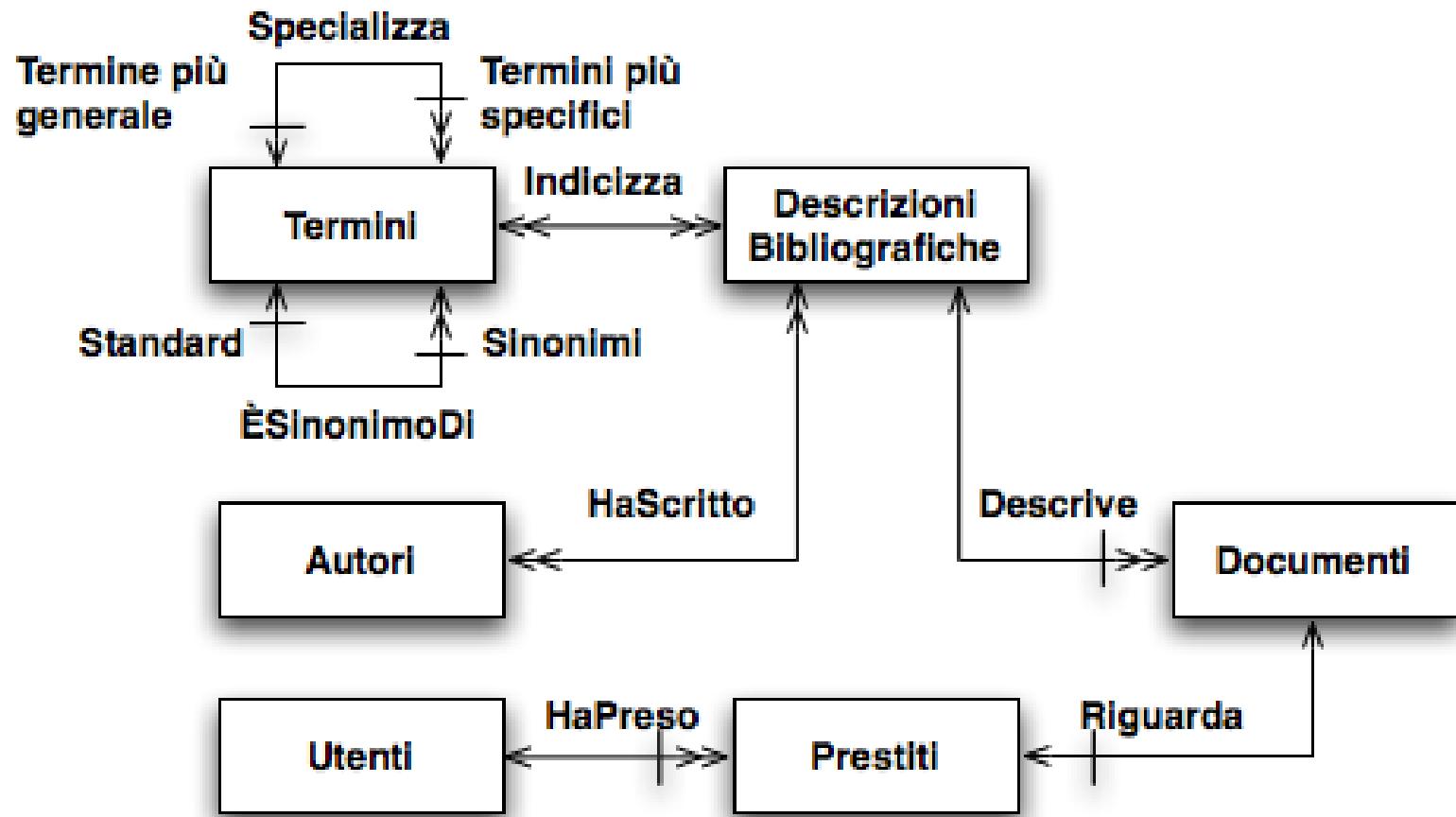
# Prima Modellazione: biblioteca - Gerarchia su utenti

- Gli utenti **regolari** possono essere studenti o docenti. Di uno studente interessa anche la matricola e di un docente anche il telefono dell'ufficio.

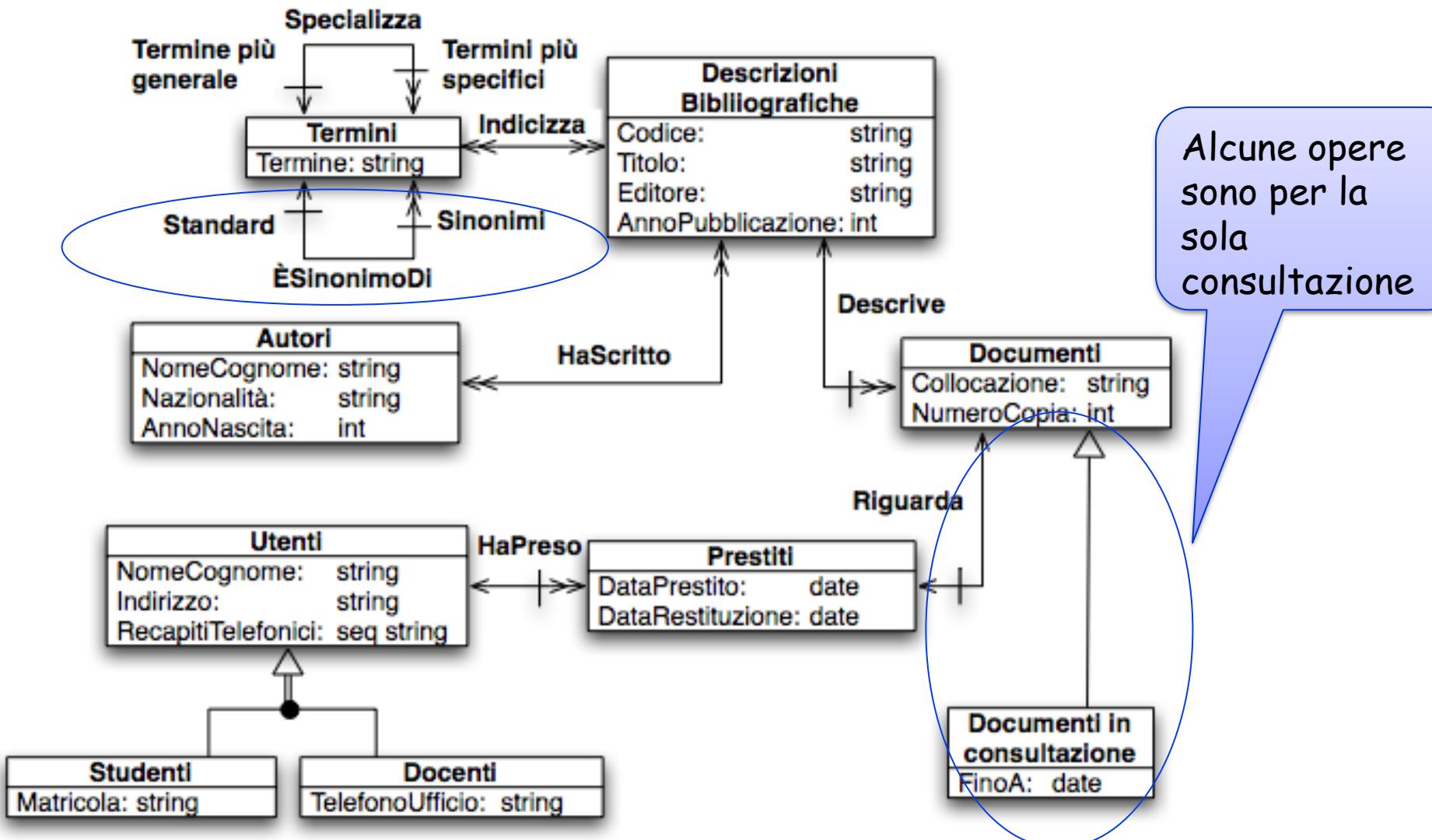


# ESEMPIO DELLA BIBLIOTECA

---



# ESEMPIO DELLA BIBLIOTECA CON SOTTOCLASSI



# ESEMPIO DELLA BIBLIOTECA CON SOTTOCLASSI

- Alcune opere sono per la sola consultazione e **possono essere presi in prestito solo da docenti**



# ESEMPIO DELLA BIBLIOTECA CON SOTTOCLASSI

- Alcune opere sono per la sola consultazione e **possono essere presi in prestito solo da docenti**



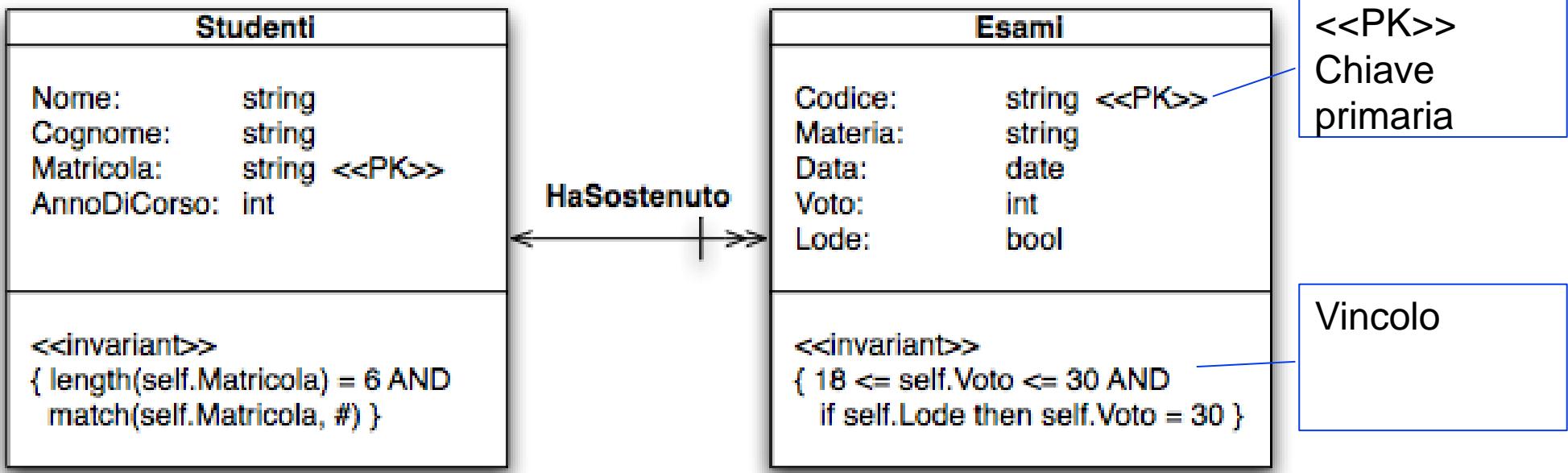
## COSA SI MODELLA: LA CONOSCENZA ASTRATTA

---

- **Conoscenza astratta:** fatti generali che descrivono:
  - la struttura della conoscenza concreta (collezioni, tipi entità, associazioni),
  - restrizioni sui valori possibili della conoscenza concreta e sui modi in cui essi possono evolvere nel tempo (vincoli d'integrità)
    - Vincoli statici e vincoli dinamici
  - regole per derivare nuovi fatti da altri noti.

# DESCRITTORE DI CLASSE CON VINCOLI

- I vincoli possono essere descritti in modo **dichiarativo** (da preferire), con formule del calcolo dei predicati, oppure mediante **controlli** da eseguire nelle operazioni (di base o degli utenti)



Esempio:

Un impiegato può essere direttore solo del dipartimento a cui afferisce

Nessun impiegato può guadagnare più del suo direttore

Ogni impiegato deve risiedere nella città in cui ha sede il dipartimento in cui lavora

# LA COSTRUZIONE DI UNA BASE DI DATI

---

- Analisi dei requisiti
- Progettazione:
  - Progettazione concettuale, logica, fisica dei dati
  - Progettazione delle applicazioni
- Realizzazione
- Noi spesso considereremo l'analisi dei requisiti una parte della progettazione

# FASI DELLA PROGETTAZIONE - RIEPILOGO

---

- Analisi dei requisiti -> specifica dei requisiti, schemi di settore
  - Progettazione concettuale -> schema concettuale
  - Progettazione logica -> schema logico
  - Progettazione fisica -> schema fisico
- 
- Tanto lo schema concettuale che quello logico contengono le viste esterne

## ANALISI DEI REQUISITI

---

- Analizza il sistema esistente e raccogli requisiti informali
- Elimina ambiguità imprecisioni e disuniformità
- Raggruppa le frasi relative a diverse categorie di dati, vincoli, e operazioni
- Costruisci un glossario
- Disegna lo schema di settore
- Specifica le operazioni
- Verifica la coerenza tra operazioni e dati

## Requisiti: documentazione descrittiva

---

- In generale, il linguaggio naturale è pieno di ambiguità e fraintendimenti. Si deve, quanto più è possibile, evitare tali ambiguità.
- In prima approssimazione si può procedere con le seguenti regole:
  - Studiare e comprendere il sistema informativo e i bisogni informativi di tutti i settori dell'organizzazione
  - scegliere il corretto livello di astrazione
  - standardizzare la struttura delle frasi
  - suddividere le frasi articolate
  - separare le frasi sui dati da quelle sulle funzioni

## Requisiti: organizzazione di termini e concetti

---

- Regole generali:
  - Eliminare ambiguità, imprecisioni e disuniformità: individuare **omonimi** e **sinonimi** e unificare i termini; rendere esplicito il **riferimento fra termini**.
  - riorganizzare le frasi per **concetti**, ovvero ottenendo diverse categorie di dati, vincoli e operazioni
  - costruire un **glossario** dei termini
  - Disegnare lo schema
  - Specificare le operazioni
  - Verificare la coerenza fra operazioni e dati.

# Regole generali per l'analisi dei requisiti

---

- scegliere il corretto livello di astrazione
- standardizzare la struttura delle frasi
- suddividere le frasi articolate in frasi più semplici
- separare le frasi sui dati da quelle sulle funzioni
- costruire un glossario dei termini
- individuare omonimi e sinonimi e unificare i termini
- rendere esplicito il riferimento fra termini
- riorganizzare le frasi per concetti

# Ambiguità e imprecisioni da correggere

## Società di formazione (1)

Si vuole realizzare una base di dati per una società che eroga corsi, di cui vogliamo rappresentare i dati dei **partecipanti** ai corsi e dei docenti. Per 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 frequentato (i corsi sono in tutto circa 200) e il giudizio finale.

Studenti o partecipanti?

Posti (non nel senso di luogo/città) dove hanno lavorato

Cosa si intende per «Periodo»?

Il committente lascia intuire che tali informazioni si riferiscono al datore di lavoro

## Società di formazione (2)

Rappresentiamo anche i **seminari** che stanno attualmente frequentando e, per ogni **giorno**, i **luoghi** e le ore dove sono tenute le lezioni. I **corsi** hanno un codice, un titolo e possono avere varie **edizioni** con date di inizio e fine e numero di partecipanti. Se gli **studenti** sono liberi professionisti, vogliamo conoscere l'area di interesse e, se lo possiedono, il **titolo**. Per quelli che lavorano alle dipendenze di altri, vogliamo conoscere invece **il loro livello e la posizione ricoperta**.

Seminari, corsi o edizioni dei corsi?

giorno o giorni della settimana?

Luogo dove sono tenute le lezioni

Studenti o partecipanti?

Titolo o titolo professionale?

### Società di formazione (3)

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.

Insegnanti o docenti?

Posto dove sono nati o luogo di nascita?

«Nome del corso» ovvero il «titolo del corso»

Recapiti telefonici oppure numeri telefonici?

# 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.	Seminario	Docente
Datori	Ente presso cui i partecipanti lavorano o hanno lavorato	Posti	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 cui vogliamo rappresentare i dati dei partecipanti ai corsi e dei docenti.

**Partecipanti**

**Corsi**

**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.

Partecipanti
Codice
CodiceFiscale
Cognome
Nome
Genere
CittaNascita
DataNascita

## 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.

## Docenti

CodiceFiscale

Cognome

Nome

Genere

CittaNascita

DataNascita

Recapiti

## 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.

## Corsi

Codice

Titolo

Edizioni?

---

## 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.

DatoriLavoro

CodiceFiscale

Cognome

Nome

Indirizzo

Citta

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.

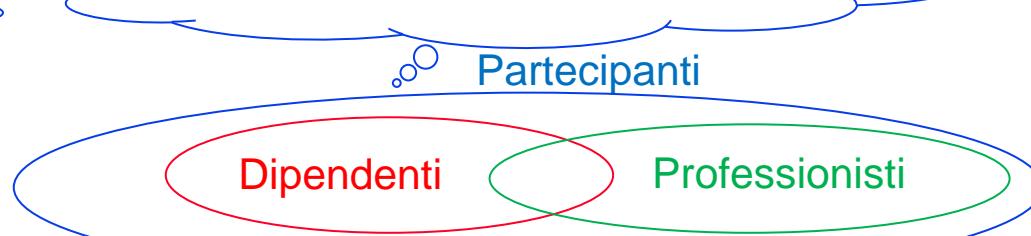
Non disgiunte e copertura?



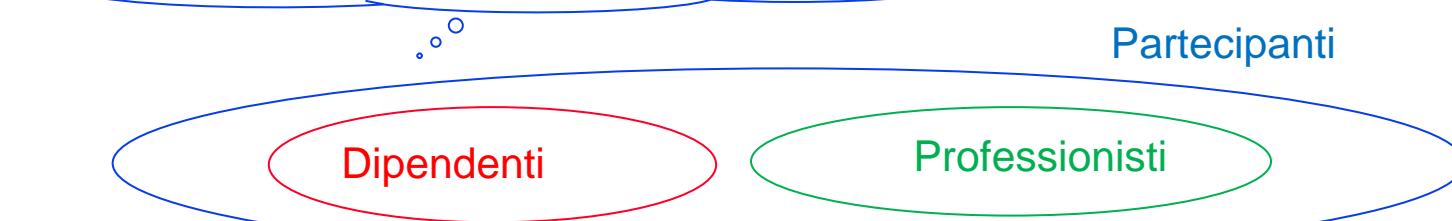
Partizione (Disgiunte e copertura)?



Non disgiunte e non copertura?



Partizione (Disgiunte e non copertura)?

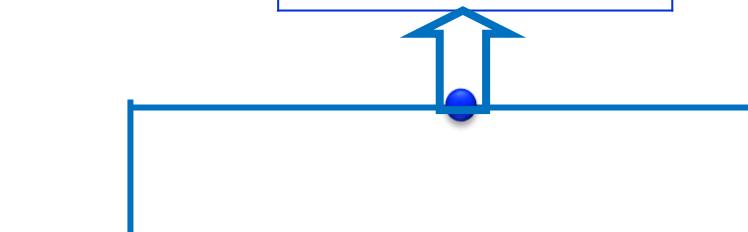


## Frasi relative a tipi specifici di partecipanti

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.

### Partecipanti

Codice
CodiceFiscale
Cognome
Nome
Genere
CittaNascita
DataNascita



### Dipendenti

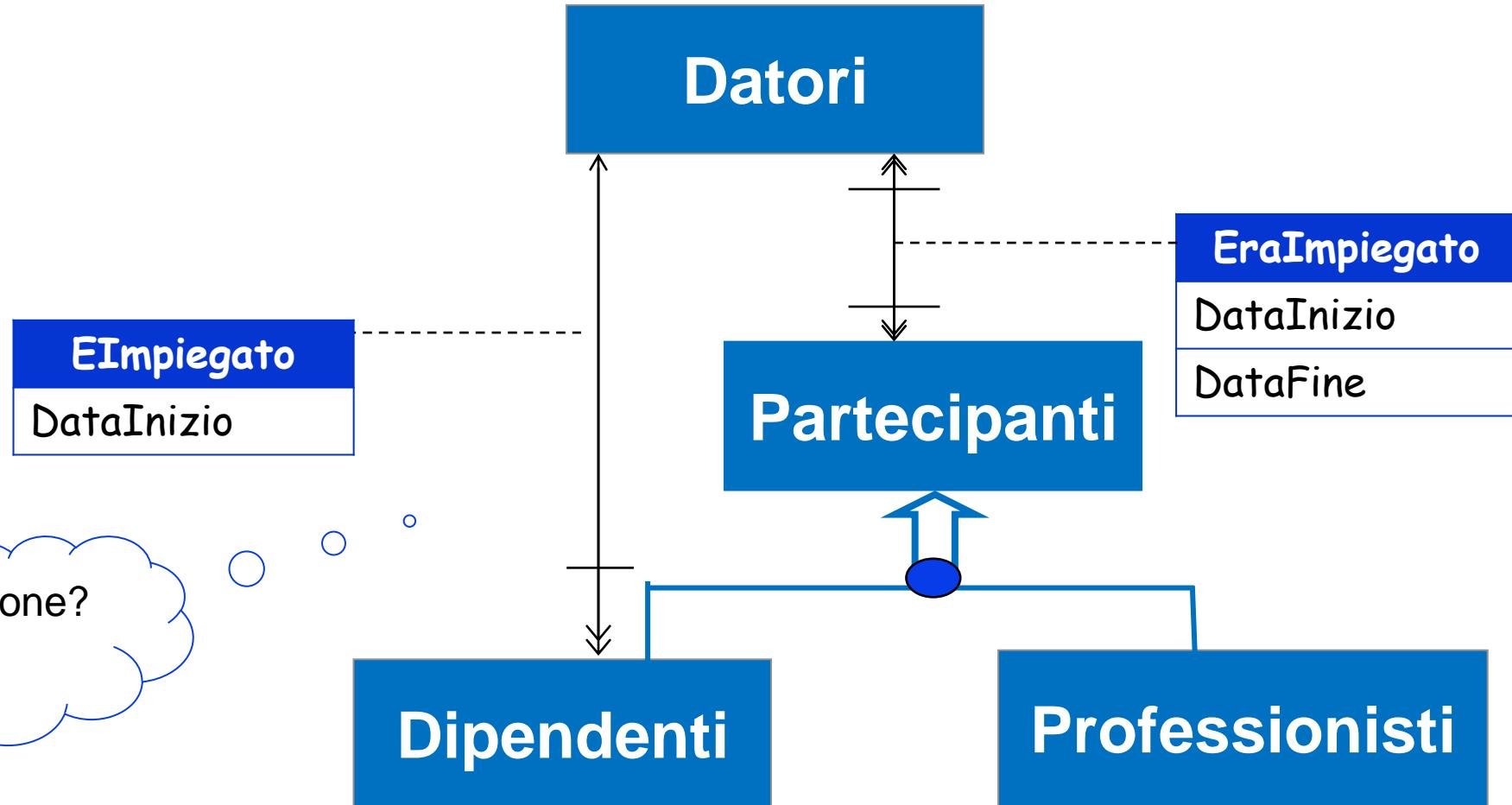
Livello
PosizioneRicoperta

### Professionisti

AreaInteresse (0,1)
TitoloProfessionale

## Frasi relative ai partecipanti

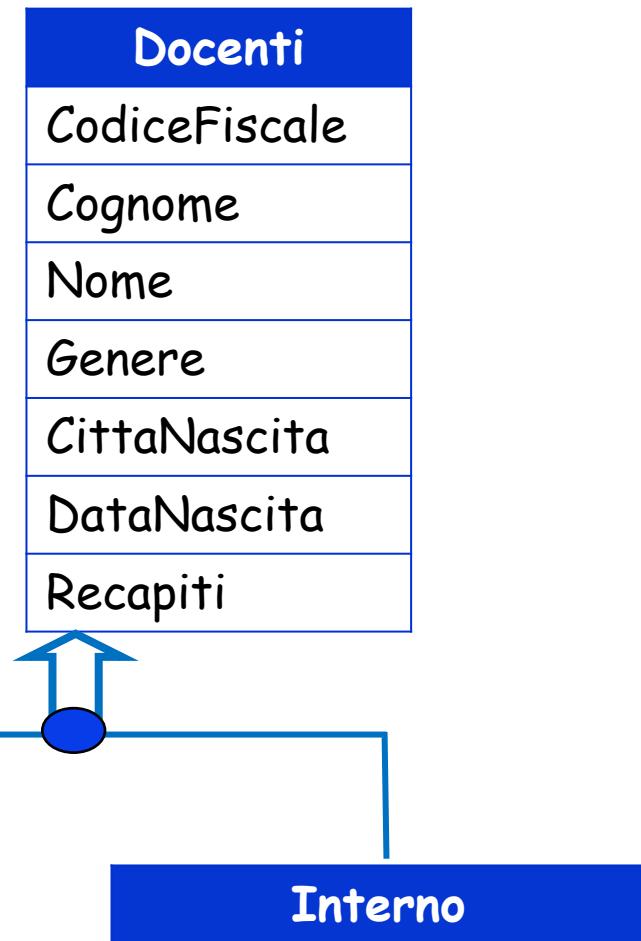
Per i partecipanti (circa 5000), .... i nomi dei loro attuali datori di lavoro e di quelli precedenti (insieme alle date di inizio e fine rapporto), ....



---

**Frasi relative ai docenti**

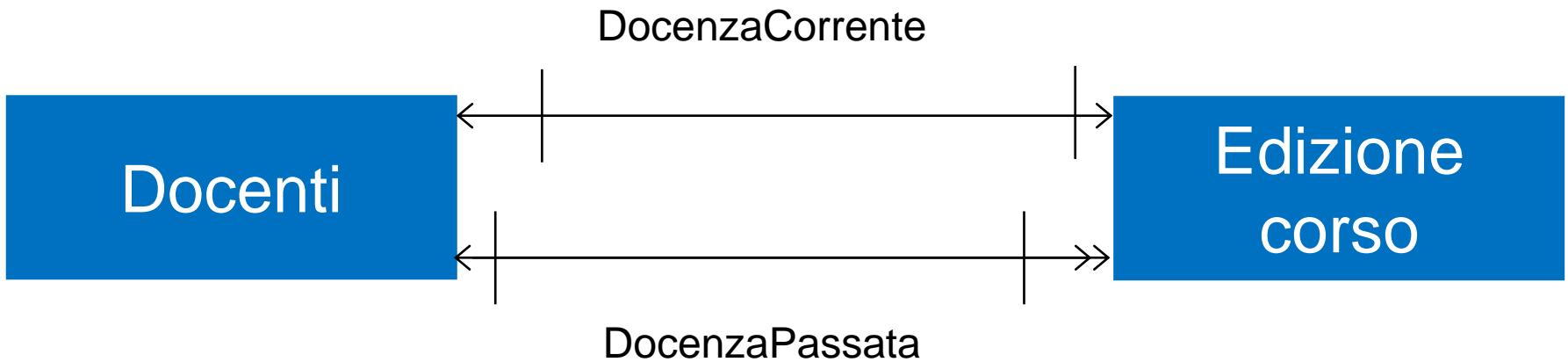
Per i docenti (circa 300), .... I docenti possono essere dipendenti interni della società di formazione o collaboratori esterni.



---

### Frasi relative ai docenti

Per i docenti (circa 300), ...., il titolo del corso che insegnano, di quelli che hanno insegnato nel passato e di quelli che possono insegnare. ...



## Frasi relative ai corsi

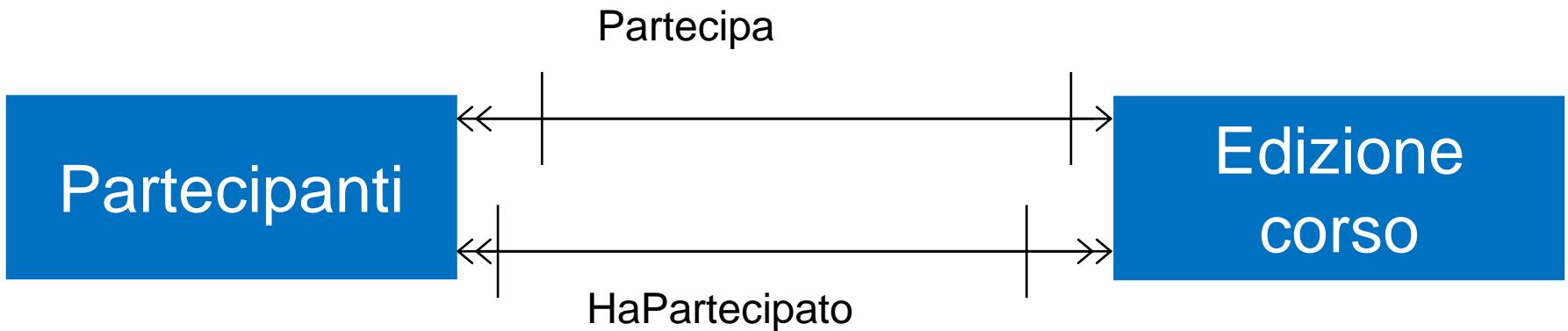
Per i corsi (circa 200), rappresentiamo il titolo e il codice, le **varie edizioni** con date di inizio e fine e, per ogni edizione, rappresentiamo il numero di partecipanti e il giorno della settimana, le aule e le ore dove sono tenute le lezioni.

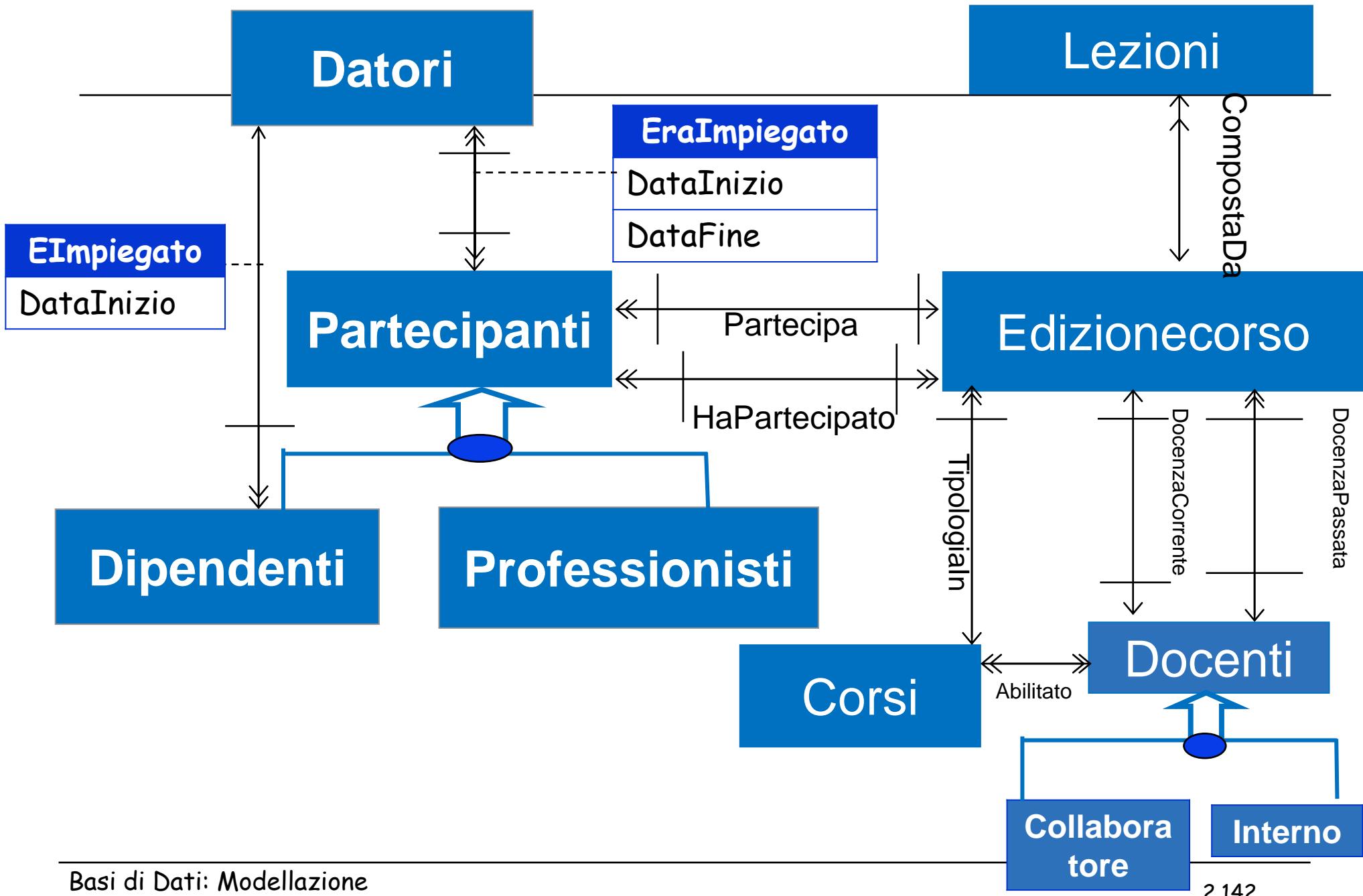


---

### Frasi relative ai partecipanti

Per i partecipanti (circa 5000), .... , le edizioni dei corsi che stanno attualmente frequentando e quelli che hanno frequentato nel passato, con la relativa votazione finale in decimi.





## Specifiche sulle operazioni

---

- Dopo aver definito i dati, occorre stabilire le specifiche sulle operazioni da effettuare sui dati.
- Bisogna utilizzare la stessa **terminologia** usata per i dati.
- Bisognerebbe indicare la **frequenza** con cui vengono effettuate certe **operazioni**.
- La conoscenza di queste informazioni è indispensabile nella fase di progettazione logica.

## Possibili operazioni e la loro frequenza

---

**OPERAZIONE 1:** inserisci un nuovo partecipante, indicando tutti i suoi dati (in media 40 volte al giorno)

**OPERAZIONE 2:** assegna un partecipante a una edizione di corso (50 volte al giorno)

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

**OPERAZIONE 4:** assegna un docente abilitato a una edizione di un corso (15 volte al giorno)

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

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

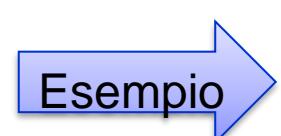
**OPERAZIONE 7:** per ogni docente, trova i partecipanti a tutti i corsi da lui insegnati (5 volte la settimana)

**OPERAZIONE 8:** effettua una statistica su tutti i partecipanti a un corso con tutte le informazioni su di essi, sulla edizione a cui ha partecipato e la rispettiva votazione (10 volte al mese)

# PROGETTAZIONE CONCETTUALE DI SCHEMI SETTORIALI

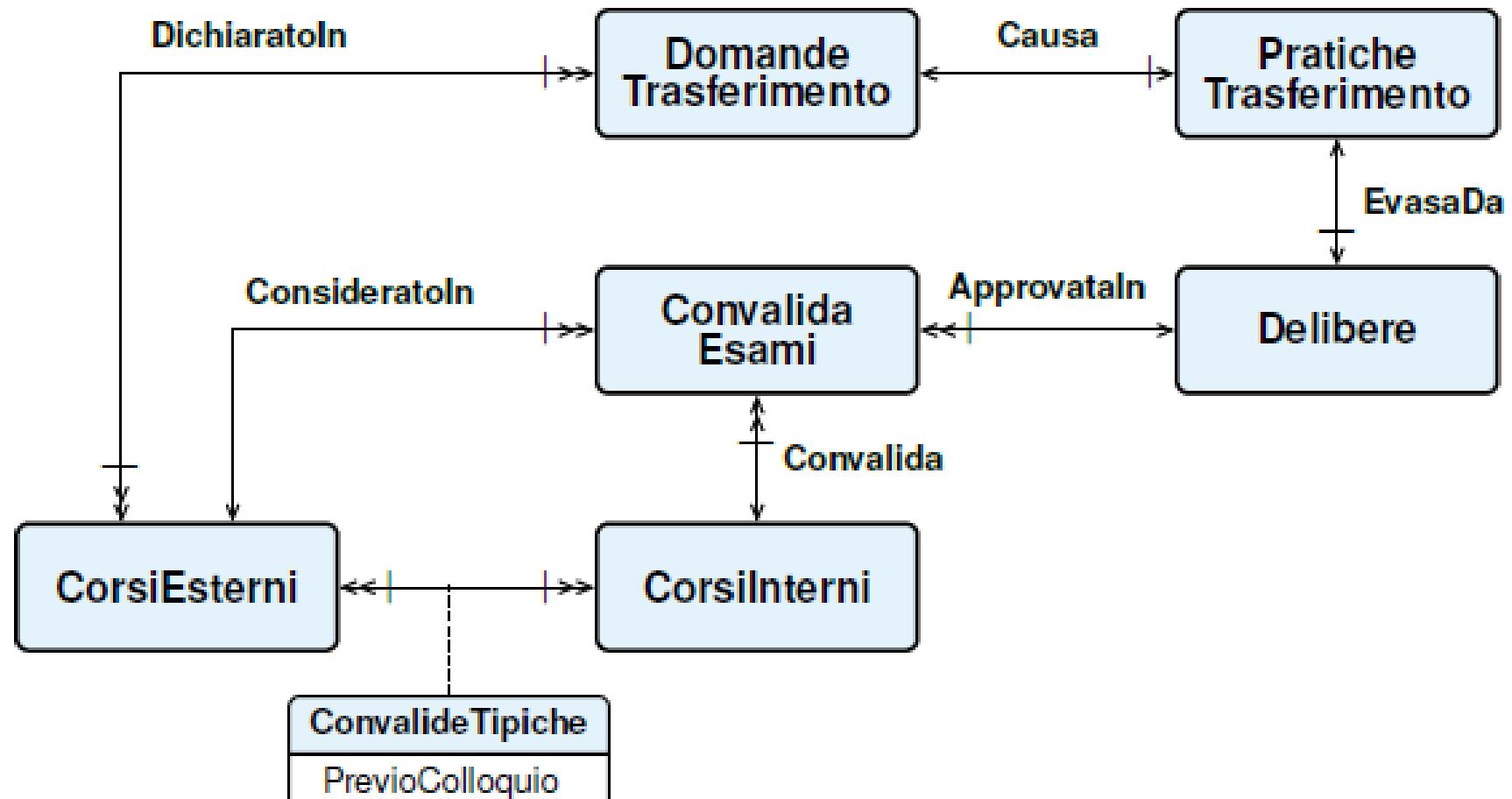
---

- Identificare le classi
- Identificare le associazioni e le loro proprietà strutturali
- Identificare gli attributi delle classi e associazioni e i loro tipi
- Elencare le chiavi
- Individuare le sottoclassi
- Individuare le generalizzazioni



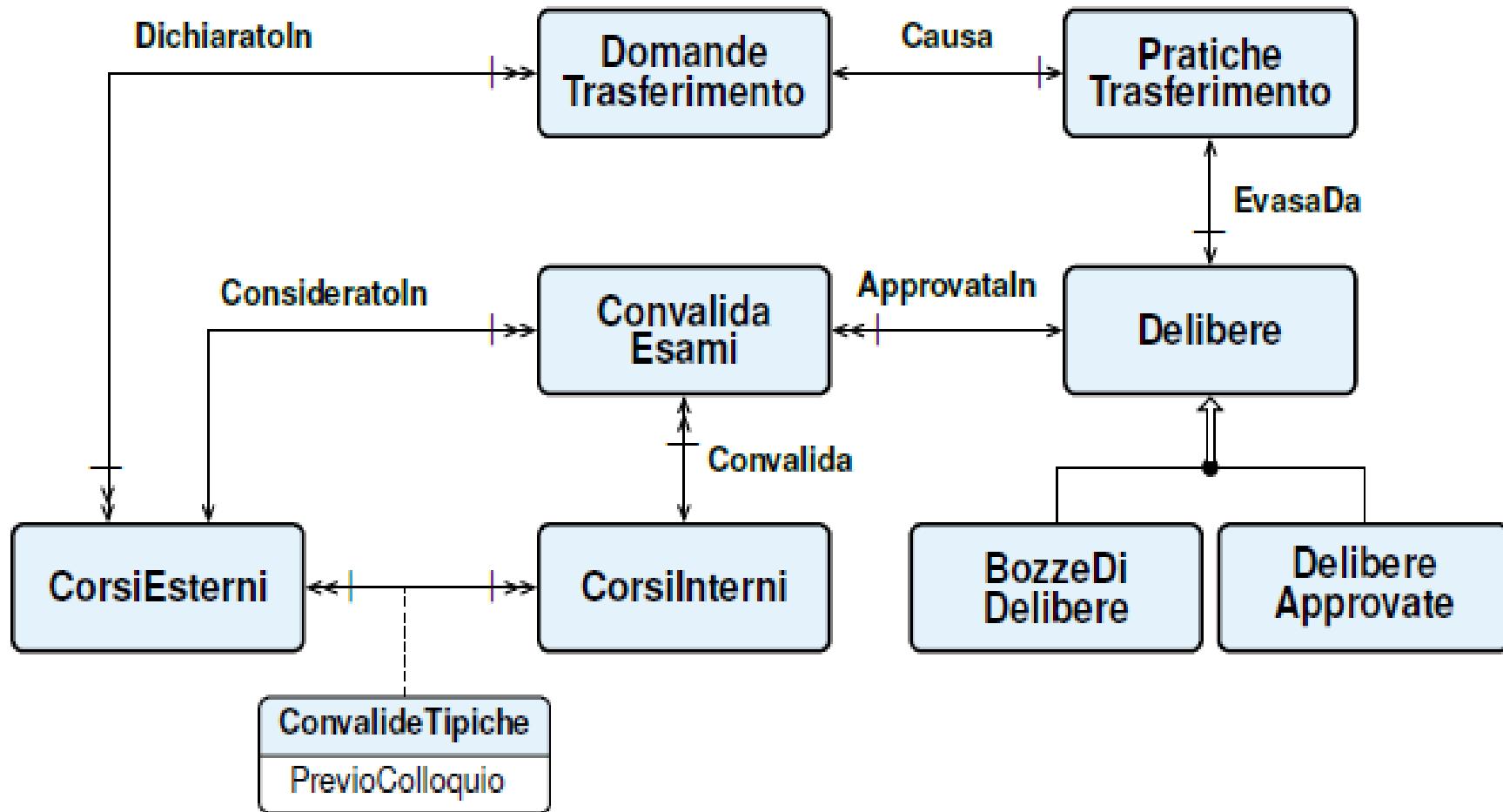
## Primo schema scheletro

---



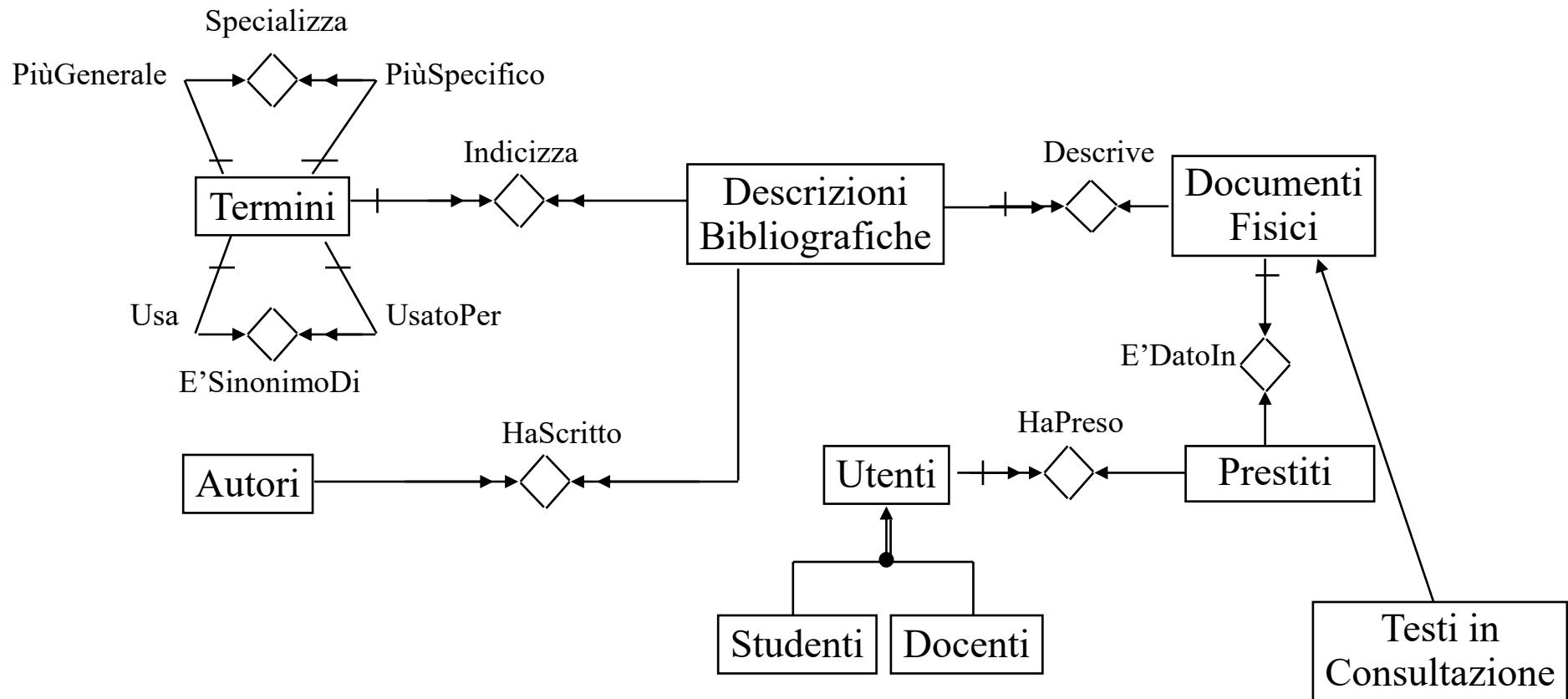
Esempio

## Individuare sottoclassi



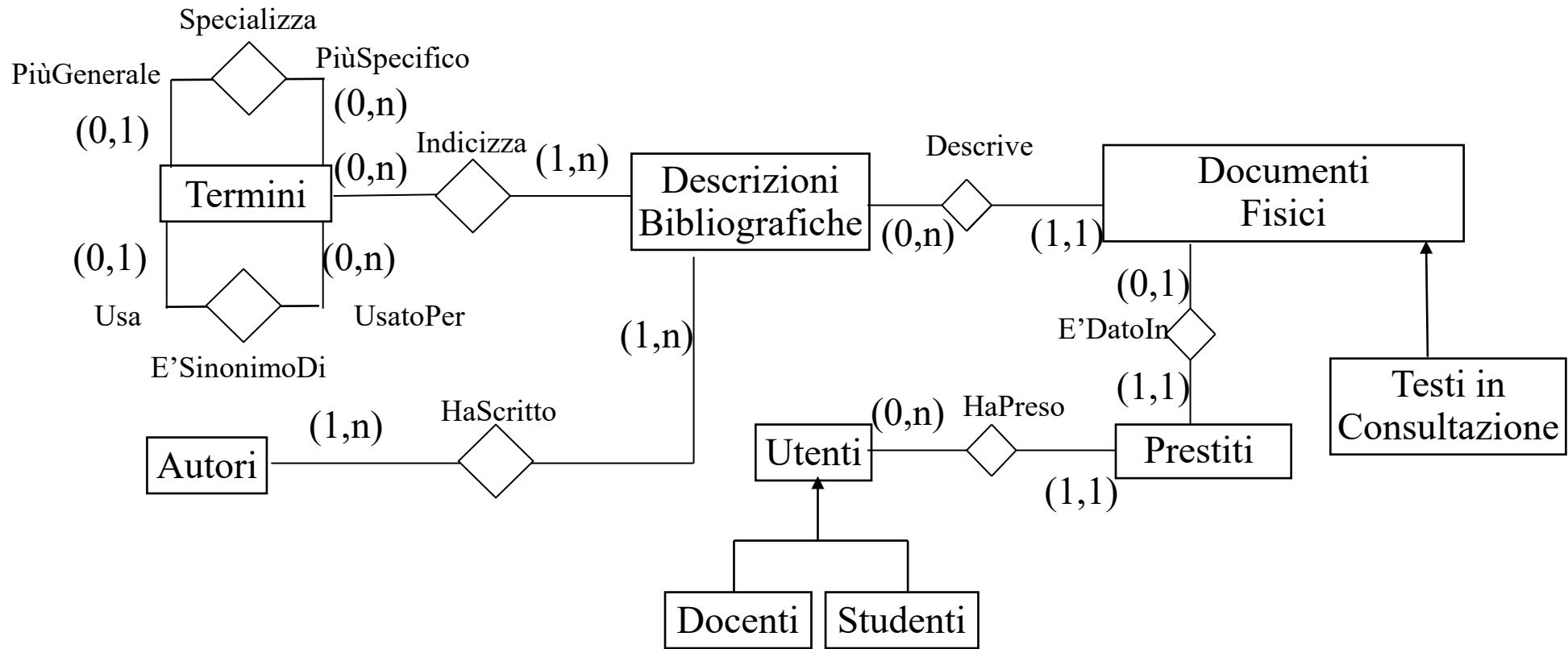
# ALTRI FORMALISMI GRAFICI

---



# ALTRI FORMALISMI GRAFICI

---



## Il modello relazionale

---

- Proposto da E. F. Codd nel 1970 per favorire l'indipendenza dei dati
- Disponibile in DBMS reali nel 1981 (non è facile implementare l'indipendenza con efficienza e affidabilità!)
- Si basa sul concetto matematico di **relazione** (con una variante)
- Le relazioni hanno naturale rappresentazione per mezzo di **tabelle**

## RELAZIONI matematica: come nella teoria degli insiemi

---

- Relazione matematica: come nella teoria degli insiemi.

$D_1, \dots, D_n$  ( $n$  insiemi anche non distinti)

- prodotto cartesiano  $D_1 \times \dots \times D_n$ :

l'insieme di tutte le  $n$ -uple  $(d_1, \dots, d_n)$  tali che  $d_1 \in D_1, \dots, d_n \in D_n$

- relazione matematica su  $D_1, \dots, D_n$ :

un sottoinsieme di  $D_1 \times \dots \times D_n$ .

- $D_1, \dots, D_n$  sono i domini della relazione

## Relazione matematica, esempio

---

$$D_1 = \{a, b\}$$

$$D_2 = \{x, y, z\}$$

prodotto cartesiano  $D_1 \times D_2$

a	x
a	y
a	z
b	x
b	y
b	z

una relazione

$$r \subseteq D_1 \times D_2$$

a	x
a	z
b	y

## Relazione matematica, proprietà

---

Una **relazione matematica** è un insieme di *n*-uple ordinate:

$(d_1, \dots, d_n)$  tali che  $d_1 \in D_1, \dots, d_n \in D_n$

Oss: una relazione è un insieme; quindi:

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

## Relazione matematica: struttura posizionale dei domini

---

*Partite*  $\subseteq$  string  $\times$  string  $\times$  int  $\times$  int

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

- Ciascuno dei domini ha due **ruoli** diversi, distinguibili attraverso la posizione:
  - La struttura è **posizionale**

## Relazione matematica: struttura non posizionale

---

$\text{Partite} \subseteq \text{string} \times \text{string} \times \text{int} \times \text{int}$

A ciascun dominio si associa un nome (**attributo**), che ne descrive il "ruolo"

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

Una **tupla** su un insieme di attributi  $X$  è una funzione  $t$  che associa a ciascun attributo  $A$  un valore del suo dominio.

Una **relazione su  $X$**  è un insieme di tuple su  $X$ .

## Tabelle e relazioni

---

- Una tabella rappresenta una **relazione** se
  - i valori di ogni colonna sono fra loro omogenei
  - le righe sono diverse fra loro
  - le intestazioni delle colonne sono diverse tra loro
- In una tabella che rappresenta una relazione
  - l'ordinamento tra le righe è irrilevante
  - l'ordinamento tra le colonne è irrilevante

**Il modello è basato su valori**

---

**Il modello relazionale è basato su valori.**

Ciò significa che i riferimenti fra dati in relazioni diverse sono rappresentati per mezzo di valori dei domini che compaiono nelle ennuple.

## Schemi e istanze

---

- In ogni base di dati si distinguono:
  - lo **schema**, sostanzialmente invariante nel tempo, che ne descrive la struttura (aspetto intensionale)
    - le intestazioni delle tabelle
  - l'**istanza**, i valori attuali, che possono cambiare anche molto rapidamente (aspetto estensionale)
    - il “corpo” di ciascuna tabella

Studenti

	<u>Nome</u>	<u>Matricola</u>	Provincia	AnnoNascita
Isaia	071523	PI	1982	
Rossi	067459	LU	1984	
Bianchi	079856	LI	1983	
Bonini	075649	PI	1984	

Schema di relazione

Istanza di Relazione o estensione della relazione

# IL MODELLO RELAZIONALE

---

- Definizione: I meccanismi per definire una base di dati con il modello relazionale sono l'**ennupla** e la **relazione**:
  - un **tipo ennupla**  $T$  è un insieme finito di coppie (Attributo, Tipo elementare);
  - se  $T$  è un tipo ennupla,  $R(T)$  è lo **schema della relazione  $R$** ;
  - lo schema di una base di dati è un **insieme di schemi di relazione  $R_i(T_i)$** ;
  - un'**istanza** di uno schema  $R(T)$  è un insieme finito di ennuple di tipo  $T$ .
- Uguaglianza di due tipi ennupla, due ennuple, due tipi relazione

## Vantaggi della struttura basata su valori

---

- indipendenza dalle strutture fisiche (si potrebbe avere anche con puntatori di alto livello) che possono cambiare dinamicamente. La rappresentazione logica dei dati (costituita dai soli valori) non fa riferimento a quella fisica
- si rappresenta solo ciò che è rilevante dal punto di vista dell'applicazione
- i dati sono portabili più facilmente da un sistema ad un altro
- i puntatori sono direzionali

## Schema (riepilogo)

---

- Schema di relazione:  
un nome  $R$  con un insieme  $T$  di attributi  $A_1, \dots, A_n$ :

$$R(T) = R(A_1, \dots, A_n)$$

- Schema di base di dati:  
insieme di schemi di relazione:

$$R = \{R_1(X_1), \dots, R_k(X_k)\}$$

Dove  $X_1, \dots, X_k$  sono insiemi di attributi

## Tupla

---

- Una **tupla** su un insieme di attributi  $T$ , denotata con  $t$ , è una funzione che associa a ciascun attributo  $A$  in  $T$  un valore del dominio di  $A$
- $t[A]$  denota il valore della tupla  $t$  sull'attributo  $A \in T$
- $t[X]$  denota i valori della tupla  $t$  sugli attributi  $X \in T$

$X = \{\text{Esame, Idoneità}\}$

Chi è  $t[X]?$  . . .

Matricola	Esame	Voto	Idoneità
936463	Inglese	NULL	Si
936462	Basi di dati	30	No
...	...	...	...

## Istanze

---

- Un'istanza di relazione o relazione su uno schema  $R(X)$  è l'insieme  $r$  di tuple su  $X$
- Un'istanza di base di dati su uno schema

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

è l'insieme delle relazioni  $r = \{r_1, \dots, r_n\}$  (con  $r_i$  relazione su  $R_i$ )

## Relazione come una tabella bidimensionale

---

### Orario

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

Orario

## Schema

Insegnamento	Docente	Aula	Ora
--------------	---------	------	-----

## Istanza

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

## Informazione incompleta

---

- Il modello relazionale impone ai dati una struttura rigida:
  - le informazioni sono rappresentate per mezzo di ennuple
  - solo alcuni formati di ennuple sono ammessi: quelli che corrispondono agli schemi di relazione

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



Valori nulli

## Informazione incompleta nel modello relazionale

---

- **valore nullo:** denota l'assenza di un valore del dominio (e non è un valore del dominio)
- $t[A]$ , per ogni attributo  $A$ , è un valore del dominio  $\text{dom}(A)$  oppure il valore nullo **NULL**
- Si possono (e devono) imporre restrizioni sulla presenza di valori nulli

<b>IdPersona</b>	<b>Stato civile</b>	<b>Coniuge</b>
936463	celibe	NULL
936462	sposato	936465

<b>Matricola</b>	<b>Esame</b>	<b>Voto</b>	<b>Idoneità</b>
936463	Inglese	NULL	Si
936462	Basi di dati	30	NULL

## Troppi valori nulli

---

studenti	Matricola	Cognome	Nome	Data di nascita
	6554	Rossi	Mario	05/12/1978
	9283	Verdi	Luisa	12/11/1979
	NULL	Rossi	Maria	01/02/1978

esami	Studente	Voto	Corso
	NULL	30	NULL
	NULL	24	02
	9283	28	01

corsi	Codice	Titolo	Docente
	01	Analisi	Mario
	02	Chimica	NULL
	NULL	Chimica	Verdi

## Vincoli di integrità

---

Esistono istanze di basi di dati che, pur **sintatticamente corrette**, non rappresentano informazioni possibili per l'applicazione di interesse e che quindi generano **informazioni senza significato**.

Esami	Studente	Voto	Lode	Corso
	276545	<b>32</b>		01
	276545	30	e lode	02
	787643	<b>27</b>	<b>e lode</b>	03
	<b>739430</b>	24		04

Studenti	Matricola	Cognome	Nome
	276545	Rossi	Mario
	<b>787643</b>	Neri	Piero
	<b>787643</b>	Bianchi	Luca

## Vincoli di integrità

---

- uno **schema relazionale** è costituito da un insieme di **schemi di relazione** e da un **insieme di vincoli** d'integrità sui possibili valori delle estensioni delle relazioni.
- Un **vincolo d'integrità** è una proprietà che deve essere soddisfatta dalle istanze che rappresentano informazioni corrette per l'applicazione.
- Un vincolo è espresso mediante una funzione booleana (un **predicato**): associa ad ogni istanza il valore **vero** o **falso**.

## Vincoli di integrità, perché?

---

- Descrizione più accurata della realtà
- Contributo alla "qualità dei dati"
- Utili nella progettazione (vedremo)
- Usati dai DBMS nella esecuzione delle interrogazioni
- Non tutte le proprietà di interesse sono rappresentabili per mezzo di vincoli formulabili in modo esplicito

## Tipi di vincoli

---

- vincoli intrarelazionali:
  - sono i vincoli che devono essere rispettati dai valori contenuti nella relazione considerata
  - vincoli su valori (o di dominio)
  - vincoli di ennupla
- vincoli interrelazionali:
  - sono i vincoli che devono essere rispettati da valori contenuti in relazioni diverse

## Vincoli di ennupla

---

- I **Vincoli di ennupla** esprimono condizioni sui valori di ciascuna ennupla, indipendentemente dalle altre ennupla
- Caso particolare:
  - **Vincoli di dominio**: coinvolgono un solo attributo

(Voto  $\geq$  18) AND (Voto  $\leq$  30)

(Voto = 30) OR NOT (Lode = "e lode")

Stipendi	Impiegato	Lordo	Ritenute	Netto
	Rossi	55.000	12.500	42.500
	Neri	45.000	10.000	35.000
	Bruni	47.000	11.000	36.000

$$\text{Lordo} = (\text{Ritenute} + \text{Netto})$$

## Identificazione delle ennuple

---

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

- non ci sono due ennuple con lo stesso valore sull'attributo Matricola
- non ci sono due ennuple uguali su tutti e tre gli attributi Cognome, Nome e Data di Nascita

## Chiave

---

- Informalmente:
  - Una **chiave** è un insieme di attributi che identificano le ennuple di una relazione
- 
- Formalmente:
  - Un insieme  $K$  di attributi è **superchiave** per  $r$  se  $r$  non contiene due ennuple (distinte)  $t_1$  e  $t_2$  con  $t_1[K] = t_2[K]$
  - $K$  è **chiave** per  $r$  se è una **superchiave minimale** per  $r$  (cioè non contiene un'altra superchiave)

## Una chiave

---

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

Matricola è una chiave:

- è superchiave
- è minimale (in questo caso contiene un solo attributo)

## Un'altra chiave

---

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

Cognome, Nome, Nascita è un'altra chiave:

- è superchiave?
- Minimale?

## Un'altra chiave?

---

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

- Non ci sono ennuple uguali su Cognome e Corso:
  - *Cognome e Corso formano una chiave?*
  - Ma è sempre vero?

## Vincoli, schemi e istanze

---

- I vincoli corrispondono a proprietà del mondo reale modellato dalla base di dati
- interessano a **livello di schema** (con riferimento cioè a tutte le istanze possibili)
- ad uno schema associamo un insieme di vincoli e consideriamo **corrette** ( valide, ammissibili) le istanze che soddisfano tutti i vincoli
- un'istanza può soddisfare altri vincoli ("per caso")

---

# Studenti

Matricola	Cognome	Nome	Corso	Nascita
-----------	---------	------	-------	---------

- chiavi:

Matricola

Cognome, Nome, Nascita

---

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

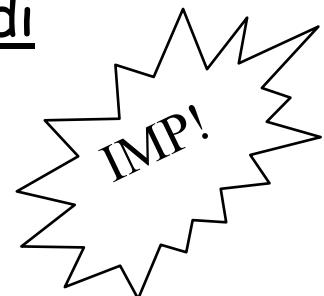
- È corretta: soddisfa i vincoli
- Ne soddisfa anche altri ("per caso"):
  - *Cognome, Corso è chiave*

## Esistenza delle chiavi

---

- Una relazione non può contenere ennuple distinte ma con valori uguali (una relazione è un sottoinsieme del prodotto cartesiano)
- Ogni relazione ha **sicuramente** come **superchiave** l'insieme di tutti gli attributi su cui è definita
- e quindi ogni relazione **ha (almeno) una chiave**
- l'esistenza delle chiavi garantisce l'accessibilità a ciascun dato della base di dati
- le chiavi permettono di correlare i dati in relazioni diverse:

il modello relazionale è basato su valori



## Chiavi e valori nulli

---

La presenza di valori nulli fra i valori di una chiave non permette

- di identificare le ennuple
- di realizzare facilmente i riferimenti da altre relazioni

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

## Chiave primaria

---

- Una **chiave primaria** è una chiave su cui non sono ammessi valori nulli
- Notazione: sottolineatura

<u>Matricola</u>	Cognome	Nome	Corso	Nascita
86765	NULL	Mario	Ing Inf	5/12/78
78763	Rossi	Mario	Ing Civile	3/11/76
65432	Neri	Piero	Ing Mecc	10/7/79
87654	Neri	Mario	Ing Inf	NULL
43289	Neri	Mario	NULL	5/12/78

## Integrità referenziale

---

- Nel modello relazionale le informazioni in relazioni diverse sono correlate attraverso valori comuni
- in particolare, vengono spesso presi in considerazione i valori delle chiavi (primarie).
- le correlazioni debbono essere "coerenti"

# Il modello relazionale

---

Studenti

Nome	<u>Matricola</u>	Provincia	AnnoNascita
Isaia	071523	PI	1982
Rossi	067459	LU	1984
Bianchi	079856	LI	1983
Bonini	075649	PI	1984

Chiave esterna

Vincolo di integrità referenziale

Esami

<u>Materia</u>	<u>Candidato*</u>	Data	Voto
BD	071523	12/01/06	28
BD	067459	15/09/06	30
FP	079856	25/10/06	30
BD	075649	27/06/06	25
LMM	071523	10/10/06	18

**studenti**

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

**esami**

Studente*	Voto	Corso*
	30	
	24	
	28	
	26	

**corsi**

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

# Infrazioni

---

<u>Codice</u>	Data	Vigile*	Stato	Numero
34321	1/2/17	3987	Fra	CB 123 AA
53524	4/3/18	3295	Ita	AB 222 CF
64521	5/4/17	3295	Ita	AX 424 DA
73321	5/2/19	9345	Fra	AX 424 DA

Vigili

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

## Vincolo di integrità referenziale

### Infrazioni

<u>Codice</u>	Data	Vigile	Stato*Numero*
34321	1/2/17	3987	Fra AX 424 DA
53524	4/3/18	3295	Ita AB 222 CF
64521	5/4/17	3295	Ita AX 424 DA
73321	5/2/19	9345	Fra AX 424 DA

Auto	<u>Stato</u>	<u>Numero</u>	Cognome	Nome
	Ita	AB 222 CF	Rossi	Mario
	Ita	AX 424 DA	Rossi	Mario
	Fra	AX 424 DA	Jacques	Dupont

## Vincolo di integrità referenziale

---

Un vincolo di **integrità referenziale** ("foreign key") fra gli attributi  $X$  di una relazione  $R_1$  e un'altra relazione  $R_2$  impone ai valori su  $X$  in  $R_1$  di comparire come valori della chiave primaria di  $R_2$

- Esempio: vincoli di integrità referenziale fra:
  - l'attributo Vigile della relazione INFRAZIONI e la relazione VIGILI
  - gli attributi Stato e Numero di INFRAZIONI e la relazione AUTO

## Violazione del vincolo di integrità referenziale

### Infrazioni

<u>Codice</u>	Data	Vigile	Stato*	Numero*
34321	1/2/17	3987	Fra	CB 123 AA
53524	4/3/18	3295	Fra	AB 222 CF
64521	5/4/17	3295	Ita	AX 424 DA
73321	5/2/19	9345	Fra	AX 424 DA

Auto	<u>Stato</u>	<u>Numero</u>	Cognome	Nome
	Ita	AB 222 CF	Rossi	Mario
	Ita	AX 424 DA	Rossi	Mario
	Fra	AX 424 DA	Jacques	Dupont

## Azioni compensative

---

- Esempio:
  - Viene eliminata una tupla dalla tabella riferita causando così una violazione
- Azioni
  - Rifiuto dell'operazione
  - Eliminazione in cascata
  - Introduzione di valori nulli

## Eliminazione in cascata

---

Impiegati

	<u>Matricola</u>	<u>Cognome</u>	<u>Progetto*</u>
	34321	Rossi	IDEA
	53524	Neri	XYZ
	64521	Verdi	NULL
	73032	Bianchi	IDEA

Progetti

	<u>Codice</u>	<u>Inizio</u>	<u>Durata</u>	<u>Costo</u>
	IDEA	01/2000	36	200
	XYZ	07/2001	24	120
	BOH	09/2001	24	150

## Introduzione di valori nulli

---

Impiegati

	<u>Matricola</u>	<u>Cognome</u>	<u>Progetto*</u>
	34321	Rossi	IDEA
	53524	Neri	NULL
	64521	Verdi	NULL
	73032	Bianchi	IDEA

Progetti

	<u>Codice</u>	<u>Inizio</u>	<u>Durata</u>	<u>Costo</u>
	IDEA	01/2000	36	200
	XYZ	07/2001	24	120
	BOH	09/2001	24	150

## Vincoli multipli su più attributi

Incidenti

	<u>Codice</u>	Data	StatoA*	NumeroA*	StatoB*	NumeroB*
	34321	1/2/95	Ita	AB 222 CF	Ita	AX 424 DA
	64521	5/4/96	Ita	AX 424 DA	Fra	CB 123 AA

Auto

	<u>Stato</u>	<u>Numero</u>	Cognome	Nome
	Ita	AX 424 DA	Bianchi	Giovanni
	Ita	AB 222 CF	Rossi	Mario
	Fra	CB 123 AA	Jacques	Dupont

- vincoli di integrità referenziale fra:

- La coppia di attributi StatoA e NumeroA di INCIDENTI e la relazione AUTO
- La coppia di attributi StatoB e NumeroB di INCIDENTI e la relazione AUTO

## Riepilogo: CHIAVI ED ASSOCIAZIONI

---

- Superchiave
- Chiave
- Chiave primaria

Esempio: (Matricola) e (Nome,Indirizzo) sono chiavi in:

- Studenti(Matricola: Int, Nome: String, Indirizzo: String)

- Chiave esterna
- Associazioni

## Esercizio

---

- Rappresentare per mezzo di una o più relazioni le informazioni per la gestione delle Prenotazioni Mediche dei Pazienti di uno Studio Medico Associato

## Esercizio

---

- Definire uno schema di basi di dati per organizzare le informazioni di una azienda che ha impiegati (ognuno con un codice fiscale, cognome, nome e data di nascita), filiali (con codice, sede e direttore (che è impiegato)). Ogni impiegato lavora presso una filiale.
- Indicare le chiavi e i vincoli di integrità referenziale dello schema. Mostrare un'istanza della base di dati e verificare che soddisfa i vincoli

## Esercizio

---

- Si considerino le informazioni per la gestione dei prestiti di una biblioteca personale.
- Il proprietario presta i libri ai propri amici, che indica con i loro nomi o soprannomi, e i cui numeri di telefono sono contenuti in una rubrica.
- I libri sono individuati attraverso i titoli.
- Quando si presta un libro si prende nota della data presunta di restituzione.
- Definire uno schema per rappresentare queste informazioni indicandone i vincoli.

## Esercizio

---

- Rappresentare con una o più relazioni le informazioni contenute nell'orario dei treni in partenza di una stazione ferroviaria: numero, orario, destinazione finale, categoria, fermate intermedie, di tutti i treni in partenza

## Esercizio

---

- Definire uno schema di base di dati che organizzi i dati necessari a generare la pagina dei programmi radiofonici di un quotidiano, con stazioni, ora e titoli dei programmi.
- Per ogni stazione sono memorizzati, oltre al nome anche la frequenza di trasmissione e la sede.

---

# TRASFORMAZIONE DI SCHEMI

## Obiettivo della progettazione logica

---

- Si tratta di "**tradurre**" lo schema concettuale in uno schema logico relazionale che rappresenti gli stessi dati in maniera **corretta ed efficiente**.
- Questo richiede una ristrutturazione del modello concettuale
- **Osservazione:**
  - Non si tratta di una pura e semplice traduzione. Infatti
    - alcuni costrutti dello schema concettuale non sono direttamente rappresentabili
    - Nel modello logico è necessario tenere conto delle prestazioni

## Dati di ingresso e uscita

---

- Ingresso:
  - schema concettuale
  - informazioni sul carico applicativo (dimensioni dei dati e caratteristiche delle operazioni)
  - modello logico
- Uscita:
  - schema logico
  - documentazione associata

## Progettazione logica relazionale

---

La trasformazione di uno schema ad oggetti in uno schema relazionale avviene eseguendo i seguenti passi:

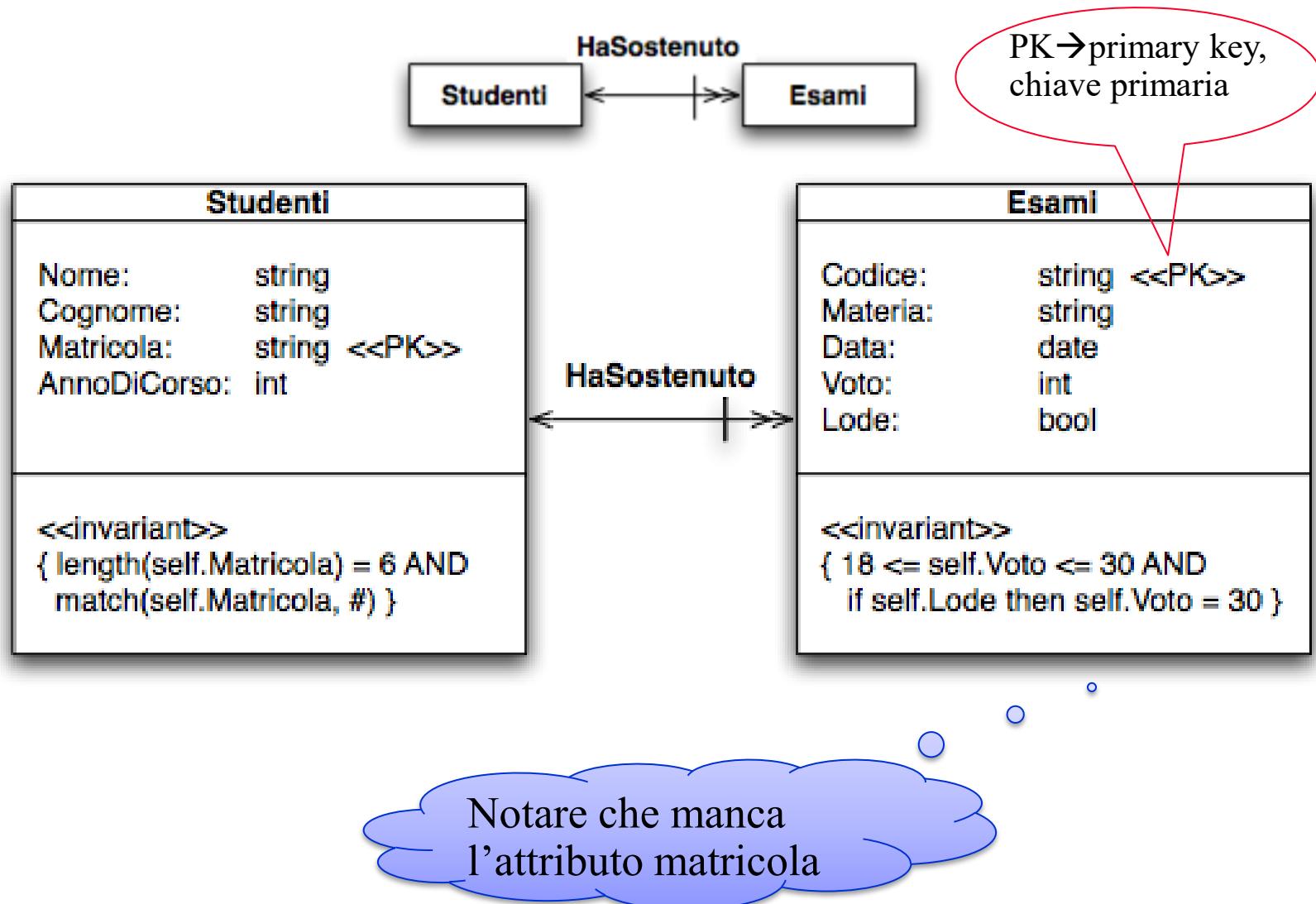
1. rappresentazione delle associazioni uno ad uno e uno a molti;
2. rappresentazione delle associazioni molti a molti o non binarie;
3. rappresentazione delle gerarchie di inclusione;
4. identificazione delle chiavi primarie;
5. rappresentazione degli attributi multivaleure;
6. appiattimento degli attributi composti.

## Obiettivo:

---

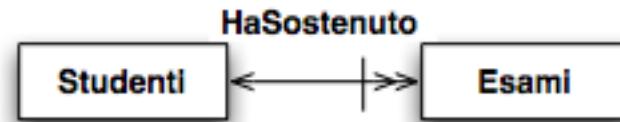
- rappresentare le stesse informazioni;
- minimizzare la ridondanza;
- produrre uno schema comprensibile, per facilitare la scrittura e manutenzione delle applicazioni.

## ESEMPIO - Schema concettuale



# ESEMPIO -

## Traduzione logica di un'associazione uno-a-molti



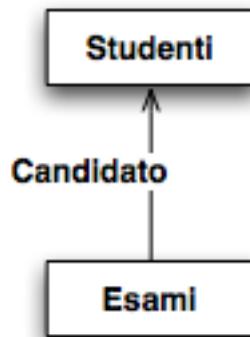
Schema:

`Studenti(Nome: string, Matricola: string, Provincia: string, AnnoNascita:int)`

`Esami(Materia: string, Candidato*: string, Data: string, Voto: int)`

Studenti

Relazioni:



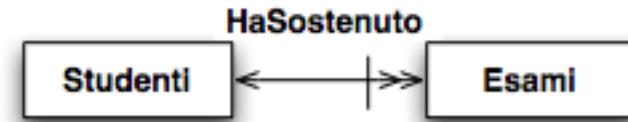
Chiave  
primaria  
Esami

Nome	Matricola	Provincia	AnnoNascita
Isaia	071523	PI	1982
Rossi	067459	LU	1984
Bianchi	079856	LI	1983
Bonini	075649	PI	1984

Materia	<u>Candidato*</u>	Data	Voto
BD	071523	12/01/06	28
BD	067459	15/09/06	30
FP	079856	25/10/06	30
BD	075649	27/06/06	25
LMM	071523	10/10/06	18

Attributo  
aggiunto

## ESEMPIO: ALTRE SOLUZIONI???



Studenti(Nome: string, Matricola: string, Provincia: string, AnnoNascita:int)

Esami(Numero :int, Materia: string, Candidato\*: string, Data: string, Voto: int)

Studenti(Nome: string, Matricola: string, Provincia: string, AnnoNascita:int,  
Esame\*:int)

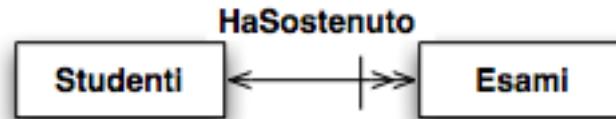
Esami(Numero :int, Materia: string, Data: string, Voto: int)



Esame e Numero corrispondono  
ad un ID dell'esame

Altra soluzione?

## ESEMPIO: ALTRE SOLUZIONI???



Studenti(Nome: string, Matricola: string, Provincia: string, AnnoNascita:int)  
Esami(Numero :int, Materia: string, Candidato\*: string, Data: string, Voto: int)

Studenti(Nome: string, Matricola: string, Provincia: string, AnnoNascita:int,  
Esame\*:int)

Esami(Numero :int, Materia: string, Data: string, Voto: int)

Studenti(Nome: string, Matricola: string, Provincia: string, AnnoNascita:int)  
Esami(Numero :int, Materia: string, Data: string, Voto: int)  
StudentiEsami(Esame\*: int, Candidato\*: string)

- Quale preferire?

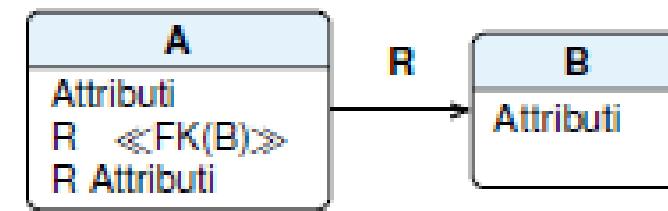
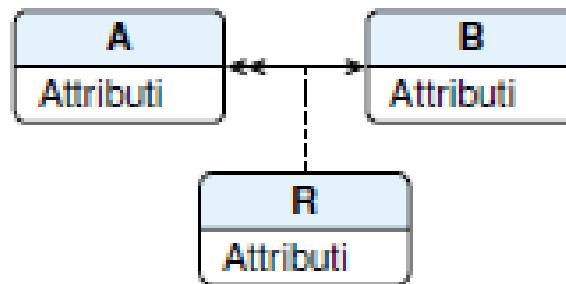
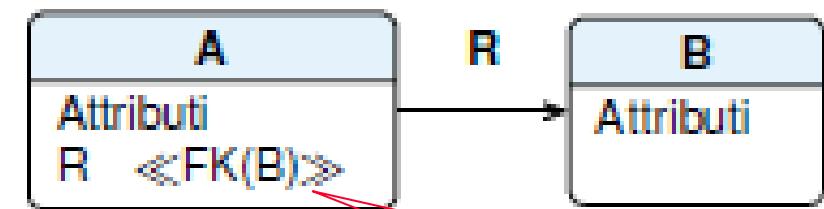
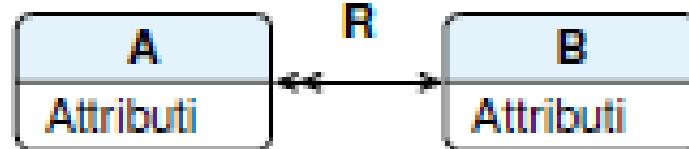
Il modello relazionale

Esame e Numero corrispondono  
ad un ID dell'esame (non al  
codice della materia)

Altra  
soluzione?

## Rappresentazione delle associazioni uno a molti

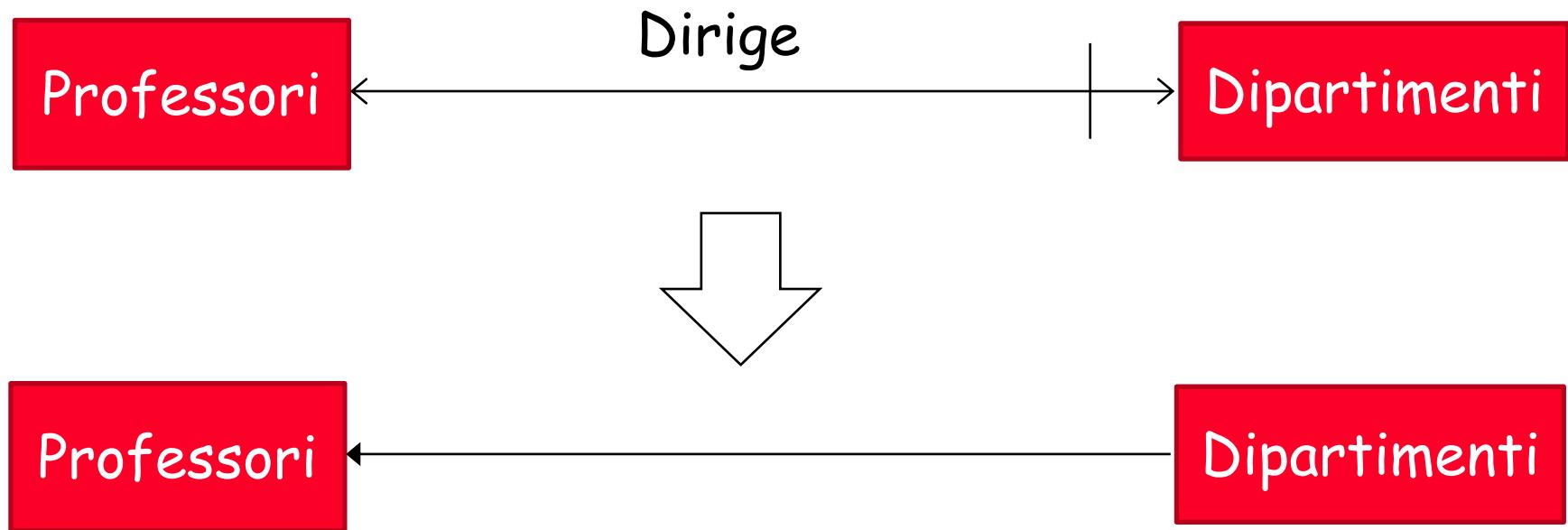
- le associazioni uno a molti si rappresentano aggiungendo agli attributi della relazione rispetto a cui l'associazione è univoca una chiave esterna che riferisce l'altra relazione.



## Esempi - Associazione 1 a 1

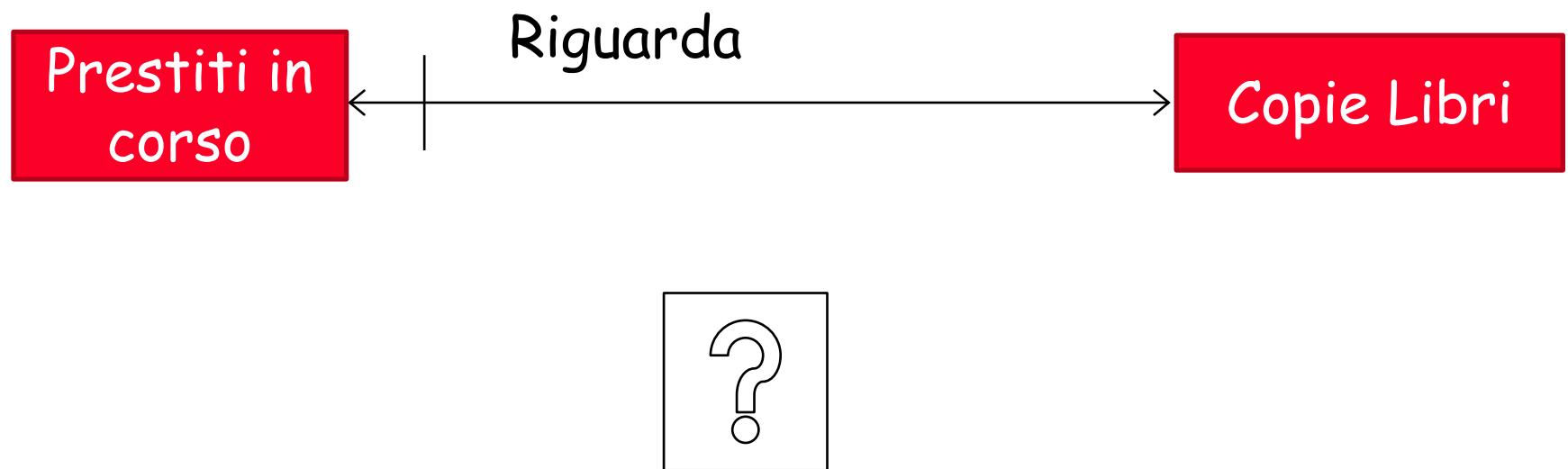
---

- Dirige(Professori, Dipartimenti) ha cardinalità (1:1):
  - Un professore può o non può dirigere un solo dipartimento
  - Un dipartimento deve avere un (solo) professore come dirigente



## Esempi

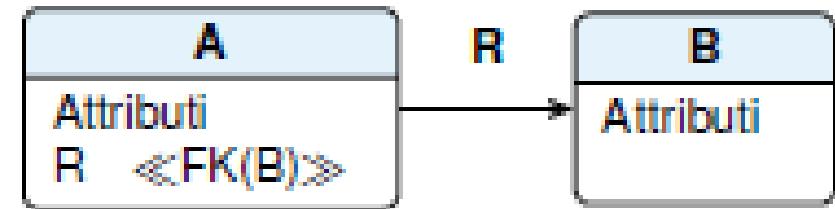
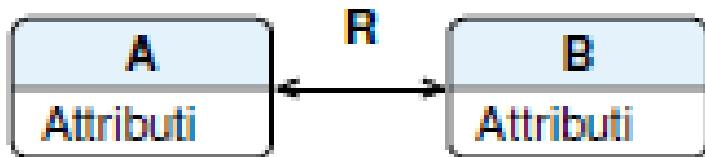
---



## Rappresentazione delle associazioni uno ad uno

---

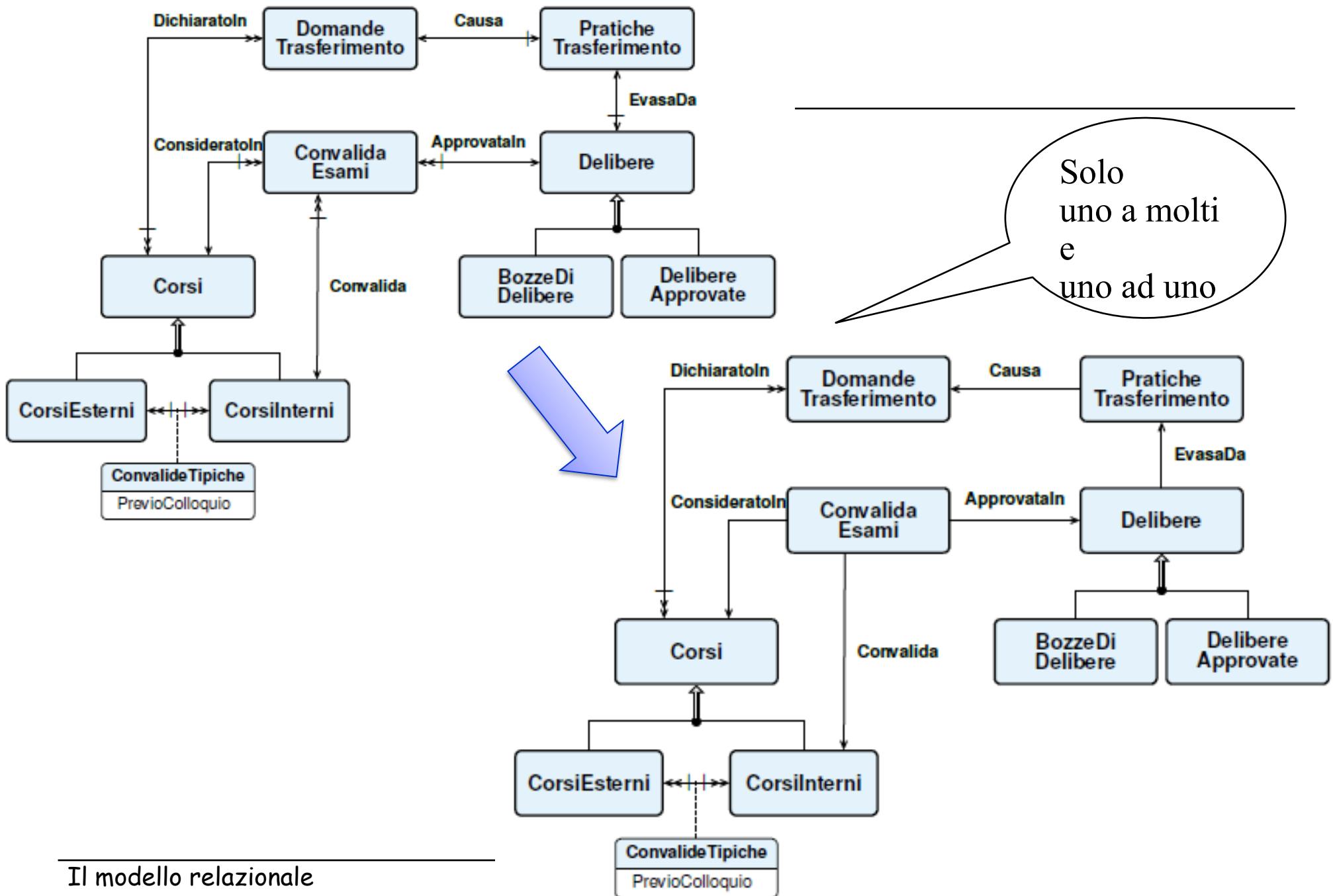
- le associazioni uno a uno si rappresentano aggiungendo la chiave esterna ad una qualunque delle due relazioni che riferisce l'altra relazione, preferendo quella rispetto a cui l'associazione è totale, nel caso in cui esista un vincolo di totalità



## Vincoli sulla cardinalità delle associazioni uno a molti e uno ad uno

---

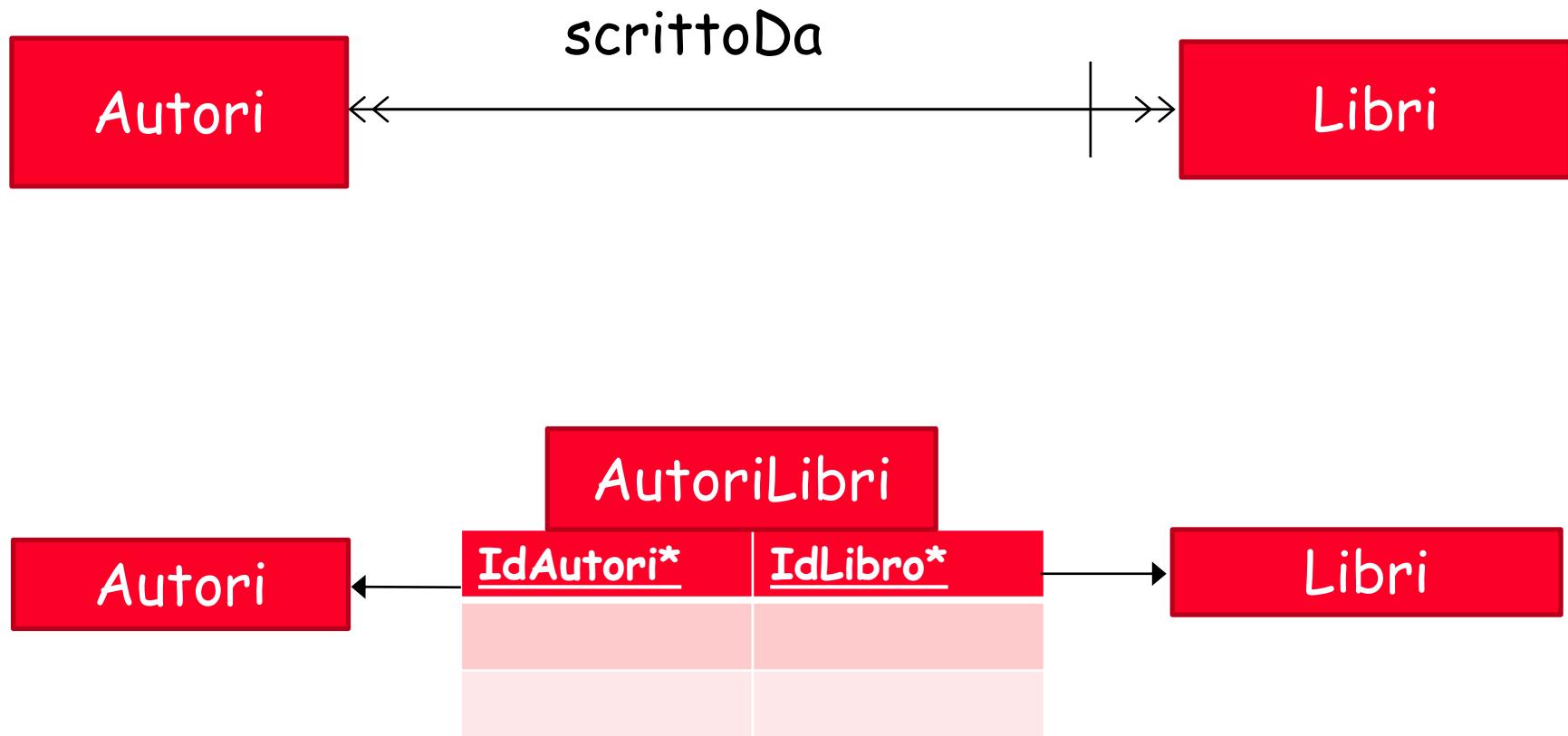
- La direzione dell'associazione rappresentata dalla chiave esterna è detta "**la diretta**" dell'associazione.
- Vincoli sulla cardinalità delle associazioni uno a molti ed uno ad uno:
  - **univocità della diretta**;
  - **totalità della diretta**: si rappresenta imponendo un vincolo **not null** sulla chiave esterna;
  - **univocità dell'inversa e totalità della diretta**: si rappresenta imponendo un vincolo not null ed un vincolo di chiave sulla chiave esterna.



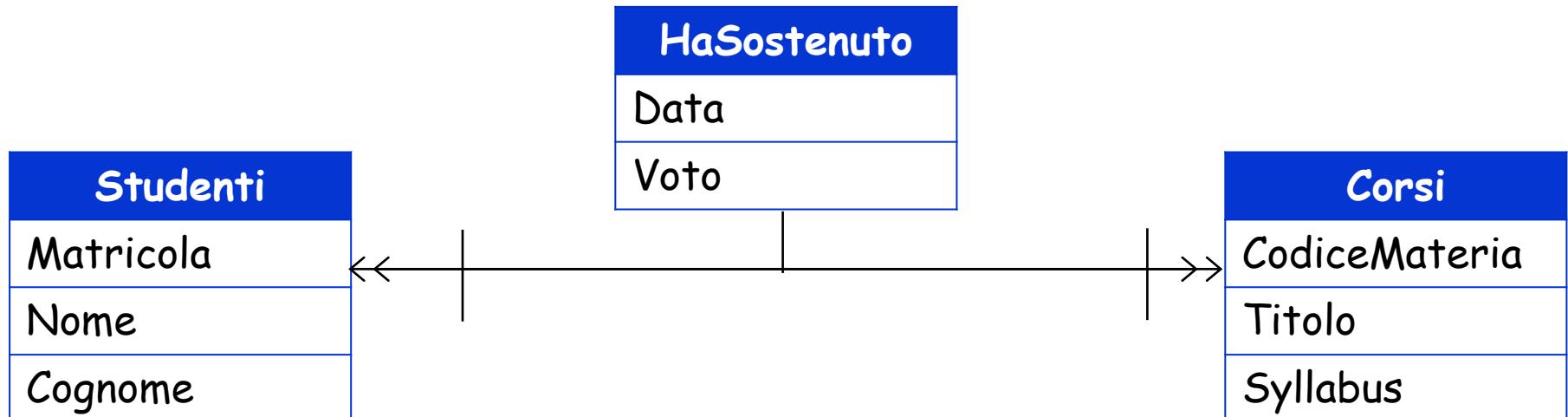
Il modello relazionale

## Esempio - Associazione molti a molti

---



## Esempio - Associazione molti a molti

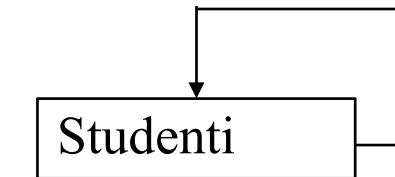


## ESEMPIO - Traduzione logico (ricorsione)

Schema:

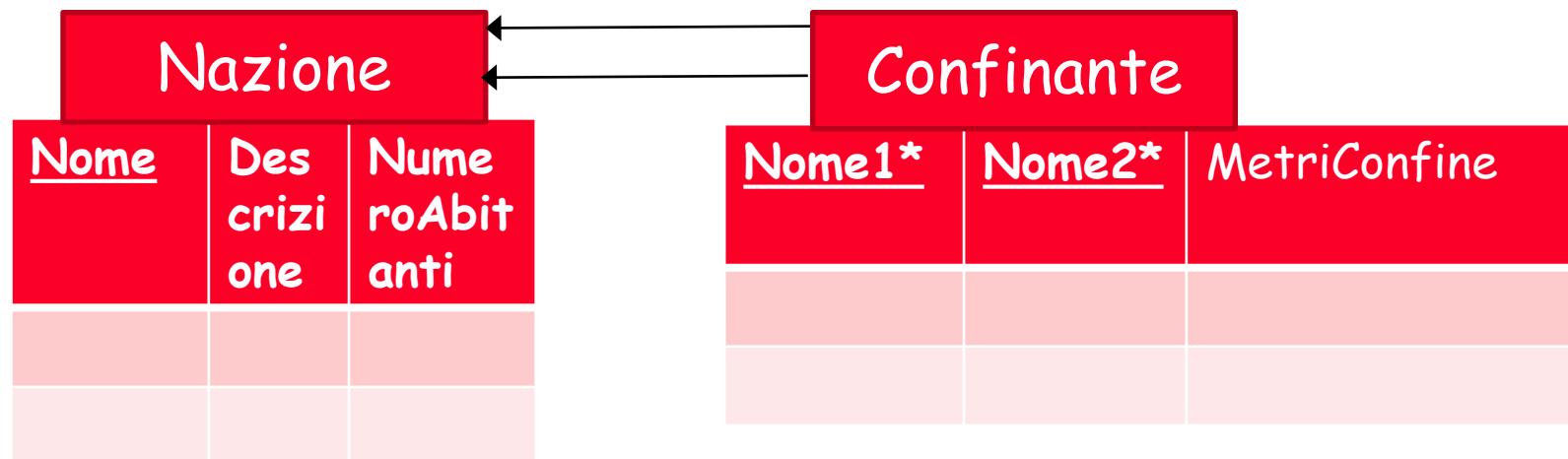
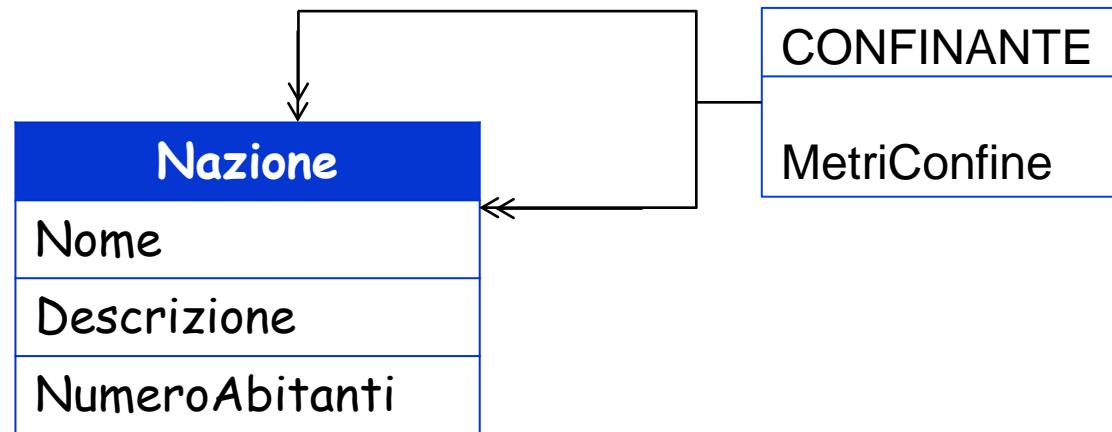
Studenti(Nome: string, Matricola: string, Provincia: string, AnnoNascita:int, ,  
TutorStudente\*: string)

Studenti



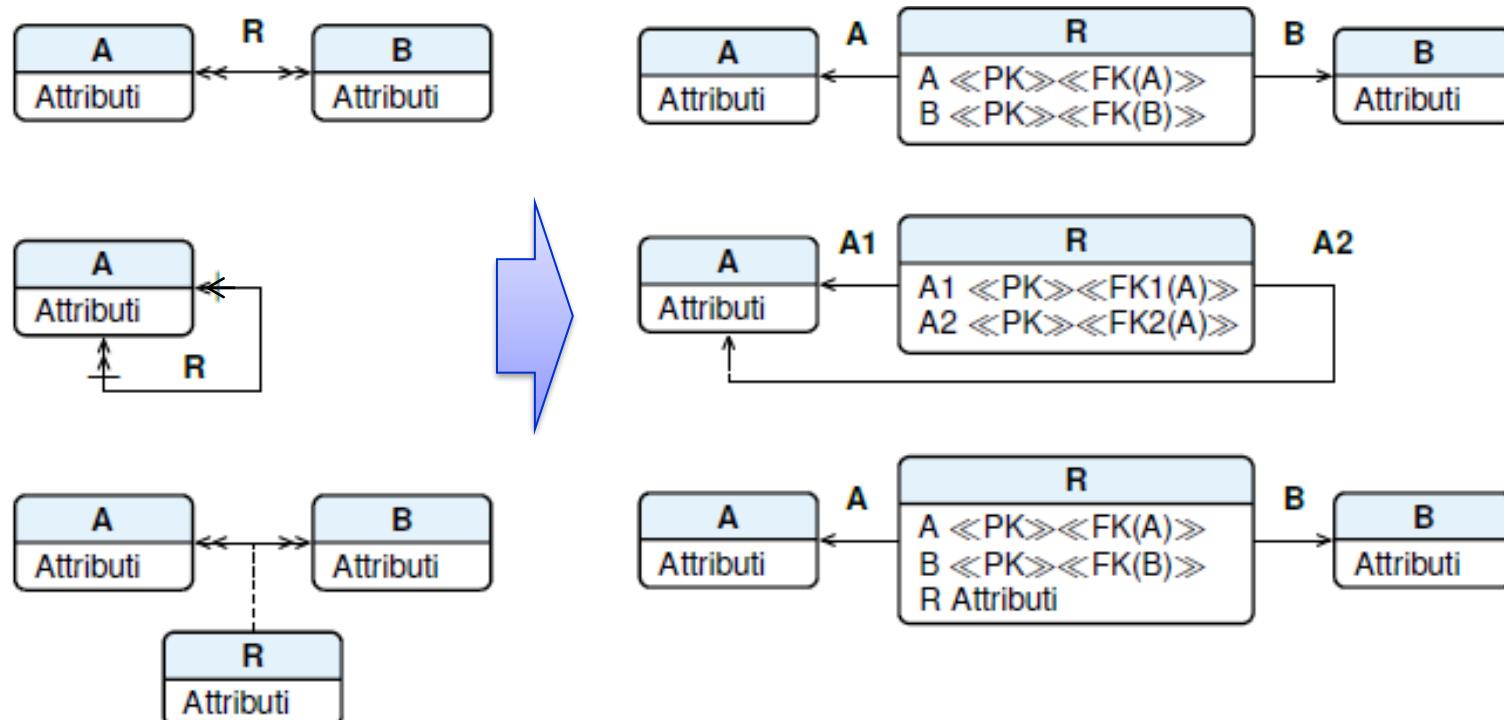
Nome	<u>Matricola</u>	Provincia	AnnoNascita	<u>MatrTutor*</u>
Isaia	071523	PI	1982	067459
Rossi	067459	LU	1984	071523
Bianchi	079856	LI	1983	071523
Bonini	075649	PI	1984	071523

## Esempio - Associazione molti a molti (ricorsione)

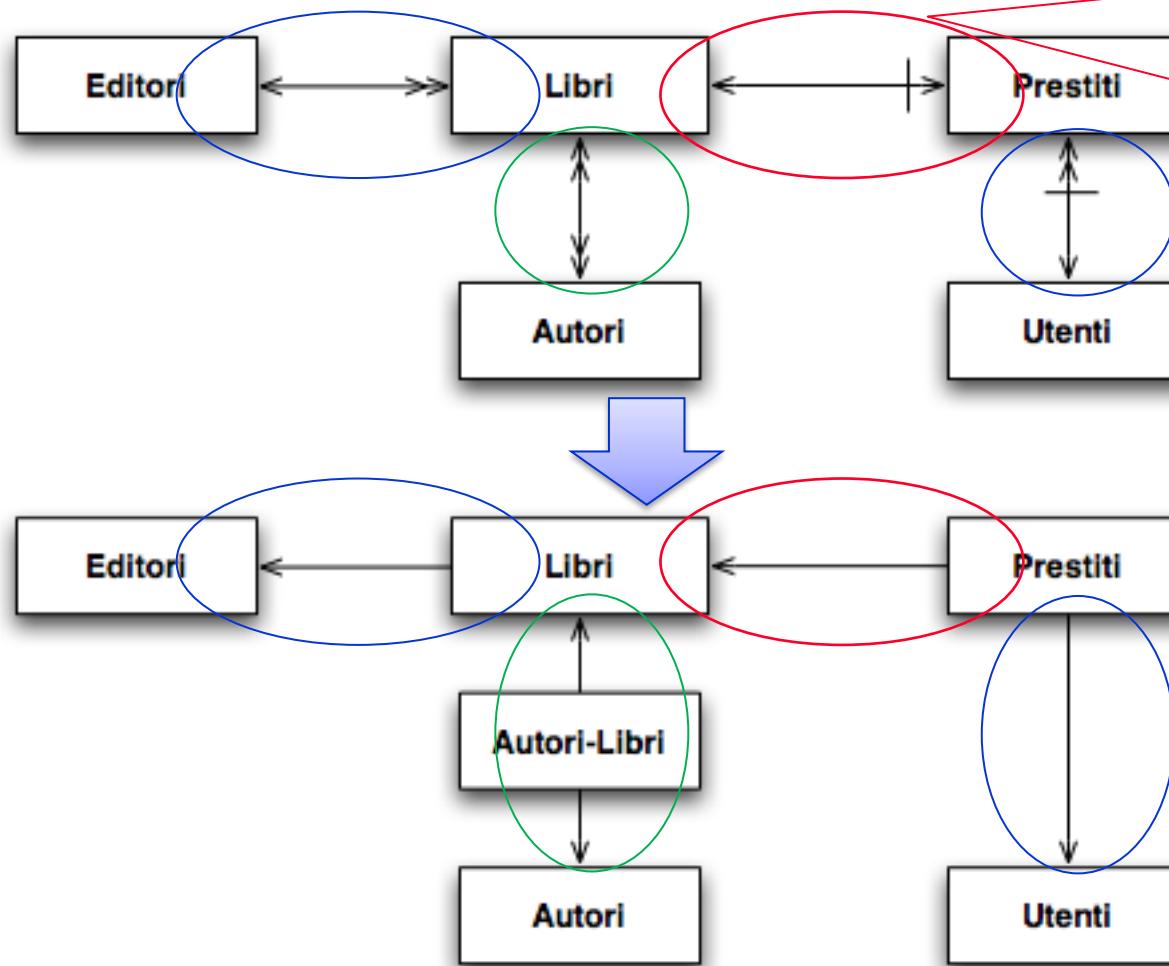


## Rappresentazione delle associazioni molti a molti

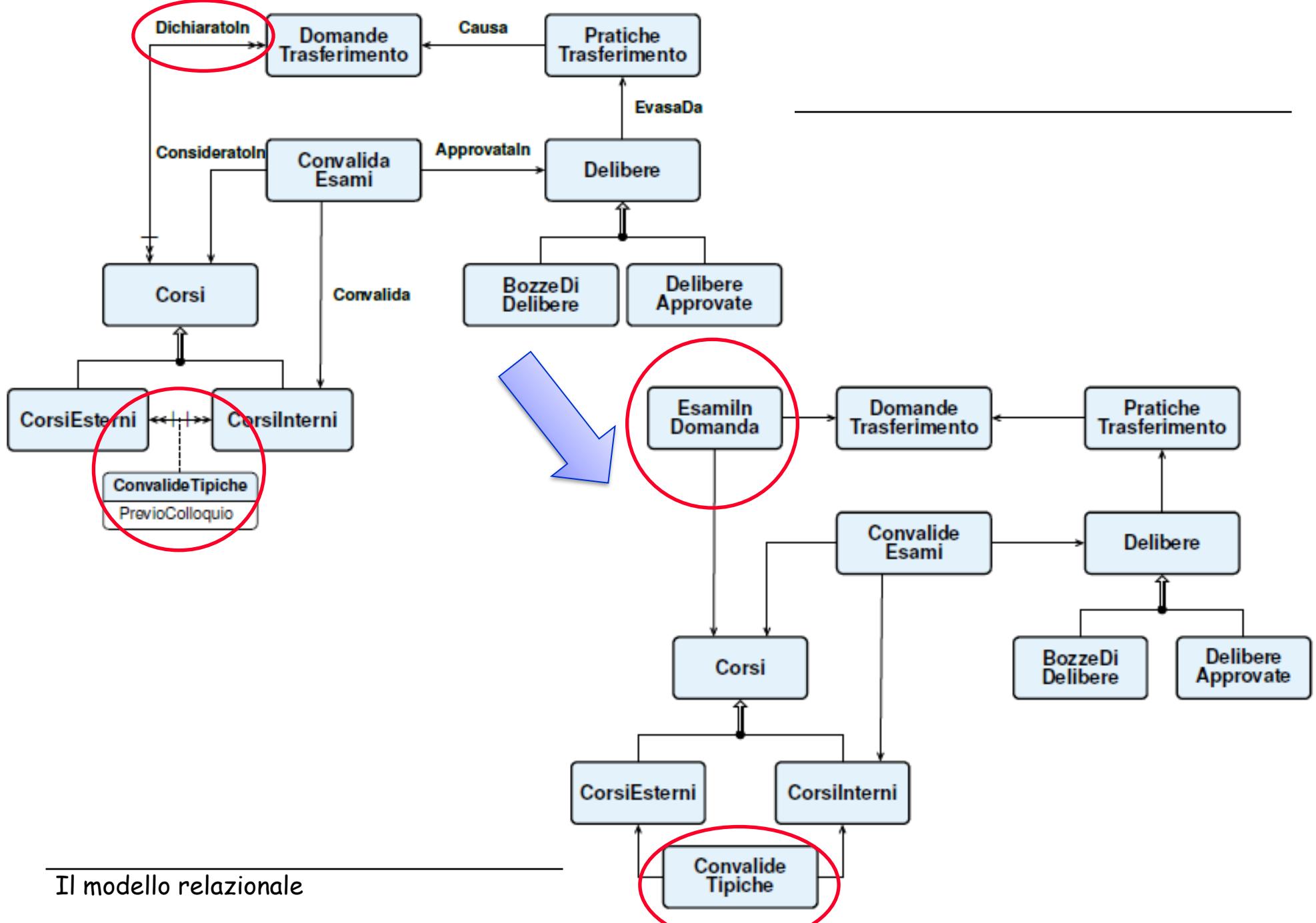
- Un'associazione molti a molti tra due classi si rappresenta aggiungendo allo schema una nuova relazione che contiene due chiavi esterne che riferiscono le due relazioni coinvolte; la chiave primaria di questa relazione è costituita dall'insieme di tutti i suoi attributi.



# TRASFORMAZIONE DI SCHEMI A OGGETTI IN RELAZIONALI



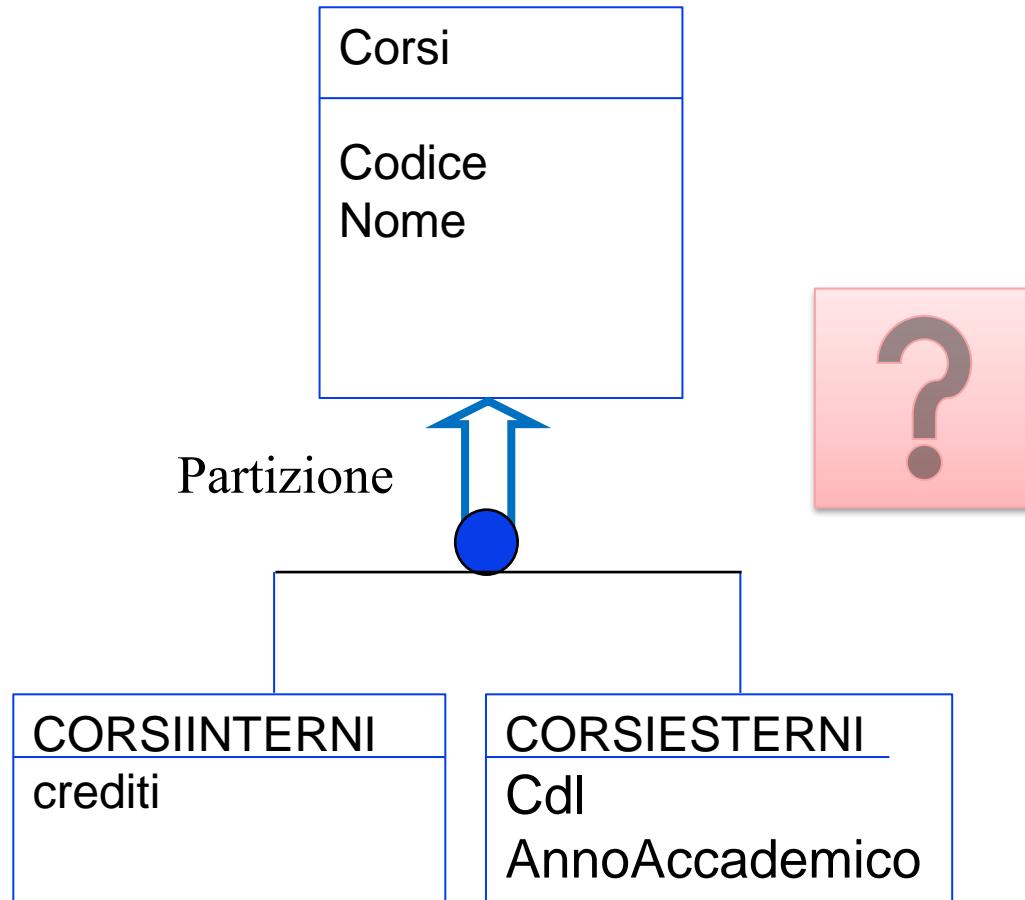
Si preferisce la direzione totale: ogni prestito ha un libro, ma non tutti i libri hanno un prestito



Il modello relazionale

## Traduzione delle gerarchie

---



## Rappresentazione delle gerarchie fra classi

---

- il modello relazionale non può rappresentare direttamente le gerarchie
- Bisogna eliminare le gerarchie, sostituendole con classi e relazioni:
  1. accorpamento delle figlie della gerarchia nel genitore  
**(relazione unica)**
  2. accorpamento del genitore della gerarchia nelle figlie  
**(partizionamento orizzontale)**
  3. sostituzione della gerarchia con relazioni  
**(partizionamento verticale)**

## Accorpamento delle figlie della gerarchia nel genitore (relazione unica)

---

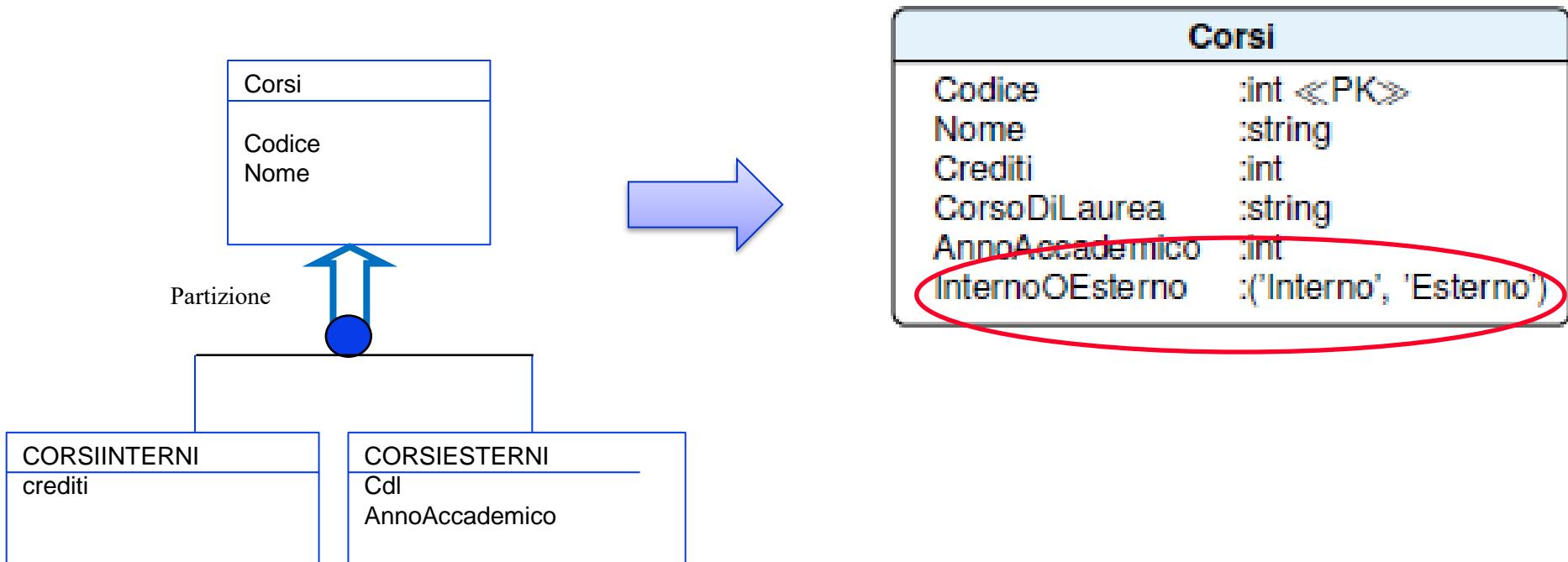
Se  $A_0$  è la classe genitore di  $A_1$  ed  $A_2$ , le classi  $A_1$  e  $A_2$  vengono eliminate ed accorpate ad  $A_0$

Ad  $A_0$  viene aggiunto un **attributo (Discriminatore)** che indica da quale delle classi figlie deriva una certa istanza, e gli **attributi di  $A_1$  ed  $A_2$**  vengono assorbiti dalla classe genitore, e assumono valore nullo sulle istanze provenienti dall'altra classe.

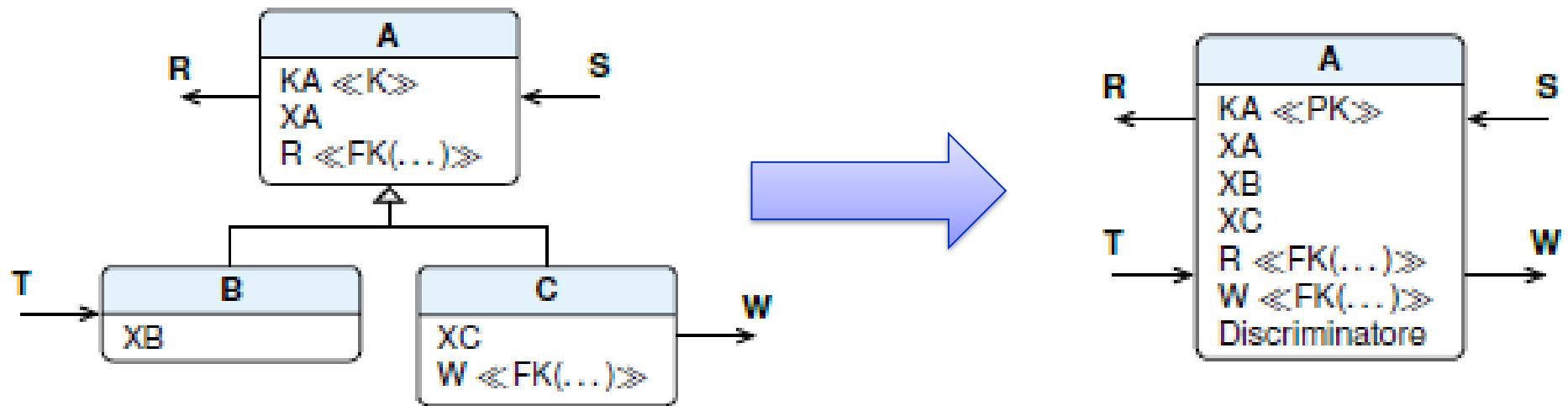
Infine, una **relazione relativa a solo una delle classi figlie** viene acquisita dalla classe genitore e avrà comunque cardinalità minima uguale a 0, in quanto gli elementi dell'altra classe non contribuiscono alla relazione.

## Accorpamento delle figlie della gerarchia nel genitore (relazione unica)

- Classe Corsi con due attributi Codice (la chiave), Nome e con due sottoclassi di tipo partizione:
  - CorsiInterni, con un attributo Crediti,
  - CorsiEsterni, con due attributi CorsoDiLaurea, AnnoAccademico.



## Accorpamento delle figlie della gerarchia nel genitore (relazione unica)



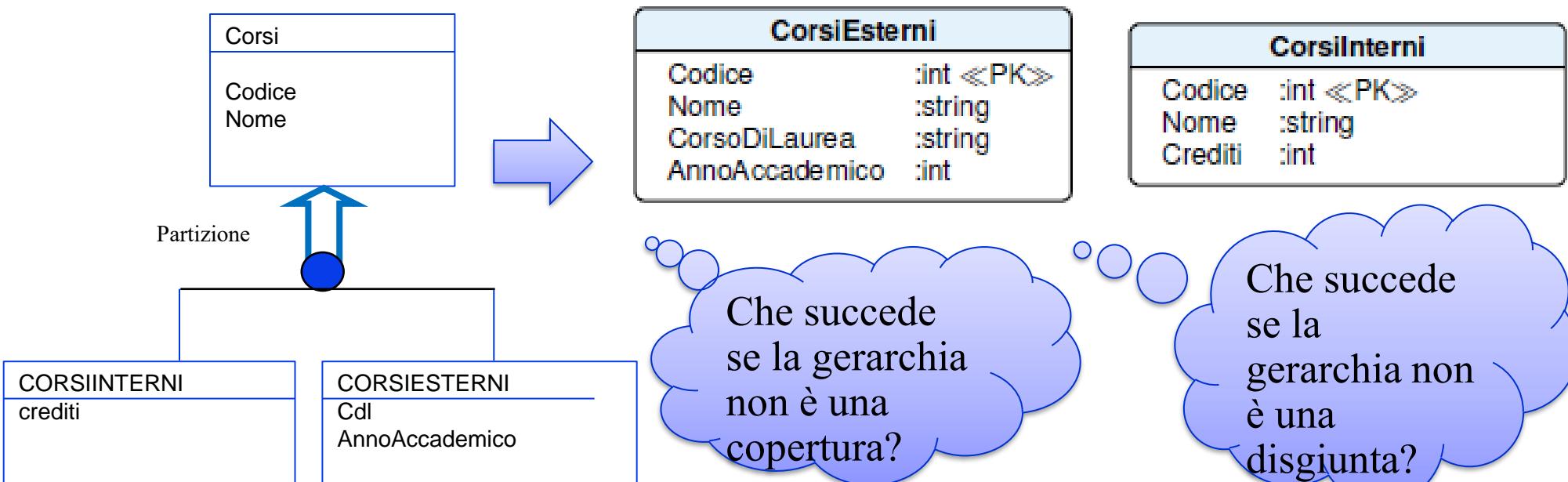
## Accorpamento del genitore della gerarchia nelle figlie **(partizionamento orizzontale)**

---

- La classe genitore  $A_0$  viene eliminata, e le classi figlie  $A_1$  ed  $A_2$  ereditano le proprietà (attributi, identificatore e relazioni) dell'classe genitore

## Accorpamento del genitore della gerarchia nelle figlie (partizionamento orizzontale)

- Classe Corsi con due attributi Codice (la chiave), Nome e con due sottoclassi di tipo partizione:
  - CorsiInterni, con un attributo Crediti,
  - CorsiEsterne, con due attributi CorsoDiLaurea, AnnoAccademico.

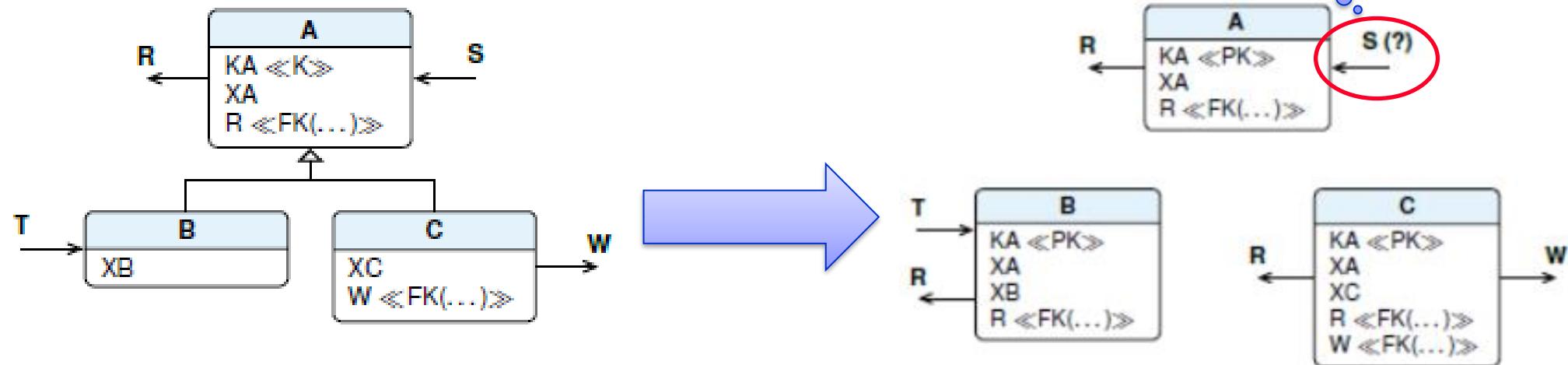


## sostituzione della gerarchia con relazioni

### Accorpamento del genitore della gerarchia nelle figlie (partizionamento orizzontale)

- il partizionamento orizzontale divide gli elementi della superclasse in più relazioni diverse, per cui non è possibile mantenere un vincolo referenziale verso la superclasse stessa; in conclusione, questa tecnica non si usa se nello schema relazionale grafico c'è una freccia che entra nella superclasse:

Che succede se esiste una chiave esterna verso la tabella A?



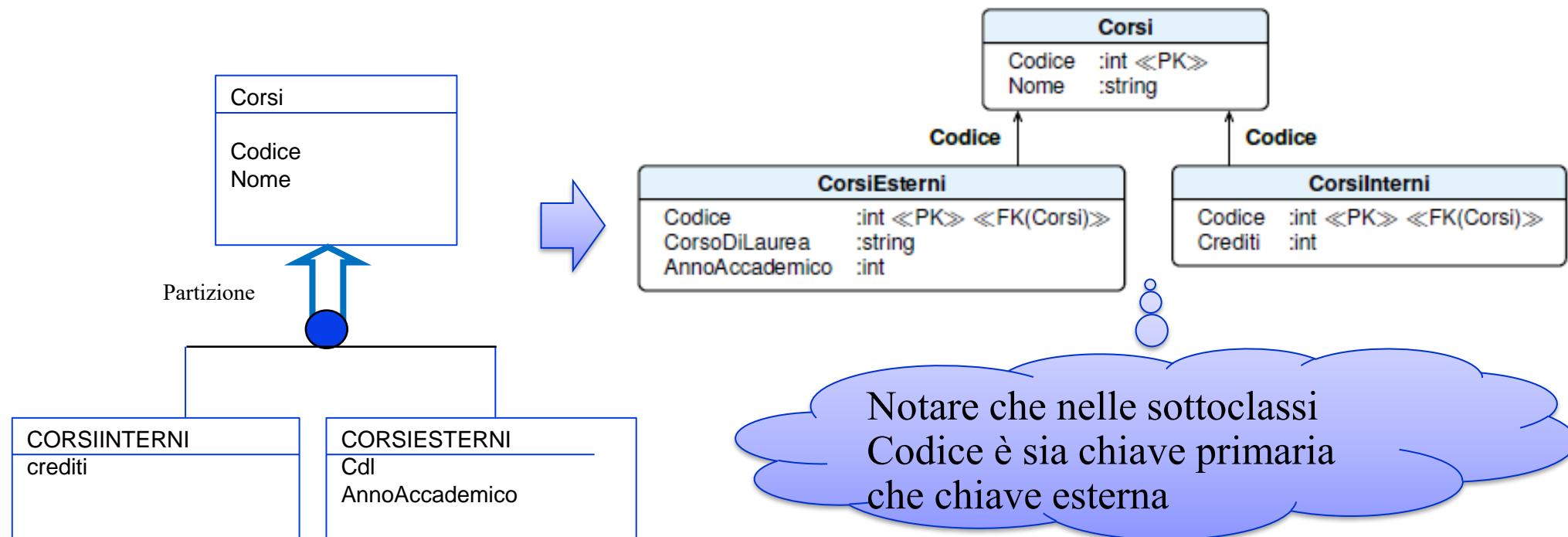
## sostituzione della gerarchia con relazioni **(partizionamento verticale)**

---

- La gerarchia si trasforma in due associazioni uno a uno che legano rispettivamente la classe genitore con le classi figlie.
- In questo caso non c'è un trasferimento di attributi o di associazioni e le classi figlie  $A_1$  ed  $A_2$  sono identificate esternamente dalla classe genitore  $A_0$ .
- Nello schema ottenuto vanno aggiunti dei vincoli: ogni occorrenza di  $A_0$  non può partecipare contemporaneamente alle due associazioni, e se la gerarchia è totale, deve partecipare ad almeno una delle due

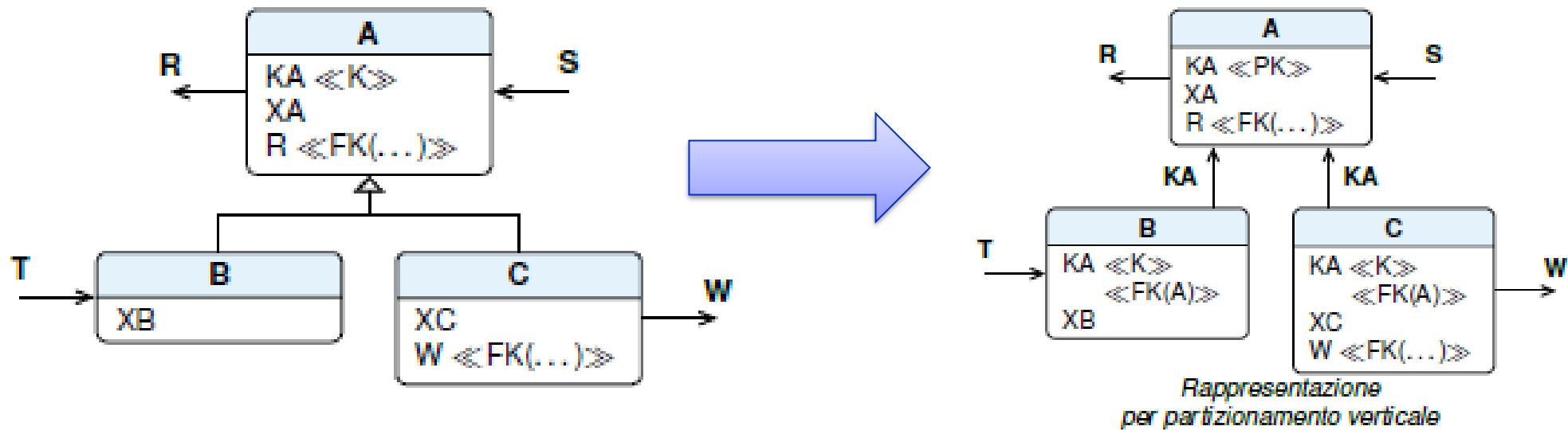
## sostituzione della gerarchia con relazioni (partizionamento verticale)

- Classe Corsi con due attributi Codice (la chiave), Nome e con due sottoclassi di tipo partizione:
  - CorsiInterni, con un attributo Crediti,
  - CorsiEsterne, con due attributi CorsoDiLaurea, AnnoAccademico.



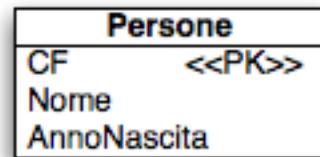
## sostituzione della gerarchia con relazioni **(partizionamento verticale)**

---

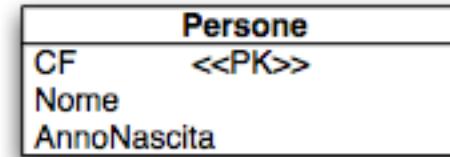


# Riepilogo: LE SOTTOCLASSI

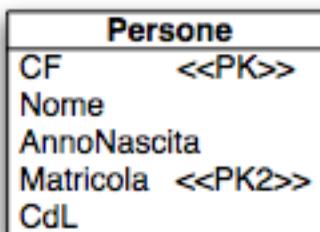
---



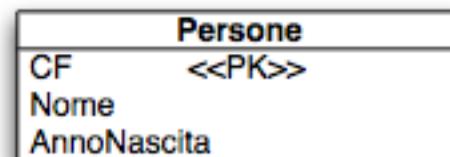
Diventa



Oppure



Oppure



## Esempio - Campo-multivalore

---

- Gestione persone con «più» indirizzi email

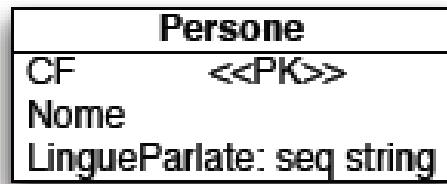
<b>CodiceFiscale</b>	<b>Email_1</b>	<b>Email_2</b>	<b>Email_3</b>	...
RSSMRT	RSS@	MRT@	RMT@	
BNCGNN	BNC@			



<b>CodiceFiscale</b>	<b>IndirizziEmail</b>
RSSMRT	RSS@
RSSMRT	MRT@
RSSMRT	RMT@
BNCGNN	BNC@

# GLI ATTRIBUTI MULTIVALORE

---



Diventa

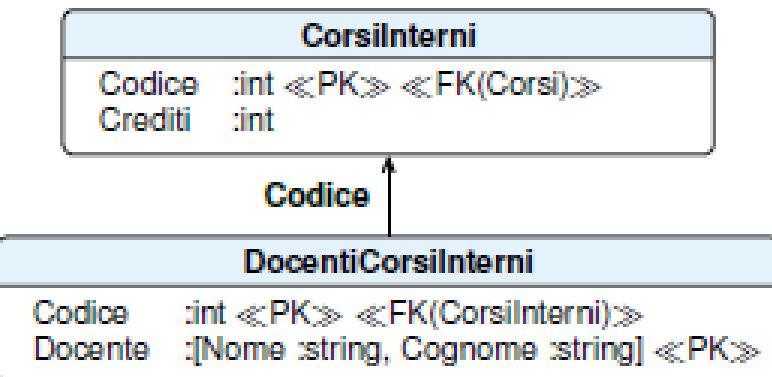
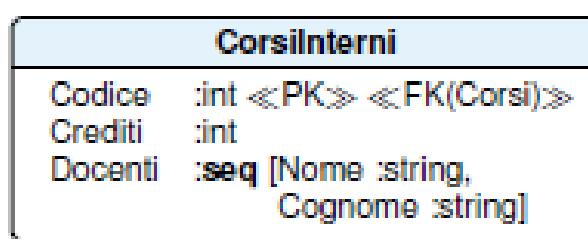


Oppure



# Rappresentazione delle proprietà multivalue

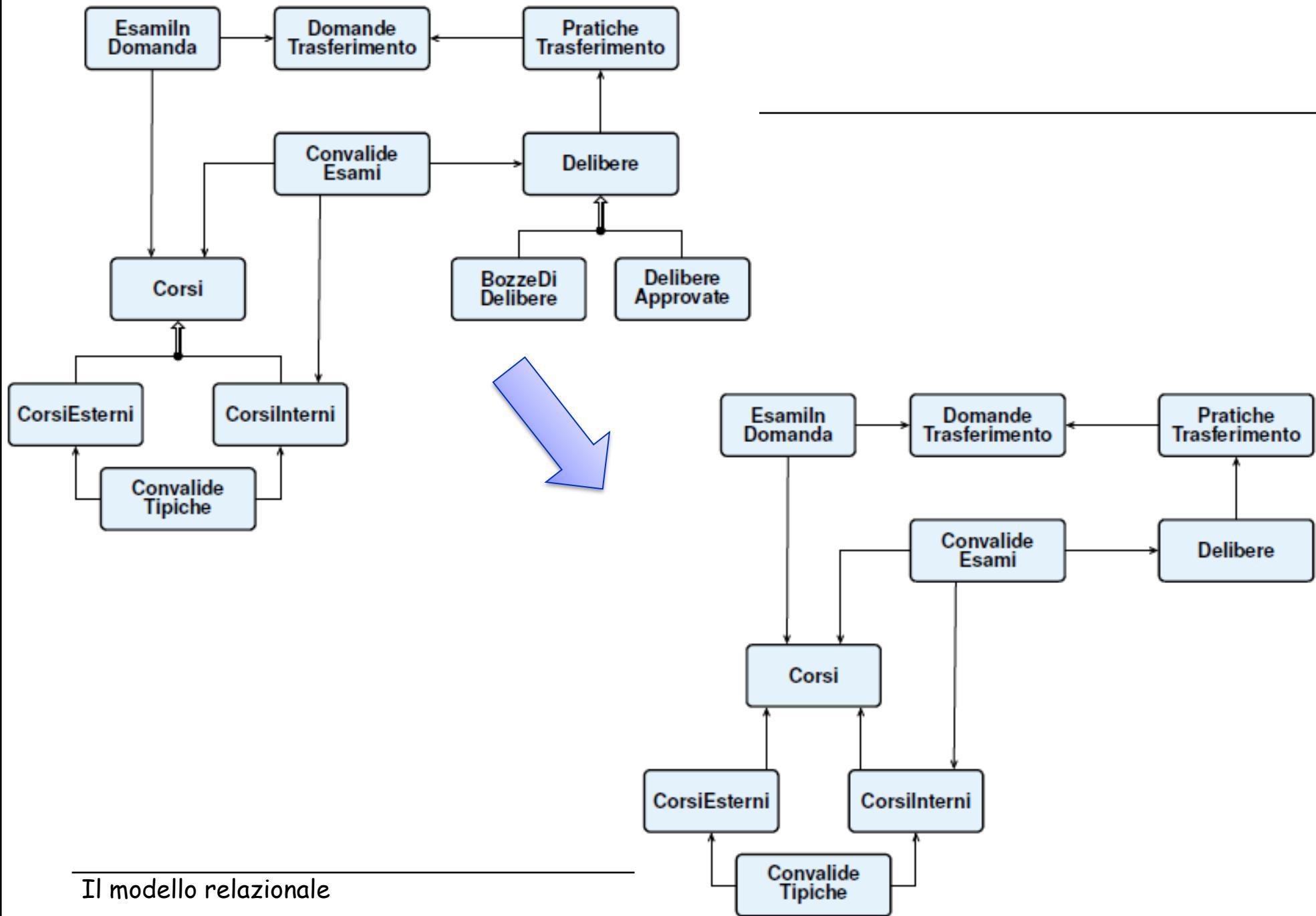
---

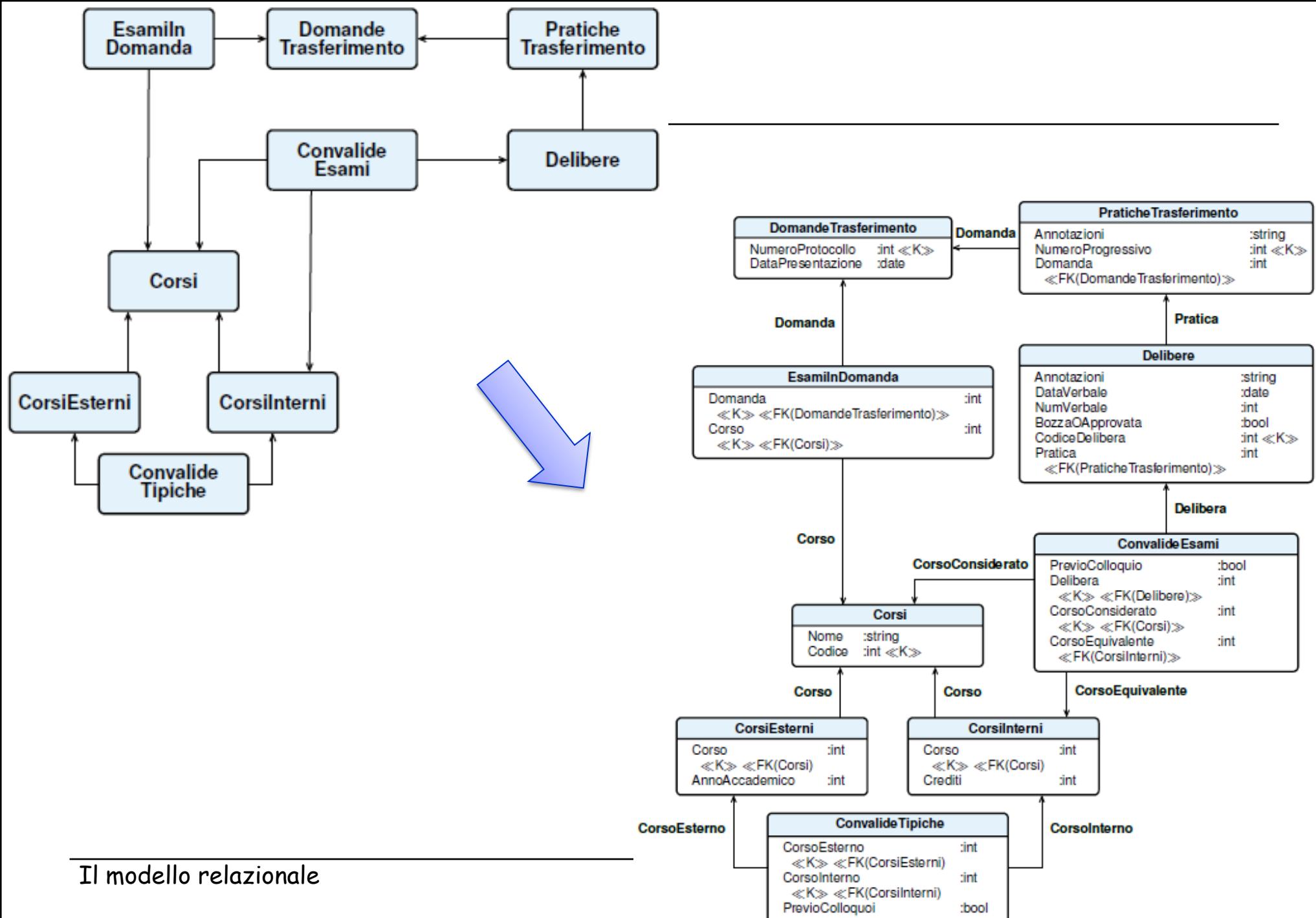


# Appiattimento degli attributi composti

---

DocentiCorsiInterni	
Codice	:int <<PK>> <<FK(CorsiInterni)>>
Nome	:string <<PK>>
Cognome	:string <<PK>>



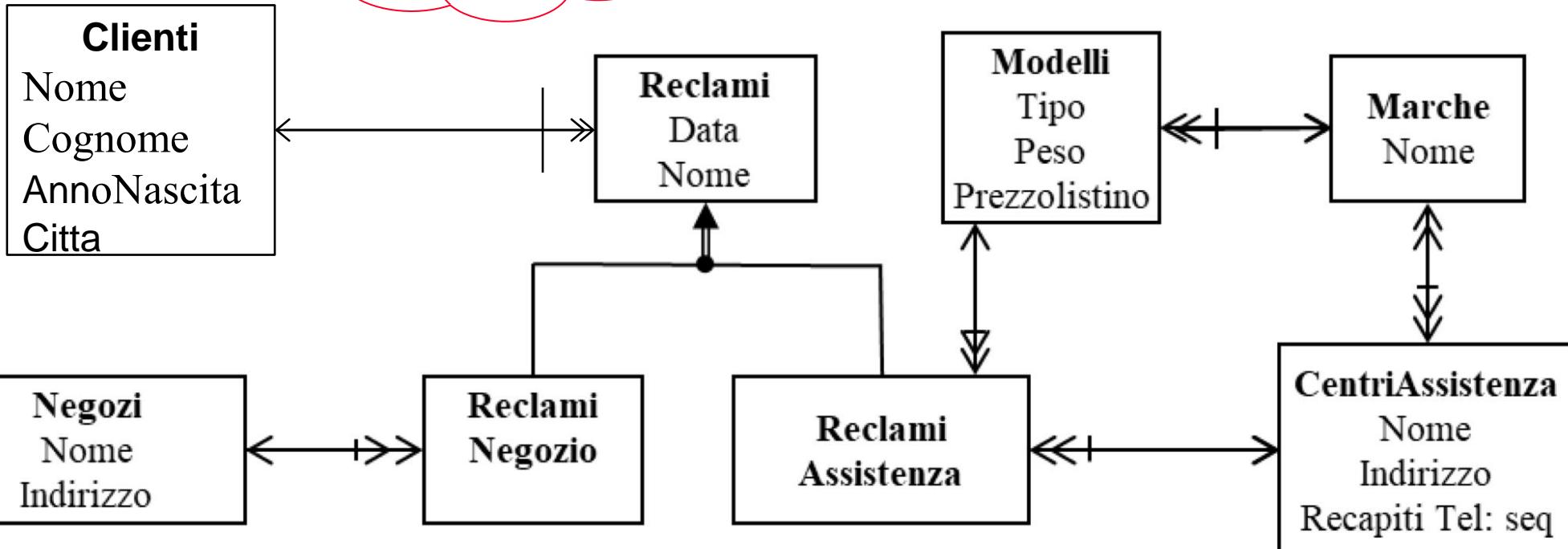


Il modello relazionale

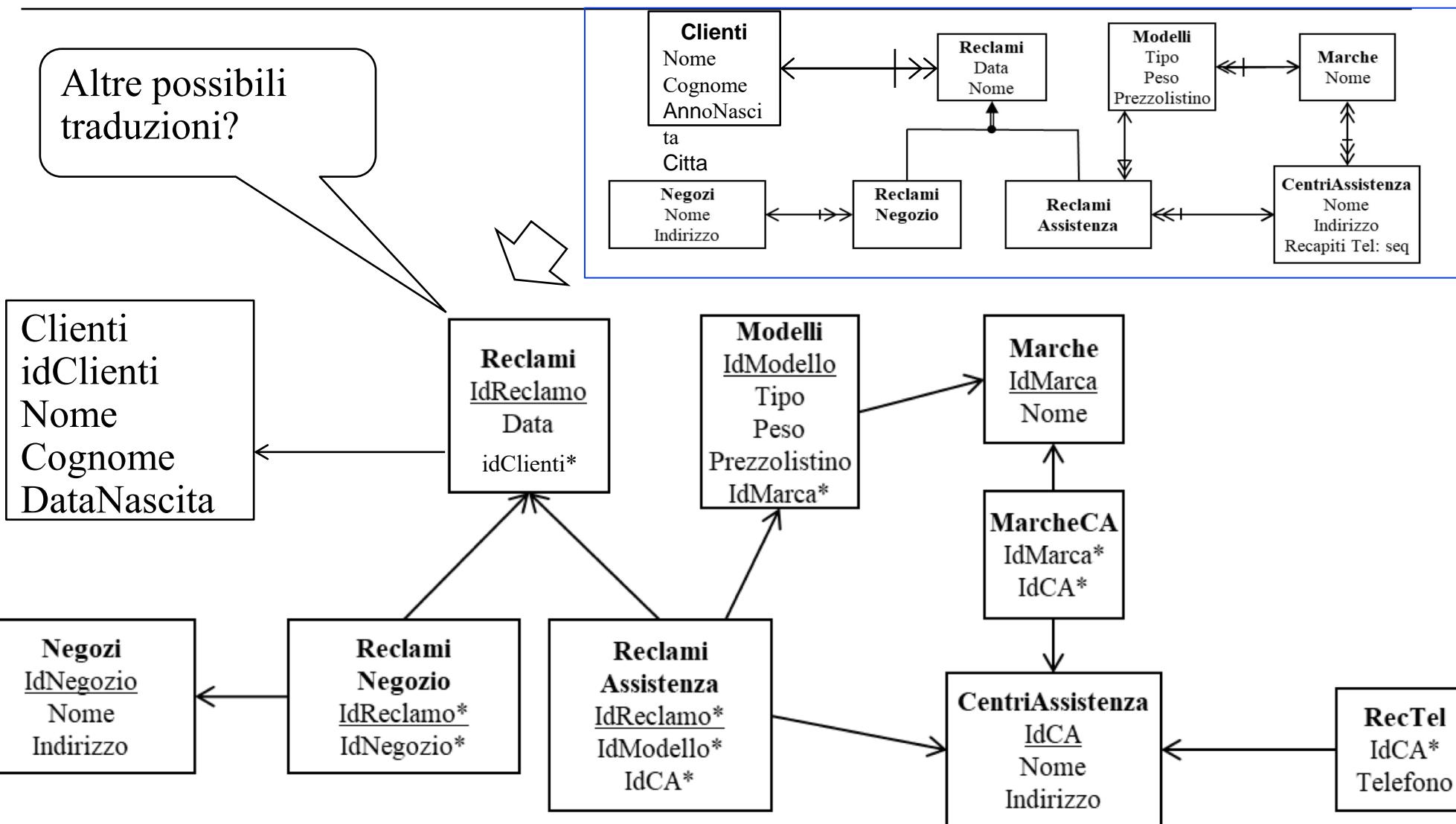
# Primo compitino di Basi di Dati - 3/4/2019 - VARIANTE

- Variante: i reclami possono essere redatti da clienti registrati nella base di dati di cui interessano nome, cognome, anno nascita, città

E lo schema logico?



# Schema logico - Primo compitino di Basi di Dati - 3/4/2019



---

# ALGEBRA RELAZIONALE

Alcuni esempi sono adattati dal libro Albano et al e  
altri da Atzeni-et al., Basi di dati

# Linguaggi per basi di dati

---

operazioni sullo schema

**DDL: data definition language**

Operazioni di creazione, cancellazione e modifica di schemi di tabelle, creazione viste, creazione indici...

operazioni sui dati

**DML: data manipulation language**

- **Data Query language** - Query o interrogazione della base di dati
- **Aggiornamento dati** - Inserimento, cancellazione e modifica di dati

## LINGUAGGI RELAZIONALI

---

- *Algebra relazionale*: insieme di operatori su relazioni che danno come risultato relazioni.
  - Non si usa come linguaggio di interrogazione dei DBMS ma come rappresentazione interna delle interrogazioni.
- *Calcolo relazionale*: linguaggio dichiarativo di tipo logico dal quale è stato derivato l'SQL.

# Operatori dell'algebra relazionale

---

- unione, intersezione, differenza
- ridenominazione
- selezione
- proiezione
- join (join naturale, prodotto cartesiano, theta-join)

## Operatori insiemistici

---

- le relazioni sono insiemi
- i risultati devono essere relazioni
- è possibile applicare **unione, intersezione, differenza** solo a relazioni definite sugli **stessi attributi (nome e tipo)**, cioè possono operare solo su tuple uniformi.

# ALGEBRA RELAZIONALE: Unione

Laureati

Matricola	Nome	Età
7274	Rossi	42
7432	Neri	54
9824	Verdi	45

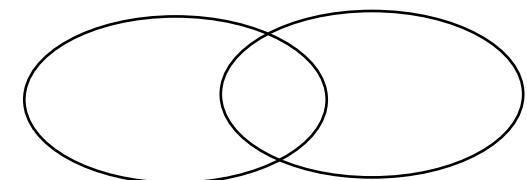
Quadri

Matricola	Nome	Età
9297	Neri	33
7432	Neri	54
9824	Verdi	45

Nota campi con lo stesso nome e stesso tipo

$\text{Laureati} \cup \text{Quadri}$

Matricola	Nome	Età
7274	Rossi	42
7432	Neri	54
9824	Verdi	45
9297	Neri	33



# ALGEBRA RELAZIONALE: Differenza

Nota campi con lo stesso nome e stesso tipo

Laureati

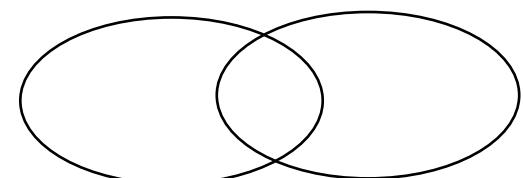
Matricola	Nome	Età
7274	Rossi	42
7432	Neri	54
9824	Verdi	45

Quadri

Matricola	Nome	Età
9297	Neri	33
7432	Neri	54
9824	Verdi	45

Laureati – Quadri

Matricola	Nome	Età
7274	Rossi	42



## ALGEBRA RELAZIONALE: UNIONE E DIFFERENZA

---

- Unione:  $R \cup S$
- Differenza:  $R - S$
- Qual è il tipo del risultato? Se  $R$  e  $S$  hanno  $n$  elementi, quanti ne ha il risultato?
- Se  $t_1$  è un'ennupla non in  $R$ , allora:
  - $R = (R \cup \{t_1\}) - \{t_1\}$

# ALGEBRA RELAZIONALE: Una unione sensata ma impossibile

---

Paternità

Padre	Figlio
Adamò	Abele
Adamò	Caino
Abramo	Isacco

Maternità

Madre	Figlio
Eva	Abele
Eva	Set
Sara	Isacco

Paternità  $\cup$  Maternità

??

## ALGEBRA RELAZIONALE: Ridenominazione

---

- operatore **monadico** (con un argomento)
- "modifica lo schema" lasciando inalterata l'istanza dell'operando
- È indicato con la lettera  $\rho$

### Paternità

Padre	Figlio
Adamo	Abele
Adamo	Caino
Abramo	Isacco

$\rho$  Genitore  $\leftarrow$  Padre (Paternità)

Genitore	Figlio
Adamo	Abele
Adamo	Caino
Abramo	Isacco

## Paternità

Padre	Figlio
Adamò	Abele
Adamò	Caino
Abramo	Isacco

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

Genitore	Figlio
Adamò	Abele
Adamò	Caino
Abramo	Isacco

## Maternità

Madre	Figlio
Eva	Abele
Eva	Set
Sara	Isacco

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

Genitore	Figlio
Eva	Abele
Eva	Set
Sara	Isacco

---

$\rho$  Genitore  $\leftarrow$  Padre (Paternità)

Genitore	Figlio
Adamo	Abele
Adamo	Caino
Abramo	Isacco

$\rho$  Genitore  $\leftarrow$  Madre (Maternità)

Genitore	Figlio
Eva	Abele
Eva	Set
Sara	Isacco

$\rho$  Genitore  $\leftarrow$  Padre (Paternità)



$\rho$  Genitore  $\leftarrow$  Madre (Maternità)

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

## ALGEBRA RELAZIONALE: PROIEZIONE

---

- operatore monadico
- produce un risultato che
  - ha parte degli attributi dell'operando
  - contiene ennuple cui contribuiscono tutte le ennuple dell'operando **ristrette agli attributi nella lista**
- Sintassi

$\pi$  ListaAttributi (Operando)

## ALGEBRA RELAZIONALE: PROIEZIONE - esempio

---

Matricola e cognome di tutti gli impiegati

Impiegati

Matricola	Cognome	
7309	Neri	
5998	Neri	
9553	Rossi	
5698	Rossi	

$\pi_{\text{Matricola}, \text{Cognome}} (\text{Impiegati})$

## ALGEBRA RELAZIONALE: PROIEZIONE

---

- Proiezione:  $\pi_{A_1 \dots A_n}(R)$
- Qual è il tipo del risultato? Se R ha n elementi quanti ne ha il risultato?
- Esempi:
  - $\pi_{\text{Nome}, \text{Matricola}}(\text{Studenti})$
  - $\pi_{\text{Nome}}(\text{Studenti})$



## ALGEBRA RELAZIONALE: PROIEZIONE - esempio (con duplicati)

Cognome e filiale di tutti gli impiegati

Matricola	Cognome	Filiale	Stipendio
7309	Neri	Napoli	55
5998	Neri	Milano	64
9553	Rossi	Roma	44
5698	Rossi	Roma	64

Notate che vi sono due impiegati con cognome Rossi sulla filiale Roma, che succede dopo la proiezione?

$\pi_{\text{Cognome}, \text{Filiale}} (\text{Impiegati})$

	Cognome	Filiale	
	Neri	Napoli	
	Neri	Milano	
	Rossi	Roma	

## Cardinalità delle proiezioni

---

- una proiezione
  - contiene al più tante ennuple quante l'operando
  - può contenerne di meno
- se  $X$  è una superchiave di  $R$ , allora  $\pi_X(R)$  contiene esattamente tante ennuple quante  $R$ .
- Se  $X$  non è superchiave, potrebbero esistere valori ripetuti su quegli attributi, che quindi vengono rappresentati una sola volta

## ALGEBRA RELAZIONALE: SELEZIONE (RESTRIZIONE)

---

- operatore monadico
- produce un risultato che
  - ha lo stesso schema dell'operando
  - contiene un sottoinsieme delle ennuple dell'operando,
  - quelle che soddisfano una condizione espressa combinando, con i connettivi logici  $\wedge$  (and),  $\vee$  (or),  $\neg$  (not), condizioni atomiche del tipo  $A \theta B$  o  $A \theta c$ , dove  $\theta$  è un operatore di confronto,  $A$  e  $B$  sono attributi su cui l'operatore  $\theta$  abbia senso,  $c$  una costante compatibile col dominio di  $A$
- È denotata con  $\sigma$ , con la condizione messa a pedice

## ALGEBRA RELAZIONALE: SELEZIONE - esempio

---

Impiegati che guadagnano più di 50

Impiegati

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

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

## ALGEBRA RELAZIONALE: SELEZIONE - esempio

---

Impiegati che guadagnano più di 50 e lavorano a Milano

Impiegati

Matricola	Cognome	Filiale	Stipendio
5998	Neri	Milano	64

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

## ALGEBRA RELAZIONALE: SELEZIONE - esempio

---

Impiegati che hanno lo stesso nome della filiale  
presso cui lavorano

Impiegati

Matricola	Cognome	Filiale	Stipendio
9553	Milano	Milano	44

$\sigma_{Cognome = Filiale}(\text{Impiegati})$

## ALGEBRA RELAZIONALE: SELEZIONE - esempio con valori nulli

### Impiegati

Matricola	Cognome	Filiale	Età
7309	Rossi	Roma	32
5998	Neri	Milano	45
9553	Bruni	Milano	NULL

Quale è il risultato?  
Che succede alla tupla con valore NULL in età?

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

- la condizione atomica è vera solo per valori non nulli

# ALGEBRA RELAZIONALE: RESTRIZIONE

---

- Restrizione (selezione):  $\sigma_{\text{Condizione}}(R)$
- Qual è il tipo del risultato? Se R ha n elementi, quanti ne ha il risultato?
- Esempi:
  - $\sigma_{\text{Nome} = 'Caio'}(\text{Studenti})$
- Composizione di operatori:
  - $\pi_{\text{Matricola}}(\sigma_{\text{Nome} = 'Caio'}(\text{Studenti}))$
- Cond ::= Espr **Theta** Espr | Cond And Cond | Not Cond
- Espr ::= Attributo | Costante | Espr **Op** Espr
- **Theta** ::= = | < | > | != | <= | >=
- **Op** ::= + | - | \* | StringConcat

## Un risultato non desiderabile

---

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

- Perché?

Perché le selezioni vengono valutate separatamente!

- Ma anche

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

- Perché?

- Perché anche le condizioni atomiche vengono valutate separatamente!



---

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

- la condizione atomica è vera solo per valori **non nulli**
- per riferirsi ai valori nulli esistono forme apposite di condizioni:

IS NULL

IS NOT NULL

- 
- A questo punto:

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

=

$$\sigma_{\text{Età}>30 \vee \text{Età}\leq 30 \vee \text{Età IS NULL}}(\text{Persone})$$

=

Persone

## Esempio

---

### Impiegati

Matricola	Cognome	Filiale	Età
5998	Neri	Milano	45
9553	Bruni	Milano	NULL

$$\sigma_{(\text{Età} > 40) \vee (\text{Età IS NULL})} (\text{Impiegati})$$

# PROIEZIONE E RESTRIZIONE

- Proiezione  $\pi_{A,B}(R)$ :

A

B



- Restrizione  $\sigma_{\text{Cond}}(R)$ :



## Selezione e proiezione

---

Combinando selezione e proiezione, possiamo estrarre interessanti informazioni da una relazione

## ALGEBRA RELAZIONALE: PROIEZIONE E SELEZIONE - esempio

---

Matricola e cognome degli impiegati che  
guadagnano più di 50

Matricola	Cognome
7309	Rossi
5998	Neri
5698	Neri

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

## Esercizio Cardinalità

---

- Considerare una relazione  $R(A; \underline{B}; C; D; E)$ . Indicare quali delle seguenti proiezioni hanno certamente lo stesso numero di ennuple di  $R$ :
  1.  $\pi_{ABCD}(R)$
  2.  $\pi_{AC}(R)$
  3.  $\pi_{BC}(R)$
  4.  $\pi_C(R)$
  5.  $\pi_{CD}(R)$ .

# ALGEBRA RELAZIONALE: PRODOTTO

Nota che i nomi dei campi sono distinti

- Prodotto:  $R \times S$

a	A
a1	A1
a2	A2

×

b	B
b1	B1
b2	B2
b3	B3

=

a	A	b	B
a1	A1	b1	B1
a1	A1	b2	B2
a1	A1	b3	B3
a2	A2	b1	B1
a2	A2	b2	B2
a2	A2	b3	B3

- Qual è il tipo del risultato? Se R ha n elementi quanti ne ha il risultato?

## Impiegati

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

## Reparti

Codice	Capo
A	Mori
B	Bruni

Impiegati × Reparti

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

Impiegati

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

Reparti

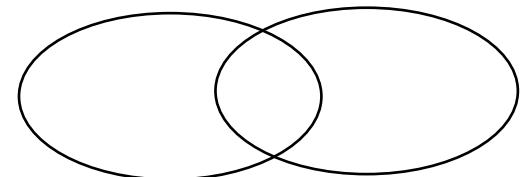
Reparto	Capo
A	Mori
B	Bruni

Impiegati × Reparti



# ALGEBRA RELAZIONALE: Intersezione - Operatore derivato

---



Laureati

Matricola	Nome	Età
7274	Rossi	42
7432	Neri	54
9824	Verdi	45

Quadri

Matricola	Nome	Età
9297	Neri	33
7432	Neri	54
9824	Verdi	45

Laureati  $\cap$  Quadri

Matricola	Nome	Età
7432	Neri	54
9824	Verdi	45



# ALGEBRA RELAZIONALE: Intersezione

- Possiamo derivare l'operatore intersezione usando gli operatori:
- $R(A,B), S(A,B)$

$$R \cap S = \{x \mid x \in R \ \exists y \in S, x = y\}$$

$\rho_S \rightarrow$  ride nomina gli attributi di S anteponendo il prefisso S

- Prodotto
- Ridenominazione
- Selezione
- Proiezione

A	B	S.A	S.B
1	2	1	2
2	3	2	4

$\rho_S S$

$$\sigma_{A=S.A \text{ AND } B=S.B}(R \times \rho_S S)$$

R $\times$ $\rho_S S$			
A	B	S.A	S.B
1	2	1	2

$$\pi_{A,B}(\sigma_{A=S.A \text{ AND } B=S.B}(R \times \rho_S S))$$

A	B
1	2

## Esempio: Prove scritte in un concorso pubblico

---

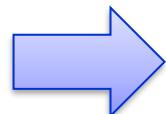
- I compiti sono anonimi e ad ognuno è associata una busta chiusa con il nome del candidato
- Ciascun compito e la relativa busta vengono contrassegnati con uno stesso numero

Numero	Voto
1	25
2	13
3	27
4	28

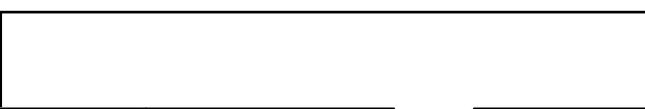
Numero	Candidato
1	Mario Rossi
2	Nicola Russo
3	Mario Bianchi
4	Remo Neri

Mario Rossi	25
Nicola Russo	13
Mario Bianchi	27
Remo Neri	28



## Esempio: Prove scritte in un concorso pubblico

---



Numero	Voto	Numero	Candidato
1	25	1	Mario Rossi
2	13	2	Nicola Russo
3	27	3	Mario Bianchi
4	28	4	Remo Neri

Numero	Candidato	Voto
1	Mario Rossi	25
2	Nicola Russo	13
3	Mario Bianchi	27
4	Remo Neri	28

## Prodotto cartesiano e chiavi esterne

Studenti

Nome	<u>Matricola</u>	Provincia	AnnoNascita
Isaia	071523	PI	1982
Rossi	067459	LU	1984
Bianchi	079856	LI	1983
Bonini	075649	PI	1984

Esami

<u>Materia</u>	<u>Matricola*</u>	Data	Voto
BD	071523	12/01/06	28
BD	067459	15/09/06	30
FP	079856	25/10/06	30
BD	075649	27/06/06	25
LMM	071523	10/10/06	18

Vincolo di integrità referenziale

Come possiamo ottenere la tabella  
EsamiStudenti  
(Matricola, nome,  
materia, data, voto)

Studenti × Esami

Tutte le righe del prodotto cartesiano sono utili?

$$\pi_{Matr, Nome, Materia, Data, Voto}(\sigma_{matricola=S.matricola}(Esami \times \rho_S Studenti))$$

# Prodotto cartesiano seguito da selezione - Rappresentazione mediante albero

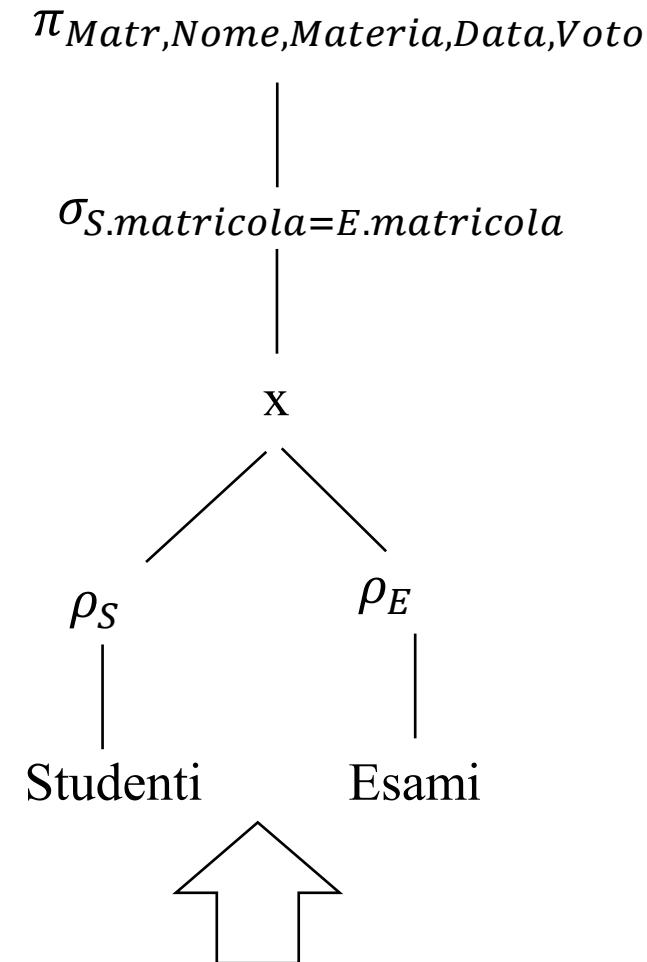
Studenti

Nome	<u>Matricola</u>	Provincia	AnnoNascita
Isaia	071523	PI	1982
Rossi	067459	LU	1984
Bianchi	079856	LI	1983
Bonini	075649	PI	1984

Esami

<u>Materia</u>	<u>Matricola*</u>	Data	Voto
BD	071523	12/01/06	28
BD	067459	15/09/06	30
FP	079856	25/10/06	30
BD	075649	27/06/06	25
LMM	071523	10/10/06	18

Vincolo di integrità referenziale



$\pi_{\text{Matr}, \text{Nome}, \text{Materia}, \text{Data}, \text{Voto}}(\sigma_{\text{matricola}=\text{S.matricola}}(\text{Esami} \times \rho_s \text{ Studenti}))$

## ALGEBRA RELAZIONALE: Join (giunzione) - Operatore derivato

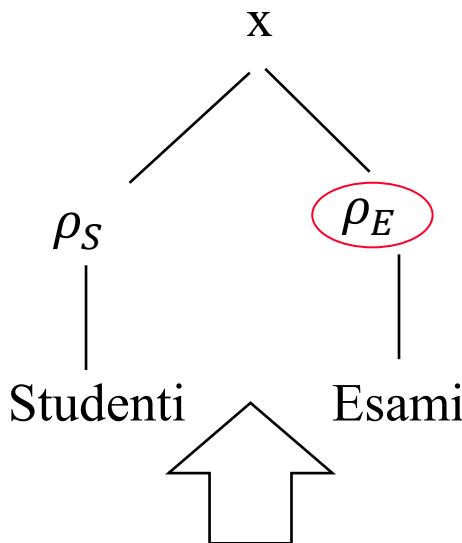
---

- Combinando selezione e proiezione, possiamo estrarre informazioni da una relazione, ma non possiamo però correlare informazioni presenti in relazioni diverse
- il **join** è l'operatore più interessante dell'algebra relazionale poiché permette di **correlare** dati in relazioni diverse

# Join (Giunzione) naturale

$\pi_{Matricola, Nome, Materia, Data, Voto}$

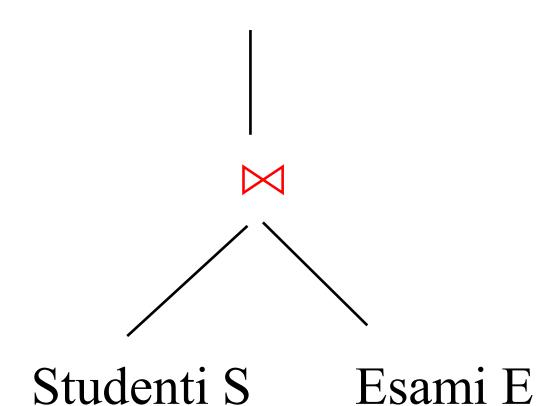
$$\sigma_{S.matricola=E.matricola}$$



Studenti	Nome	<u>Matricola</u>	Provincia	AnnoNascita
Esami	<u>Materia</u>	<u>Matricola*</u>	Data	Voto

$\pi_{Matricola, Nome, Materia, Data, Voto}$

Join naturale



$\pi_{Matr, Nome, Materia, Data, Voto}(\sigma_{matricola=S.matricola}(Esami \times \rho_S Studenti))$

## ALGEBRA RELAZIONALE: Join naturale

---

- operatore binario (generalizzabile) che correla dati in relazioni diverse,  
**sulla base di valori uguali in attributi con lo stesso nome.**
- Produce un risultato:
  - sull'unione degli attributi degli operandi
  - con ennuple che sono ottenute combinando le ennuple degli operandi  
con valori uguali sugli attributi in comune
- $R_1(X_1), R_2(X_2)$
- $R_1 \bowtie R_2$  è una relazione su  $X_1 \cup X_2$

$R_1 \bowtie R_2 = \{ t \text{ su } X_1 \cup X_2 \mid \text{esistono } t_1 \in R_1 \text{ e } t_2 \in R_2$

$\text{con } t[X_1] = t_1 \text{ e } t[X_2] = t_2 \}$

## Join naturale e attributi uguali

---

Studenti

Nome	<u>Matricola</u>	Provincia	Data
------	------------------	-----------	------

Esami

Materia	<u>Matricola*</u>	Data	Voto
---------	-------------------	------	------

Vincolo di integrità referenziale°

Che succede?

Join:  $R_1 \bowtie R_2$

---

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

Reparto	Capo
A	Mori
B	Bruni

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

- ogni ennupla contribuisce al risultato:
  - **join completo**

## ALGEBRA RELAZIONALE: Un join non completo

---

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

Reparto	Capo
B	Mori
C	Bruni

Impiegato	Reparto	Capo
Neri	B	Mori
Bianchi	B	Mori

## ALGEBRA RELAZIONALE: Un join vuoto

---

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

Impiegato	Reparto	Capo

## ALGEBRA RELAZIONALE: Un join completo, con $n \times m$ ennuple

---

Impiegato	Reparto
Rossi	B
Neri	B

Reparto	Capo
B	Mori
B	Bruni

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

## ALGEBRA RELAZIONALE: Cardinalità del join

---

$$R_1(A,B), R_2(B,C)$$

- Il join di  $R_1$  e  $R_2$  contiene un numero di ennuple compreso fra zero e il prodotto di  $|R_1|$  e  $|R_2|$ :

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

- Se il join fra  $R_1$  ed  $R_2$  è completo, allora contiene un numero di ennuple almeno uguale al massimo fra  $|R_1|$  e  $|R_2|$ :
- Se il join coinvolge una chiave  $B$  di  $R_2$ , allora il numero di ennuple è compreso fra zero e  $|R_1|$ :

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

- Se il join coinvolge una chiave  $B$  di  $R_2$  e un vincolo di integrità referenziale tra attributi di  $R_1$  ( $B$  in  $R_1$ ) e la chiave di  $R_2$ , allora il numero di ennuple è pari a  $|R_1|$ :

$$|R_1 \bowtie R_2| = |R_1|$$

## ALGEBRA RELAZIONALE: Join, una difficoltà

---

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

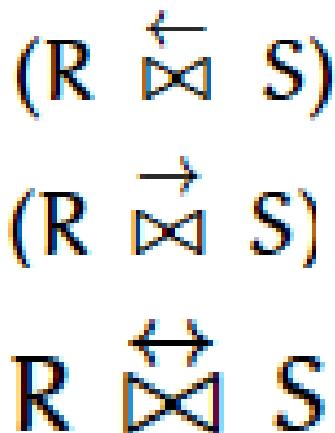
Impiegato	Reparto	Capo
Neri	B	Mori
Bianchi	B	Mori

alcune ennuple non contribuiscono al risultato: vengono "tagliate fuori"

## ALGEBRA RELAZIONALE: Join esterno

---

- Il **join esterno** estende, con valori nulli, le ennuple che verrebbero tagliate fuori da un join (**interno**)
- esiste in tre versioni:
  - sinistro, destro, completo
- Sinistro (left): mantiene tutte le ennuple del primo operando, estendendole con valori nulli, se necessario
- Destro (right): ... del secondo operando ...
- Completo (full): ... di entrambi gli operandi ...



## Impiegati

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

## Reparti

Reparto	Capo
B	Mori
C	Bruni

Impiegati  $\bowtie_{\text{LEFT}}$  Reparti

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

Impiegati

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

Reparti

Reparto	Capo
B	Mori
C	Bruni

Impiegati  $\bowtie_{RIGHT}$  Reparti

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

---

Impiegati

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

Reparti

Reparto	Capo
B	Mori
C	Bruni

Impiegati  $\bowtie_{\text{FULL}}$  Reparti

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

## ALGEBRA RELAZIONALE: Join (naturale) e proiezioni

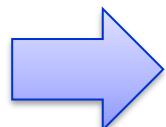
---

$R_1(X_1), R_2(X_2)$

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

$R(X), X = X_1 \cup X_2$

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



## Join e Proiezioni

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

Impiegati  
 $\bowtie$  Reparti

$\pi_{\text{impiegato},$   
 $\text{reparto}}$   
 $(\text{Impiegati}$   
 $\bowtie \text{ Reparti})$

$\pi_{\text{reparto}, \text{capo}}$   
 $(\text{Impiegati}$   
 $\bowtie \text{ Reparti})$

## Impiegati

### Impiegato

Rossi  
 Neri  
 Bianchi

### Reparto

A  
 B  
 B

## Reparti

### Reparto

B  
 C

### Capo

Mori  
 Bruni

### Impiegato

Neri  
 Bianchi

### Reparto

B  
 B

### Capo

Mori  
 Mori

### Impiegato

Neri  
 Bianchi

### Reparto

B  
 B

### Reparto

B

### Capo

Mori

### Impiegato

Neri  
 Bianchi

### Reparto

B  
 B

Le **proiezioni**  
 del join su  $X_1$  e  $X_2$   
 danno luogo a  
 tabelle diverse da  
 quelle da cui il join è  
 stato ottenuto.

## Proiezioni e join

Mostriamo che  
 $R \subseteq (\pi_{X_1}(R)) \bowtie (\pi_{X_2}(R))$

Il join delle proiezioni  
di una tabella  
può dare luogo a  
una tabella più grande

$\pi_{\text{impiegato, reparto}}(\text{newImpiegatiReparti})$

Impiegato	Reparto
Neri	B
Bianchi	B
Verdi	A

$\pi_{\text{reparto, capo}}(\text{newImpiegatiReparti})$

Reparto	Capo
B	Mori
B	Bruni
A	Bini

newImpiegatiReparti

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

$(\pi_{\text{impiegato, reparto}}(\text{newImpiegatiReparti}))$

$\bowtie$

$(\pi_{\text{reparto, capo}}(\text{newImpiegatiReparti}))$

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

## ALGEBRA RELAZIONALE: Prodotto cartesiano

---

- un join naturale su relazioni senza attributi in comune
- contiene sempre un numero di ennuple pari al prodotto delle cardinalità degli operandi (le ennuple sono tutte combinabili)

## Theta-join ed equi-join

- **Prodotto cartesiano:** concatena tuple non necessariamente correlate dal punto di vista semantico.

Impiegati

Impiegato	Reparto
Rossi	A
Neri	A
Neri	B

Reparti

Codice	Capo
A	Venere
B	Marte

Impiegati x Reparti

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

Il prodotto cartesiano è più utile se fatto seguire da una selezione.

Prodotto cartesiano seguito dalla selezione che mantiene solo le tuple con valori uguali sull'attributo:

Reparto di Impiegati

e

Codice di Reparti

## ALGEBRA RELAZIONALE: Theta-join

---

- Il prodotto cartesiano, in pratica, ha senso solo se seguito da selezione:

$$\sigma_{\text{Condizione}}(R_1 \bowtie R_2)$$

- L'operazione viene chiamata **theta-join** e può essere sintatticamente indicata con

$$R_1 \bowtie_{\text{Condizione}} R_2$$

Le due scritture sono equivalenti

## ALGEBRA RELAZIONALE: Perché "theta-join"?

---

- La condizione  $C$  è spesso una congiunzione ( $\wedge$ ) di atomi di confronto  $A_1 \vartheta A_2$  dove  $\vartheta$  è uno degli operatori di confronto ( $=, >, <, \dots$ )
- se l'operatore è sempre l'uguaglianza ( $=$ ) allora si parla di **equi-join**

## Join (Giunzione)

---

$\pi_{Matr, Nome, Materia, Data, Voto}$

$\sigma_{S.matricola=E.matricola}$

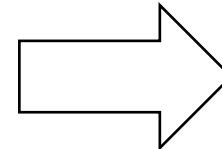
X

$\rho_S$

Studenti

$\rho_E$

Esami



$\pi_{Matr, Nome, Materia, Data, Voto}$

$\bowtie_{S.matricola=E.matricola}$

Studenti S

Esami E

$\pi_{Matr, Nome, Materia, Data, Voto}(\sigma_{matricola=S.matricola}(Esami \times \rho_S Studenti))$

## Impiegati

## Reparti

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

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

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

---

## Impiegati

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

## Reparti

Reparto	Capo
A	Mori
B	Bruni

Impiegati  $\bowtie$  Reparti

## ALGEBRA RELAZIONALE: Join naturale ed equi-join

---

Impiegati

Impiegato	Reparto
-----------	---------

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

Impiegati  $\bowtie$  Reparti

Impiegati

Impiegato	Reparto
-----------	---------

Reparti

Codice	Capo
--------	------

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

$(\pi_{\text{Impiegato}, \text{Reparto}, \text{Capo}}$

$(\text{Impiegati} \bowtie \rho_{\text{Reparto} \leftarrow \text{Codice}}(\text{Reparti})$

## Esercizio 1

---

IMPIEGATI(Codice, Nome, Cognome, Età)

Trovare gli impiegati che guadagnano più di 40 mila euro

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

Matricola	Nome	Età	Stipendio
7309	Rossi	34	45
5698	Bruni	43	42
4076	Mori	45	50
8123	Lupi	46	60

## Esercizio 2

---

**IMPIEGATI(Codice, Nome, Cognome, Età)**

- Trovare codice, nome ed età degli impiegati che guadagnano più di 40 mila euro

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

Matricola	Nome	Età
7309	Rossi	34
5698	Bruni	43
4076	Mori	45
8123	Lupi	46

## Esercizio

---

Impiegati

<u>Matricola</u>	Nome	Età	Stipendio
7309	Rossi	34	45
5998	Bianchi	37	38
9553	Neri	42	35
5698	Bruni	43	42
4076	Mori	45	50
8123	Lupi	46	60

Supervisione

<u>Impiegato*</u>	<u>Capo*</u>
7309	5698
5998	5698
9553	4076
5698	4076
4076	8123

## ALGEBRA RELAZIONALE: Self Join

---

Supponiamo di considerare la seguente relazione

Genitori	
Genitore	Figlio
Luca	Anna
Maria	Anna
Giorgio	Luca
Silvia	Maria
Enzo	Maria

e di volere ottenere una relazione Nonno-Nipote.

E' ovvio che in questo caso abbiamo bisogno di utilizzare due volte la stessa tabella

## ALGEBRA RELAZIONALE: Self Join

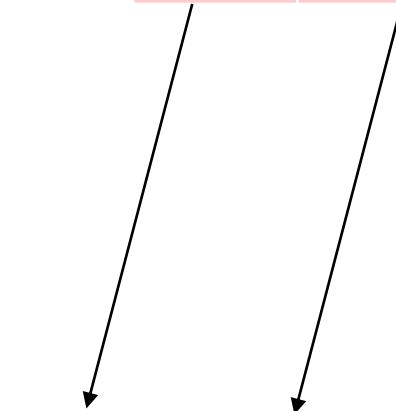
Tuttavia  $\text{Genitore} \bowtie \text{Genitore} = \text{Genitore}$ , poiché tutti gli attributi coincidono.

In questo caso è utile effettuare una ridenominazione:

$\rho_{\text{Nonno}, \text{Genitore} \leftarrow \text{Genitore}, \text{Figlio}}(\text{Genitore})$

A questo punto effettuando un natural join con la tabella Genitore, si ottiene l'informazione cercata

Genitore	Figlio
Luca	Anna
Maria	Anna
Giorgio	Luca
Silvia	Maria
Enzo	Maria



Nonno	Genitore
Luca	Anna
Maria	Anna
Giorgio	Luca
Silvia	Maria
Enzo	Maria

Genitore	Figlio
Luca	Anna
Maria	Anna
Giorgio	Luca
Silvia	Maria
Enzo	Maria

## ALGEBRA RELAZIONALE: Self Join

$\rho_{\text{Nonno}, \text{Genitore} \leftarrow \text{Genitore}, \text{Figlio}} (\text{Genitore}) \bowtie \rho_{\text{Nipote} \leftarrow \text{Figlio}} (\text{Genitore})$

Nonno	Genitore
Luca	Anna
Maria	Anna
Giorgio	Luca
Silvia	Maria
Enzo	Maria



Genitore	Nipote
Luca	Anna
Maria	Anna
Giorgio	Luca
Silvia	Maria
Enzo	Maria



Nonno	Genitore	Nipote
Giorgio	Luca	Anna
Silvia	Maria	Anna
Enzo	Maria	Anna

Eventualmente si può effettuare una proiezione

$\pi_{\text{Nonno}, \text{Nipote}} (\rho_{\text{Nonno}, \text{Genitore} \leftarrow \text{Genitore}, \text{Figlio}} (\text{Genitore}) \bowtie \rho_{\text{Nipote} \leftarrow \text{Figlio}} (\text{Genitore}))$

Nonno	Nipote
Giorgio	Anna
Silvia	Anna
Enzo	Anna

## ALGEBRA RELAZIONALE: ALTRI OPERATORI - Riepilogo

---

- Ridenominazione  $\rho_{A \leftarrow B}(R)$
- Operatori Derivati:
  - intersezione:  $R \cap S$
  - giunzione:  $R \vee_{R.A=S.B} S$  oppure  $R \bowtie_{R.A=S.B} S$
  - giunzione naturale:  $R \vee S$  oppure  $R \bowtie S$

## Esempio: Join Studenti ed Esami

The diagram illustrates the join operation between two tables:

- Studenti Table:** Contains columns Matricola, Nome, and Cognome. Data: (1, Luca, Rossi), (2, Maria, Bianchi), (3, Giorgio, Viola), (4, Silvia, Neri), (5, Enzo, Verdi).
- Esami Table:** Contains columns Matricola and Voto. Data: (1, 25), (3, 30), (2, 23), (1, 29), (4, 29), (1, 26), (5, 30), (4, 30).
- Result Table:** Shows the joined data where rows from both tables are combined based on matching Matricola values. Data: (Luca, Rossi, 1, 25), (Giorgio, Viola, 3, 30), (Maria, Bianchi, 2, 23), (Luca, Rossi, 1, 29), (Silvia, Neri, 4, 29), (Luca, Rossi, 1, 26), (Enzo, Verdi, 5, 30), (Silvia, Neri, 4, 30).

Problemi: Matricola, Nome, cognome, voto degli studenti:

- con (almeno un) **voto maggiore di 28** (quantificatore esistenziale)
- **non** hanno mai ottenuto un **voto maggiore di 28** (differenza)
- Nomi degli studenti che hanno ottenuto **solo voti maggiore di 28** (quantificatore universale)

## Esempio: Join Studenti ed Esami

The diagram illustrates the join operation between two tables:

- Studenti Table:** Contains columns Matricola, Nome, and Cognome. Data: (1, Luca, Rossi), (2, Maria, Bianchi), (3, Giorgio, Viola), (4, Silvia, Neri), (5, Enzo, Verdi).
- Esami Table:** Contains columns Matricola and Voto. Data: (1, 25), (3, 30), (2, 23), (1, 29), (4, 29), (1, 26), (5, 30), (4, 30).
- Result Table:** Shows the joined data where rows from both tables are combined based on matching Matricola values. Data: (Luca, Rossi, 1, 25), (Giorgio, Viola, 3, 30), (Maria, Bianchi, 2, 23), (Luca, Rossi, 1, 29), (Silvia, Neri, 4, 29), (Luca, Rossi, 1, 26), (Enzo, Verdi, 5, 30), (Silvia, Neri, 4, 30).

Problemi: Matricola, Nome, cognome, voto degli studenti:

- con (almeno un) **voto maggiore di 28** (quantificatore esistenziale)
- **non** hanno mai ottenuto un **voto maggiore di 28** (differenza)
- Nomi degli studenti che hanno ottenuto **solo voti maggiore di 28** (quantificatore universale)

# Quantificatore esistenziale - Parte I

Problema: Matricola, Nome, cognome, voto degli studenti: con (almeno un) voto maggiore di 28 (quantificatore esistenziale)

Matri cola	Nome	Cognome
1	Luca	Rossi
2	Maria	Bianchi
3	Giorgio	Viola
4	Silvia	Neri
5	Enzo	Verdi



Matricola	Voto
1	25
3	30
2	23
1	29
4	29
1	26
5	30
4	30

Nome	Cognome	Matricola	Voto
Luca	Rossi	1	25
Giorgio	Viola	3	30
Maria	Bianchi	2	23
Luca	Rossi	1	29
Silvia	Neri	4	29
Luca	Rossi	1	26
Enzo	Verdi	5	30
Silvia	Neri	4	30

Cloud: Luca Rossi va eliminato?

?

Cloud: Join e selezione



## Esempio di trasformazione su quantificatore esistenziale

Matricola, Nome, cognome, materia, data, voto degli studenti con **voto maggiore di 28**.

In che punto applichiamo la restrizione sul voto?

Studenti	Nome	<u>Matricola</u>	Provincia	AnnoNascita
----------	------	------------------	-----------	-------------

Esami	<u>Materia</u>	<u>Matricola*</u>	Data	Voto
-------	----------------	-------------------	------	------

$\pi_{Matr, Nome, Materia, Data, Voto}$

$\bowtie_{S.matricola=E.matricola}$

Studenti S

Esami E

$\pi_{Matr, Nome, Materia, Data, Voto}$

$\sigma_{Voto > 28}$

$\bowtie_{S.matricola=E.matricola}$

Studenti S

Esami E

$\pi_{Matr, Nome, Materia, Data, Voto}$

$\bowtie_{S.matricola=E.matricola}$

$\sigma_{Voto > 28}$

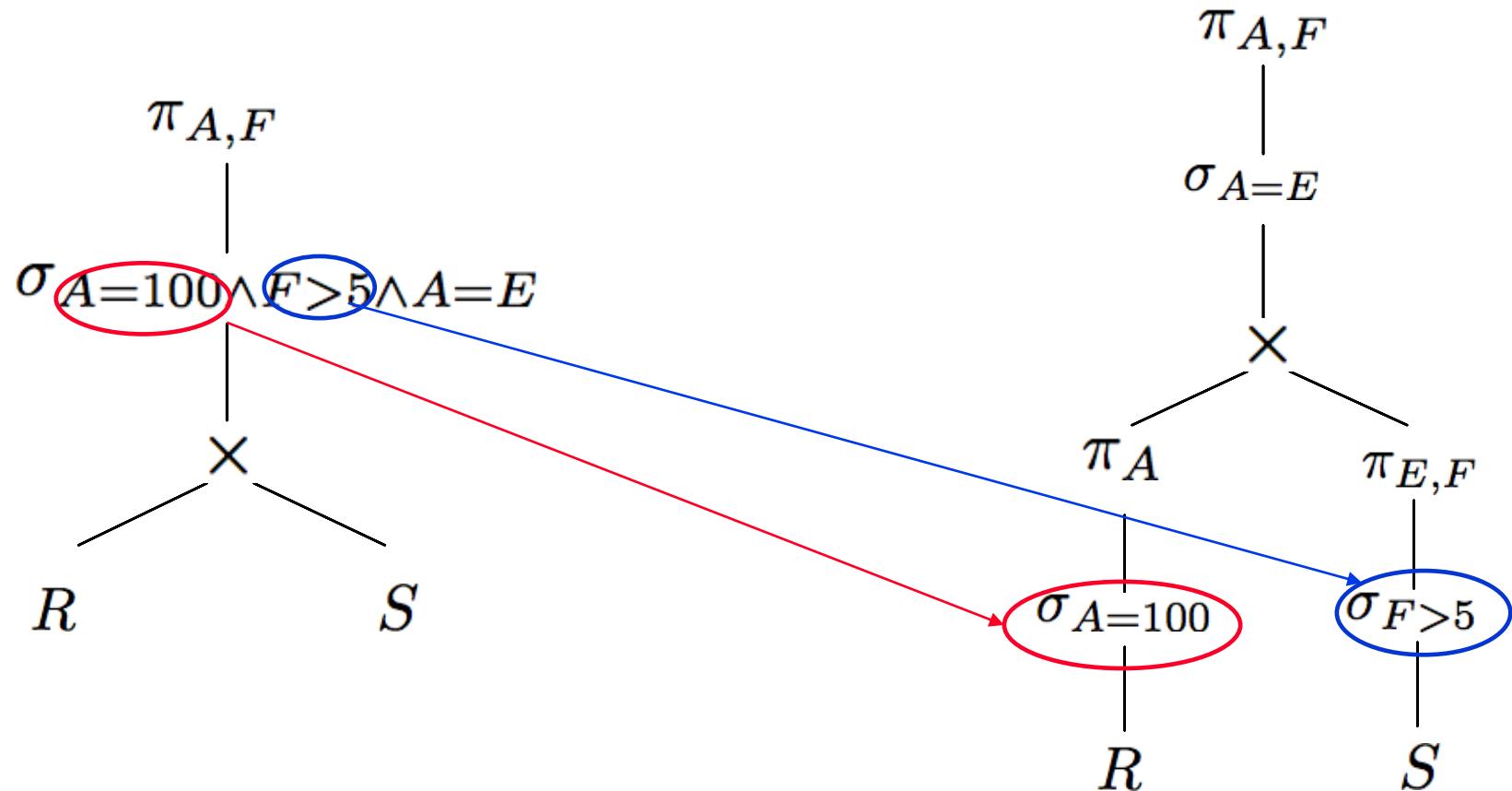
Studenti S

Esami E

# ALBERI LOGICI E TRASFORMAZIONI ALGEBRICHE

- Consideriamo le relazioni  $R(A, B, C, D)$  e  $S(E, F, G)$  e l'espressione:

$$\pi_{A,F}(\sigma_{A=100 \wedge F>5 \wedge A=E}(R \times S))$$



## Esempio 2 - Differenza - Parte I

Problema: studenti che **non** hanno mai ottenuto un **voto maggiore di 28** (differenza)

Matricola	Nome	Cognome
1	Luca	Rossi
2	Maria	Bianchi
3	Giorgio	Viola
4	Silvia	Neri
5	Enzo	Verdi



Matricola	Voto
1	25
3	30
2	23
1	29
4	29
1	26
5	30
4	30

Nome	Cognome	Matricola	Voto
Luca	Rossi	1	25
Giorgio	Viola	3	30
Maria	Bianchi	2	23
Luca	Rossi	1	29
Silvia	Neri	4	29
Luca	Rossi	1	26
Enzo	Verdi	5	30
Silvia	Neri	4	30

Selezioniamo gli studenti che hanno preso voti >28

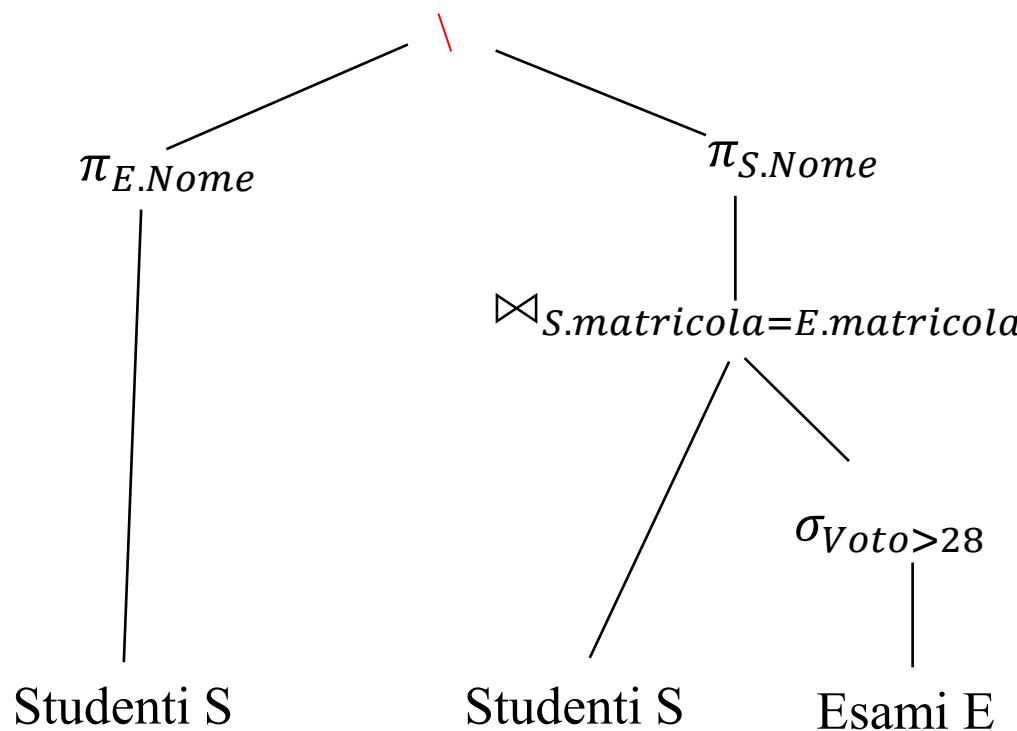
Faccio la differenza fra gli studenti e questo insieme

## Esempio 2 - Differenza - Parte II

Nome degli studenti che **non** hanno mai ottenuto un **voto maggiore di 28**.

Studenti	Nome	<u>Matricola</u>	Provincia	AnnoNascita
----------	------	------------------	-----------	-------------

Esami	<u>Materia</u>	<u>Matricola*</u>	Data	Voto
-------	----------------	-------------------	------	------



## Esempio 3 - Quantificatore universale - Parte I

Problema: studenti che hanno ottenuto **solo voti maggiore di 28** (quantificatore universale)

Output: Giorgio, Silvia, Enzo

Matricola	Nome	Cognome
1	Luca	Rossi
2	Maria	Bianchi
3	Giorgio	Viola
4	Silvia	Neri
5	Enzo	Verdi



Matricola	Voto
1	25
3	30
2	23
1	29
4	29
1	26
5	30
4	30

Nome	Cognome	Matricola	Voto
Luca	Rossi	1	25
Giorgio	Viola	3	30
Maria	Bianchi	2	23
Luca	Rossi	1	29
Silvia	Neri	4	29
Luca	Rossi	1	26
Enzo	Verdi	5	30
Silvia	Neri	4	30

Selezioniamo gli studenti che hanno preso voti  $\leq 28$

Faccio come prima la differenza fra gli studenti e questo insieme?

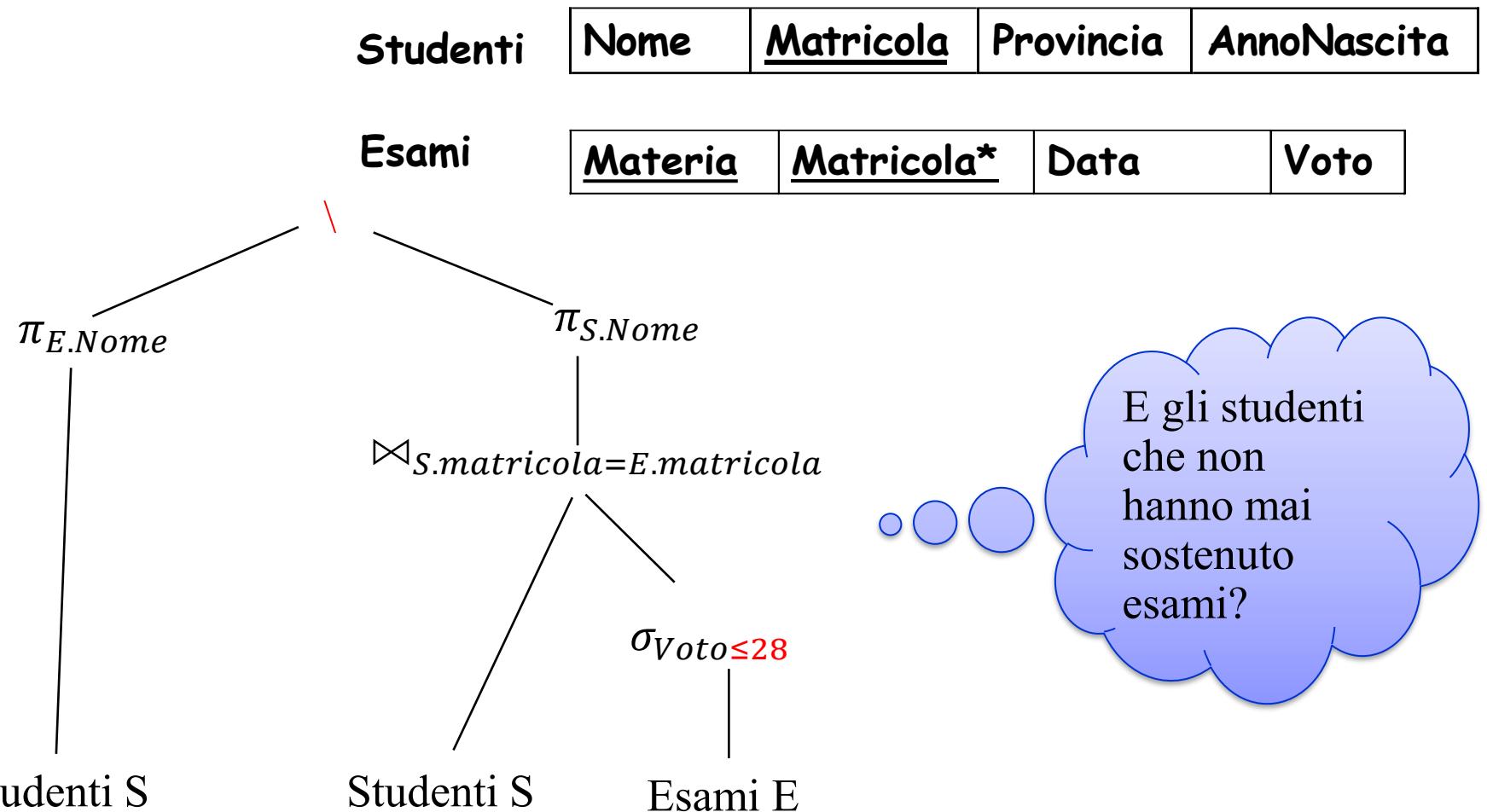
Il modello relazionale

Giorgio	Viola
Silvia	Neri
Enzo	Verdi

4.85

## Esempio 3 - quantificatore universale - Parte II

Nomi degli studenti che hanno ottenuto **solo** voti maggiore di 28.



## Esempio 3 - Quantificatore universale - Parte I

Studenti che hanno sostenuto almeno un esame

Problema: studenti che hanno ottenuto solo voti maggiore di 28 (escludendo quelli che non hanno sostenuto esami).

Matri cola	Nome	Cognome
1	Luca	Rossi
2	Maria	Bianchi
3	Giorgio	Viola
4	Silvia	Neri
5	Enzo	Verdi

Matricola	Voto
1	25
3	30
2	23
1	29
4	29
1	26
4	30

Nome	Cognome	Matricola	Voto
Luca	Rossi	1	25
Giorgio	Viola	3	30
Maria	Bianchi	2	23
Luca	Rossi	1	29
Silvia	Neri	4	29
Luca	Rossi	1	26
Silvia	Neri	4	30

Nota che Enzo non ha sostenuto esami!

Esami degli Studenti con voto  $\leq 28$

Nome	Cognome	Matricola	Voto
Luca	Rossi	1	25
Maria	Bianchi	2	23
Luca	Rossi	1	26

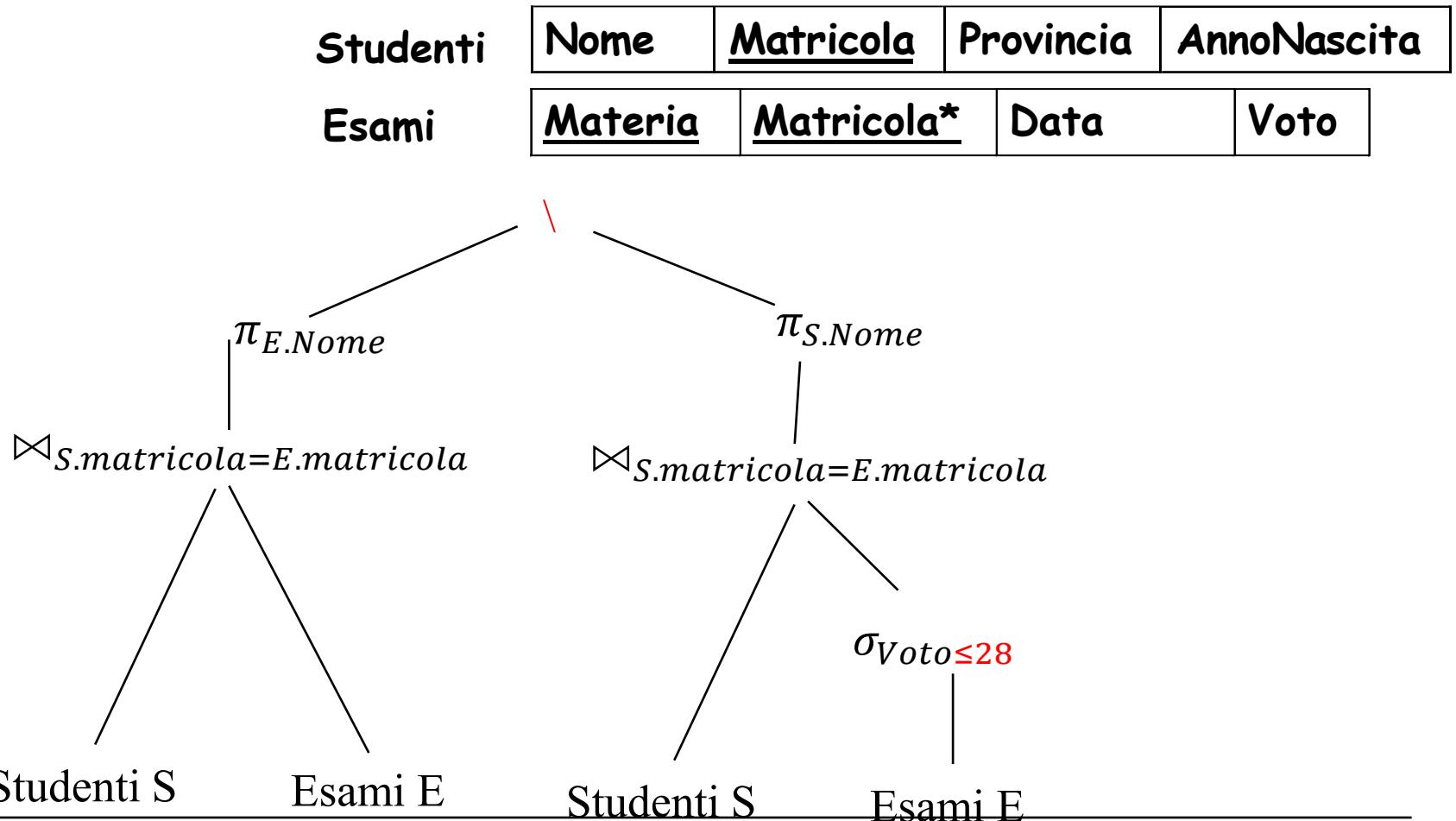
Il modello relazionale

Faccio la differenza fra gli studenti che hanno sostenuto almeno un esame e questo insieme

4.87

## Esempio 4 - quantificatore universale - Parte III

Nomi degli studenti che hanno ottenuto solo voti maggiore di 28 (escludendo quelli che non hanno sostenuto esami).



## Non Distributività della proiezione rispetto alla differenza

- In generale  $\pi_A(R_1 - R_2) \not\leftrightarrow \pi_A(R_1) - \pi_A(R_2)$
- Se  $R_1$  e  $R_2$  sono definite su  $AB$ , e contengono tuple uguali su  $A$  e diverse su  $B$

Impl

Impiegato	Capo
Neri	Mori
Bianchi	Bruni
Verdi	Bini

Imp2

Impiegato	Capo
Neri	Rossi
Bianchi	Bordeaux
Verdi	Blu

$$\pi_A(\text{Impl} - \text{Imp2}) \equiv \pi_A(\text{Impl}) - \pi_A(\text{Imp2}) ?$$

Dipende da chi è A....

Se  $R_1$  e  $R_2$  sono definite su  $AB$  e contengono tuple uguali su  $A$  e diverse su  $B$ , NO

## ALGEBRA RELAZIONALE: RAGGRUPPAMENTO

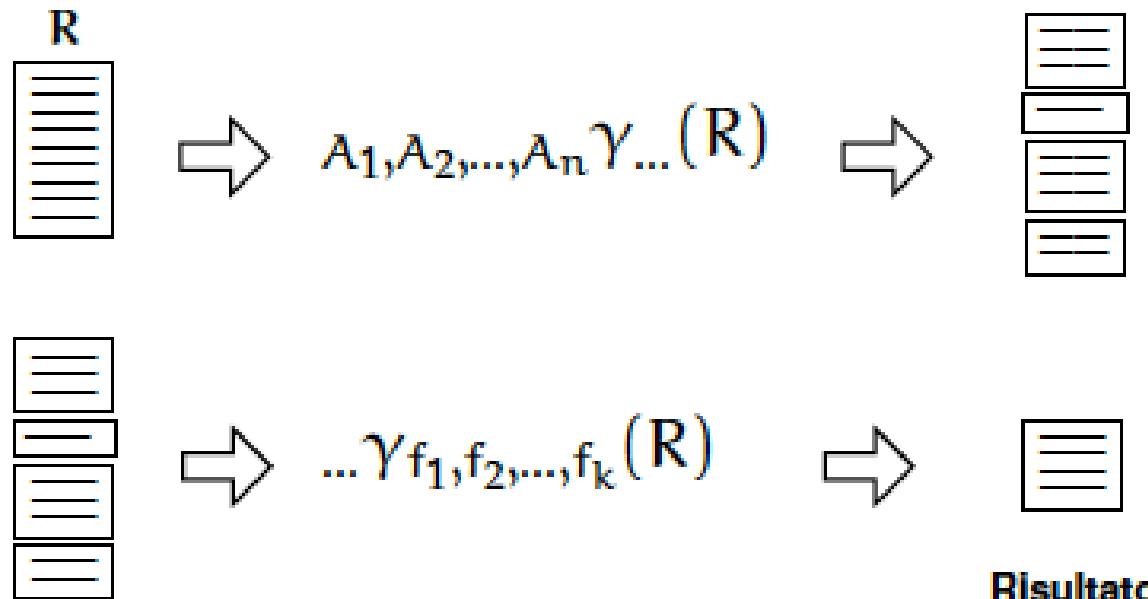
---

- Raggruppamento:  $\{A_i\} \times \{f_i\}(R)$
- Gli  $A_i$  sono attributi di  $R$  e le  $f_i$  sono espressioni che usano funzioni di aggregazione (min, max, count, sum, ...)
- Il valore del raggruppamento è una relazione calcolata come segue
  - Si partizionano le ennuple di  $R$  mettendo nello stesso gruppo tutte le ennuple con valori uguali degli  $A_i$
  - Si calcolano le espressioni  $f_i$  per ogni gruppo
  - Per ogni gruppo si restituisce una sola ennupla con attributi i valori degli  $A_i$  e delle espressioni  $f_i$

# ALGEBRA RELAZIONALE: RAGGRUPPAMENTO

---

$$\{A_i\} \gamma_{\{f_i\}}(R)$$



# ESECUZIONE DEL RAGGRUPPAMENTO

---

- Per ogni candidato: numero degli esami, voto minimo, massimo e medio:

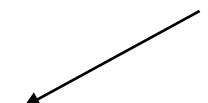
$\{\text{Candidato}\} \gamma_{\{\text{count}(*), \text{min}(\text{Voto}), \text{max}(\text{Voto}), \text{avg}(\text{Voto})\}} (\text{Esami})$

Materia	Candidato	Voto	Docente
DA	1	20	10
LFC	2	30	20
MTI	1	30	30
LP	2	20	40

Materia	Candidato	Voto	Docente
DA	1	20	10
MTI	1	24	30
LFC	2	30	20
LP	2	20	40



Candidato	Count(*)	min(Voto)	max(Voto)	avg(Voto)
1	2	20	24	22
2	2	20	30	25



# Esempio su due attributi

Clienti

IDCLIENTE	COGNOME	NOME
192	Zavoli	Luigi
97	Grassi	Maria
114	Di Santo	Luigi
42	Zavoli	Luigi
5	Di Santo	Luigi
138	Grassi	Maria
12	Zavoli	Maria

Raggruppamento per nome

NOME	COUNT (*)
Maria	3
Luigi	4

Raggruppamento per cognome

COGNOME	COUNT (*)
Zavoli	3
Grassi	2
Di Santo	2

Concettualmente diversi:  
prima si raggruppa per  
cognome (risp. Nome) e la  
partizione ottenuta si  
partiziona ulteriormente  
per nome (risp. Cognome).

Il risultato è uguale  
perché la partizione finale  
(cognome-nome) è uguale a  
quella (nome-cognome)

Raggruppamento per  
cognome, nome

COGNOME	NOME	COUNT (*)
Zavoli	Luigi	2
Di Santo	Luigi	2
Zavoli	Maria	1
Grassi	Maria	2

Raggruppamento  
per nome,  
cognome

NOME	COGNOME	COUNT (*)
Maria	Grassi	2
Luigi	Zavoli	2
Luigi	Di Santo	2
Maria	Zavoli	1

## RAGGRUPPAMENTO: misura e dimensione

---

- Posso raggruppare per voto?

Materia	Candidato	Voto	Docente
DA	1	20	10
LFC	2	30	20
MTI	1	30	30
LP	2	20	40

- Cosa succede se calcolo  $\min(voto)$ ?

## RAGGRUPPAMENTO su chiave primaria

---

- Posso raggruppare per matricola?

Nome	<u>Matricola</u>	Provincia	AnnoNascita
Isaia	071523	PI	1982
Rossi	067459	LU	1984
Bianchi	079856	LI	1983
Isaia	075649	PI	1984

- Cosa succede se calcolo count(\*)?

# RAGGRUPPAMENTO su chiave primaria

Studenti

Nome	<u>Matricola</u>	Provincia	AnnoNascita
------	------------------	-----------	-------------

Esami

<u>Materia</u>	<u>Matricola*</u>	Data	Voto
----------------	-------------------	------	------

Ridenominazione  
nella proiezione

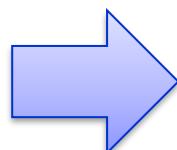
Per ogni studente,  
visualizzare nome,  
cognome e numero  
degli esami sostenuti

Funziona?

$$\pi_{S.Nome, S.Cognome, \text{count}(*)} \text{as } nEsami$$
$$\{nome, cognome\} \gamma \{\text{count}(\cdot)\}$$
$$\bowtie_{S.matricola = E.matricola}$$

Studenti S

Esami E



## RAGGRUPPAMENTO su chiave primaria

Studenti

<b>Nome</b>	<b>Matricola</b>	<b>Provincia</b>	<b>AnnoNascita</b>
-------------	------------------	------------------	--------------------

Esami

<b>Materia</b>	<b>Matricola*</b>	<b>Data</b>	<b>Voto</b>
----------------	-------------------	-------------	-------------

- Per ogni studente, visualizzare matricola, nome, cognome e numero degli esami sostenuti

$\pi_{S.Matricola,S.Nome,S.Cognome,count(*)} as nEsami$

$\{matricola, nome, cognome\} \gamma_{\{count(*)\}}$

$\bowtie_{S.matricola=E.matricola}$

Studenti

Il modello relazionale

U

$\pi_{S.Matricola,S.Nome,S.Cognome,0} as nEsami$

$\pi_{S.Matricola,S.Nome,S.Cognome}$

Studenti

$\pi_{S.Matricola,S.Nome,S.Cognome}$

$\bowtie_{S.matricola=E.matricola}$

Studenti

Esami

4.97

## TRASFORMAZIONI ALGEBRICHE

---

- Basate su regole di equivalenza fra espressione algebriche
- Consentono di scegliere diversi ordini di join e di anticipare proiezioni e restrizioni.
- Alcuni esempi con la relazione  $R(A, B, C, D)$ :

$$\pi_A(\pi_{A,B}(R)) \equiv \pi_A(R)$$

$$\sigma_{C_1}(\sigma_{C_2}(R)) \equiv \sigma_{C_1 \wedge C_2}(R)$$

$$\sigma_{C_1 \wedge C_2}(R \times S) \equiv \sigma_{C_1}(R) \times \sigma_{C_2}(S)$$

$$R \times (S \times T) \equiv (R \times S) \times T$$

$$(R \times S) \equiv (S \times R)$$

$$\sigma_C(x\gamma_F(R)) \equiv x\gamma_F(\sigma_C(R))$$

## Atomizzazione delle selezioni

---

$$\sigma_{F1 \wedge F2}(E) = \sigma_{F1}(\sigma_{F2}(E))$$

Una selezione congiuntiva può essere sostituita  
da una cascata di selezioni atomiche

## Idempotenza delle Proiezioni

---

Una proiezione può essere trasformata in una cascata di proiezioni che eliminano i vari attributi in fasi diverse

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

se E è definita su un insieme di attributi che contiene Y oltre che X.

Non ha effetto sull'efficienza. Ha effetto sulla leggibilità della query.

## Anticipazione della selezione rispetto al Join (Pushing Selection down)

---

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

se F fa riferimento solo ad attributi di E<sub>2</sub>.

Aumenta l'efficienza della query perché la selezione riduce il numero delle righe di E2 prima del join.

## Anticipazione della proiezione rispetto al join (Pushing Projections down)

---

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

Se  $E_1$  e  $E_2$  definite rispettivamente su  $X_1$  e  $X_2$ ,  
 $Y_2 \subseteq X_2$  e gli attributi in  $X_2 - Y_2$  non sono  
coinvolti nel join.

Non ha effetto sull'efficienza ma sulla  
leggibilità

## Distributività della selezione rispetto all'unione

$$\sigma_F(E_1 \cup E_2) = \sigma_F(E_1) \cup \sigma_F(E_2)$$

## Distributività della selezione rispetto alla differenza

---

$$\sigma_F(E_1 - E_2) = \sigma_F(E_1) - \sigma_F(E_2)$$

## Distributività della proiezione rispetto all'unione

---

$$\pi_X(E_1 \cup E_2) = \pi_X(E_1) \cup \pi_X(E_2)$$

# Non Distributività della proiezione rispetto alla differenza

In generale

$$\pi_A(R_1 - R_2) \not\leftrightarrow \pi_A(R_1) - \pi_A(R_2)$$

Se  $R_1$  e  $R_2$  sono definite sull'insieme di attributi  $X=AB$ , e contengono tuple uguali su  $A$  e diverse su  $B$

# Esempio

---

Imp1

Impiegato	Capo
Neri	Mori
Bianchi	Bruni
Verdi	Bini

Imp2

Impiegato	Capo
Neri	Rossi
Bianchi	Bordeaux
Verdi	Blu

$$\pi_A (Imp1 - Imp2) \equiv \pi_A (Imp1) - \pi_A (Imp2) ?$$

Dipende da chi è A....

Se  $R_1$  e  $R_2$  sono definite su AB e contengono tuple uguali su A e diverse su B, NO

## Inglobamento di una selezione in un prodotto cartesiano a formare un join

---

$$\sigma_F(R_1 \bowtie R_2) \equiv R_1 \bowtie_F R_2.$$

## Altre equivalenze

- $\sigma_{F_1 \vee F_2}(R) \equiv \sigma_{F_1}(R) \cup \sigma_{F_2}(R).$
- $\sigma_{F_1 \wedge F_2}(R) \equiv \sigma_{F_1}(R) \cap \sigma_{F_2}(R).$
- $\sigma_{F_1 \wedge \neg F_2}(R) \equiv \sigma_{F_1}(R) - \sigma_{F_2}(R).$

Si noti infine che valgono proprietà commutativa e associativa di tutti gli operatori binari (unione, intersezione, join) tranne la differenza.

## OPERATORI ALGEBRICI NON INSIEMISTICI

---

- $\pi^b_{\{A_i\}}(R)$ : proiezione multiinsiemistica (senza eliminazione dei duplicati)
- $\tau_{\{A_i\}}(R)$ : ordinamento

## Calcolo relazionale su ennuple

---

- Il calcolo relazionale è un linguaggio che permette di definire il risultato di un'interrogazione come l'insieme di quelle ennuple che soddisfano una certa condizione  $\phi$ .
- L'insieme delle matricole degli studenti che hanno superato qualcuno degli esami elencati nella relazione Materie(Materia), si può definire come:  
$$\{t.\text{Matricola} \mid t \in \text{Studenti}, \exists m \in \text{Materie}. \exists e \in \text{ProveEsami}. e.\text{Candidato} = t.\text{Matricola} \wedge e.\text{Materia} = m.\text{Materia}\}$$
- Che è equivalente a

$$\pi_{\text{Matricola}}(\text{Studenti} \bowtie_{\text{Matricola} = \text{Candidato}} (\text{ProveEsami} \bowtie \text{Materie}))$$

## Esercizio 3 - Trovare le matricole dei capi

---

IMPIEGATI(Matricola, Nome, Età, Stipendio)

Supervisione(Impiegato\*, Capo\*)

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



## Esercizio 3 - Trovare le matricole dei capi - Soluzione

IMPIEGATI(Matricola,  
Nome, Età, Stipendio)

Supervisione(Impiegato\*,  
Capo\*)

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

Impiegati

Matricola	Nome	Età	Stipendio
7309	Rossi	34	45
5998	Bianchi	37	38
9553	Neri	42	35
5698	Bruni	43	42
4076	Mori	45	50
8123	Lupi	46	60

Supervisione

Impiegato	Capo
7309	5698
5998	5698
9553	4076
5698	4076
4076	8123

## Esercizio 3 - Trovare le matricole dei capi - Soluzione

---

IMPIEGATI(Matricola, Nome, Età, Stipendio)

Supervisione(Impiegato\*, Capo\*)

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

$\pi_{\text{matricole}}($

$\pi_{\text{Capo}}(\text{Supervisione}$

$\bowtie \text{Impiegato} = \text{Matricola} (\sigma_{\text{Stipendio} > 40}(\text{Impiegati})) )$

## Esercizio 4 - Trovare nome e stipendio dei capi

---

IMPIEGATI(Matricola, Nome, Età, Stipendio)

Supervisione(Impiegato\*, Capo\*)

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

$\pi_{\text{Nome}, \text{Stipendio}} ($

Impiegati  $\bowtie$  Matricola=Capo

$\pi_{\text{Capo}} (\text{Supervisione}$

$\bowtie$  Impiegato=Matricola ( $\sigma_{\text{Stipendio} > 40} (\text{Impiegati})$ ) )

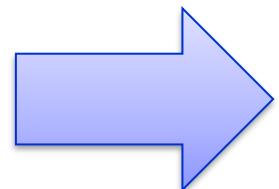
## Esercizio 5

---

IMPIEGATI(Matricola, Nome, Età, Stipendio)

Supervisione(Impiegato\*, Capo\*)

Trovare le matricole dei capi i cui impiegati guadagnano **tutti** più di 40 mila euro



## Esercizio 5

IMPIEGATI(Matricola, Nome, Età, Stipendio)

Supervisione(Impiegato\*, Capo\*)

- Trovare le matricole dei capi i cui impiegati guadagnano **tutti** più di 40 mila euro

Non si può esprimere direttamente nell'algebra, poiché mancano i "quantificatori universali")

$\pi_{\text{Capo}}(\text{Supervisione})$  -

$\pi_{\text{Capo}}(\text{Supervisione}$

$\bowtie$  Impiegato=Matricola

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

Assumiamo per semplicità che non ci siano valori NULL in Stipendio

Capi che hanno almeno un Impiegato che guadagna meno di 40

## Esercizio 6

IMPIEGATI(Matricola, Nome, Età, Stipendio)

Supervisione(Impiegato\*, Capo\*)

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

$\pi_{\text{Matr}, \text{Nome}, \text{Stip}, \text{MatrC}, \text{NomeC}, \text{StipC}} ($

$\sigma_{\text{Stipendio} > \text{StipC}} ($

$\rho_{\text{MatrC}, \text{NomeC}, \text{StipC}, \text{EtàC}} \leftarrow \text{Matr}, \text{Nome}, \text{Stip}, \text{Età} (\text{Impiegati})$

$\bowtie \text{ MatrC} = \text{Capo}$

(Supervisione  $\bowtie$  Impiegato=Matricola Impiegati)) )

## Esercizio 7

**IMPIEGATI(Matricola, Nome, Età, Stipendio)**

**Supervisione(Impiegato\*, Capo\*)**

Trovare quali sono gli impiegati che hanno stipendio **massimo**

$\pi_{\text{Matricola}}(\text{Impiegati}) -$

$\pi_{\text{Matricola}}(\text{Impiegati}$

$\bowtie \text{ Stip} < \text{Stip1}$

$(P_{\text{Matr1}, \text{Nome1}, \text{Eta1}, \text{Stip1}} \leftarrow \text{Matr, Nome, Stip, Età}$   
**(Impiegati))**



Partiamo dagli  
impiegati che  
**non** hanno lo  
stipendio  
massimo

---

# La Normalizzazione

Materiale adattato dal libro Albano et al e  
dal libro Atzeni-et al., Basi di dati

---

# Parte I

# TEORIA RELAZIONALE: INTRODUZIONE

---

- Due metodi per produrre uno schema relazionale:
  - a) Partire da un buon schema a oggetti e tradurlo
  - b) Partire da uno schema relazionale fatto da altri e modificarlo o completarlo
- Teoria della progettazione relazionale: studia cosa sono le "anomalie" e come eliminarle (**normalizzazione**).
- È particolarmente utile se si usa il metodo (b). È moderatamente utile anche quando si usa il metodo (a).

# UNA TABELLA

---

N Inv	Stanza	Resp	Oggetto	Produttore	Descrizione
1012	256	Ghelli	Mac Mini	Apple	Personal Comp
1015	312	Albano	Dell XPS M1330	Dell	Notebook 2 GHZ
1034	256	Ghelli	Dell XPS M1330	Dell	Notebook 2GB
1112	288	Leoni	Mac Mini 2	Apple	Personal Comp

È fatta male? Perché? Come si può correggere?

## SCHEMI CON ANOMALIE

---

- Esempio:
  - StudentiEdEsami(Matricola, Nome, Provincia, AnnoNascita, Materia, Voto)
- Anomalie:
  - Ridondanze
  - Potenziali inconsistenze
  - Anomalie nelle inserzioni
  - Anomalie nelle eliminazioni

## SCHEMI CON ANOMALIE

---

- Esempio:
  - StudentiEdEsami(Matricola, Nome, Provincia, AnnoNascita, Materia, Voto)
- Anomalie:
  - Ridondanze
  - Potenziali inconsistenze
  - Anomalie nelle inserzioni/eliminazioni
- Soluzione: dividiamo lo schema in due tavelle.
  - Studenti ( Matricola, **Nome**, Provincia, AnnoNascita)
  - Esami (**Nome**, Materia, Voto)



Va bene?

## SCHEMI CON ANOMALIE

---

- Esempio:
  - StudentiEdEsami(Matricola, Nome, Provincia, AnnoNascita, Materia, Voto)
- Anomalie:
  - Ridondanze
  - Potenziali inconsistenze
  - Anomalie nelle inserzioni/eliminazioni
- Soluzione: dividiamo lo schema in due tavelle.
  - Studenti ( **Matricola**, Nome, Provincia, AnnoNascita)
  - Esami ( **Matricola**, Materia, Voto)



## SCHEMI CON ANOMALIE

---

- Esempio:
  - StudentiEdEsami(Matricola, Nome, Provincia, AnnoNascita, Materia, Voto)
  - Quali sono le relazioni fra i diversi campi?

## OBIETTIVI

---

- Nozione base: dipendenze funzionali
- Obiettivi della teoria:
  - **Equivalenza** di schemi: in che misura si può dire che uno schema rappresenta un altro
  - **Qualità** degli schemi (forme normali)
  - **Trasformazione** degli schemi (normalizzazione di schemi)
- Ipotesi dello **schema di relazione universale**:
  - Tutti i fatti sono descritti da attributi di un'unica relazione (**relazione universale**), cioè gli attributi hanno un significato globale.
  - Definizione: Lo schema di **relazione universale** U di una base di dati relazionale ha come attributi l'unione degli attributi di tutte le relazioni della base di dati.

## Obiettivi: Forme normali

---

- Una **forma normale** è una proprietà di una base di dati relazionale che ne garantisce la “qualità”, cioè l'assenza di determinati difetti
- Quando una relazione **non è normalizzata**:
  - presenta ridondanze,
  - si presta a comportamenti poco desiderabili durante gli aggiornamenti
- La **normalizzazione** è una procedura che permette di trasformare schemi non normalizzati in schemi che soddisfano una forma normale

## Perché questi fenomeni indesiderabili? - Parte I

<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

### Ridondanza

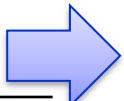
Lo stipendio di ciascun impiegato è ripetuto in tutte le ennuple relative.

Questo perché lo stipendio dipende solo dall'Impiegato.

Il costo del bilancio per ogni progetto è ripetuto.

### Anomalia di aggiornamento

Se lo stipendio di un impiegato varia, è necessario andarne a modificare il valore in diverse ennuple



## Perché questi fenomeni indesiderabili? - Parte II

<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

Posso consentire valori null per il progetto?

### Anomalia di cancellazione

Se un impiegato interrompe la partecipazione a tutti i progetti, dobbiamo cancellare tutte le ennuple in cui appare, e in questo modo l'impiegato non sarà più presente nel database

### Anomalia di inserimento

Un nuovo impiegato non può essere inserito finché non gli viene assegnato un progetto

# Linee Guida per una corretta progettazione - Parte I

---

## Semantica degli attributi

- Si progetti ogni schema relazionale in modo tale che **sia semplice spiegarne il significato**. Non si uniscano attributi provenienti da più tipi di classi e tipi di associazione in una unica relazione.

## Ridondanza

- Si progettino gli schemi relazionale in modo che nelle relazioni **non siano presenti anomalie di inserimento, cancellazione o modifica**. Se sono presenti delle anomalie (che si vuole mantenere), le si rilevi chiaramente e ci si assicuri che i programmi che aggiornano la base di dati operino correttamente

## Linee Guida per una corretta progettazione - Parte II

---

### Valori Nulli

- Per quanto possibile, si eviti di porre in relazione di base attributi i cui valori possono essere (frequentemente) nulli. Se è inevitabile, ci si assicuri che essi si presentino solo in casi eccezionali e che non riguardino una maggioranza di tuple nella relazione

### Tuple spurie

- Si progettino schemi di relazione in modo tale che essi possano essere riuniti tramite **JOIN** con condizioni di uguaglianza su attributi che sono o **chiavi primarie** o **chiavi esterne** in modo da garantire che non vengano generate tuple spurie. Non si abbiano relazioni che contengono attributi di «accoppiamento» diversi dalle combinazioni chiave esterna-chiave primaria.

## DIPENDENZE FUNZIONALI

---

- Per formalizzare la nozione di schema senza anomalie, occorre una descrizione formale della **semantica** dei fatti rappresentati in uno schema relazionale.
  - ◦ ◦
- **Istanza valida di R:** è una nozione **semantica**, che dipende da ciò che sappiamo del dominio del discorso (non estensionale, non deducibile da alcune istanze dello schema)
- Nozione fondamentale: **dipendenza funzionale**

## Dipendenza funzionale (informale)

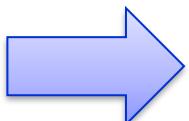
---

- Istanza valida  $r$  su  $R(T)$
- Siano  $X$  e  $Y$  due sottoinsiemi non vuoti di  $T$
- esiste in  $r$  una dipendenza funzionale da  $X$  a  $Y$  se, per ogni coppia di ennuple  $t_1$  e  $t_2$  di  $r$  con gli stessi valori su  $X$ , risulta che  $t_1$  e  $t_2$  hanno gli stessi valori anche su  $Y$
- La dipendenza funzionale da  $X$  a  $Y$  si denota con  $X \rightarrow Y$

Esempio:

Persone (CodiceFiscale, Cognome, Nome, DataNascita)

CodiceFiscale  $\rightarrow$  Cognome, Nome



## Dipendenza funzionale vs chiave - Parte I

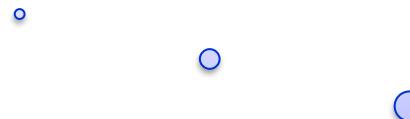
---

- Istanza valida r su R(T) - Siano X e Y due sottoinsiemi non vuoti di T
- esiste in r una dipendenza funzionale da X a Y ( $X \rightarrow Y$ ) se, per ogni coppia di ennuple  $t_1$  e  $t_2$  di r con gli stessi valori su X, risulta che  $t_1$  e  $t_2$  hanno gli stessi valori anche su Y

Esempio:

StudentiEdEsami(Matricola, Nome, Provincia, AnnoNascita,  
Materia, Voto)

Matricola  $\rightarrow$  Nome, Provincia, AnnoNascita



## DIPENDENZE FUNZIONALI (formalmente)

---

- Dato uno schema  $R(T)$  e  $X, Y \subseteq T$ , una **dipendenza funzionale** ( DF ) fra gli attributi  $X$  e  $Y$ , è un vincolo su  $R$  sulle istanze della relazione, espresso nella forma

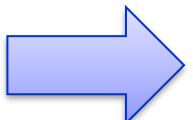
$$X \rightarrow Y,$$

i.e.  $X$  determina funzionalmente  $Y$  o  $Y$  è determinato da  $X$ , se per ogni istanza valida  $r$  di  $R$  un valore di  $X$  determina in modo univoco un valore di  $Y$ :

$\forall r$  istanza valida di  $R$ .

$$\forall t_1, t_2 \in r. \text{ se } t_1[X] = t_2[X] \text{ allora } t_1[Y] = t_2[Y]$$

- In altre parole: un'istanza  $r$  di  $R(T)$  soddisfa la dipendenza  $X \rightarrow Y$ , (o  $X \rightarrow Y$  vale in  $r$ ), se per ogni coppia di ennuple  $t_1$  e  $t_2$  di  $r$ ,  
se  $t_1[X] = t_2[X]$  allora  $t_1[Y] = t_2[Y]$



## Esempio: dipendenze funzionale

Esiste la DF  
Dipartimento → Indirizzo?

∀ r istanza valida di R.

✓  $\forall t_1, t_2 \in r. \text{ se } t_1[X] = t_2[X] \text{ allora } t_1[Y] = t_2[Y]$

CodCorso	Titolo	CFU	Anno	Semestre	Codice Docente	Dipartimento	Indirizzo
1	Basi di Dati	6	2022	I	A1	Informatica	Via Roma
2	Basi di Dati	6	2023	II	A4	Informatica	Via Roma
3	Algebra	12	2022	I	A1	Informatica	Via Roma
4	Algebra	12	2023	I	A4	Informatica	Via Roma
5	Basi di Dati	6	2021	I	A1	Informatica	Via Roma
6	Basi di Dati	6	2020	I	A1	Informatica	Via Roma
7	Algebra	12	2023	II	A4	Matematica	Via Bianchi

Basi di Dati: la normalizzazione titolo → Semestre?

Esiste la DF

Esiste la DF  
Titolo, Anno → Semestre?

## Esempio

---

$\forall r$  istanza valida di R.

$$\forall t_1, t_2 \in r. \text{ se } t_1[X] = t_2[X] \text{ allora } t_1[Y] = t_2[Y]$$

- Questa tabella soddisfa la dipendenza funzionale

Matricola  $\rightarrow$  Cognome

Matricola	Cognome
1	Rossi
2	Verdi
3	Rossi
4	Viola

## DIPENDENZE FUNZIONALI vs Chiave - Esempio

---

$\forall r$  istanza valida di R.

$\forall t1, t2 \in r. \text{ se } t1[X] = t2[X] \text{ allora } t1[Y] = t2[Y]$

Esempio:

StudentiEdEsami(Matricola, Nome, Provincia, AnnoNascita, Materia, Voto)

- Matricola è una chiave?
- Materia è una chiave?
- Matricola  $\rightarrow$  Nome, Provincia, AnnoNascita
- Esempio:

Studenti ( Matricola, Nome, Provincia, AnnoNascita)

---

## DIPENDENZE FUNZIONALI

---

- Dato uno schema  $R(T)$  e  $X, Y \subseteq T$ , una dipendenza funzionale ( DF ) è un vincolo su  $R$  del tipo  $X \rightarrow Y$ ,

$\square \forall r$  istanza valida di  $R$ .

$$\forall t_1, t_2 \in r. \text{ se } t_1[X] = t_2[X] \text{ allora } t_1[Y] = t_2[Y]$$

Si dice che

- un'istanza  $r_0$  di  $R$  soddisfa la DF  $X \rightarrow Y$  ( $r_0 \models X \rightarrow Y$ ) se
    - la proprietà vale per  $r_0$ :  $\forall t_1, t_2 \in r. \text{ se } t_1[X] = t_2[X] \text{ allora } t_1[Y] = t_2[Y]$
  - e che un'istanza  $r_0$  di  $R$  soddisfa un insieme  $F$  di DF
    - se per ogni  $X \rightarrow Y \in F$ , vale  $r_0 \models X \rightarrow Y$ :
- $$r_0 \models X \rightarrow Y \text{ sse } \forall t_1, t_2 \in r_0. \text{ se } t_1[X] = t_2[X] \text{ allora } t_1[Y] = t_2[Y]$$

## Esempio

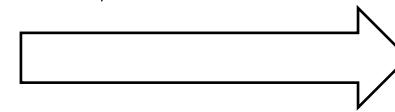
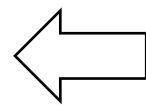
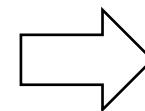
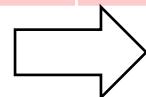
<u>Impiegato</u>	<u>Stipendio</u>	<u>Progetto</u>	<u>Bilancio</u>	<u>Funzione</u>
Rossi	20	Marte	2	tecnico
Verdi	35	Giove	15	progettista
Verdi	35	Venere	15	progettista
Neri	55	Venere	15	direttore
Neri	55	Giove	15	consulente
Neri	55	Marte	2	consulente
Mori	48	Marte	2	direttore
Mori	48	Venere	15	progettista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

Abbiamo usato un'unica relazione per rappresentare informazioni eterogenee

- gli impiegati con i relativi stipendi ( $\text{Impiegato} \rightarrow \text{Stipendio}$ )
- i progetti con i relativi bilanci ( $\text{Progetto} \rightarrow \text{Bilancio}$ )
- le partecipazioni degli impiegati ai progetti con le relative funzioni ( $\text{Impiegato Progetto} \rightarrow \text{Funzione}$ ). Anomala?

# UNA TABELLA

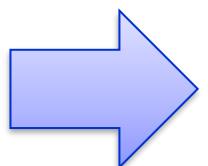
N Inv	Stanza	Resp	Oggetto	Produttore	Descrizione
1012	256	Ghelli	Mac Mini	Apple	Personal Comp
1015	312	Albano	Dell XPS M1330	Dell	Notebook 2 GHZ
1034	256	Ghelli	Dell XPS M1330	Dell	Notebook 2GB
1112	288	Leoni	Mac Mini 2	Apple	Personal Comp



## ESEMPIO - Parte I

---

- DotazioniLibri(CodiceLibro, NomeNegozio, IndNegozio, Titolo, Quantità)
- DF:  
 $\{ \text{CodiceLibro} \rightarrow \text{Titolo}$   
 $\text{NomeNegozio} \rightarrow \text{IndNegozio}$   
 $\text{CodiceLibro}, \text{NomeNegozio} \rightarrow \text{IndNegozio}, \text{Titolo}, \text{Quantità} \}$



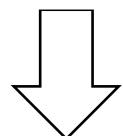
## Dipendenze funzionali atomiche - Asimmetria - Esempio Parte II

---

Ogni dipendenza funzionale  $X \rightarrow A_1 A_2 \dots A_n$ , si può scomporre nelle dipendenze funzionali  $X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_n$

Le dipendenze funzionali del tipo  $X \rightarrow A$  si chiamano **dipendenze funzionali atomiche**.

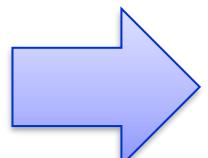
- DotazioniLibri(CodiceLibro, NomeNegozio, IndNegozio, Titolo, Quantità)  
 $\text{CodiceLibro}, \text{NomeNegozio} \rightarrow \text{IndNegozio}, \text{Titolo}, \text{Quantità}$



$\text{CodiceLibro}, \text{NomeNegozio} \rightarrow \text{IndNegozio},$

$\text{CodiceLibro}, \text{NomeNegozio} \rightarrow \text{Titolo}$

$\text{CodiceLibro}, \text{NomeNegozio} \rightarrow \text{Quantità}$



## Esempio Parte III - Dipendenza funzionali e ridondanza

---

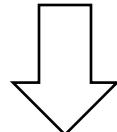
DotazioniLibri(CodiceLibro, NomeNegozio, IndNegozio, Titolo, Quantità)

- DF:

{ CodiceLibro → Titolo

NomeNegozio → IndNegozio

CodiceLibro, NomeNegozio → IndNegozio, Titolo, Quantità }



{ CodiceLibro → Titolo

NomeNegozio → IndNegozio

CodiceLibro, NomeNegozio → Quantità }

## Altro esempio - Parte I

Articoli (Kit, Componente, Tipo, QuantComp, PrezzoComp, Fornitore, PrezzoTot)

Kit	Componente	Tipo	QuantComp	PrezzoComp	Fornitore	PrezzoTot
Libreria	Legno	Noce	50	10	A	4400
Libreria	Bulloni	Acciaio	200	1	B	4400
Libreria	Vetro	Cristallo	3	50	C	4400
Scaffale	Legno	Mogano	37	15	A	555
PC	Bulloni	Acciaio	25	1	B	700
PC	Tastiera	A3000	3	30	D	700
PC	Mouse	B2000	5	45	D	700
Scrivania	Legno	Noce	10	8	B	500
Scrivania	Maniglie	Rame	10	24	B	500
Tavolo	Legno	Noce	4	10	A	600
	...		...			...

PrezzoTot è il Prezzo di vendita

Assumiamo che:

Il **tipo** si riferisce ad una sola componente

Chiave:  
**Kit, Tipo**

Sono chiavi:

- Quantcomp, PrezzoComp
  - Tipo, PrezzoTot
- ???

Quali sono le dipendenze funzionali?

## Altro esempio - Parte II

Assumiamo che:  
Il tipo si riferisce ad una sola componente

Ridondanze:

- PrezzoTot è ripetuto in ogni tupla che si riferisce allo stesso kit
- PrezzoComp è ripetuto in ogni tupla che ha lo stesso valore di Tipo e Fornitore
- Componente è ripetuto in ogni tupla che ha lo stesso Tipo

Quali sono le dipendenze funzionali?

- Tipo → Componente
- Kit → PrezzoTot
- Kit,Tipo → PrezzoComponente, QuantComp, Fornitore

Kit	Componente	Tipo	QuantC omp	Prezzo Comp	Fornit ore	Prezz oTot
Libreria	Legno	Noce	50	10	A	4400
Libreria	Bulloni	Acciaio	200	1	B	4400
Libreria	Vetro	Cristallo	3	50	C	4400
Scaffale	Legno	Mogano	37	15	A	555
PC	Bulloni	Acciaio	25	1	B	700
PC	Tastiera	A3000	3	30	D	700
PC	Mouse	B2000	5	45	D	700
Scrivania	Legno	Noce	10	8	B	500
Scrivania	Maniglie	Rame	10	24	B	500
Tavolo	Legno	Noce	4	10	A	600
	...		...			...



## Altro esempio - Parte III - Decomposizione

---

- Una decomposizione della relazione che non presenti ridondanze e senza perdita di informazione
- Dipendenze funzionali
  - Tipo → Componente
  - Kit → PrezzoTot
  - Kit,Tipo → PrezzoComponente, QuantComp, Fornitore

Kit	Componente	Tipo	QuantC omp	Prezzo Comp	Fornitore	Prezz oTot
-----	------------	------	---------------	----------------	-----------	---------------

Kit	Tipo	QuantCo mp	Fornitore	Tipo	Prezzo Comp	Forn itore	Kit	Prezzo Tot	Tipo	Compone nte
-----	------	---------------	-----------	------	----------------	---------------	-----	---------------	------	----------------

## Altro esempio - Dipendenza funzionali banali

---

<b><u>Impiegato</u></b>	<b>Stipendio</b>	<b><u>Progetto</u></b>	<b>Bilancio</b>	<b>Funzione</b>
-------------------------	------------------	------------------------	-----------------	-----------------

La dipendenza funzionale del tipo

Impiegato Progetto → Progetto

è sempre valida, per cui si tratta di una DF "banale"

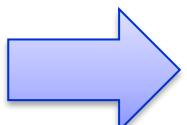
X → A è non banale se A non è contenuta in X

Siamo interessati alle dipendenze funzionali non banali.

# ESPRIMERE LE DIPENDENZE FUNZIONALI

---

- Consideriamo:  $\text{NomeNegozio} \rightarrow \text{IndNegozio}$
- Espressione diretta ( $P \Rightarrow Q$ ):
  - Se in due righe il  $\text{NomeNegozio}$  è uguale, anche l' $\text{IndNegozio}$  è uguale:
    - $\text{NomeNegozio}_\equiv \Rightarrow \text{IndNegozio}_\equiv$
- Per contrapposizione ( $\neg Q \Rightarrow \neg P$ ):
  - Se l' $\text{IndNegozio}$  è diverso allora il  $\text{NomeNegozio}$  è diverso:
    - $\text{IndNegozio}_\neq \Rightarrow \text{NomeNegozio}_\neq$
- Per assurdo:
  - Non possono esserci due nuple in  $\text{DotazioniLibri}$  con  $\text{NomeNegozio}$  uguale e  $\text{IndNegozio}$  diverso:
    - $\text{Not } (\text{NomeNegozio}_\equiv \wedge \text{IndNegozio}_\neq)$
    - $\text{NomeNegozio}_\equiv \wedge \text{IndNegozio}_\neq \Rightarrow \text{False}$



# MANIPOLAZIONE DI CLAUSOLE

---

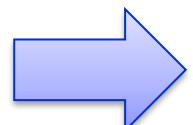
- Sono equivalenti:
  - $\text{NomeNegozi}_\equiv \Rightarrow \text{IndNegozi}_\equiv$
  - $\text{IndNegozi}_\neq \Rightarrow \text{NomeNegozi}_\neq$
  - $\text{NomeNegozi}_\equiv \wedge \text{IndNegozi}_\neq \Rightarrow \text{False}$
- In generale:
  - $A \Rightarrow B \Leftrightarrow A \wedge \neg B \Rightarrow \text{False} \Leftrightarrow \neg B \Rightarrow \neg A$
- Più in generale, in ogni clausola  $A \wedge B \Rightarrow E \vee F$  posso spostare le sottoformule da un lato all'altro, negandole
- Quindi sono equivalenti:
  - $\text{NomeNegozi}_\equiv \wedge \text{CodiceLibro}_\equiv \Rightarrow \text{Quantità}_\equiv$
  - $\text{NomeNegozi}_\equiv \wedge \text{CodiceLibro}_\equiv \wedge \text{Quantità}_\neq \Rightarrow \text{False}$
  - $\text{CodiceLibro}_\equiv \wedge \text{Quantità}_\neq \Rightarrow \text{NomeNegozi}_\neq$
  - $\text{NomeNegozi}_\equiv \wedge \text{Quantità}_\neq \Rightarrow \text{CodiceLibro}_\neq$

Importante!

## ESEMPIO CdL

---

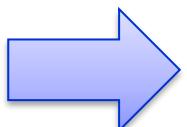
- Orari(CodAula, NomeAula, Piano, Posti, Materia, CDL, Docente, Giorno, OraInizio, OraFine)
- In un dato momento, un docente si trova al più in un'aula
- Non è possibile che due docenti diversi siano nella stessa aula contemporaneamente
- Se due lezioni si svolgono su due piani diversi appartengono a due corsi di laurea diversi
- Se due lezioni diverse si svolgono lo stesso giorno per la stessa materia, appartengono a due CDL diversi (lezioni diverse:  $\text{not}(\text{CodAula}_\_ \wedge \text{NomeAula}_\_ \wedge \dots)$ )



## ESEMPIO CdL - Prima domanda

---

- Orari(CodAula, NomeAula, Piano, Posti, Materia, CDL, Docente, Giorno, OraInizio, OraFine)
  - In un dato momento, un docente si trova al più in un'aula.
- 
- Due domande:
    - Che vuole dire momento?
    - Che tipo di implicazione vi aspettate?

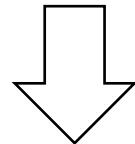


## Soluzione ESEMPIO CdL - 1

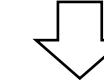
- Orari(CodAula, NomeAula, Piano, Posti, Materia, CDL, Docente, Giorno, OraInizio, OraFine)
- In un dato momento, un docente si trova al più in un'aula.

$\text{NOT} ([\text{Giorno}, \text{OraInizio}]_ \subseteq \wedge \text{Docente}_ \subseteq \Rightarrow \text{Aula}_ \neq) \Rightarrow \text{FALSE}$

$\text{NOT} ([\text{Giorno}, \text{OraFine}]_ \subseteq \wedge \text{Docente}_ \subseteq \Rightarrow \text{Aula}_ \neq) \Rightarrow \text{FALSE}$



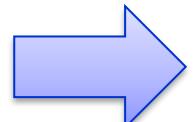
$A \wedge B \wedge C_ \neq \Rightarrow \text{FALSE}$



$A \wedge B \Rightarrow C_ \subseteq$

$[\text{Giorno}, \text{OraInizio}]_ \subseteq \wedge \text{Docente}_ \subseteq \Rightarrow \text{Aula}_ \subseteq$

$[\text{Giorno}, \text{OraFine}]_ \subseteq \wedge \text{Docente}_ \subseteq \Rightarrow \text{Aula}_ \subseteq$



## Soluzione ESEMPIO CdL - 2

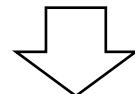
---

- Orari(CodAula, NomeAula, Piano, Posti, Materia, CDL, Docente, Giorno, OraInizio, OraFine)
- Non è possibile che due docenti diversi siano nella stessa aula contemporaneamente

Non posso avere:

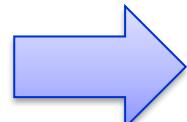
$$[\text{Giorno}, \text{OraInizio}]_e \wedge \text{Aula}_e \wedge \text{Docente}_x \Rightarrow \text{FALSE}$$

$$[\text{Giorno}, \text{OraFine}]_e \wedge \text{Aula}_e \wedge \text{Docente}_x \Rightarrow \text{FALSE}$$



$$[\text{Giorno}, \text{OraInizio}]_e \wedge \text{Aula}_e \Rightarrow \text{Docente}_e$$

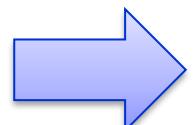
$$[\text{Giorno}, \text{OraFine}]_e \wedge \text{Aula}_e \Rightarrow \text{Docente}_e$$



## Soluzione ESEMPIO CdL - 3

---

- Orari(CodAula, NomeAula, Piano, Posti, Materia, CdL, Docente, Giorno, OraInizio, OraFine)
  - Se due lezioni si svolgono su due piani diversi appartengono a due corsi di laurea diversi
- 
- $\text{Piano}_z \Rightarrow \text{CdL}_z$
  - Che equivale a
  - $\text{CdL}_z \Rightarrow \text{Piano}_z$



## Soluzione ESEMPIO CdL - 4

- Orari(CodAula, NomeAula, Piano, Posti, Materia, CDL, Docente, Giorno, OraInizio, OraFine)
- Se due lezioni diverse si svolgono lo stesso giorno per la stessa materia, appartengono a due CDL diversi (lezioni diverse:  $\text{not}(\text{CodAula}_1 \wedge \text{NomeAula}_1 \wedge \dots)$ )
- $\text{Materia}_1 \wedge \text{Giorno}_1 \Rightarrow \text{CdL}_1 ???$
- $\text{Lezioni}_1 \wedge \text{Materia}_1 \wedge \text{Giorno}_1 \Rightarrow \text{CdL}_1$  cioè  
 $\text{Materia}_1 \wedge \text{Giorno}_1 \wedge \text{CdL}_1 \Rightarrow \text{Lezioni}_1$
- $\text{Materia}_1 \wedge \text{Giorno}_1 \wedge \text{CdL}_1 \Rightarrow \text{CodAula}, \text{NomeAula}, \text{Piano}, \text{Posti}, \text{Materia}, \text{CDL}, \text{Docente}, \text{Giorno}, \text{OraInizio}, \text{OraFine}$
- $\text{Materia}_1 \wedge \text{Giorno}_1 \wedge \text{CdL}_1 \Rightarrow \text{CodAula}, \text{NomeAula}, \text{Piano}, \text{Posti}, \text{Docente}, \text{OraInizio}, \text{OraFine}$



## Riepilogo

---

- Qualità degli schemi relazionali
- Anomalie
- Dipendenze funzionali  $X \rightarrow Y$ 
  - $r \models X \rightarrow Y$  se  $\forall t1, t2 \in r$ . se  $t1[X] = t2[X]$  allora  $t1[Y] = t2[Y]$
  - $r$  soddisfa un insieme  $F$  di DF ( $r \models F$ )
  - $F \models X \rightarrow Y$

Attenzione: una dipendenza funzionale è un vincolo.

Non vi possono essere gradi di libertà!

---

# **PARTE II**

# DIPENDENZE FUNZIONALI

---

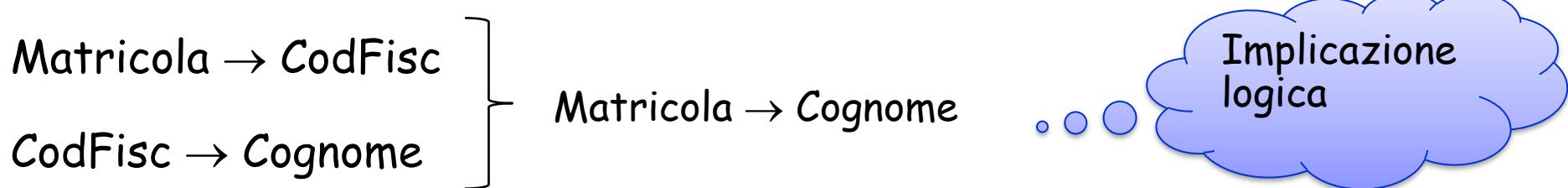
- Notazione:
  - $R \langle T, F \rangle$  denota uno schema con attributi  $T$  e dipendenze funzionali  $F$ .
- Le DF sono una proprietà semantica, cioè dipendono dai fatti rappresentati e non da come gli attributi sono combinati in schemi di relazione.
- Si parla di DF **completa** quando  $X \rightarrow Y$  e per ogni  $W \subset X$ , non vale  $W \rightarrow Y$ .
- Se  $X$  è una **superchiave**, allora  $X$  determina ogni altro attributo della relazione:  $X \rightarrow T$
- Se  $X$  è una **chiave**, allora  $X \rightarrow T$  è una DF completa

## PROPRIETÀ DELLE DF

---

Da un insieme  $F$  di DF, in generale altre DF sono 'implicate' da  $F$ .

Esempio:



- Dipendenze implicate (definizione):

Sia  $F$  un insieme di DF sullo schema  $R$ , diremo che

$F$  implica logicamente  $X \rightarrow Y$  ( $F \models X \rightarrow Y$ ),

se ogni istanza  $r$  di  $R$  che soddisfa  $F$  soddisfa anche  $X \rightarrow Y$ .

- Dipendenze banali:

implicate dal vuoto, es.  $\{\} \models X \rightarrow X$

## ESEMPIO

---

- Sia  $r$  un'istanza di  $R\langle T, F \rangle$ , con  $F = \{X \rightarrow Y, X \rightarrow Z\}$  e  $X, Y, Z \subseteq T$ .

Sia  $X' \subseteq X$ . Altre DF sono soddisfatte da  $r$ ,

- ad es.

- $X \rightarrow X'$  (DF banale) e

- $X \rightarrow YZ$ , infatti

$$t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$$

$$t_1[X] = t_2[X] \Rightarrow t_1[Z] = t_2[Z]$$

$$t_1[X] = t_2[X] \Rightarrow t_1[YZ] = t_2[YZ]$$

Pertanto  $\{X \rightarrow Y, X \rightarrow Z\} \models X \rightarrow YZ$

- Altro esempio:  $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$
- $\models$  denota l'implicazione logica

## REGOLE DI INFERNZA

---

- Come derivare DF implicate logicamente da F?
  - usando un insieme di regole di inferenza.
- "Assiomi" (sono in realtà regole di inferenza) di Armstrong:
  - Se  $Y \subseteq X$ , allora  $X \rightarrow Y$  (Riflessività R)
  - Se  $X \rightarrow Y$ ,  $Z \subseteq T$ , allora  $XZ \rightarrow YZ$  (Arricchimento A)
  - Se  $X \rightarrow Y$ ,  $Y \rightarrow Z$ , allora  $X \rightarrow Z$  (Transitività T)

# DERIVAZIONE

## Definizione

Sia  $F$  un insieme di DF, diremo che  $X \rightarrow Y$  sia derivabile da  $F$  ( $F \vdash X \rightarrow Y$ ), se  $X \rightarrow Y$  può essere inferito da  $F$  usando gli assiomi di Armstrong.

- Si dimostra che valgono anche le regole:

- $\{X \rightarrow Y, X \rightarrow Z\} \vdash X \rightarrow YZ$  (**unione U**)
- $Z \subseteq Y \quad \{X \rightarrow Y\} \vdash X \rightarrow Z$  (**decomposizione D**)

- L'unione:  $\{X \rightarrow Y, X \rightarrow Z\} \vdash X \rightarrow YZ$  (**unione U**)

1.  $X \rightarrow Y$  (per ipotesi)
2.  $X \rightarrow XY$  (per arricchimento da 1)
3.  $X \rightarrow Z$  (per ipotesi)
4.  $XY \rightarrow YZ$  (per arricchimento da 3)
5.  $X \rightarrow YZ$  (per transitività da 2, 4)

Se  $Y \subseteq X$ , allora  $X \rightarrow Y$  (**Riflessività R**)

Se  $X \rightarrow Y, Z \subseteq T$ ,  
allora  $XZ \rightarrow YZ$   
(**Arricchimento A**)

Se  $X \rightarrow Y, Y \rightarrow Z$ ,  
allora  $X \rightarrow Z$   
(**Transitività T**)

# DERIVAZIONE

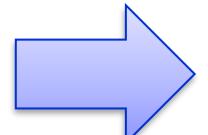
---

## Definizione

- Sia  $F$  un insieme di DF, diremo che  $X \rightarrow Y$  sia derivabile da  $F$  ( $F \vdash X \rightarrow Y$ ), se  $X \rightarrow Y$  può essere inferito da  $F$  usando gli assiomi di Armstrong.
- Una **derivazione** di  $f$  da  $F$  è una sequenza finita  $f_1, \dots, f_m$  di dipendenze, dove  $f_m = f$  e 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.

Si dimostra che valgono anche le regole:

- $\{X \rightarrow Y, X \rightarrow Z\} \vdash X \rightarrow YZ$  (**unione U**)
- $Z \subseteq Y \quad \{X \rightarrow Y\} \vdash X \rightarrow Z$  (**decomposizione D**)
- Da **Unione e Decomposizione** si ricava che se  $Y = A_1A_2\dots A_n$  allora
  - $X \rightarrow Y \Leftrightarrow \{X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_n\}$



## ESEMPIO

---

- $R(A B C D)$
- $F = \{A \rightarrow B, BC \rightarrow D\}$
- $AC$  è una superchiave? Ovvero  $AC \rightarrow ABCD$  ?

1.  $A \rightarrow B$  ipotesi 1
2.  $AC \rightarrow BC$  da 1 per Arr (C)
3.  $BC \rightarrow D$  ipotesi 2
4.  $BC \rightarrow BCD$  da 3 per Arr (BC)
5.  $AC \rightarrow BCD$  da 2+4 per Trans
6.  $AC \rightarrow ABCD$  da 5 per Arr (A)

Se  $Y \subseteq X$ , allora  $X \rightarrow Y$  (Riflessività R)

Se  $X \rightarrow Y, Z \subseteq T$ ,  
allora  $XZ \rightarrow YZ$   
(Arricchimento Arr)

Se  $X \rightarrow Y, Y \rightarrow Z$ ,  
allora  $X \rightarrow Z$   
(Transitività Trans)

# CORRETTEZZA E COMPLETEZZA DEGLI ASSIOMI DI ARMSTRONG

---

- **Teorema:** Gli assiomi di Armstrong sono corretti e completi.
- Attraverso gli assiomi di Armstrong, si può mostrare l'equivalenza della nozione di **implicazione logica** ( $\mid=$ ) e di quella di **derivazione** ( $\mid-$ ):  
se una dipendenza è derivabile con gli assiomi di Armstrong allora è anche implicata logicamente (**correttezza** degli assiomi), e viceversa se una dipendenza è implicata logicamente allora è anche derivabile dagli assiomi (**completezza** degli assiomi).
- **Correttezza** degli assiomi:

$$\forall f, \quad F \mid- f \Rightarrow F \mid= f$$

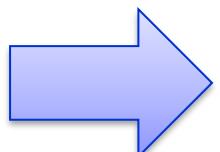
- **Completezza** degli assiomi:

$$\forall f, \quad F \mid= f \Rightarrow F \mid- f$$

## CHIUSURA DI UN INSIEME F

---

- **Definizione** Dato un insieme F di DF, la **chiusura di F**, denotata con **F<sup>+</sup>**, è:  
$$F^+ = \{ X \rightarrow Y \mid F \vdash X \rightarrow Y \}$$
- Un problema che si presenta spesso è quello di decidere se una dipendenza funzionale appartiene a F<sup>+</sup> (problema dell'implicazione):
  - la sua **risoluzione** con l'algoritmo banale (di generare F<sup>+</sup> applicando ad F ripetutamente gli assiomi di Armstrong) ha una complessità esponenziale rispetto al numero di attributi dello schema.



## CHIUSURA DI UN INSIEME F

---

- **Definizione** Dato un insieme F di DF, la **chiusura di F**, denotata con **F<sup>+</sup>**, è:

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

- **Definizione** Dato  $R\langle T, F \rangle$ , e  $X \subseteq T$ , la **chiusura di X rispetto ad F**, denotata con  $X_F^+$ , (o  $X^+$ , se F è chiaro dal contesto) è

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



## CHIUSURA DI UN INSIEME F

---

- **Definizione** Dato un insieme F di DF, la **chiusura di F**, denotata con  $F^+$ , è:

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

- **Definizione** Dato  $R \langle T, F \rangle$ , e  $X \subseteq T$ , la **chiusura di X rispetto ad F**, denotata con  $X_F^+$ , (o  $X^+$ , se F è chiaro dal contesto) è

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

- **Problema dell'implicazione:** controllare se una DF  $V \rightarrow W \in F^+$

Un algoritmo efficiente per risolvere il problema dell'implicazione senza calcolare la chiusura di F scaturisce dal seguente teorema.

**Teorema/osservazione**  $F \vdash X \rightarrow Y \Leftrightarrow Y \subseteq X_F^+$ .

## Algoritmo per calcolare $X_F^+$ - Idea

---

Sia  $X$  un insieme di attributi e  $F$  un insieme di dipendenze. Vogliamo calcolare  $X_F^+$

1. Inizializziamo  $X^+$  con l'insieme  $X$
2. Se fra le dipendenze di  $F$  c'è una dipendenza  $Y \rightarrow A$  con  $Y \subseteq X^+$  allora si inserisce  $A$  in  $X^+$ , ossia  $X^+ = X^+ \cup \{A\}$
3. Si ripete il passo 2 fino a quando non ci sono altri attributi da aggiungere a  $X^+$
4. Si dà in output  $X_F^+ = X^+$

## CHIUSURA LENTA

- Un semplice algoritmo per calcolare  $X^+$  (ne esiste uno migliore di complessità di tempo  $O(ap)$ ) è
- **Algoritmo CHIUSURA LENTA**

input             $R \langle T, F \rangle X \subseteq T$

output           $X^+$

begin

$X^+ = X$     Inizializziamo  $X^+$  con l'insieme  $X$

    while ( $X^+$  cambia) do

        for  $W \rightarrow V$  in  $F$  with  $W \subseteq X^+$  and  $V \not\subseteq X^+$

            do  $X^+ = X^+ \cup V$

end

1. Si dà in output  $X_F^+ = X^+$

fino a quando non ci sono altri attributi da aggiungere a  $X^+$

Se fra le dipendenze di  $F$  c'è una dipendenza  $W \rightarrow V$  con  $W \subseteq X^+$  allora si inserisce  $V$  in  $X^+$ , ossia  $X^+ = X^+ \cup \{V\}$

## Terminazione dell'algoritmo

---

L'algoritmo termina perché ad ogni passo viene aggiunto un nuovo attributo a  $X^+$ . Essendo gli attributi in numero finito, a un certo punto l'algoritmo deve fermarsi

Per dimostrare la correttezza, si dimostra che  $X_F^+ = X^+$   
(per induzione)

## ESEMPIO

---

- $F = \{DB \rightarrow E, B \rightarrow C, A \rightarrow B\}$ , trovare  $(AD)^+$ :
- Vogliamo conoscere gli attributi che sono determinati funzionalmente da un insieme di dipendenze  $A$  e  $D$ .

$$X^+ = AD$$

$$X^+ = ADB$$

$$X^+ = ADBE$$

$$X^+ = ADBEC$$

Se fra le dipendenze di  $F$  c'è una dipendenza  $Y \rightarrow A$  con  $Y \subseteq X^+$  allora si inserisce  $A$  in  $X^+$ , ossia  $X^+ = X^+ \cup \{A\}$

## CHIAVI E ATTRIBUTI PRIMI

---

- **Definizione** Dato lo schema  $R \langle T, F \rangle$ , diremo che un insieme di attributi  $W \subseteq T$  è una **chiave candidata** di  $R$ , se
  - $W \rightarrow T \in F^+$  (W superchiave)
  - $\forall V \subset W, V \rightarrow T \notin F^+$  (se  $V \subset W$ ,  $V$  non superchiave)
- **Attributo primo** : attributo che appartiene ad almeno una chiave

## ESEMPIO - superchiave?

- $F = \{DB \rightarrow E, B \rightarrow C, A \rightarrow B\}$ , trovare  $(AD)^+$ :

$X^+ = AD$

$X^+ = ADB$

$X^+ = ADBE$

$X^+ = ADBEC$

- **AD** è superchiave?

- Si poiché contiene tutti gli attributi

- A è superchiave?

- $A \rightarrow B, A \rightarrow BC$ , si ferma  $\rightarrow$  non è superchiave

- ABD è superchiave?

- $(ABD)^+$  è analoga a  $(AD)^+$ , perché ABD è più grande di AD, quindi è superchiave

- ABC è superchiave?

- ABC stesso, quindi non è superchiave



Esempio: chiave?

---

<u>Impiegato</u>	<u>Stipendio</u>	<u>Progetto</u>	<u>Bilancio</u>	<u>Funzione</u>
------------------	------------------	-----------------	-----------------	-----------------

$F = \{ \text{Impiegato} \rightarrow \text{Stipendio}, \text{Progetto} \rightarrow \text{Bilancio},$   
 $\text{Impiegato Progetto} \rightarrow \text{Funzione} \}$

$\{\text{Impiegato}\}^+ = \{\text{Impiegato}, \text{Stipendio}\}$

$\{\text{Progetto}\}^+ = \{\text{Progetto}, \text{Bilancio}\}$

$\{\text{Impiegato}, \text{Progetto}\}^+ = \{\text{Impiegato}, \text{Progetto}, \text{Stipendio}, \text{Bilancio}, \text{Funzione}\}$

$K = (\text{Impiegato Progetto})$  è chiave. Infatti genera tutto l'insieme  $U$  e nessuno dei suoi sottoset di attributi lo genera

---

## CHIAVI E ATTRIBUTI PRIMI

---

- **Definizione** Dato lo schema  $R \langle T, F \rangle$ , diremo che un insieme di attributi  $W \subseteq T$  è una **chiave candidata** di  $R$ , se
  - $W \rightarrow T \in F^+$  (W superchiave)
  - $\forall V \subset W, V \rightarrow T \notin F^+$  (se  $V \subset W$ ,  $V$  non superchiave)
- **Attributo primo** : attributo che appartiene ad almeno una chiave
- **Complessità**
  - Il problema di trovare tutte le chiavi di una relazione richiede un algoritmo di complessità esponenziale nel caso peggiore
  - Il problema di controllare se un attributo è primo è NP-completo

## Proprietà interessanti per trovare tutte le chiavi

---

L'algoritmo per trovare tutte le chiavi si basa su due proprietà:

1. Se un attributo A di T **non** appare a destra di alcuna dipendenza in F, allora A **appartiene** ad ogni chiave di R
2. Se un attributo A di T **appare a destra** di qualche dipendenza in F, ma **non appare a sinistra** di alcuna dipendenza non banale, allora **A non appartiene ad alcuna chiave.**

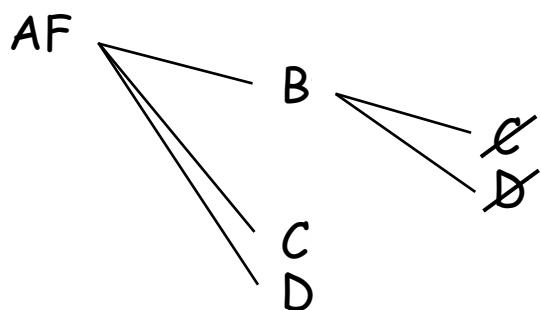
Domande:

- Posso sfruttare queste due proprietà per trovare una chiave?
- Quale è il punto di partenza?

## TROVARE TUTTE LE CHIAVI di R

---

- Sia  $F = \{C \rightarrow D, CF \rightarrow B, D \rightarrow C, F \rightarrow E\}$
- Ogni chiave deve contenere AF; le chiavi sono in  $AF \cdot P(BCDE) = AF^{BCDE}$   
(nel testo:  $AF::(BCDE)$ )
- $AF^+ = AFE$ ; ogni chiave in  $AF^{BCD} - \{AF\}$
- Candidati:  $AF^{BCD} - \{AF\} = AF^{BCD} + AFC^D + AFD$



$AF^+ = AFE$	$BCDE - AFE = BCD$
$AFB^+ = AFBE$	$CD - AFBE = CD$
no: $AFC$ chiave	
no: $AFD$ chiave	
$AFC^+ = AFCDBE$	<b>AFC chiave</b>
$AFD^+ = AFDCEB$	<b>AFD chiave</b>

---

# **PARTE III**

---

# Copertura Canonica

## COPERTURA DI INSIEMI DI DF

---

- **Definizione:** Due insiemi di DF,  $F$  e  $G$ , sullo schema  $R$  sono **equivalenti**,  
 $F \equiv G$ , sse  $F^+ = G^+$ .
- Se  $F \equiv G$ , allora  $F$  è una **copertura** di  $G$  (e  $G$  una copertura di  $F$ ).

**Esempio:**

studenti(matricola, CF, Cognome, Nome, Anno)

## COPERTURA DI INSIEMI DI DF - Parte I - attributo estraneo

---

- **Definizione** Sia  $F$  un insieme di DF:
  1. Data una  $X \rightarrow Y \in F$ , si dice che  $X$  contiene un **attributo estraneo**  $A_i$  sse  $(X - \{A_i\}) \rightarrow Y \in F^+$ , cioè  $F \vdash (X - \{A_i\}) \rightarrow Y$

(Come facciamo a stabilire che in una DF del tipo  $AX \rightarrow B$  l'attributo  $A$  è estraneo? Per verificare se  $A$  è estraneo calcoliamo  $X^+$  e verifichiamo se include  $B$ , ovvero se basta  $X$  a determinare  $B$ )

- **Esempio:**

Orari(CodAula, NomeAula, Piano, Posti, Materia, CDL, Docente, Giorno, Ora)

  - se vale
    - Docente, Giorno, Ora  $\rightarrow$  CodAula
    - Docente, Giorno  $\rightarrow$  Ora
  - allora
    - Docente, Giorno  $\rightarrow$  CodAula
    - (quindi) nella prima dipendenza Ora è attributo estraneo



## Attributo estraneo - Esempio

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

- In  $AB \rightarrow C$ , l'attributo B è estraneo?
- Calcoliamo  $A^+$  e verifichiamo se include C, ovvero se basta X a determinare C

$$A^+ = A$$

$$A^+ = AB \text{ poiché } A \rightarrow B \text{ e } A \subseteq A^+$$

$$A^+ = ABC \text{ poiché } AB \rightarrow C \text{ e } AB \subseteq A^+$$

C dipende solo da A, ovvero in  $AB \rightarrow C$  l'attributo B è estraneo (perché a sua volta dipendente da A:  $A \rightarrow B$ )

◦ ◦ ◦

*A<sup>+</sup> include  
l'attributo C*

L'insieme di DF può essere riscritto come:  $F' = \{A \rightarrow C, A \rightarrow B\}$

Nota che

$$(AB)^+ = AB$$

$$(AB)^+ = ABC \text{ poiché } AB \rightarrow C \text{ e } AB \subseteq A^+$$

## COPERTURA DI INSIEMI DI DF - Parte II - ridondante

---

- **Definizione** Sia  $F$  un insieme di DF:

2.  $X \rightarrow Y$  è una **dipendenza ridondante** sse  $(F - \{X \rightarrow Y\})^+ = F^+$ ,

Equivalentemente  $F - \{X \rightarrow Y\} \vdash X \rightarrow Y$

(Come facciamo a stabilire che una DF del tipo  $X \rightarrow A$  è ridondante? La eliminiamo da  $F$ , calcoliamo  $X^+$  e verifichiamo se include  $A$ , ovvero se con le DF che restano riusciamo ancora a dimostrare che  $X$  determina  $A$ )

- **Esempio:**

Orari(CodAula, NomeAula, Piano, Posti, Materia, CDL, Docente, Giorno, Ora)

- se vale

- $\text{Docente, Giorno, Ora} \rightarrow \text{CodAula}$

- $\text{CodAula} \rightarrow \text{NomeAula}$

- è inutile avere anche

- $\text{Docente, Giorno, Ora} \rightarrow \text{NomeAula}$



## DF ridondanti - Esempio

---

- $F = \{B \rightarrow C, B \rightarrow A, C \rightarrow A\}$
- $B \rightarrow A$  è ridondante
- Poiché possiamo dedurla da  $B \rightarrow C$  e  $C \rightarrow A$

## COPERTURA DI INSIEMI DI DF - Parte III

- **Definizione** Sia  $F$  un insieme di DF:
  1. Data una  $X \rightarrow Y \in F$ , si dice che  $X$  contiene un **attributo estraneo**  $A_i$  sse  $(X - \{A_i\}) \rightarrow Y \in F^+$ , cioè  $F \vdash (X - \{A_i\}) \rightarrow Y$
  2.  $X \rightarrow Y$  è una **dipendenza ridondante** sse
$$(F - \{X \rightarrow Y\})^+ = F^+,$$

Equivalentemente  $F - \{X \rightarrow Y\} \vdash X \rightarrow Y$

- **Altro Esempio:** studenti(matricola, CF, Cognome, Nome, Anno)
    - Matricola  $\rightarrow$  cognome
    - Matricola, cognome  $\rightarrow$  nome (cognome è estraneo)
- è equivalente a
- Matricola  $\rightarrow$  cognome
  - Matricola  $\rightarrow$  nome

## Esempio

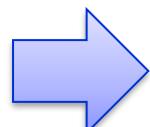
---

$$F_1 = \{A \rightarrow B, AB \rightarrow C, A \rightarrow C\}$$

$$F_2 = \{A \rightarrow B, AB \rightarrow C\}$$

$$F_3 = \{A \rightarrow B, A \rightarrow C\}$$

- In  $F_1$  vi è ridondanza? Presenta attributi estranei?
- In  $F_2$  vi è ridondanza? Presenta attributi estranei?
- In  $F_3$  vi è ridondanza? Presenta attributi estranei?



## Esempio - Soluzione

---

$$F_3 = \{A \rightarrow B, A \rightarrow C\}$$

- $F_3$  non presenta attributi estranei
- 

$$F_2 = \{A \rightarrow B, AB \rightarrow C\}$$

- $F_2$  non è ridondante ma presenta un attributo estraneo, perché B può essere eliminato dal primo membro della seconda dipendenza ( $A^+ = A$  ;  $A^+ = AB$  ;  $A^+ = ABC$ ) (quindi equivale a  $F_3$ )
- 

$$F_1 = \{A \rightarrow B, AB \rightarrow C, A \rightarrow C\}$$

- $F_1$  è ridondante, perché  $\{A \rightarrow B, AB \rightarrow C\}$  implica  $A \rightarrow C$  (quindi equivale a  $F_2$ )
-

## COPERTURA DI INSIEMI DI DF - Parte IV

---

- **Definizione** Sia  $F$  un insieme di DF:
  1. Data una  $X \rightarrow Y \in F$ , si dice che  $X$  contiene un attributo *estraneo*  $A_i$  sse  $(X - \{A_i\}) \rightarrow Y \in F^+$ , cioè  $F \vdash (X - \{A_i\}) \rightarrow Y$
  2.  $X \rightarrow Y$  è una dipendenza *ridondante* sse
$$(F - \{X \rightarrow Y\})^+ = F^+,$$

Equivalentemente

$$F - \{X \rightarrow Y\} \vdash X \rightarrow Y$$

$F$  è detta una **copertura canonica** sse

- la parte destra di ogni DF in  $F$  è un attributo;
- non esistono attributi estranei;
- nessuna dipendenza in  $F$  è ridondante.

# ESISTENZA DELLA COPERTURA CANONICA

- Teorema

Per ogni insieme di dipendenze  $F$  esiste una copertura canonica.

- Algoritmo per calcolare una copertura canonica:

- Trasformare le dipendenze nella forma  $X \rightarrow A$
- Eliminare gli attributi estranei
- Eliminare le dipendenze ridondanti

Si sostituisce l'insieme dato con quello equivalente che ha tutti i secondi membri costituiti da singoli attributi (dipendenze atomiche)

Per ogni dipendenza si verifica se esistono attributi eliminabili dal primo membro.

Data una  $X \rightarrow Y \in F$ , si dice che  $X$  contiene un attributo estraneo  $A_i$ , sse  $(X - \{A_i\}) \rightarrow Y \in F^+$ , cioè  $F \not\models (X - \{A_i\}) \rightarrow Y$

$X \rightarrow Y$  è una dipendenza ridondante sse  $(F - \{X \rightarrow Y\})^+ = F^+$ , Equivalentemente  $F - \{X \rightarrow Y\} \vdash X \rightarrow Y$

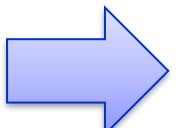
## Esempio ESISTENZA DELLA COPERTURA CANONICA - Parte I

---

Impiegato (Matricola, Cognome, Grado, Retribuzione,  
Dipartimento, Supervisore, Progetto, Anzianità)

Consideriamo il seguente insieme di dipendenze  
funzionali

$\{M \rightarrow RSDG, MS \rightarrow CD, G \rightarrow R, D \rightarrow S, S \rightarrow D, MPD \rightarrow AM\}$



## Esempio ESISTENZA DELLA COPERTURA CANONICA - Parte II

Impiegato (Matricola, Cognome, Grado, Retribuzione, Dipartimento, Supervisore, Progetto, Anzianità)

$$\{M \rightarrow RSDG, MS \rightarrow CD, G \rightarrow R, D \rightarrow S, S \rightarrow D, MPD \rightarrow AM\}$$

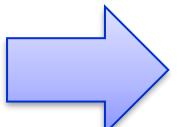
1. Creiamo le dipendenze funzionali atomiche

$$\{M \rightarrow R, M \rightarrow S, M \rightarrow D, M \rightarrow G, MS \rightarrow C, MS \rightarrow D, G \rightarrow R, D \rightarrow S, S \rightarrow D, MPD \rightarrow A, MPD \rightarrow M\}$$

2. Eliminare gli attributi estranei:

- è possibile eliminare  $S$  dal primo membro di  $MS \rightarrow C$  e  $MS \rightarrow D$  perché  $M \rightarrow S$  (si ottiene da  $M \rightarrow D$  e  $D \rightarrow S$ )
- È possibile eliminare  $D$  dal primo membro di  $MPD \rightarrow A$  e  $MPD \rightarrow M$  poiché  $M \rightarrow D$

$$\{M \rightarrow R, M \rightarrow S, M \rightarrow D, M \rightarrow G, M \rightarrow C, M \rightarrow D, G \rightarrow R, D \rightarrow S, S \rightarrow D, MP \rightarrow A, MP \rightarrow M\}$$



## Esempio ESISTENZA DELLA COPERTURA CANONICA - Parte III

Impiegato (Matricola, Cognome, Grado, Retribuzione, Dipartimento, Supervisore, Progetto, Anzianità)

$\{M \rightarrow R, M \rightarrow S, M \rightarrow D, M \rightarrow G, MS \rightarrow C, MS \rightarrow D, G \rightarrow R,$   
 $D \rightarrow S, S \rightarrow D, MPD \rightarrow A, MPD \rightarrow M\}$



Eliminazione degli attributi estranei

$\{M \rightarrow R, M \rightarrow S, M \rightarrow D, M \rightarrow G, M \rightarrow C, M \rightarrow D, G \rightarrow R,$   
 $D \rightarrow S, S \rightarrow D, MP \rightarrow A, MP \rightarrow M\}$

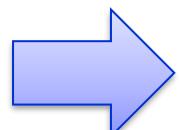
3. Si trova l'insieme di dipendenza funzionali non ridondante: eliminiamo le dipendenze ottenibili da altre:

$M \rightarrow R$  (deriva da  $M \rightarrow G$  e  $G \rightarrow R$ )

$M \rightarrow S$  (deriva da  $M \rightarrow D$  e  $D \rightarrow S$ )

$M \rightarrow D$  (Perché già  $M \rightarrow D$ )

$MP \rightarrow M$  (Perché  $M$  compare a primo membro)



## Esempio ESISTENZA DELLA COPERTURA CANONICA - Parte IV

Impiegato (Matricola, Cognome, Grado, Retribuzione, Dipartimento, Supervisore, Progetto, Anzianità)

$\{M \rightarrow RSDG, MS \rightarrow CD, G \rightarrow R, D \rightarrow S, S \rightarrow D, MPD \rightarrow AM\}$

Eliminazione degli attributi estranei



$\{M \rightarrow R, M \rightarrow S, M \rightarrow D, M \rightarrow G, MS \rightarrow C, MS \rightarrow D, G \rightarrow R,$   
 $D \rightarrow S, S \rightarrow D, MPD \rightarrow A, MPD \rightarrow M\}$

Eliminazione di:  $M \rightarrow R, M \rightarrow S, M \rightarrow D, MP \rightarrow M$



$\{M \rightarrow R, M \rightarrow S, M \rightarrow D, M \rightarrow G, M \rightarrow C, M \rightarrow D, G \rightarrow R,$   
 $D \rightarrow S, S \rightarrow D, MP \rightarrow A, MP \rightarrow M\}$



$\{M \rightarrow D, M \rightarrow G, M \rightarrow C, G \rightarrow R, D \rightarrow S, S \rightarrow D, MP \rightarrow A\}$

## Riassunto

---

- Qualità degli schemi relazionali e anomalie
- Dipendenze funzionali  $X \rightarrow Y$

$r \models X \rightarrow Y$  se  $\forall t_1, t_2 \in r.$

se  $t_1[X] = t_2[X]$  allora  $t_1[Y] = t_2[Y]$

- Chiusura di un insieme di dipendenze

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

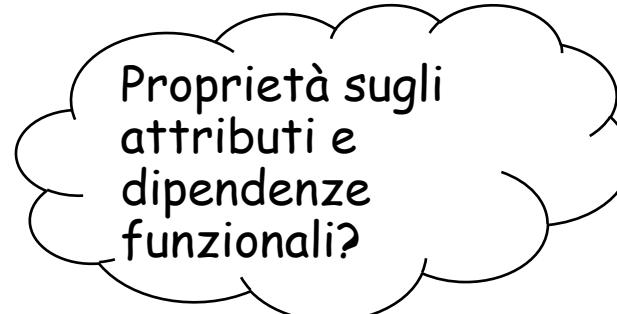
- Chiave e attributi primi
- Copertura canonica (attributi estranei, dipendenze ridondanti)

## Trovare una qualsiasi chiave

---

- $T = \{A, B, C, D, E, F\}$
- $F = \{C \rightarrow D, CF \rightarrow B, D \rightarrow C, F \rightarrow E\}$

Quale proprietà possiamo usare?



---

# RIEPILOGO

## Normalizzazione dei dati

---

- Le **ridondanze** sui dati possono essere di due tipi:
  - **Ridondanza concettuale** → non ci sono duplicazioni dello stesso dato, ma sono memorizzate **informazioni che possono essere ricavate da altre già contenute nel DB.**
  - **Ridondanza logica** → esistono **duplicazioni sui dati, che possono generare anomalie nelle operazioni sui dati ...**

## Normalizzazione dei dati

---

- Le dipendenze funzionali sono definite a livello di schema e non a livello di istanza!

Matricola	Cognome	Corso	Voto
1244	Rossi	Basi di Dati	18
1567	Bianchi	Programmazione	22
1898	Bianchi	Analisi I	20
2040	Verdi	Programmazione	22
2121	Verdi	Basi di Dati	18
2678	Bruni	Analisi I	20

- Dipendenza funzionale **Corso → Voto?** NO!

## Normalizzazione dei dati

---

- Le dipendenze funzionali hanno sempre un verso!

<b>Matricola</b>	<b>Studente</b>	<b>Corso</b>	<b>Docente</b>	<b>Voto</b>
1244	Rossi	Basi di Dati	Ghelli	18
1567	Bianchi	Programmazione	Messina	22
1898	Bianchi	Analisi I	Palermo	20
2040	Verdi	Programmazione	Messina	22
2121	Verdi	Basi di Dati	Ghelli	18
2678	Bruni	Analisi I	Palermo	20

- Dipendenza funzionale **Corso → Docente** ? Può essere, occorre considerare le specifiche del sistema ...



## Normalizzazione dei dati

---

Le dipendenze funzionali hanno sempre un verso!

<u>Matricola</u>	Studente	<u>Corso</u>	Docente	Voto
1244	Rossi	Basi di Dati	Roma	18
1567	Bianchi	Programmazione	Messina	22
1898	Bianchi	Analisi I	Palermo	20
2040	Verdi	Programmazione	Messina	22
2121	Verdi	Basi di Dati	Roma	18
2678	Bruni	Analisi I	Palermo	20
4354	Bruni	Architetture	Roma	28

➤ Corso → Docente? OK    Docente → Corso? NO!

## Normalizzazione dei dati

---

- Le dipendenze funzionali sono una **generalizzazione** del vincolo di **chiave (e di superchiave)**.
- Data una tabella con schema  $R(X)$ , con superchiave  $K$ .
  - Esiste un vincolo di dipendenza funzionale tra  $K$  e qualsiasi attributo dello schema  $r$ .

$$K \rightarrow X_1, \quad X_1 \subseteq X$$

# Riepilogo: ESISTENZA DELLA COPERTURA CANONICA

- Teorema

Per ogni insieme di dipendenze  $F$  esiste una copertura canonica.

- Algoritmo per calcolare una copertura canonica:

- Trasformare le dipendenze nella forma  $X \rightarrow A$
- Eliminare gli attributi estranei
- Eliminare le dipendenze ridondanti

$X \rightarrow Y$  è una dipendenza ridondante sse ( $F - \{X \rightarrow Y\}$ )<sup>+</sup> =  $F^+$ ,  
Equivalentemente  $F - \{X \rightarrow Y\} \vdash X \rightarrow Y$

Si sostituisce l'insieme dato con quello equivalente che ha tutti i secondi membri costituiti da singoli attributi (dipendenze atomiche)

Per ogni dipendenza si verifica se esistono attributi eliminabili dal primo membro.  
Data una  $X \rightarrow Y \in F$ , si dice che  $X$  contiene un attributo estraneo  $A_i$  sse  $(X - \{A_i\}) \rightarrow Y \in F^+$ , cioè  $F \not\vdash (X - \{A_i\}) \rightarrow Y$

---

# Decomposizione di Schemi

**IN GENERALE, PER ELIMINARE ANOMALIE DA UNO SCHEMA OCCORRE DECOMPORLO IN SCHEMI PIÙ PICCOLI "EQUIVALENTI"**

# Esempio di decomposizione

L'intuizione è che si devono "estrarre" gli attributi che sono determinati da attributi non chiave ovvero "creare uno schema per ogni funzione"

<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

Impiegato → Stipendio

<u>Impiegato</u>	<u>Stipendio</u>
Rossi	20
Verdi	35
Neri	55
Mori	48
Bianchi	48

Progetto → Bilancio

<u>Progetto</u>	<u>Bilancio</u>
Marte	2
Giove	15
Venere	15

Impiegato Progetto → Funzione

<u>Impiegato</u>	<u>Progetto</u>	<u>Funzione</u>
Rossi	Marte	tecnico
Verdi	Giove	progettista
Verdi	Venere	progettista
Neri	Venere	direttore
Neri	Giove	consulente
Neri	Marte	consulente
Mori	Marte	direttore
Mori	Venere	progettista
Bianchi	Venere	progettista
Bianchi	Giove	direttore

## Non sempre è così facile - Esempio

---

La soluzione non è tuttavia sempre così semplice, bisogna fare anche altre considerazioni; ad esempio, operando come prima:

<u>Impiegato</u>	<u>Progetto</u>	<u>Sede</u>
Rossi	Marte	Roma
Verdi	Giove	Milano
Verdi	Venere	Milano
Neri	Saturno	Milano
Neri	Venere	Milano

Ammette le due dipendenze funzionali

Impiegato → Sede

Progetto → Sede

## Decomponiamo sulla base delle dipendenze

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

Impiegato → Sede  
Progetto → Sede



Impiegato → Sede

Impiegato	Sede
Rossi	Roma
Verdi	Milano
Neri	Milano

Progetto → Sede

Progetto	Sede
Marte	Roma
Giove	Milano
Saturno	Milano
Venere	Milano

## Proviamo a ricostruire (mediante join)

$T_1$

Impiegato	Sede
Rossi	Roma
Verdi	Milano
Neri	Milano



$T_2$

Progetto	Sede
Marte	Roma
Giove	Milano
Saturno	Milano
Venere	Milano

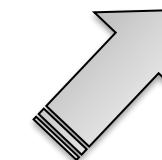
Decomposizione con perdite:  
si generano delle tuple spurie dopo il join.

$T_1$  join  $T_2$

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

Queste due tuple sono in più (spurie)

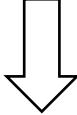
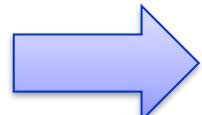
Impiegato	Progetto	Sede
Rossi	Marte	Roma
Verdi	Giove	Milano
Verdi	Venere	Milano
Neri	Saturno	Milano
Neri	Venere	Milano



Diversa dalla relazione di partenza!

## DECOMPOSIZIONE DI SCHEMI

---

- **Definizione** Dato uno schema  $R(T)$ ,  
 $\rho = \{R_1(T_1), \dots, R_k(T_k)\}$  è una **decomposizione** di  $R$  sse  $T_1 \cup \dots \cup T_k = T$ :
  - StudentiEdEsami(**Matricola**, **Nome**, Provincia, AnnoNascita, **Materia**, Voto)  

  - {Studenti(**Matricola**, **Nome**, Provincia, AnnoNascita),  
Esami(**Matricola**, **Materia**, **Voto**)}
  - {Studenti(**Matricola**, **Nome**, Provincia, AnnoNascita),  
Esami(**Nome**, **Materia**, **Voto**)}
  - {Studenti(**Matricola**, **Nome**, Provincia, AnnoNascita),  
Esami(**Materia**, **Voto**)}
- 

## DECOMPOSIZIONE DI SCHEMI: Conservazione dei dati e delle dipendenze

---

- In generale, per eliminare anomalie da uno schema occorre decomporlo in schemi più piccoli "equivalenti"
- **Definizione** Dato uno schema  $R(T)$ ,  
 $\rho = \{R_1(T_1), \dots, R_k(T_k)\}$  è una decomposizione di  $R$  sse  $\cup T_i = T$ :
- Due proprietà desiderabili di una decomposizione:
  - conservazione dei dati (*nozione semantica*)
  - conservazione delle dipendenze

## DECOMPOSIZIONE DI SCHEMI - Conservazione dei dati

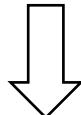
---

- Decomposizioni che **preservano i dati**:
- Definizione:  $\rho = \{R_1(T_1), \dots, R_k(T_k)\}$  è una decomposizione di uno schema  $R(T)$  che **preserva i dati** sse per ogni *istanza valida*  $r$  di  $R$ :  
$$r = (\pi_{T_1} r) \vee (\pi_{T_2} r) \vee \dots \vee (\pi_{T_k} r)$$
- Dalla definizione di giunzione naturale scaturisce il seguente risultato:
- Teorema: Se  $\rho = \{R_1(T_1), \dots, R_k(T_k)\}$  è una decomposizione di  $R(T)$ , allora per ogni istanza  $r$  di  $R$ :  
$$r \subseteq (\pi_{T_1} r) \vee (\pi_{T_2} r) \vee \dots \vee (\pi_{T_k} r)$$

## DECOMPOSIZIONE DI SCHEMI - Con Perdita di informazione

---

StudentiEdEsami(Matricola, Nome, Provincia, AnnoNascita, Materia, Voto)



{Studenti(Matricola, Nome, Provincia, AnnoNascita),  
Esami(Nome, Materia, Voto)}

Cosa succede quando si fa la giunzione?

Nessuna tupla si perde, ma...?

Questa decomposizione crea tuple spurie: ci sono n-uple in più.

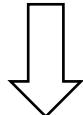
Si pensi al caso di due persone con lo stesso nome che hanno sostenuto esami diversi, cosa succede dopo la giunzione?

Perdita di informazione!

## DECOMPOSIZIONE DI SCHEMI - Con Perdita di informazione

---

StudentiEdEsami(Matricola, Nome, Provincia, AnnoNascita, Materia, Voto)



{Studenti(Matricola, Nome, Provincia, AnnoNascita),  
Esami(Materia, Voto)}

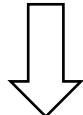
Si perde l'informazione sullo studente.

Perdita di informazione!

## DECOMPOSIZIONE DI SCHEMI - Senza Perdita di informazione

---

StudentiEdEsami(**Matricola**, **Nome**, Provincia, AnnoNascita, Materia, Voto)



{Studenti(**Matricola**, Nome, Provincia, AnnoNascita),  
Esami(**Matricola**, Materia, Voto)}

Perché non perdo informazione con questa decomposizione?

La chiave è l'unico modo per avere una decomposizione senza perdita di informazione.

## ESEMPIO DI DECOMPOSIZIONE - non preserva i dati

- Sia  $r$  qui sotto un'istanza valida di  $R(ABC)$ :

	A	B	C
$r =$	a1	b	c1
	a2	b	c2

Allora la decomposizione  $\{R(AB), R(BC)\}$ :

$\pi_{T_1} r =$	A	B	
	a1	b	
	a2	b	

$\pi_{T_2} r =$	B	C	
	b	c1	
	b	c2	

non preserva i dati, infatti  $\pi_{T_1} r \vee \pi_{T_2} r \supseteq r$

$r =$	A	B	C
	a1	b	c1
	a1	b	c2
	a2	b	c1
	a2	b	c2

## Decomposizione senza perdita

---

Uno schema  $R(X)$  si **decompon**e **sen**a Perdita dei dati negli schemi  $R1(X1)$  ed  $R2(X2)$  se, per ogni possibile istanza  $r$  di  $R(X)$ , il join naturale delle proiezioni di  $r$  su  $X1$  ed  $X2$  produce la tabella di partenza. (cioè non contiene *ennuple spurie*)

$$\rho_{X1}(r) \bowtie \rho_{X2}(r) = r$$

## Decomposizione senza perdita

---

$$\rho_{X_1}(r) \triangleright\triangleleft \rho_{X_2}(r) = r$$

La decomposizione senza perdita è garantita se l'insieme degli attributi comuni alle due relazioni ( $X_1 \cap X_2$ ) è chiave per almeno una delle due relazioni composte.

Ad esempio,  $Sede=(Progetto, Sede) \cap (Impiegato, Sede)$  non è chiave per nessuna delle due relazioni

Impiegato	Sede
Rossi	Roma
Verdi	Milano
Neri	Milano

Progetto	Sede
Marte	Roma
Giove	Milano
Saturno	Milano
Venere	Milano

(decomposizione con perdita)

## Senza perdita

Sia  $r$  una relazione su un insieme di attributi  $X$  e siano  $X_1$  e  $X_2$  due sottoinsiemi di  $X$  la cui unione sia pari a  $X$  stesso;

Inoltre, sia  $X_0$  l'intersezione di  $X_1$  e  $X_2$ , allora:

- $r$  si decomponе senza perdita su  $X_1$  e  $X_2$  se soddisfa la dipendenza funzionale  $X_0 \rightarrow X_1$  oppure la dipendenza funzionale  $X_0 \rightarrow X_2$

## Motivazione

---

### Teorema (non formale):

Se l'insieme degli attributi comuni alle due relazioni ( $X_1 \cap X_2$ ) è chiave per almeno una delle due relazioni decomposte allora la decomposizione è senza perdita

### Dimostrazione (non formale)

- Supponiamo  $r$  sia una relazione sugli attributi  $ABC$  e consideriamo le sue proiezioni  $r_1$  su  $AB$  e  $r_2$  su  $AC$ .
- Supponiamo che  $r$  soddisfi la dipendenza funzionale  $A \rightarrow C$ . Allora  $A$  è chiave per  $r$  su  $AC$  e quindi non ci sono in tale proiezione due tuple diverse sugli stessi valori di  $A$ .



## Motivazione

---

Il join costruisce tuple a partire dalle tuple nelle due proiezioni

- Sia  $t=(a,b,c)$  una tupla del join di  $r_1$  e  $r_2$ .
- Mostriamo che appartiene ad  $r$  (cioè non è spuria).
  - $t$  è ottenuta mediante join da  $t_1=(a,b)$  di  $r_1$  e  $t_2=(a,c)$  su  $r_2$ .
  - Allora per la definizione di proiezione, esistono due tuple in  $r$ ,  $t'_1 = (a,b,*)$  e  $t'_2 = (a,*,c)$  (dove  $*$  sta per un valore non noto).
  - Poiché  $A \rightarrow C$ , allora esiste un solo valore in  $C$  associato al valore  $a$ . Dato che  $(a,c)$  compare nella proiezione, questo valore è proprio  $c$ .
  - Ma allora nella tupla  $t'_1$  il valore incognito deve essere proprio  $c$ . Quindi  $(a,b,c)$  appartiene a  $r$ .

# DECOMPOSIZIONI BINARIE

---

- Teorema
- Sia  $R \langle T, F \rangle$  uno schema di relazione, la decomposizione  $\rho = \{R_1(T_1), R_2(T_2)\}$  preserva i dati sse
  - $T_1 \cap T_2 \rightarrow T_1 \in F^+$  oppure  $T_1 \cap T_2 \rightarrow T_2 \in F^+$ .
- Esistono criteri anche per decomposizioni in più di due schemi.

## Decomposizione senza perdita

- Anche se una decomposizione è senza perdite, può comunque presentare dei problemi di conservazione delle dipendenze ...
- Ad esempio,  $\text{Impiegato} = (\text{Impiegato}, \text{Sede}) \cap (\text{Impiegato}, \text{Progetto})$

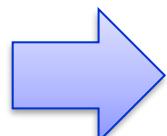
<u>Impiegato</u>	Progetto	Sede
Rossi	Marte	Roma
Verdi	Giove	Milano
Verdi	Venere	Milano
Neri	Venere	Milano
Neri	Saturno	Milano



IMPIEGATO → SEDE	
<u>Impiegato</u>	Sede
Rossi	Roma
Verdi	Milano
Neri	Milano

<u>Impiegato</u>	Progetto
Rossi	Marte
Verdi	Giove
Verdi	Venere
Neri	Venere
Neri	Saturno

- Con questa decomposizione, non ho tuple spurie ...



## Proviamo a decomporre senza perdita

---

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

Impiegato → Sede  
~~Progetto → Sede~~

Impiegato	Sede
Rossi	Roma
Verdi	Milano
Neri	Milano

Impiegato	Progetto
Rossi	Marte
Verdi	Giove
Verdi	Venere
Neri	Saturno
Neri	Venere

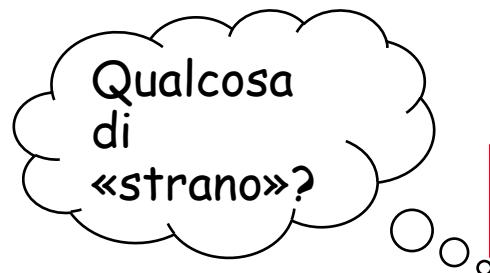
In questa decomposizione: trascuriamo la seconda dipendenza funzionale



## Un altro problema - Parte I

- Supponiamo di voler inserire una nuova tupla che specifica la partecipazione dell'impiegato Neri (che opera a Milano) al progetto Marte (lo schema non lo impedisce)

Insert into ImpProg value (Neri, Marte)



Impiegato	Sede
Rossi	Roma
Verdi	Milano
Neri	Milano

Impiegato	Progetto
Rossi	Marte
Verdi	Giove
Verdi	Venere
Neri	Saturno
Neri	Venere
Neri	Marte

Impiegato → Sede

Progetto → Sede

Domanda: Che accade se aggiungo l'impiegato Neri al progetto Marte?

## Un altro problema - Parte II

IMPIEGATO → SEDE	
Impiegato	Sede
Rossi	Roma
Verdi	Milano
Neri	Milano



Impiegato	Progetto
Rossi	Marte
Verdi	Giove
Verdi	Venere
Neri	Venere
Neri	Saturno
Neri	Marte

Viene violata la seconda dipendenza funzionale (che per il momento avevamo tenuto in sospeso)

Progetto → Sede

Una decomposizione **conserva le dipendenze se ciascuna delle dipendenze funzionali dello schema originario coinvolge attributi che compaiono tutti insieme in uno degli schemi decomposti**

Nell'esempio considerato  
Progetto → Sede  
non è conservata

=

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

## Esempio di query di verifica

---

- Se una DF non si preserva diventa più complicato capire quali sono le modifiche del DB che non violano la FD stessa
- In generale si devono prima eseguire query SQL di verifica (o, meglio, fare uso di trigger )
- Bisogna verificare che il progetto (Marte) sia presso la stessa sede dell'impiegato (Neri). A tal fine bisogna trovare un impiegato che lavora al progetto Marte

<u>Impiegato</u>	<u>Sede</u>
Rossi	Roma
Verdi	Milano
Neri	Milano

<u>Impiegato</u>	<u>Progetto</u>
Rossi	Marte
Verdi	Giove
Verdi	Venere
Neri	Venere
Neri	Saturno
Neri	Marte

`SELECT * -- OK se restituisce una tupla  
FROM Impiegati I  
WHERE  
I.Impiegato = 'Neri' AND I.Sede IN (  
SELECT I1.Sede  
FROM Impiegati I1, ImpProg IP  
WHERE I1.Impiegato = IP.Impiegato  
AND IP.Progetto = 'Marte')`

## Qualità delle decomposizioni

Una decomposizione:

- deve essere senza perdita, per garantire la ricostruzione delle informazioni originarie
- dovrebbe preservare le dipendenze, per semplificare il mantenimento dei vincoli di integrità originari

Nell'esempio, questo suggerisce di inserire anche:

- Va sempre eseguita una query, ma più semplice

<u>Impiegato</u>	<u>Sede</u>	<u>Impiegato</u>	<u>Progetto</u>
Rossi	Roma	Rossi	Marte
Verdi	Milano	Verdi	Giove
Neri	Milano	Neri	Venere
		Neri	Saturno
		Neri	Marte

Progetti

<u>Progetto</u>	<u>Sede</u>
Marte	Roma
Giove	Milano
Venere	Milano
Saturno	Milano

SELECT \* -- OK se restituisce una tupla  
FROM Impiegati I, Progetti P  
WHERE

I.Impiegato = 'Neri' AND  
P.Progetto = 'Marte' AND  
I.Sede = P.Sede

## Conservazione delle dipendenze

---

Una decomposizione **conserva le dipendenze** se ciascuna delle dipendenze funzionali dello schema originario coinvolge attributi che compaiono tutti insieme in uno degli schemi composti

Nell'esempio considerato

Progetto → Sede

non è conservata

## ESEMPIO - Preservare le dipendenze?

- Telefoni(Prefisso, Numero, Località, Abbonato, Via)

{Pref Num → Loc Abb Via, Loc → Pref} cioè:      °    °    ○

(Prefisso,  
Numero) è  
chiave

- Si consideri la decomposizione:

$\rho = \{ \text{Tel} < \{N, L, A, V\}, F1 >$ ,

Pref<{L, P}, F2 >} con

$F1 = \{LN \rightarrow A \ V\}$  e  $F2 = \{L \rightarrow P\}$

Numero	Località	Abbonato	Via
5348	Padova	Gino	Pascoli
5348	Vigonza	Ignazio	Silone
2344	Venezia	Lino	Leopardi
2122	Padova	Ciro	Pavese

Prefisso	Località
49	Padova
49	Vigonza
41	Venezia

- Preserva dati ma non le dipendenze: PN → L non è deducibile da F1 e F2.
- Esistono istanze valide della decomposizione che non sono proiezione di una istanza valida della relazione originale

# PROIEZIONE DELLE DIPENDENZE

- Definizione
- Dato lo schema  $R(T, F)$ , e  $T_1 \subseteq T$ , la **proiezione** di  $F$  su  $T_1$  è

$$\pi_{T_1}(F) = \{X \rightarrow Y \in F^+ \mid XY \subseteq T_1\}$$

## Esempio

Sia  $R(A, B, C)$  e  $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$ .

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

$$\pi_{AC}(F) \equiv \{A \rightarrow C, C \rightarrow A\}$$

## Algoritmo banale per il calcolo di $\pi_{T_1}(F)$ :

for each  $Y \subseteq T_1$  do ( $Z := Y^+$ ; output  $Y \rightarrow Z \cap T_1$ )

Potrebbe sembrare che la decomposizione  $\rho = \{R1(A, B), R2(A, C)\}$  non preservi le dipendenze perché  $B$  e  $C$  non appaiono insieme in uno schema della decomposizione, invece da  $B \rightarrow A$  e  $A \rightarrow C$  si ha  $B \rightarrow C$

# PRESERVAZIONE DELLE DIPENDENZE

---

- **Definizione** Dato lo schema  $R \langle T, F \rangle$ , la decomposizione  $\rho = \{R_1, \dots, R_n\}$  **preserva le dipendenze** sse l'unione delle dipendenze in  $\pi_{T_i}(F)$  è una copertura di  $F$ .
- **Proposizione** Dato lo schema  $R \langle T, F \rangle$ , il problema di stabilire se la decomposizione  $\rho = \{R_1, \dots, R_n\}$  preserva le dipendenze ha complessità di tempo polinomiale.
- Un teorema importante:  
Sia  $\rho = \{R_i \langle T_i, F_i \rangle\}$  una decomposizione di  $R \langle T, F \rangle$  che **preserva le dipendenze** e tale che  $T_j$ , per qualche  $j$ , è una superchiave per  $R \langle T, F \rangle$ .  
Allora  $\rho$  preserva i dati.

## ESEMPIO (ridondanza?)

---

- Telefoni(Prefisso, Numero, Località, Abbonato, Via)

$$F = \{P \text{ } N \rightarrow L \text{ } A \text{ } V, L \rightarrow P\}$$

- Si consideri la decomposizione:

$\rho = \{\text{Tel}\langle\{N, L, A, V\}, F1\rangle, \text{Pref}\langle\{L, P\}, F2\rangle\}$  con

- $F1 = \{LN \rightarrow A \text{ } V\}$
- $F2 = \{L \rightarrow P\}$
- Preserva dati ma non le dipendenze:  $PN \rightarrow L$  non è deducibile da  $F1$  e  $F2$ .

→

Localita	Numero	Abbonato	Via
Pisa	506070	Rossi	Via Roma
Calci	506070	Verdi	Lungarno

→

Localita	Prefisso
Pisa	050
Calci	050

## Qualità delle decomposizioni

---

- ▶ Una decomposizione dovrebbe sempre soddisfare le seguenti proprietà:
  - ▶ la **decomposizione senza perdita**, che garantisce la ricostruzione delle informazioni originarie senza generazione di tuple spurie
  - ▶ la **conservazione delle dipendenze**, che garantisce il mantenimento dei vincoli di integrità (di dipendenza funzionale) originari

---

# Forme normali

## Forme normali

---

- Una **forma normale** è una proprietà di una base di dati relazionale che ne garantisce la “qualità”, cioè l'assenza di determinati difetti
- Quando una relazione **non è normalizzata**:
  - presenta ridondanze,
  - si presta a comportamenti poco desiderabili durante gli aggiornamenti

## FORME NORMALI

---

- **1FN:**
  - Impone una restrizione sul tipo di una relazione: ogni attributo ha un tipo elementare.
- **2FN, 3FN e FNBC:**
  - Impongono restrizioni sulle dipendenze.
  - FNBC (Boyce-Codd) è la più naturale e la più restrittiva.

## Forma normale di Boyce e Codd (BCNF)

---

Una relazione  $r$  è in **forma normale di Boyce e Codd (BCNF)** se, per ogni dipendenza funzionale (non banale)  $X \rightarrow Y$  definita su di essa,  $X$  contiene una chiave  $K$  di  $r$  (è una superchiave)

La forma normale richiede che i concetti in una relazione siano omogenei (solo proprietà direttamente associate alla chiave)

La relazione sugli Impiegati

<u>Impiegato</u>	Stipendio	<u>Progetto</u>	Bilancio	Funzione
------------------	-----------	-----------------	----------	----------

non è in forma normale di Boyce and Codd perché esiste la dipendenza funzionale

$\text{Impiegato} \rightarrow \text{Stipendio}$  (Impiegato non è (super)chiave per la relazione )

---

## FORME NORMALI di Boyce-Codd

---

- **FNBC:**
  - Intuizione: se esiste in R una dipendenza  $X \rightarrow A$  non banale ed X non è chiave, allora X modella l'identità di un'entità diversa da quelle modellate dall'intera R
  - Ad esempio, in  
StudentiEdEsami(Matricola, Nome, Provincia, AnnoNascita, Materia, Voto)
    - Matricola  $\rightarrow$  Nome e Matricola non è (super)chiave.
      - il Nome dipende dalla Matricola che non è chiave.

- **Definizione**
- $R \langle T, F \rangle$  è in BCNF  $\Leftrightarrow$   
per ogni  $X \rightarrow A \in F^+$ , con  $A \notin X$  (non banale),  $X$  è una superchiave.
- **Teorema**  $R \langle T, F \rangle$  è in BCNF  $\Leftrightarrow$  per ogni  $X \rightarrow A \in F$  non banale,  
 $X$  è una superchiave.
- **Esempi:**
  - Docenti(CodiceFiscale, Nome, Dipartimento, Indirizzo)  $\{CF \rightarrow ND; D \rightarrow I\}$
  - Impiegati(Codice, Qualifica, NomeFiglio)  $\{C \rightarrow Q\}$



## L'ALGORITMO DI ANALISI

---

- $R< T, F >$  è decomposta in:  $R_1(X, Y)$  e  $R_2(X, Z)$  e su di esse si ripete il procedimento; esponenziale.
- input:  $R< T, F >$ , con  $F$  copertura canonica
- output: **decomposizione in BCNF che preserva i dati**

$$\rho = \{R< T, F >\}$$

while esiste in  $\rho$  una  $R_i< T_i, F_i >$  non in BCNF per la DF  $X \rightarrow A$   
do

$$T_a = X^+$$

$$F_a = \pi_{T_a}(F_i)$$

$$T_b = T_i - X^+ + X \quad \leftarrow$$

attenzione: errore nel vecchio libro

$$F_b = \pi_{T_b}(F_i)$$

$$\rho = \rho - R_i + \{ R_a < T_a, F_a >, R_b < T_b, F_b > \}$$

( $R_a$  ed  $R_b$  sono nomi nuovi)

end

---

## Osservazione

---

$$T_a = X^+$$

$$T_b = T_i - X^+ + X$$

Perché aggiungiamo X?

## PROPRIETA' DELL'ALGORITMO

---

- Preserva i dati, ma non necessariamente le dipendenze
- Esempi di decomposizioni senza perdita di dipendenze:
  - Docenti(CodiceFiscale, Nome, Dipartimento, Indirizzo), { $CF \rightarrow ND$ ;  $D \rightarrow I$ }  
 $(CF)^+ = CF \cup D \cup I$       è chiave  
 $(D)^+ = D \cup I$       non è chiave

Decompongono

- $R1(D, I)$ :                           $R2(CF, N, D)$
- $F1 = \{ D \rightarrow I \}$                    $F2 = \{ CF \rightarrow ND \}$

## PROPRIETA' DELL'ALGORITMO

---

- Preserva i dati, ma non necessariamente le dipendenze
- Esempi di decomposizioni senza perdita di dipendenze:
  - Impiegati(Codice, Qualifica, NomeFiglio) { $C \rightarrow Q$ }
  - $R1(\underline{C}, Q); \quad R2(C, NF)$
  - $F1 = \{ C \rightarrow Q \} \quad F2 = \{ \}$



Dato che non perdo  
dipendenze funzionali,  
posso fare la  
proiezione  
approssimata su F

## Esempio

---

- Telefoni (Prefisso, Numero, Località, Abbonato, Via)

$$F = \{ PN \rightarrow LAV, \quad L \rightarrow P \}$$

- Ad esempio:

- Pisa → 050
- Milano → 02

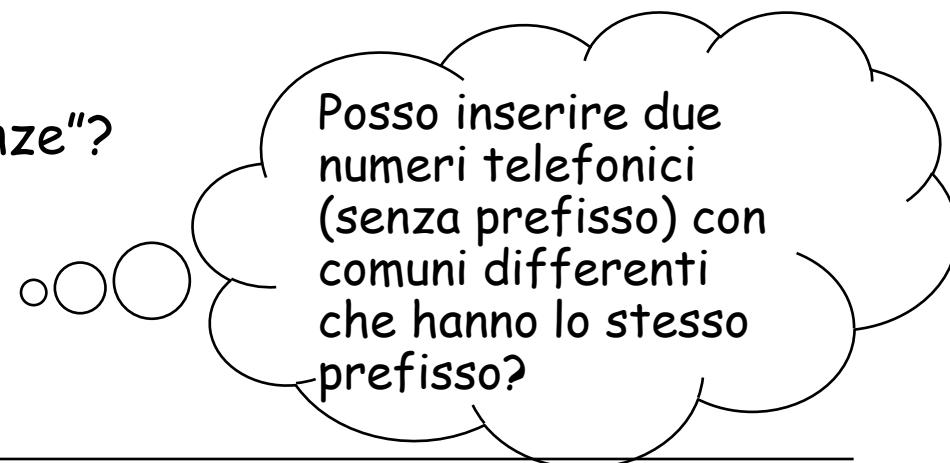
Preserva Dati e Dipendenze?



## PROPRIETA' DELL'ALGORITMO (cont.)

---

- Decomposizione con **perdita di dipendenze**:
- Telefoni(Prefisso, Numero, Località, Abbonato, Via),  
 $\{P \ N \rightarrow L \ A \ V, \ L \rightarrow P\}$ 
  - $R1(L, P); R2(L, N, A, V)$
  - Preserva dati ma non le dipendenze:  **$PN \rightarrow L$  non è deducibile da F1 e F2.**
- Cosa vuole dire "non preserva le dipendenze"?
  - $R1 = \{(Pisa, 050); (Calci, 050)\}$
  - $R2 = \{(Pisa, 506070, Rossi, Piave), (Calci, 506070, Bianchi, Isonzo)\}$



## Esercizio I

---

Si consideri il seguente schema relazionale  $R < ABCDE$ ,  $F = \{ CE \rightarrow A, D \rightarrow E, CB \rightarrow E, CE \rightarrow B \} >$  Applicare l'algoritmo di analisi e dire se dati e dipendenze sono stati preservati.

## Esercizio II

---

Si consideri il seguente schema relazionale  $R \leftarrow ABCDE, \{ AC \rightarrow D, BD \rightarrow A, BD \rightarrow E, A \rightarrow B \} \rightarrow$ . Applicare l'algoritmo di analisi e dire se dati e dipendenze sono stati preservati.

## Esercizio I

---

Si consideri il seguente schema relazionale  $R < ABCDE, F = \{ CE \rightarrow A, D \rightarrow E, CB \rightarrow E, CE \rightarrow B \} >$  Applicare l'algoritmo di analisi e dire se dati e dipendenze sono stati preservati.

Consideriamo  $CE \rightarrow A$ .  $CE^+ = CEAB$  (CE non è chiave), per cui decomponiamo:

$R1(CEAB)$  (gli attributi di  $CE^+$ ),  $R2(CED)$

(In  $R2$  tutti gli altri attribuiti ( $D$ ) e la chiave esterna ( $CE$ ))

Proiettiamo le dipendenze (approssimate su  $F$ ):

$R1 < CEAB, \{ CE \rightarrow A, CB \rightarrow E, CE \rightarrow B \} >$  (Proiezione in  $F$ )

$R2 < CED, \{ D \rightarrow E \} >$  (Proiezione in  $F$ )

$CE^+ = CEAB$  e  $CB^+ = CBEA$ , per cui  $R1$  è in BCNF

$D^+ = DE$ , per cui  $R2$  va ancora decomposta:  $R2 < CED, \{ D \rightarrow E \} > \rightarrow R3, R4$

La decomposizione è quindi:  $\{ R1(CBEA), R3(DE), R4(DC) \}$ .

La decomposizione preserva dati e dipendenze ed è in questo caso è la stessa prodotta dall'algoritmo di sintesi (che vedremo dopo).

## Esercizio II

---

Si consideri il seguente schema relazionale  $R < ABCDE, \{ AC \rightarrow D, BD \rightarrow A, BD \rightarrow E, A \rightarrow B \} >$ . Applicare l'algoritmo di analisi e dire se dati e dipendenze sono stati preservati.

Consideriamo  $AC \rightarrow D$ .  $AC^+ = ACDBE$  ( $AC$  è chiave)

Consideriamo  $BD \rightarrow A$ .  $BD^+ = BDAE$  ( $BD$  non è chiave), per cui decomponiamo:

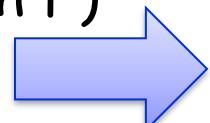
R1( $BDAE$ ) (gli attributi di  $BD^+$ ),

R2( $BDC$ ) (tutti gli altri attribuiti ( $C$ ) e la chiave esterna  $BD$ )

Proiettiamo le dipendenze (approssimate su F):

R1  $< BDAE, \{ BD \rightarrow A, BD \rightarrow E, A \rightarrow B \} >$  (Proiezione approx. in F)

R2  $< BDC, \{ \} >$  (Proiezione approx in F)



## Esercizio II - Soluzione Parte II

Si consideri il seguente schema relazionale

$R < ABCDE, \{ AC \rightarrow D, BD \rightarrow A, BD \rightarrow E, A \rightarrow B \} >$ .



Proiettiamo le dipendenze (approssimate su F):

$R1 < BDAE, \{ BD \rightarrow A, BD \rightarrow E, A \rightarrow B \} >$  (Proiezione in F)

$R2 < BDC, \{ \} >$  (Proiezione in F)

La dipendenza  $AC \rightarrow D$  si perde!

In R1:  $BD^+ = BDAE$  ( $BD$  è chiave), ma  $A \rightarrow B$  viola la BCNF ( $A^+ = AB$ ), per cui proiettando su R1 ( $BD \rightarrow A, BD \rightarrow E$  si perdono) abbiamo:

$R3 < AB, \{ A \rightarrow B \}, R4(ADE, \{ \} ) >$

La decomposizione è quindi  $\{ R2(BDC, \{ \}), R3(AB, \{ A \rightarrow B \}), R4(ADE, \{ \} ) \}$  che preserva i dati ma non le dipendenze.

## Riepilogo

---

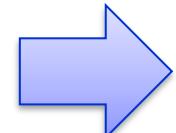
- Boyce-Codd Normal Form
  - Anomalia:
    - dipendenza  $X \rightarrow A$  non banale, con  $X$  non superchiave,
    - $X$  identità di un'entità con attributi  $X^+$  diversa da quelle modellate dall'intera  $R$
    - Es:
- StudentiEdEsami(Matricola, Nome, Prov, AnnoNascita, Materia, Voto)
  - con Matricola  $\rightarrow$  Nome, Prov, AnnoNascita
- Schema in BCNF = schema privo di anomalie
- Algoritmo di normalizzazione

## Una relazione non normalizzata

Data una relazione non in FNBC, è sempre possibile ottenere una decomposizione in FNBC?

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

- **Progetto Sede → Dirigente:** ogni progetto ha più dirigenti che ne sono responsabili, ma in sedi diverse, e ogni dirigente può essere responsabile di più progetti; però per ogni sede, un progetto ha un solo responsabile
- **Dirigente → Sede:** ogni dirigente opera presso una sede
- **Dirigente → Sede** è una dipendenza funzionale ma **Dirigente** non è una (super)chiave. Quindi la relazione non è in BCNF.



## La decomposizione è problematica

---

Dirigente	<u>Progetto</u>	<u>Sede</u>
Rossi	Marte	Roma
Verdi	Giove	Milano
Verdi	Marte	Milano
Neri	Saturno	Milano
Neri	Venere	Milano

Progetto Sede → Dirigente  
Dirigente → Sede

- **Progetto Sede → Dirigente** coinvolge tutti gli attributi e quindi nessuna decomposizione può preservare tale dipendenza.
- quindi **in alcuni casi la BCNF "non è raggiungibile"**
- **Occorre ricorrere a una forma normale indebolita**

## Algoritmi per la normalizzazione

---

- Quando si hanno diverse DF è difficile ragionare sullo schema, ed è quindi altrettanto difficile operare manualmente buone decomposizioni
- La terza forma normale (3NF) è un target di normalizzazione che consente di ottenere automaticamente:
  - decomposizioni senza perdita
  - decomposizioni che preservano tutte le dipendenze

## La terza forma normale

---

- Una relazione  $r$  è in terza forma normale (3NF) se, per ogni FD (non banale)  $X \rightarrow Y$  definita su  $r$ , è verificata almeno una delle seguenti condizioni:
  - $X$  contiene una chiave K di  $r$  (come nella BCNF)
  - Oppure ogni attributo in  $Y$  è contenuto in almeno una chiave K di  $r$

## Una relazione in terza forma normale

---

<u>Dirigente</u>	<u>Progetto</u>	<u>Sede</u>
Rossi	Marte	Roma
Verdi	Giove	Milano
Verdi	Marte	Milano
Neri	Saturno	Milano
Neri	Venere	Milano

Progetto Sede → Dirigente

Dirigente → Sede

Nella prima dipendenza funzionale il primo membro della dipendenza (**Progetto, Sede**) è una chiave, nella seconda il secondo membro (**Sede**) è contenuto in una chiave. Quindi la relazione è in **terza forma normale**

## BCNF e terza forma normale

---

(SVANTAGGI) La 3FN è **meno restrittiva** della FNBC

- ✧ **Tollerà alcune ridondanze** ed anomalie sui dati.
  - ✧ Es. per ogni occorrenza di un dirigente viene ripetuta la sua sede
- ✧ **Certifica meno lo qualità** dello schema ottenuto.

(VANTAGGI) La 3FN è **sempre ottenibile**, qualsiasi sia lo schema di partenza.

- ✧ **COME?** Algoritmo di **normalizzazione** in TFN!

## TERZA FORMA NORMALE

---

- **Definizione:**  $R<T, F>$  è in **3FN** se per ogni  $X \rightarrow A \in F^+$ , con  $A \notin X$ ,  $X$  è una superchiave o  $A$  è primo.
- Nota:
  - La 3FN ammette una dipendenza non banale e non-dachiave se gli attributi a destra sono primi;
  - la BCNF non ammette mai nessuna dipendenza non banale e non-dachiave.
- **Teorema:**

$R<T, F>$  è in 3FN se per ogni  $X \rightarrow A \in F$  non banale,  
allora  $X$  è una superchiave oppure  $A$  è primo.

## ESEMPI (Non sono in 3FN)

---

- Definizione:  $R \langle T, F \rangle$  è in 3FN se per ogni  $X \rightarrow A \in F^+$ , con  $A \notin X$ ,  $X$  è una superchiave o  $A$  è primo.
- Non sono in 3FN (e quindi, neppure in BCNF)
  - Docenti(CodiceFiscale, Nome, Dipartimento, Indirizzo)
    - $\{ CF \rightarrow N \ D; D \rightarrow I \}$
  - Impiegati(Codice, Qualifica, NomeFiglio)
    - $\{ C \rightarrow Q \}$

## ESEMPI (in 3FN, ma non in BCNF)

---

- Sono in 3FN, ma non in BCNF:
  - Telefoni(Prefisso, Numero, Località, Abbonato, Via)
    - $F = \{P\ N \rightarrow L\ A\ V, \quad L \rightarrow P\}$
    - Chiavi = {PN, LN}
  - Esami(Matricola, Telefono, Materia, Voto)
    - Matricola Materia  $\rightarrow$  Voto
    - Matricola  $\rightarrow$  Telefono
    - Telefono  $\rightarrow$  Matricola
    - Chiavi:
      - Matricola Materia,
      - Telefono Materia

## L'ALGORITMO DI SINTESI: VERSIONE BASE

---

Input: Un insieme  $R$  di attributi e un insieme  $F$  di dipendenze su  $R$ .

Output: Una decomposizione  $\rho = \{S_i\}_{i=1..n}$  di  $R$  tale che preservi dati e dipendenze e ogni  $S_i$  sia in 3NF, rispetto alle proiezioni di  $F$  su  $S_i$ .

begin

**Passo 1.** Trova una copertura canonica  $G$  di  $F$  e pon  $\rho = \{ \}$ .

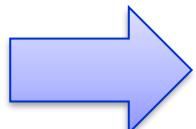
**Passo 2.** Sostituisci in  $G$  ogni insieme  $X \rightarrow A_1, \dots, X \rightarrow A_h$  di dipendenze con lo stesso determinante, con la dipendenza  $X \rightarrow A_1 \cdot \dots \cdot A_h$ .

**Passo 3.** Per ogni dipendenza  $X \rightarrow Y$  in  $G$ , metti uno schema con attributi  $XY$  in  $\rho$ .

**Passo 4.** Elimina ogni schema di  $\rho$  contenuto in un altro schema di  $\rho$ .

**Passo 5.** Se la decomposizione non contiene alcuno schema i cui attributi costituiscono una superchiave per  $R$ , aggiungi ad essa lo schema con attributi  $W$ , con  $W$  una chiave di  $R$ .

end



## L'ALGORITMO DI SINTESI: VERSIONE BASE

- Dati R di attributi, ed un insieme di dipendenze F, l'algoritmo di sintesi di schemi in terza forma normale procede come segue:

Impiegato (Matricola, Cognome, Grado, Retribuzione, Dipartimento, Supervisore, Progetto, Anzianità)

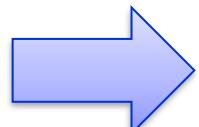
{ $M \rightarrow RSDG$ ,  $MS \rightarrow CD$ ,  $G \rightarrow R$ ,  $D \rightarrow S$ ,  $S \rightarrow D$ ,  $MPD \rightarrow AM$ }

STEP 1. Costruire una copertura canonica G di F.

$F = \{M \rightarrow RSDG, MS \rightarrow CD, G \rightarrow R, D \rightarrow S, S \rightarrow D, MPD \rightarrow AM\}$

$G = \{M \rightarrow D, M \rightarrow G, M \rightarrow C, G \rightarrow R, D \rightarrow S, S \rightarrow D, MP \rightarrow A\}$

Vedi slide  
copertura  
canonica



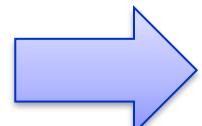
## L'ALGORITMO DI SINTESI: VERSIONE BASE

---

$F = \{M \rightarrow R, SDG, MS \rightarrow CD, G \rightarrow R, D \rightarrow S, S \rightarrow D, MPD \rightarrow AM\}$

$G = \{M \rightarrow D, M \rightarrow G, M \rightarrow C, G \rightarrow R, D \rightarrow S, S \rightarrow D, MP \rightarrow A\}$

- STEP 2a. Decomporre  $G$  nei sottoinsiemi  $G^{(1)}, G^{(2)}, \dots, G^{(n)}$ , tali che ad ogni sottoinsieme appartengano dipendenze con gli stessi lati sinistri (facoltativo)
- $G^{(1)} = \{M \rightarrow D, M \rightarrow G, M \rightarrow C\}$
- $G^{(2)} = \{G \rightarrow R\}$
- $G^{(3)} = \{D \rightarrow S\}$
- $G^{(4)} = \{S \rightarrow D\}$
- $G^{(5)} = \{MP \rightarrow A\}$



## L'ALGORITMO DI SINTESI: VERSIONE BASE

---

$$G = \{M \rightarrow D, M \rightarrow G, M \rightarrow C, G \rightarrow R, D \rightarrow S, S \rightarrow D, MP \rightarrow A\}$$

- STEP 2b sostituisci in  $G$  ogni insieme  $X \rightarrow A_1, \dots, X \rightarrow A_h$  di dipendenze con lo stesso determinante, con la dipendenza  $X \rightarrow A_1 \dots A_h$ .

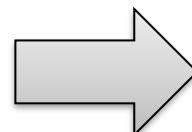
$$G^{(1)} = \{M \rightarrow D, M \rightarrow G, M \rightarrow C\}$$

$$G^{(1)} = \{M \rightarrow DGC\}$$

$$G^{(2)} = \{G \rightarrow R\}$$

$$G^{(2)} = \{G \rightarrow R\}$$

$$G^{(3)} = \{D \rightarrow S\}$$



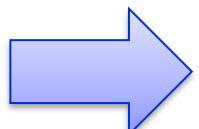
$$G^{(3)} = \{D \rightarrow S\}$$

$$G^{(4)} = \{S \rightarrow D\}$$

$$G^{(4)} = \{S \rightarrow D\}$$

$$G^{(5)} = \{MP \rightarrow A\}$$

$$G^{(5)} = \{MP \rightarrow A\}$$



## L'ALGORITMO DI SINTESI: VERSIONE BASE

---

- STEP 3. Trasformare ciascun  $G^{(i)}$  in una relazione  $R^{(i)}$  con gli attributi contenuti in ciascuna dipendenza.

- Il lato sinistro diventa la chiave della relazione.

- Step 2

Step 3

- $G^{(1)} = \{M \rightarrow DGC\}$ :

$R^{(1)}(\underline{MDGC})$

- $G^{(2)} = \{G \rightarrow R\}$ :

$R^{(2)}(\underline{GR})$

- $G^{(3)} = \{D \rightarrow S\}$ :

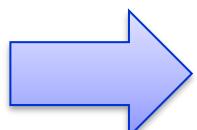
$R^{(3)}(\underline{SD})$

- $G^{(4)} = \{S \rightarrow D\}$ :

$R^{(4)}(\underline{SD})$

- $G^{(5)} = \{MP \rightarrow A\}$ :

$R^{(5)}(\underline{MPA})$



# L'ALGORITMO DI SINTESI: VERSIONE BASE

- STEP 4. Si eliminano schemi contenuti in altri.

(se allo step precedente due o più lati sinistri delle dipendenze si implicano a vicenda, si fondono i relativi insiemi.)

## Step 2

- $G^{(1)} = \{M \rightarrow DGC\}$ :

## Step 3

$R^{(1)}(\underline{MDGC})$

- $G^{(2)} = \{G \rightarrow R\}$ :

$R^{(2)}(\underline{GR})$

- $G^{(3)} = \{D \rightarrow S\}$ :

$R^{(3)}(\underline{DS})$

- $G^{(4)} = \{S \rightarrow D\}$ :

$R^{(4)}(\underline{SD})$

- $G^{(5)} = \{MP \rightarrow A\}$ :

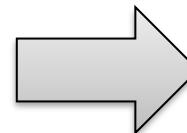
$R^{(5)}(\underline{MPA})$

## Step 4

$R^{(1)}(\underline{MDGC})$



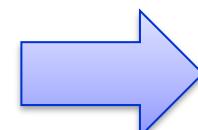
$G^{(3)} = \{D \rightarrow S, S \rightarrow D\}$



$R^{(2)}(\underline{GR})$

$R^{(3)}(\underline{DS})$

$R^{(5)}(\underline{MPA})$



## L'ALGORITMO DI SINTESI: VERSIONE BASE

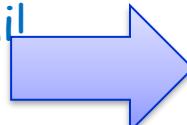
---

- STEP 5. Se nessuna relazione  $R^{(i)}$  così ottenuta contiene una (super)chiave K di R(U), inserire una nuova relazione  $R^{(n+1)}$  contenente gli attributi della chiave.
- Impiegato (Matricola, Cognome, Grado, Retribuzione, Dipartimento, Supervisore, Progetto, Anzianità)

$$F = \{M \rightarrow RSDG, MS \rightarrow CD, G \rightarrow R, D \rightarrow S, S \rightarrow D, MPD \rightarrow AM\}$$

$$G = \{M \rightarrow D, M \rightarrow G, M \rightarrow C, G \rightarrow R, D \rightarrow S, S \rightarrow D, MP \rightarrow A\}$$

- la chiave è costituita da: (MP).
- Dallo step 4:  $R^{(1)}(MDGC)$     $R^{(2)}(GR)$     $R^{(3)}(SD)$     $\textcolor{red}{R^{(5)}(MPA)}$
- $\textcolor{blue}{R^{(5)}(MPA)}$  contiene la chiave → non c'è necessità di aggiungere altre relazioni!



## L'ALGORITMO DI SINTESI: VERSIONE BASE

---

- In conclusione, data la relazione: **R(MGCRDSPA)**, con dipendenze funzionali:

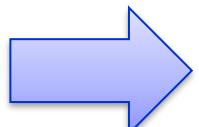
Impiegato (Matricola, Cognome, Grado, Retribuzione, Dipartimento,  
Supervisore, Progetto, Anzianità)

$$F = \{M \rightarrow RSDG, MS \rightarrow CD, G \rightarrow R, D \rightarrow S, S \rightarrow D, MPD \rightarrow AM\}$$

$$G = \{M \rightarrow D, M \rightarrow G, M \rightarrow C, G \rightarrow R, D \rightarrow S, S \rightarrow D, MP \rightarrow A\}$$

La sua decomposizione in 3FN è la seguente:

- $R^{(1)}(MDGC) \quad R^{(2)}(GR) \quad R^{(3)}(SD) \quad R^{(4)}(MPA)$



## LE DF NON BASTANO: DIPENDENZE MULTIVALORE

<b>Impiegati</b>
<b>Codice</b>
<b>Stipendi: seq num</b>
<b>NomeFigli: seq string</b>

- Impiegati(Codice, StoriaStipendio, NomeFiglio)

c1	s1	n1
c1	s1	n2
c1	s2	n1
c1	s2	n2

Verdi	1000	Giorgio
Verdi	1000	Anna
Verdi	1400	Giorgio
Verdi	1400	Anna
Rossi	1900	Gino

- La coesistenza di due proprietà multivalue INDIPENDENTI, fa sì che per ogni impiegato esistono tante ennuple quante sono le possibili coppie di valori di Qualifica e NomeFiglio.

<b>Impiegati</b>
<b>Codice</b>
<b>Qualifiche: seq num</b>
<b>NomeFigli: seq string</b>

<b>Impiegati</b>
<b>Codice</b>
<b>Posizioni: seq (Qualifica, NomeDirigente)</b>

Tre casi di relazioni con proprietà multivalori. Si possono risolvere usando le decomposizioni?

## Le DF non bastano: DF multivalue - Esempio 1

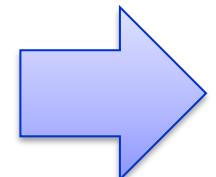
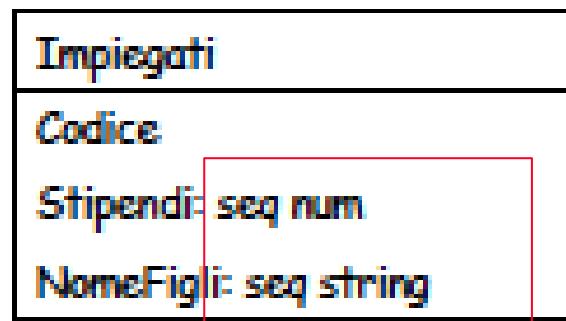
- Impiegati(Codice, StoriaStipendio, NomeFiglio)

c1	s1	n1
c1	s1	n2
c1	s2	n1
c1	s2	n2

Cogn, StoriaStip,Figlio

Verdi	1000	Giorgio
Verdi	1000	Anna
Verdi	1400	Giorgio
Verdi	1400	Anna
Rossi	1900	Gino

- La coesistenza di due proprietà multivalue **indipendenti**, fa sì che per ogni impiegato esistano tante ennuple quante sono le possibili coppie di valori di Stipendio e NomeFiglio.



## Le DF non bastano: DF multivalue - Esempio 1

---

- Impiegati(Codice, StoriaStipendio, NomeFiglio)

c1	s1	n1
c1	s1	n2
c1	s2	n1
c1	s2	n2

Verdi	1000	Giorgio
Verdi	1000	Anna
Verdi	1400	Giorgio
Verdi	1400	Anna
Rossi	1900	Gino

- Decomponendo lo schema in due sottoschemi in modo da modellare separatamente le proprietà multivalori indipendenti, si avrebbe una base di dati priva di anomalie:
  - StipendiImpiegati(Codice, StoriaStipendio)
  - FigliImpiegati(Codice, NomeFiglio)

## LE DF NON BASTANO: DIPENDENZE MULTIVALORE

---

- Altro esempi:

Impiegati

Codice

Qualifiche: seq num

NomeFigli: seq string

Impiegati

Codice

Posizioni: seq (Qualifica,  
NomeDirigente)



Si possono risolvere usando  
le decomposizioni?

---

# SQL

Materiale adattato dal libro Albano et al e  
dal libro Atzeni-et al., Basi di dati

## Interrogazione di una base dati

---

- L'interrogazione di una base di dati è uno degli aspetti più importanti del linguaggio SQL.
- I comandi di interrogazione, o **QUERY**, permettono di effettuare una ricerca dei dati presenti nel database che soddisfano particolari condizioni, richieste dall'utente

## SQL

---

```
SELECT s.Nome, e.Data  
FROM Studenti s, Esami e  
WHERE e.Materia = 'BD' AND e.Voto = 30 AND e.Matricola =  
s.Matricola
```

```
SELECT s.Nome As Nome, 2018 - s.AnnoNascita As Eta,  
0 As NumeroEsami  
FROM Studenti s  
WHERE NOT EXISTS (SELECT *  
                   FROM Esami e  
                   WHERE e.Matricola = s.Matricola )
```

## SQL: Structured Query Language

---

- SQL è stato definito nel 1973 ed è oggi il linguaggio universale dei sistemi relazionali
- Standard: SQL-84, SQL-89, SQL-92 (o SQL2), SQL:1999 (o SQL3) (ANSI/ISO)
- SQL-92: entry, intermediate e full SQL.
- SQL:1999: a oggetti.
- SQL: DDL, DML, Query language.

# Una distinzione terminologica (separazione fra dati e programmi)

---

Nei linguaggi di interrogazione di basi di dati  
distinguiamo tra

## Data Manipulation Language (DML)

per l'interrogazione e l'aggiornamento di  
(**istanze** di) basi di dati

Select insegnamento

From Orario

Where docente="Mario Rossi"

## Data Definition Language (DDL)

per la definizione di **schemi** (logici, esterni,  
fisici) e altre operazioni generali

```
CREATE TABLE orario (
    insegnamento CHAR(20) ,
    docente      CHAR(20) ,
    aula         CHAR(4) ,
    ora          CHAR(5)
)
```

## Istruzione SELECT per l'interrogazione

---

- Le query vengono effettuate mediante il comando SELECT.
- Sintassi:

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

Bisogna specificare:

- La "target list" cioè la lista degli attributi interessati.
- clausola **FROM** per stabilire in quale/quali tabella/e sono contenuti i dati che ci occorrono.
- clausola **WHERE** per esprimere le condizioni che i dati cercati devono soddisfare.

## Target List (lista degli attributi)

---

- Specificare la target list corrisponde a scegliere alcuni degli attributi della tabella o delle tabelle considerate.
- Implementa l'operazione di **proiezione** dell'algebra relazionale

Rubrica

Matr	Cognome	Nome	Email	tel

Select Cognome, nome, tel  
FROM Rubrica

## Where

---

- La clausola **WHERE** serve a scegliere le righe della tabella che soddisfano una certa condizione.
- In questo modo la clausola where implementa la **selezione** dell'algebra relazionale

Rubrica

Matr	Cogn	Nome	Email	tel
...	...	Pippo	...	...
...	...	Mario	...	...
...	...	Silvia	...	...
...	...	Mario	...	...
...	...	Marco	...	...
...	...	Mario	...	...

SELECT \*  
FROM Rubrica  
WHERE nome='Mario'

## From

---

- La clausola **FROM** ha lo scopo di scegliere quali sono le tabelle del database da cui vogliamo estrarre le nostre informazioni.
- Nel caso in cui le tabelle elencate sono due, la clausola **FROM**, insieme con la clausola **WHERE**, che stabilisce quali righe delle due tabelle bisogna accoppiare, implementa il **theta join**

**SELECT \***

**FROM Studenti, Esami**

**Where Studenti.Matricola=Esami.Studente**

Studenti

Matricola	Nome	Cognome	Indirizzo

Esami

cod.Materia	Nome	Docente	studente

## SELECT

---

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

La query considera il prodotto cartesiano tra le tabelle in *ListaTabelle* (**JOIN**).

Fra queste seleziona solo le righe che soddisfano la *Condizione* (**SELEZIONE**) e infine valuta le espressioni specificate nella target list *ListaAttributi* (**PROIEZIONE**).

La SELECT implementa quindi gli operatori di Proiezione, Selezione e Join dell'algebra Relazionale, ecc.

## SQL PER INTERROGARE: SELECT FROM WHERE

---

- SQL è un calcolo su **multiinsiemi**.

- Il comando base dell'SQL:

```
SELECT [DISTINCT] Attributo {, Attributo}  
FROM Tabella [Identificatore] {, Tabella [Identificatore]}  
[WHERE Condizione]
```

- Semantica: prodotto + restrizione + proiezione.

- Un attributo  $A$  di una tabella "R"  $x$  si denota come:

- $A$  oppure
- $R.A$  oppure
- $x.A$



Ridenominazione  
della tabella R

## LA LISTA DEGLI ATTRIBUTI

---

- Attributi ::= \*
  - | Expr [[AS] Nuovonome] {, Expr [[AS] Nuovonome]}
- Expr ::= [Ide.]Attributo | Const
  - | ( Expr ) | [-] Expr [Op Expr]
  - | COUNT(\*)
  - | AggrFun ( [DISTINCT] [Ide.]Attributo)
- e AS x: dà un nome alla colonna di e
- AggrFun ::= SUM | COUNT | AVG | MAX | MIN
- AggrFun: o si usano tutte funzioni di aggregazione (e si ottiene un'unica riga) o non se ne usa nessuna.

## LA LISTA DELLE TABELLE

---

- Le tabelle si possono combinare usando:
  - "," (prodotto): FROM T1,T2
  - Giunzioni di vario genere:
    - Studenti s JOIN Esami e ON s.Matricola=e.Matricola
    - Studenti s JOIN Esami e USING Matricola
    - Studenti s NATURAL JOIN Esami e
    - Studenti s LEFT JOIN Esami e ON s.Matricola=e.Matricola
    - LEFT JOIN - USING
    - NATURAL LEFT JOIN
    - RIGHT JOIN
    - FULL JOIN



Il join può essere realizzato anche attraverso la clausola WHERE

## LA CONDIZIONE

---

- Combinazione booleana di predicati tra cui:
  - Expr Comp Expr
  - Expr Comp ( Sottoselect che torna un valore)
  - [NOT] EXISTS (Sottoselect)
  - Expr Comp (ANY | ALL) (Sottoselect)
  - Expr [NOT] IN ( Sottoselect) (oppure IN (v1,..,vn))
- Comp: <, =, >, <>, <=, >=

# SINTASSI DELLA SELECT

---

- Sottoselect:

```
SELECT [DISTINCT] Attributi  
      FROM Tabelle  
      [WHERE Condizione]  
      [GROUP BY A1,...,An [HAVING Condizione]]
```

- Select:

```
Sottoselect  
  { (UNION | INTERSECT | EXCEPT)  
    Sottoselect }  
  [ ORDER BY Attributo [DESC] {, Attributo [DESC]} ]
```

## Esempio: proiezione

- Visualizzare il nome e l'età dei dati presenti nella tabella Persone

SELECT nome, eta

FROM Persone

Nome	Eta
ANDREA	27
ALDO	25
MARIA	55
ANNA	50
FILIPPO	26
LUIGI	50
FRANCO	60
OLGA	30
SERGIO	85
LUISA	75

## Persone

NOME	ETA	REDDITO
Andrea	27	21
Aldo	25	15
Maria	55	42
Anna	50	35
Filippo	26	29
Luigi	50	40
Franco	60	20
Olga	30	41
Sergio	85	35
Luisa	75	87

## ESEMPI: Proiezione

---

- Trovare il nome, la matricola e la provincia degli studenti:

```
SELECT Nome, Matricola, Provincia  
FROM    Studenti
```

Nome	Matricola	Provincia
Isaia	171523	PI
Rossi	167459	LU
Bianchi	179856	LI
Bonini	175649	PI

## Selezione (Restrizione), la clausola where

---

- Con SQL è anche possibile effettuare la selezione dell'algebra relazionale mediante la clausola **WHERE**.
- La clausola **WHERE** permette infatti di specificare le condizioni che devono soddisfare le righe cercate.
- Ovviamente la clausola WHERE è opzionale, e se si omette, si selezionano tutte le righe della tabella specificata
- **Esempio:** Nome, età e reddito delle persone con meno di trent'anni:

SELECT nome, eta, reddito

FROM persone

WHERE eta < 30

## Esempio query

- Nome e reddito delle persone con meno di trenta anni

$\pi_{\text{Nome}, \text{Reddito}}(\sigma_{\text{Eta} < 30}(\text{Persone}))$

SELECT nome, reddito

FROM persone

WHERE eta < 30

Nome	Reddito
Andrea	21
Aldo	15
Filippo	29

## Persone

NOME	ETA	REDDITO
Andrea	27	21
Aldo	25	15
Maria	55	42
Anna	50	35
Filippo	26	29
Luigi	50	40
Franco	60	20
Olga	30	41
Sergio	85	35
Luisa	75	87

## Esempio

---

- Visualizzare tutte le colonne della tabella Maternita.

SELECT \*

From Maternita

Se si desidera visualizzare **tutti gli attributi** della tabella, si può semplificare la target list indicando la lista con un semplice **asterisco \***

Maternita

Madre	Figlio
Luisa	Luigi
Luisa	Maria
Anna	Olga
Anna	Filippo
Maria	Andrea
Maria	Aldo

## ESEMPI: Restrizione

---

- Trovare tutti i dati degli studenti di Pisa:

```
SELECT *  
FROM Studenti  
WHERE Provincia = 'PI'
```

Nome	Matricola	Provincia	AnnoNascita
Isaia	171523	PI	1996
Bonini	175649	PI	1996

- Trovare la matricola, l'anno di nascita e il nome degli studenti di Pisa (*Proiezione+Restrizione*):

```
SELECT Nome, Matricola, AnnoNascita  
FROM Studenti  
WHERE Provincia = 'PI'
```

Nome	Matricola	AnnoNascita
Isaia	171523	1996
Bonini	175649	1996

## ESEMPI: Prodotto e giunzione

Vedremo il join più in dettaglio più avanti

- Trovare tutte le possibili coppie Studente-Esame:

```
SELECT *  
FROM Studenti, Esami
```

- Trovare tutte le possibili coppie Studente - Esame sostenuto dallo studente:

```
SELECT *  
FROM Studenti s, Esami e  
WHERE s.Matricola = e.Matricola
```

- Trovare il nome e la data degli esami per gli studenti che hanno superato l'esame di BD con 30:

```
SELECT Nome, Data  
FROM Studenti s, Esami e  
WHERE e.Materia = 'BD' AND e.Voto = 30  
AND e.Matricola = s.Matricola
```

## Alias delle colonne

---

- L'alias serve a dare a una colonna un nome diverso rispetto a quello che è utilizzato nella definizione della tabella.
- Implementa l'operatore **p** (**Ridenominazione**) dell'algebra relazionale
- Può essere usata opzionalmente la parola chiave **AS** tra il nome della colonna e l'alias richiede necessariamente le virgolette se l'alias ha degli spazi.

Select Figlio **as** Figlio\_M  
From Maternita

Select Figlio 'Figlio madre'  
From Maternita

## Condizioni

---

- La condizione che segue il costrutto where è una condizione Booleana. Fa quindi uso di
  - Operatori di confronto  $=$ ,  $<$ ,  $>$ ,  $\neq$  (oppure  $\neq$ ),  $\leq$ ,  $\geq$
  - Connettori logici **AND**, **OR**, **NOT**
  - Operatori **BETWEEN**, **IN**, **LIKE**, **IS NULL**

## Operatori di confronto, esempi

---

Selezionare le persone che si chiamano Mario

```
SELECT *  
FROM Persone  
Where nome='Mario'
```

Persone

NOME	ETA	REDDITO
------	-----	---------

Selezionare gli impiegati che guadagnano più di 1200 euro

```
SELECT *  
FROM Impiegato  
WHERE Stip>1200
```

Stipendio

NOME	Cognome	Stip
------	---------	------

## Operatori di confronto, esempi

---

Selezionare il nome e cognome delle persone che non hanno età superiore a 30 anni

```
SELECT Nome, Cognome  
FROM Persone  
WHERE eta<=30
```

Persone

NOME	ETA	REDDITO
------	-----	---------

Selezionare tutti gli impiegati tranne quelli che lavorano nel dipartimento di Produzione.

```
SELECT *  
FROM Impiegato  
WHERE dipartimento <> 'Produzione'
```

Impiegati

Matricola	Cognome	Dipart	Stip
-----------	---------	--------	------

## Uso dell'operatore BETWEEN

---

- **BETWEEN** consente la selezione di righe con attributi in un particolare range.
- Esempio: cercare gli impiegati (nome e dipartimenti) che guadagnano tra 1000 e 1500 euro e visualizzare nome e dipartimento:

**Impiegati**

Matricola	nome	Dipart	Stip
-----------	------	--------	------

SELECT nome, Dipart

FROM Impiegato

Where Stip **between** 1000 and 1500

## Uso dell'operatore IN

---

- E' usato per selezionare righe che hanno un attributo che assume valori contenuti in una lista.

### ESEMPIO

Selezionare i nomi dei professori che tengono i corsi di BD1, BD2, Algoritmi e Sistemi

```
SELECT *
FROM Insegnanti
WHERE corso IN ('BD1', 'BD2', 'Algoritmi', 'Sistemi')
```

## Connettori logici, Esempi

---

Selezionare le persone che hanno più di 30 anni e che non abitano a Pisa

Persone

NOME	ETA	DIPART	CITTA
------	-----	--------	-------

```
SELECT *  
FROM Persone  
WHERE (eta>30) AND NOT(citta='Pisa')
```

Selezionare gli impiegati che lavorano nel dipartimento di produzione e quelli che lavorano in segreteria

```
SELECT *  
FROM Persone  
WHERE Dipart= 'Produzione' OR Dipart= 'Segreteria'
```

## Uso dell'operatore LIKE

---

LIKE è usato per effettuare ricerche “wildcard” (ossia con un simbolo jolly) di una stringa di valori.

Le condizioni di ricerca possono contenere letterali, caratteri o numeri.

Esistono due tipi di wildcard:

% denota zero o più caratteri.

\_ denota un carattere.

## Uso dell'operatore LIKE

---

Esempio: Fra le persone elencate nella tabella Persone, determinare quelle il cui nome termina per 'a'

SELECT \*

FROM Persone

Where nome LIKE '%a'

NOME	ETA	REDDITO
Andrea	27	21
Maria	55	42
Anna	50	35
Olga	30	41
Luisa	75	87

Esempio: Selezionare le sequenze di DNA in cui ci sono almeno due simboli 'G' che distano 3 caratteri

SELECT \*

FROM DNA

WHERE Sequenza LIKE '%G\_\_G%'

## Simbolo escape

---

- A volte può succedere che uno dei simboli coinvolti nel pattern matching sia proprio \_ oppure %.
- In questo caso, si sceglie un simbolo che non è ammesso fra i simboli della stringa (supponiamo '#') detto **simbolo escape**, nell'espressione per la ricerca si fa precedere il simbolo \_ o % cercato dal simbolo escape, e poi si specifica che '#' e' il simbolo di escape.
- Esempio: Modelli il cui nome inizia per C\_F

SELECT \*

FROM Modelli

WHERE nome\_modello LIKE 'C#\_F%' ESCAPE '#'

## IL VALORE NULL

---

- Il valore di un campo di un'ennupla può mancare per varie ragioni; SQL fornisce il valore speciale NULL per tali situazioni.
- La presenza del NULL introduce dei problemi:
  - occorrono dei predicati per controllare se un valore è/non è NULL.
  - la condizione "reddito>8" è vera o falsa quando il reddito è uguale a NULL? Cosa succede degli operatori AND, OR e NOT?
  - Occorre una logica a 3 valori (vero, falso e unknown).
  - Va definita opportunamente la semantica dei costrutti. Ad es. il WHERE elimina le ennuple che non rendono vera la condizione.
  - Nuovi operatori sono utili (es. giunzioni esterne)

## Is Null

---

- L'operatore **IS NULL** verifica se il contenuto di un operando è null.
  - A IS NULL è true se A vale NULL, false altrimenti
  - A IS NOT NULL è true se A ha un valore noto, false altrimenti
- **Esempio:**

Veicoli

Targa	Cod_mod	Cod_cat	Cilindrata	Cod_comb.	cav.Fisc	Velocita	Posti	Imm
-------	---------	---------	------------	-----------	----------	----------	-------	-----

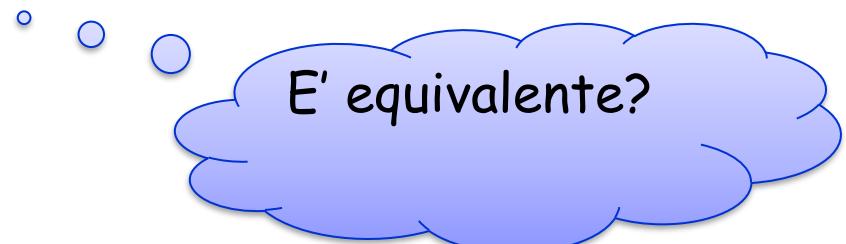
```
SELECT *
FROM Veicoli
WHERE Cilindrata IS NULL
```

### Impiegati

Matricola	Cognome	Filiale	Età
5998	Neri	Milano	45
9553	Bruni	Milano	NULL

$\sigma$  Età > 40 OR Età IS NULL (Impiegati)

Select \*  
FROM Impiegati  
WHERE Eta>40 or Eta=null



## Logica predici a tre valori: True, false, sconosciuto (NULL)

---

- I valori NULL rappresentano l'assenza di informazione, cioè:
  - Valore sconosciuto
  - Valore non disponibile
  - Attributo non applicabile
- Ciascun valore NULL è quindi considerato diverso dagli altri valori NULL
- Il risultato del confronto logico con un valore che è NULL da come output: sconosciuto (UNKNOWN), indicato con S

NOT	
V	F
F	V
S	S

OR	V	F	S
V	V	V	V
F	V	F	S
S	V	S	S

AND	V	F	S
V	V	F	S
F	F	F	F
S	S	F	S

Esempio

OR	V	F	S	AND	V	F	S
V	V	V	V	V	V	F	S
F	V	F	S	F	F	F	F
S	V	S	S	S	S	F	S

- `SELECT * FROM Persone WHERE Nome=Luigi OR AnnoNascita=2022;`

Persone

Nome	Cognome	AnnoNascita	LuogoDiNascita
Luigi	Bianchi	NULL	Roma
Mario	Verdi	2022	Pisa
Marco	Rossi	NULL	NULL

T OR S → T

F OR T → T

F OR S → S

- `SELECT * FROM Persone WHERE Nome=Luigi AND AnnoNascita=2022;`

Persone

Nome	Cognome	AnnoNascita	LuogoDiNascita
Luigi	Bianchi	NULL	Roma
Mario	Verdi	2022	Pisa
Marco	Rossi	NULL	NULL

T AND S → S

F AND T → F

F AND S → F

OR	V	F	S	AND	V	F	S
V	V	V	V	V	V	F	S
F	V	F	S	F	F	F	F
S	V	S	S	S	S	F	S

Esempio

- SELECT \* FROM Persone WHERE AnnoNascita IS NULL

Nome	Cognome	AnnoNascita	LuogoDiNascita	Persone
Luigi	Bianchi	NULL	Roma	ok
Mario	Verdi	2022	Pisa	
Marco	Rossi	NULL	NULL	ok

- SELECT \* FROM Persone WHERE AnnoNascita IS NULL AND/OR Luogo IS NULL

Nome	Cognome	AnnoNascita	Luogo	Persone
Luigi	Bianchi	NULL	Roma	Restituita dall'OR
Mario	Verdi	2022	Pisa	
Marco	Rossi	NULL	NULL	Restituita dall'AND e dall'OR

OR	V	F	S
V	V	V	V
F	V	F	S
S	V	S	S

## Valore NULL e operatori logici

MATRICOLA	NOME	COGNOME	DATAISCRIZIONE
282320	Gianna	Neri	04-MAG-20
369871	Mario	Rossi	04-MAG-20
515140	Mario	Verdi	
090456	Mario	Bianchi	04-MAG-20
579555	Luigi	Rossi	
018701	Luca	Bianchi	04-MAG-20

Select \* from studenti  
where cognome='Rossi'

MATRICOLA	NOME	COGNOME	DATAISCRIZIONE
369871	Mario	Rossi	04-MAG-20
579555	Luigi	Rossi	

Select \* from studenti  
where cognome='Rossi' or  
dataiscrizione=NULL

MATRICOLA	NOME	COGNOME	DATAISCRIZIONE
369871	Mario	Rossi	04-MAG-20
579555	Luigi	Rossi	(null)

Select \* from studenti  
where cognome='Rossi' or  
dataiscrizione IS NULL

MATRICOLA	NOME	COGNOME	DATAISCRIZIONE
369871	Mario	Rossi	04-MAG-20
515140	Mario	Verdi	
579555	Luigi	Rossi	

AND	V	F	S
V	V	F	S
F	F	F	F
S	S	F	S

## Valore NULL e operatori logici

MATRIC	NOME	COGNOME	DATAISCRIT
282320	Gianna	Neri	04-MAG-20
369871	Mario	Rossi	04-MAG-20
515140	Mario	Verdi	
090456	Mario	Bianchi	04-MAG-20
579555	Luigi	Rossi	
018701	Luca	Bianchi	04-MAG-20

Select \* from studenti  
where cognome='Rossi'

MATRIC	NOME	COGNOME	DATAISCRIT
369871	Mario	Rossi	04-MAG-20
579555	Luigi	Rossi	

Select \* from studenti  
where cognome='Rossi' **and**  
dataiscrizione **IS NULL**

MATRIC	NOME	COGNOME	DATAISCRIT
579555	Luigi	Rossi	

Select \* from studenti  
where cognome='Rossi' **and**  
dataiscrizione=**NULL**

nessuna riga selezionata

## Forma negativa

---

Tutti gli operatori descritti sono presenti anche in forma Negativa, con ovvio significato.

- NOT BETWEEN
- NOT IN
- NOT LIKE
- NOT NULL

## Espressione nella target list

---

- Esempio:
- Evidenziare i cognomi degli impiegati e il loro stipendio aumentato del 20%

IMPIEGATO

Nome	Cognome	Stip
Mario	Rossi	1200
Luigi	Verdi	1130
Maria	Bianchi	1450
Luisa	Gialli	1300

Select cognome, stip \*120/100  
FROM Impiegato

Cognome	Stip * 120/100
Rossi	1440
Verdi	1356
Bianchi	1740
Gialli	1560

## Espressione nella target list con ridenominazione

---

- Esempio:
- Evidenziare i cognomi degli impiegati e il loro stipendio aumentato del 20%.
- Rinominare la colonna corrispondente al salario aumentato con la denominazione "Aumento".

Select cognome,  $\text{stip} * 120 / 100$  Aumento  
FROM Impiegato

IMPIEGATO

Nome	Cognome	Stip
Mario	Rossi	1200
Luigi	Verdi	1130
Maria	Bianchi	1450
Luisa	Gialli	1300

Cognome	Aumento
Rossi	1440
Verdi	1356
Bianchi	1740
Gialli	1560

---

# ORDINAMENTO OPERATORI AGGREGATI E RAGGRUPPAMENTO

## Ordinamento del risultato

---

E' possibile dare un ordinamento del risultato di una select. L'ordinamento si può effettuare in base a un attributo, e può essere crescente o decrescente. La sintassi è la seguente:

```
SELECT lista_attributi  
FROM nome_tabella  
WHERE condizioni  
ORDER BY Attributo [ASC/DESC]
```

Le righe vengono ordinate in base al campo Attributo in maniera crescente o decrescente secondo se è data la specifica **ASC** o **DESC**. **ASC** è il default. Secondo il tipo dell'attributo, l'ordinamento è quello più naturale su quel dominio.

## Esempio

---

Nome e reddito delle persone con meno di trenta anni in ordine alfabetico

```
SELECT nome, reddito  
FROM persone  
WHERE reddito < 30  
ORDER BY nome
```

Persone

Nome	Reddito
Andrea	21
Aldo	15
Filippo	30

Persone

Nome	Reddito
Aldo	15
Andrea	21

## Doppio ordinamento

---

Si può voler ordinare i dati in base a una certa chiave (attributo) e poi ordinare i dati che coincidono su quella chiave in base a un'altra chiave (attributo).

Ordinare gli studenti in base al loro cognome, in modo tale che due persone con lo stesso cognome siano ordinate in base al nome, e persone con lo stesso nome e cognome siano ordinate in base all'ordine inverso della data di nascita

```
Select *  
From Studenti  
Order by cognome [asc], nome [asc] , nascita desc
```

## Operatori aggregati

---

- Nella target list possiamo avere anche espressioni che calcolano valori a partire da insiemi di ennuple e che restituiscono una tabella molto particolare, costituita da un singolo valore scalare.
- SQL-2 prevede 5 possibili operatori aggregati:
  - Conteggio (COUNT),
  - Minimo (MIN),
  - Massimo (MAX),
  - Media (AVG),
  - Somma (SUM)

## Espressioni aritmetica e valori NULL

---

- Se un argomento è NULL, lo è anche l'intera espressione
  - Esempio: Prezzo \* 1,22 - Sconto
- 
- Nelle funzioni di gruppo, in generale i valori NULL sono ignorate.
  - Esempio: somma(prezzi)
- 
- Quindi, può accedere che:
    - $\text{Sum}(\text{Prezzo} + \text{IVA})$  è diversa da  $\text{Sum}(\text{Prezzo}) + \text{sum}(\text{IVA})$

Prezzo	IVA
1000	21
NULL	15
2000	30

## Operatori aggregati: COUNT

---

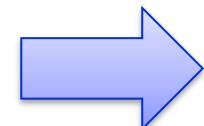
COUNT restituisce il numero di righe della tabella o il numero di valori di un particolare attributo

Esempio: Il numero di figli di Franco:

```
SELECT count(*) as NumFigliDiFranco  
      FROM Paternita  
     WHERE Padre = 'Franco'
```

l'operatore aggregato (count) viene applicato al risultato dell'interrogazione:

```
SELECT *  
      FROM Paternita  
     WHERE Padre = 'Franco'
```



## Paternità

Padre	Figlio
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo

```
SELECT *  
FROM Paternita  
WHERE Padre = 'Franco'
```

Padre	Figlio
Franco	Andrea
Franco	Aldo

count

```
SELECT count(*)  
as NumFigliDiFranco  
FROM Paternita  
WHERE Padre = 'Franco'
```

NumFigliDiFranco
2

## (\*), ALL e DISTINCT

---

Mediante le specifiche (\*), ALL e DISTINCT è possibile contare

(\*): tutte le righe selezionate;

ALL: tutti i valori non nulli delle righe selezionate;

DISTINCT: tutti i valori non nulli distinti delle righe selezionate.

Se la specifica viene omessa, il default è ALL.

# EsamiBD

---

Contare il numero di studenti iscritti al corso di BD e di  
Laboratorio di Basi di Dati

```
SELECT count(*) as "NumStud"  
FROM EsamiBD
```

NumStud
5

Contare il numero di esami di BD superati positivamente

```
SELECT count([ALL] BD) "ContaBD"  
FROM EsamiBD
```

ContaBD
3

Numero di voti distinti dati all'esame di LBD

```
SELECT count(distinct LBD) "ContDistLBD"  
FROM EsamiBD
```

ContDistLBD
2

## Max e Min

---

Le funzioni **MAX** e **MIN** calcolano rispettivamente il maggiore e il minore degli elementi di una colonna. L'argomento delle funzioni max e min può anche essere un'espressione aritmetica.

### Esempio

L'età della persona più anziana nella tabella persone

```
SELECT max(eta)  
FROM Persone
```

Persone		
NOME	ETA	REDDITO

Il più basso dei voti assegnati all'esame di BD

```
SELECT min(BD)  
FROM EsamiBD
```

Studente	BD	LBD

## Sum

---

La funzione **SUM** calcola la somma dei valori di una colonna.

Le specifiche **ALL** e **DISTINCT** permettono di sommare tutti i valori non nulli o tutti i valori distinti.

Il default in mancanza di specifiche è **ALL**.

**Esempio:**

Calcolare la somma degli stipendi mensili degli impiegati del settore Produzione.

**Impiegati**

Matricola	Cognome	Dipart	Stip
-----------	---------	--------	------

```
SELECT SUM (ALL stipendio)
FROM Impiegati
WHERE Dipart= 'Produzione'
```

## AVG

---

La funzione **AVG** calcola la media (average) dei valori **non nulli** di una colonna.

Le specifiche **ALL** e **DISTINCT** servono a calcolare la media fra tutti i valori o tra i valori distinti. Il default è **ALL**.

**Esempio:** Calcolare la media degli stipendi degli impiegati del dipartimento di Produzione e che hanno meno di 30 anni

Impiegati			
Matricola	Cognome	Dipart	Stip

```
SELECT AVG(stipendi)
FROM Impiegati
WHERE Dipart= 'Produzione' AND eta<30
```

## Operatori aggregati e valori nulli

---

```
SELECT AVG(reddito) AS Redditomedio  
FROM persone
```

Persone

Nome	Eta	Reddito
Andrea	27	30
Aldo	25	NULL
Maria	55	36
Anna	50	36

Redditomedio
34

I valori nulli non vengono considerati nella media

## Operatori aggregati e target list

---

Non è possibile utilizzare in una stessa select una proiezione su alcuni attributi della tabella considerata e operatori aggregati sulla stessa tabella.

- un'interrogazione **scorretta**:

```
SELECT nome, max(redito)  
FROM persone
```



Attenzione! Errore grave!  
Le funzioni di  
aggregazione non possono  
essere usate insieme ad  
attributi normali

- di chi sarebbe il nome? La target list deve essere omogenea.
- E' **corretta** invece la seguente:

```
SELECT min(eta), avg(redito)  
FROM persone
```

## ESEMPI: ordinamenti e funzioni di aggregazione

---

- Studenti ordinati per Nome

```
SELECT      *
FROM        Studenti
ORDER BY    Nome;
```

- Numero di elementi di Studenti

```
SELECT  count(*)
FROM    Studenti;
```

- Anno di nascita minimo, massimo e medio degli studenti:

```
SELECT  min(AnnoNascita), max(AnnoNascita), avg(AnnoNascita)
FROM    Studenti;
```

## Group by

---

A volte può essere richiesto di calcolare operatori aggregati non per l'intera tabella, ma raggruppando le righe i cui valori coincidono su un certo attributo.

Per esempio, vogliamo sapere la media degli stipendi degli impiegati per ogni dipartimento. In tal caso si può utilizzare la clausola **GROUP BY**.

```
SELECT Dipart, AVG(stipendio)  
FROM Impiegati  
GROUP BY Dipart
```

Dipart	AVG(stipendio)
Produzione	1330
Amministrazione	1505
Distribuzione	1810
Direzione	2500



## Semantica degli operatori di raggruppamento (1)

- La query è innanzitutto eseguita senza operatori aggregati e senza GROUP BY:

```
SELECT Dipart, stipendio  
FROM Impiegati
```

Quindi il risultato è diviso in sottoinsiemi aventi gli stessi valori per gli attributi indicati nel GROUP BY (Dipart nel nostro caso)

Quindi l'operatore aggregato è calcolato su ogni sottoinsieme:



# IL RAGGRUPPAMENTO

---

- Per ogni materia, trovare nome della materia e voto medio:
  - Per ogni materia:
    - Un attributo della materia
    - Una funzione aggregata sugli esami della materia
- Soluzione:

```
SELECT e.Materia, avg(e.Voto)
```

```
FROM Esami e
```

```
GROUP BY e.Materia
```



Nota che dato che  
ho raggruppato  
per materia posso  
nella target list  
aggiungere  
Materia

## Osservazione

---

- Quando si effettua un raggruppamento, questo deve essere effettuato su tutti gli elementi della target list che non sono operatori aggregati (ossia sull'insieme degli attributi puri).
- Questo ha senso perché nel risultato deve apparire una riga per ogni «gruppo»

Esempio:

SELECT Dipart, AVG(stipendio)

FROM Impiegati

GROUP BY Dipart

## Raggruppamenti e target list

- scorretta

```
SELECT padre, avg(f.reddito), p.reddito  
      FROM persone f JOIN paternita ON figlio = nome  
      JOIN persone p ON padre = p.nome  
GROUP BY padre
```

- corretta

```
SELECT padre, avg(f.reddito)  
      FROM persone f JOIN paternita ON figlio = nome JOIN  
            persone p ON padre = p.nome  
GROUP BY padre
```

## Persone

NOME	ETA	REDDITO
Andrea	27	21
Aldo	25	15
Maria	55	42
Anna	50	35
Filippo	26	29
Luigi	50	40
Franco	60	20
Olga	30	41
Sergio	85	35
Luisa	75	87

## Paternità

PADRE	FIGLIO
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo

# IL RAGGRUPPAMENTO

---

- Per ogni studente, nome e voto medio:

```
SELECT s.Nome, avg(e.Voto)  
FROM Studenti s, Esami e  
WHERE s.Matricola = e.Matricola  
GROUP BY s.Matricola, ...
```

- È necessario scrivere:
  - GROUP BY s.Matricola, **s.Nome**
  - **Gli attributi espressi non aggregati nella select (s.Nome) devono essere inclusi tra quelli citati nella GROUP BY (s.Matricola, s.Nome)**
  - Gli attributi aggregati (avg(e.Voto)) vanno scelti tra quelli non raggruppati



# Esempio - Parte I - Due tabelle

Clienti

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	IDCLIENTE	NUMBER (38, 0)	No	(null)	1	(null)
2	USERNAME	VARCHAR2 (20 BYTE)	No	(null)	2	(null)
3	PASSWORD	VARCHAR2 (10 BYTE)	No	(null)	3	(null)
4	NOME	VARCHAR2 (20 BYTE)	No	(null)	4	(null)
5	COGNOME	VARCHAR2 (20 BYTE)	No	(null)	5	(null)
6	DATANASCITA	DATE	No	(null)	6	(null)

idCliente è chiave primaria

Username è univoca

Cognome non può essere NULL

```
select idCliente, username, cognome, nome  
from clienti  
group by idCliente, username, cognome, nome
```

IDCLIENTE	USERNAME	COGNOME	NOME
1	192g.zavoli	Zavoli	Gasperino
2	42l.zavoli	Zavoli	Leano
3	1l.chiricozzi	Chiricozzi	Livio
4	2b.gubinelli	Gubinelli	Brillante
5	3a.borrelli	Borrelli	Assuntino
6	4g.malagigi	Malagigi	Gilma
7	12s.zavoli	Zavoli	Signorina

## Esempio - Parte II - Raggruppamento per cognome

Clienti

IDCLIENTE	USERNAME	COGNOME	NOME
1	192 g.zavoli	Zavoli	Gasperino
2	421.zavoli	Zavoli	Leano
3	11.chiricozzi	Chiricozzi	Livio
4	2b.gubinelli	Gubinelli	Brillante
5	3a.borrelli	Borrelli	Assuntino
6	4g.malagigi	Malagigi	Gilma
7	12s.zavoli	Zavoli	Signorina

Contiamo quante persone (cognomi e conteggio) hanno lo stesso cognome  
(stampare cognome e count(\*))

```
select clienti.Cognome, count(*)  
from clienti  
group by clienti.Cognome
```

COGNOME	COUNT(*)
1 Zavoli	3
2 Gubinelli	1
3 Malagigi	1
4 Chiricozzi	1
5 Borrelli	1

## Esempio - Parte III - Raggruppiamo per chiavi

idCliente è chiave primaria  
Username è univoco

IDCLIENTE	USERNAME	COGNOME	NOME	Clienti
1	192 g.zavoli	Zavoli	Gasperino	
2	42 l.zavoli	Zavoli	Leano	
3	11.chiricozzi	Chiricozzi	Livio	
4	2 b.gubinelli	Gubinelli	Brillante	
5	3 a.borrelli	Borrelli	Assuntino	
6	4 g.malagigi	Malagigi	Gilma	
7	12 s.zavoli	Zavoli	Signorina	

Contiamo quante persone hanno lo stesso idCliente (o username) e quanti hanno lo stesso username

```
select clienti.idCliente, count(*)  
from clienti  
group by clienti.idCliente
```

IDCLIENTE	COUNT(*)
1	42
2	1
3	2
4	12
5	4
6	3
7	192

```
select username, count(*)  
from clienti  
group by username
```

USERNAME	COUNT(*)
1 b.gubinelli	1
2 l.zavoli	1
3 s.zavoli	1
4 l.chiricozzi	1
5 g.zavoli	1
6 g.malagigi	1
7 a.borrelli	1

Posso fare un'unica query con idCliente e username nella target list?

## Esempio - Parte IV

Clienti

IDCLIENTE	USERNAME	COGNOME	NOME
1	192 g.zavoli	Zavoli	Gasperino
2	42 l.zavoli	Zavoli	Leano
3	1 l.chiricozzi	Chiricozzi	Livio
4	2 b.gubinelli	Gubinelli	Brillante
5	3 a.borrelli	Borrelli	Assuntino
6	4 g.malagigi	Malagigi	Gilma
7	12 s.zavoli	Zavoli	Signorina

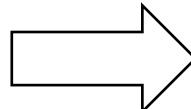
Attenzione:  
Errore grave

Contiamo quante persone hanno lo stesso idCliente, ma stampiamo anche il suo username

```
select idCliente, count(*), username  
from clienti  
group by idCliente
```



```
select idCliente, count(*), username  
from clienti  
group by idCliente, username
```



IDCLIENTE	COUNT(*)	USERNAME
1	1	l.l.chiricozzi
2	3	l.a.borrelli
3	4	l.g.malagigi
4	192	l.g.zavoli
5	42	l.l.zavoli
6	12	l.s.zavoli
7	2	l.b.gubinelli

ORA-00979: non è un'espressione GROUP BY  
00979. 00000 - "not a GROUP BY expression"  
\*Cause:  
\*Action:  
Errore alla riga: 1, colonna: 29

## Esempio - Parte V

Clienti

IDCLIENTE	USERNAME	COGNOME	NOME
1	192	Zavoli	Gasperino
2	42	Zavoli	Leano
3	11.chiricozzi	Chiricozzi	Livio
4	2b.gubinelli	Gubinelli	Brillante
5	3a.borrelli	Borrelli	Assuntino
6	4g.malagigi	Malagigi	Gilma
7	12s.zavoli	Zavoli	Signorina

```
select idCliente, count(*), username  
from clienti  
group by idCliente, username
```

IDCLIENTE	COUNT(*)	USERNAME
1	1	11.chiricozzi
2	3	1a.borrelli
3	4	1g.malagigi
4	192	1g.zavoli
5	42	1l.zavoli
6	12	1s.zavoli
7	2	1b.gubinelli

Che valori ottengo su count se sostituisco username con cognome?

```
select idCliente, count(*), cognome  
from clienti  
group by idCliente, cognome
```

IDCLIENTE	COUNT(*)	COGNOME
1	4	Malagigi
2	1	Chiricozzi
3	42	Zavoli
4	12	Zavoli
5	2	Gubinelli
6	192	Zavoli
7	3	Borrelli

## Esempio - Parte VI

Clienti

IDCLIENTE	COGNOME	NOME
192	Zavoli	Luigi
97	Grassi	Maria
114	Di Santo	Luigi
42	Zavoli	Luigi
5	Di Santo	Luigi
138	Grassi	Maria
12	Zavoli	Maria

```
select cognome, nome, count(*)  
from Clienti  
group by cognome, nome
```

COGNOME	NOME	COUNT(*)
1 Zavoli	Luigi	2
2 Di Santo	Luigi	2
3 Zavoli	Maria	1
4 Grassi	Maria	2

```
select nome, count(*)  
from Clienti  
group by nome
```

NOME	COUNT (*)
Maria	3
Luigi	4

```
select cognome, count(*)  
from Clienti  
group by cognome
```

COGNOME	COUNT(*)
1 Zavoli	3
2 Grassi	2
3 Di Santo	2

Concettualmente diversi:  
prima si raggruppa per  
cognome (risp. Nome) e la  
partizione ottenuta si  
partiziona ulteriormente  
per nome (risp. Cognome).  
Il risultato è uguale  
perché la partizione  
finale (cognome-nome) è  
uguale a quella (nome-  
cognome)

```
select nome, cognome, count(*)  
from Clienti  
group by nome, cognome
```

NOME	COGNOME	COUNT(*)
Maria	Grassi	2
Luigi	Zavoli	2
Luigi	Di Santo	2
Maria	Zavoli	1

## Condizioni sui gruppi, clausola HAVING

- Si possono applicare condizioni sul valore aggregato per ogni gruppo. Si può realizzare mediante la clausola HAVING.
- Esempio: I dipartimenti la cui media degli stipendi è maggiore di 1700 euro

```
Select dipart, AVG(stipendio)  
FROM Impiegati  
Group by Dipart  
HAVING AVG(stipendio)>1700
```

Dipart	AVG(stipendio)
Amministrazione	1505
Distribuzione	1810
Direzione	2500
Produzione	1330

Dipart	AVG(stipendio)
Distribuzione	1810
Direzione	2500

HAVING AVG(stipendio)>1700

## Where o Having

---

- In generale se la condizione coinvolge un attributo, si usa la clausola where, mentre se coinvolge un operatore aggregato si usa la clausola having.

EsamiBD (matricola, nome, cognome, città, voto, età)

- Le città per cui la media dei voti dei suoi studenti di meno di 21 anni è maggiore di 26

SELECT città, avg(voto)

FROM EsamiBD

WHERE età < 21

GROUP BY città

HAVING avg(voto) > 26

Attenzione! Having solo in presenza  
di un group by, e dopo di esso

## Sintassi, riassumiamo

---

SelectSQL ::=

```
select ListaAttributi O Espressioni  
from ListaTabelle  
[ where CondizioniSemplici ]  
[ group by ListaAttributiDiRaggruppamento ]  
[ having CondizioniAggregate ]  
[ order by ListaAttributiDiOrdinamento ]
```

# IL RAGGRUPPAMENTO

---

- SELECT ... FROM ... WHERE ... GROUP BY A<sub>1</sub>...,A<sub>n</sub> [HAVING condizione]
- Semantica:
  - Esegue le clausole FROM - WHERE
  - Partiziona la tabella risultante rispetto all'uguaglianza su tutti i campi A<sub>1</sub>...A<sub>n</sub> (**solo in questo caso, si assume NULL = NULL**)
  - Elimina i gruppi che non rispettano la clausola HAVING
  - Da ogni gruppo estrae una riga usando la clausola SELECT

## LA CLAUSOLA HAVING: IMPORTANTE

---

- Attenzione:
  - Se la SELECT contiene sia espressioni aggregate (MIN, COUNT...) che attributi non aggregati, allora **DEVE** essere presente la clausola GROUP BY
  - Le clausole HAVING e SELECT citano solo:
    - espressioni su attributi di raggruppamento;
    - funzioni di aggregazione applicate ad attributi non di raggruppamento.

## ESECUZIONE DI GROUP BY

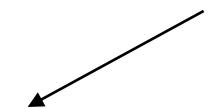
```
SELECT Matricola, count(*) AS NEsami, min(Voto), max(Voto), avg(Voto)
FROM Esami
GROUP BY Matricola
HAVING count(*) > 1;
```

Materia	Matricola	Voto	Docente
DA	1	20	10
LFC	2	30	20
MTI	1	30	30
LP	2	20	40



Materia	Matricola	Voto	Docente
DA	1	20	10
MTI	1	30	30
LFC	2	30	20
LP	2	20	40

Matricola	NEsami	min(Voto)	max(Voto)	Avg(Voto)
1	2	20	30	25
2	2	20	30	25



---

# SQL E ALGEBRA RELAZIONALE

## SQL -> ALGEBRA

SELECT DISTINCT  $S_A, S_{FA}$   
FROM R, S  
WHERE  $W_C$   
GROUP BY  $G_A$   
HAVING  $H_C$   
ORDER BY  $O_A$ ;

**ORDER BY**  $O_A$

**DISTINCT**

**SELECT**  $S_A, S_{FA}$

**HAVING**  $H_C$

**GROUP BY**  $G_A$

**WHERE**  $W_C$

**FROM** R, S

Comando **SELECT**

$\tau_{O_A}$

$\delta$

$\pi^b_{S_A \cup S_{FA}}$

$\sigma_{H_C}$

$G_A \gamma_{S_{FA} \cup H_C}$

$\sigma_{W_C}$

X

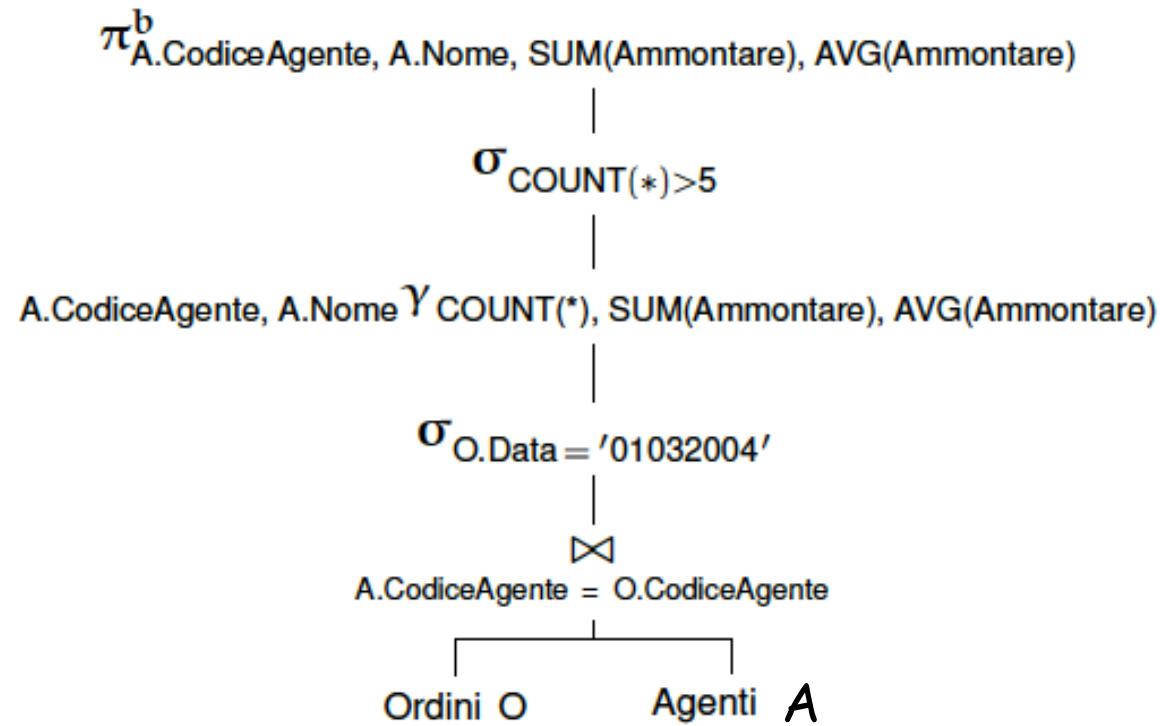


Albero logico

## Esempio

Trovare il codice e nome degli agenti con più di cinque ordini in data 1/3/2004 e, degli ordini fatti, il totale e la media dell'ammontare:

```
SELECT A.CodiceAgente, A.Nome,  
       SUM(Ammontare), AVG(Ammontare)  
  
FROM Ordini O, Agenti A  
  
WHERE A.CodiceAgente = O.CodiceAgente  
      AND  
      O.Data = '01032004'  
  
GROUP BY O.CodiceAgente, A.Nome  
  
HAVING COUNT() > 5;
```



## Riferimento alle colonne

---

- Spesso nel riferimento alle colonne selezionate nel join è necessario specificare da quale delle tabelle la colonna è stata estratta, al fine di evitare ambiguità.
- La sintassi usata è

NomeTabella.NomeColonna

## Ridenominazioni

---

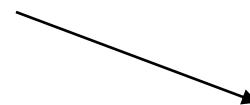
Di solito in una select che definisce una join possono essere necessarie ridenominazioni

- nel prodotto cartesiano (ossia ridenominare le tabelle coinvolte)
- nella target list (ossia ridenominare gli attributi)

```
SELECT X.A1 AS B1, Y.A2 AS B2 ...
FROM Tab1 X, Tab2 Y, Tab1 Z
WHERE X.A2 = Y.A3 AND ...
```



Ridenominazioni  
di colonne



Ridenominazione  
di tabelle

---

# JOIN

## Cross Join

---

- Il cross-join implementa il prodotto cartesiano.
- Si realizza semplicemente mediante una select che utilizza le due (o più) tabelle coinvolte, senza specificare nessuna condizione di join.

Select categorie.\* , Fabbriche.\*  
From Categorie, Fabbriche

## Join fra due tavole

---

- Se due tavole del database contengono dei dati in comune, possono essere correlate mediante un'operazione di **JOIN**.
- Le colonne delle due tavole che creano la correlazione rappresentano la stessa entità, ossia i loro valori appartengono allo **stesso dominio**.
- In genere le colonne delle due tavole considerate sono legate da un vincolo di chiave esterna (ma non è obbligatorio)

## Join

---

- IL **join** (o **equi-join**) fra due tavelle è una terza tabella le cui righe sono tutte e sole quelle ottenute dal prodotto cartesiano delle righe delle due tavelle di partenza i cui valori delle colonne espresse dalla condizione di join sono uguali.
- In SQL il join viene realizzato mediante una particolare forma del **SELECT**, in cui
- Nella clausola **FROM** vengono indicate le due tavelle coinvolte
- Nella clausola **WHERE** viene espresso il collegamento fra le due tavelle, mediante la condizione di join

## Join, sintassi

---

Siano Tab1(A1,A2) Tab2(A3,A4) due relazioni

SELECT Tab1.A1, Tab2.A4

FROM Tab1, Tab2

WHERE Tab1.A2 = Tab2.A3

Tab1

A1	A2
A	B
C	D
E	F
G	H

Tab2

A3	A4
F	I
D	L
D	M
B	N

Tab1  $\bowtie_{\text{Tab1.A2}=\text{Tab2.A3}}$  Tab2

A1	A4
A	N
C	L
C	M
E	I

## Join e algebra relazionale

---

Tab1(A1,A2) Tab2(A3,A4)

SELECT Tab1.A1, Tab2.A4

FROM Tab1, Tab2

WHERE Tab1.A2 = Tab2.A3

Traduce l'espressione dell'algebra relazionale

$\pi_{A1, A4} (\sigma_{A2 = A3} (Tab1 \bowtie Tab2))$

- Quindi il join consiste di:
  - Un prodotto cartesiano (FROM)
  - Una selezione (WHERE)
  - Una proiezione (SELECT)

## Esempio

Supponiamo che nel registro automobilistico siano presenti le seguenti tabelle:

### Categorie

<u>Cod_cat</u>	Nome_cat
----------------	----------

Veicoli

Targa	Cod_mod	Categoria*	Cilindrata	Cod_comb	cav.Fis	Velocita	Posti	Imm
-------	---------	------------	------------	----------	---------	----------	-------	-----

Vogliamo selezionare per ciascun veicolo la descrizione della relativa categoria. In questo caso devono essere coinvolte le due tabelle. Le due tabelle sono legate da un vincolo di chiave esterna tra gli attributi *Cod\_cat* (in Categorie) e *Categoria* (in Veicoli)

```
SELECT targa, Veicoli.Categoria, nome_cat  
FROM Categorie, Veicoli  
WHERE Veicoli.Categoria = Categorie.Cod_cat  
[Categoria=cod_cat]
```

Targa	Categoria	Nome_cat
-------	-----------	----------



# Veicoli

Targa	Categoria
A24353Q	01
F63457T	03
D2343GH	01
T94756U	02
W34985U	02
L23843K	01

```
SELECT targa, Veicoli.categoria, nome_cat
FROM Categorie, Veicoli
WHERE Veicoli.Categoria = Categorie.cod_cat
```

## Categorie

Veicoli.Categoria= categorie.cod\_cat

Cod_cat	Nome_cat
01	Autovettura
02	Rimorchio
03	Motociclo
04	Furgone

Targa	Categoria	Nome_cat
A24353Q	01	Autovettura
F63457T	03	Motociclo
D2343GH	01	Autovettura
T94756U	02	Rimorchio
W34985U	02	Rimorchio
L23843K	01	Autovettura

## Join, Esempio

---

```
SELECT targa, categorie.cod_cat, nome_cat → Scelta colonne  
FROM Categorie, Veicoli → Tabelle selezionate  
WHERE Veicoli.Categoria = Categorie.cod_cat → Condizione di join
```

La condizione di join può essere presente assieme ad altre condizioni mediante il connettore logico AND. Per esempio

```
SELECT targa, Veicoli.categoria, nome_cat  
FROM Categorie, Veicoli  
WHERE Veicoli.categoria = Categorie.cod_cat  
AND Cilindrata > 1600
```

## Join, Esempio

---

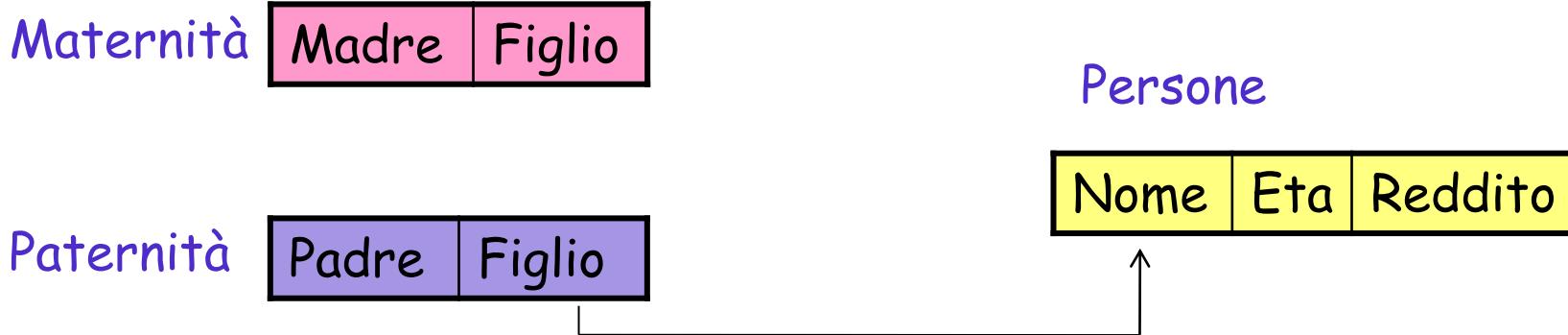
Se si vogliono selezionare tutte le colonne delle due tabelle si può sempre usare la notazione nome\_tabella.\*

```
SELECT Categorie.*, Veicoli.*  
FROM Categorie, Veicoli  
WHERE Veicoli.categoria = Categorie.cod_cat
```

In tal caso figureranno entrambi i campi Veicoli.Categoria e Categorie.cod\_cat (denominati rispettivamente Categoria e cod\_cat)

## Join, Esempio 2

- Elencare i padri di persone che guadagnano più di 2000 euro al mese (reddito/12)


$$\pi_{\text{Padre}}(\text{paternita} \bowtie_{\text{Figlio}=\text{Nome}} (\sigma_{\text{Reddito}/12 > 2000} (\text{persone}))$$

```
SELECT distinct padre
FROM persone, paternita
WHERE figlio = nome AND
      reddito/12 > 2000
```

## Inner-Join

---

L' **INNER JOIN** è un' operazione di join in cui la condizione non sia necessariamente una condizione di uguaglianza.

### Modelli

Cod_mod	Nome_mod	Cod_Fab	Num_vers	Cil_media
---------	----------	---------	----------	-----------

### Veicoli

Targa	Cod_mod*	Cod_cat	Cilindrata	Cod_comb.	cav.Fisc	Velocita	Posti	Imm
-------	----------	---------	------------	-----------	----------	----------	-------	-----

Tutti i Veicoli la cui la cilindrata è minore della cilindrata media del loro modello

```
SELECT Veicoli.*  
FROM Modelli, Veicoli  
WHERE Veicoli.cod_mod=Modelli.cod_mod  
and Cil_media> cilindrata
```

## Inner-Join

---

L'INNER JOIN è un'operazione di join in cui la condizione non sia necessariamente una condizione di uguaglianza.

Esempio:

Supponendo di avere una tabella impiegati e una reparti, trovare il nome dei reparti in cui non lavora Mario Rossi.

```
select rep.nome  
from impiegati as imp join reparti as rep on  
    imp.IdReparto <> rep.IdReparto  
where imp.cognome = 'Rossi' and imp.nome = 'Mario';
```

## Self-join

---

- Un caso molto particolare di Join è quello che mette in relazione una tabella con se stessa. Questo si può ottenere ridenominando due volte la tabella con due nomi diversi, trattando le due copie come se si trattasse di due tabelle diverse. In questo caso si parla di **SELF JOIN**

```
SELECT X.A1, Y.A4  
FROM Tab1 X, Tab2 Y, Tab1 Z  
WHERE X.A2 = Y.A3 AND X.A2 = Z.A1
```

## Self Join, Esempio

---

Trovare tutte le coppie di veicoli che sono dello stesso modello.

Veicoli

Targa	Cod_mod	Cod_cat	Cilindrata	Cod_comb.	cav.Fisc	Velocita	Posti	Imm
-------	---------	---------	------------	-----------	----------	----------	-------	-----

```
Select V1.Targa, V2.Targa  
From Veicoli V1, Veicoli V2  
Where V1.cod_mod=V2.cod_mod  
and V1.Targa>V2.Targa
```



Condizione di  
join

Per evitare che nel risultato ci siano coppie contenenti due volte la stessa macchina e le coppie che si ottengono scambiando l'ordine di coppie già esistenti

## Esempio Self-join

Trovare i colleghi di Mario Rossi (cioè quelli che lavorano nello stesso reparto)

```
Select imp1.nome, imp1.cognome , Imp1.Reparto,Imp2.Nome, Imp2.Cognome, Imp2.Reparto  
from impiegati imp1 join impiegati imp2  
on (imp1.idreparto = imp2.idreparto)
```

```
Where imp2.nome='Mario' and imp2.cognome = 'Rossi'  
and imp1.matricola <> imp2.matricola;
```

Impiegati imp1

Nome	Cognome	Reparto
Mario	Rossi	Informatica
Luigi	Bianchi	Agraria
Mario	Rossi	Agraria
Marco	Verdi	Informatica
Luca	Viola	Agraria

Impiegati imp2

Nome	Cognome	Reparto
Mario	Rossi	Informatica
Luigi	Bianchi	Agraria
Mario	Rossi	Agraria
Marco	Verdi	Informatica
Luca	Viola	Agraria

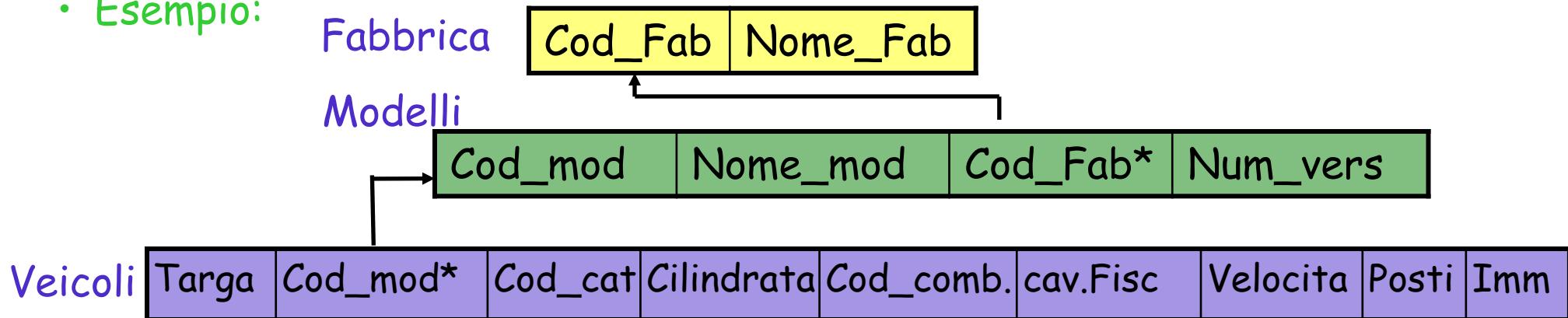
Imp1 join Imp2

Imp1.Nome	Imp1.Cognome	Imp1.Reparto	Imp2.Nome	Imp2.Cognome	Imp2.Reparto

## Join su più tabelle

---

- Talvolta un'interrogazione può coinvolgere più di due tabelle.
- Esempio:

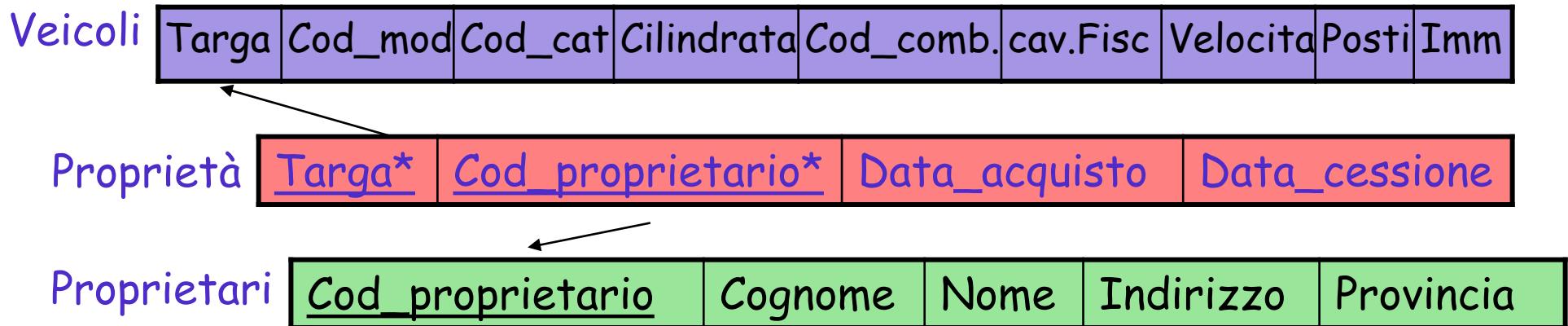


Nome della fabbrica del modello dell'automobile con targa W34534R

```
Select Targa, Nome_mod, nome_fab  
FROM Modelli M, Veicoli V, Fabbrica F  
Where M.cod_mod=V.cod_mod and M.cod_fab=F.cod_fab  
and V.targa= 'W34534R'
```

## Esempio

---



Cognome e nome del proprietario e cilindrata della macchina la cui targa è W34534R.

```
Select nome, cognome, cilindrata  
FROM Proprietari P, Proprietà PR, Veicoli V  
Where V.targa=PR.targa  
    and PR. Cod_Proprietario=P.cod_proprietario  
    and targa = 'W34534R'
```

## Join usando la clausola 'JOIN'

---

- Oltre alla forma vista, nei DBMS più moderni, per effettuare il join di due tabella è possibile utilizzare una forma più esplicita (standard ANSI).
- La sintassi è la seguente

```
SELECT Attributi  
FROM Tab1 JOIN Tab2  
ON CondizioneDiJoin
```

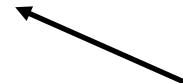
## Esempio

---

### Categorie

<u>Cod_cat</u>	Nome_cat
----------------	----------

Veicoli



Targa	Cod_mod	Categoria*	Cilindrata	Cod_comb.	cav.Fisc	Velocita	Posti	Imm
-------	---------	------------	------------	-----------	----------	----------	-------	-----

Selezionare per ciascun veicolo la descrizione della relativa categoria utilizzando l'operatore JOIN.

```
SELECT targa, Veicoli.Categoria, nome_cat  
FROM Categorie JOIN Veicoli  
ON Veicoli.Categoria = Categorie.Cod_cat
```

Targa	Categoria	Nome_cat
-------	-----------	----------

## Equi-Join e Natural Join

---

- Se dobbiamo operare una **equi-join**, ossia un join la cui condizione sia una condizione di uguaglianza, e che sia anche un **Natural Join**, ossia un join creato su tutte le colonne che hanno il medesimo nome in entrambe le tavelle, possiamo utilizzare la seguente sintassi

```
SELECT listaAttributi  
FROM Tab1 NATURAL JOIN Tab2
```

## Esempio

---

DEPUTATI (Codice, Cognome, **Nome**, CodCommissione\*, Provincia\*, Collegio\*)

COLLEGI (Provincia\*, Numero\*, **Nome**)

PROVINCE (Sigla, **Nome**, Regione\*)

REGIONI (Codice, **Nome**)

COMMISSIONI (CodCommissione, **Nome**, Presidente)

Cosa succede se facciamo il natural join fra Deputati e Commissioni?

SELECT \* FROM DEPUTATI NATURAL JOIN COMMISSIONI

È equivalente a

SELECT \* FROM DEPUTATI, COMMISSIONI where Deputati.CodCommissione =  
Commissioni.CodCommissione

???

## Using

---

- Può succedere comunque che nelle tabelle coinvolte ci siano più attributi con lo stesso nome, e col Natural Join tutte queste coppie di attributi presi dalla due tabelle vengono identificate.
- Se invece vogliamo operare una join in cui la condizione riguarda solo una o alcune di queste coppie, si usa la clausola **USING** seguita dall'elenco degli attributi coinvolti nella condizione

```
SELECT lista attributi  
FROM Tab1 JOIN Tab2  
USING (attr1,attr2,...)
```

## Esempio

---

Modelli

Cod_mod	Nome_mod	Cod_Fab	Num_vers	Cil_media
---------	----------	---------	----------	-----------

Veicoli

Targa	Cod_mod*	Cod_cat	Cilindrata	Cod_comb.	cav.Fisc	Velocita	Posti	Imm
-------	----------	---------	------------	-----------	----------	----------	-------	-----

Il nome del modello del veicolo di targa ZX2345

```
SELECT Nome_mod  
FROM Modelli JOIN Veicoli  
USING (Cod_mod)  
WHERE targa="ZX2345"
```

## Outer Join

---

- Quando vengono correlate mediante una join delle tabelle con colonne contenenti dati in comune, è possibile che un valore sia presente in una delle colonne e non nell'altra.
- Effettuando un equi-join la riga corrispondente a tale valore viene scartato.
- In alcuni casi invece può essere necessario mantenere questi valori. Per fare questo si deve effettuare un **outer join**.

Select lista\_attributi

From Tab1 **LEFT [OUTER] JOIN** Tab2

Select lista\_attributi

From Tab1 **RIGHT [OUTER] JOIN** Tab2

Select lista\_attributi

From Tab1 **FULL [OUTER] JOIN** Tab2

# Outer join, Esempio

---

Categorie

<u>Cod_cat</u>	Nome_cat

Veicoli

Targa	Cod_mod*	Cod_cat	Cilindrata	Cod_comb.	cav.Fisc	Velocita	Posti	Imm

Ottenerne i codici e nomi delle categorie di macchine della tabella veicoli e anche quelli per cui non esiste nessun veicolo.

```
Select Categorie.* , Veicoli.*  
From Categorie LEFT [OUTER] JOIN Veicoli  
    ON Categorie.cod_cat=Veicoli.Cod_cat
```

```
Select Categorie.* , Veicoli.*  
From Veicoli RIGHT [OUTER] JOIN Categorie  
    ON Categorie.cod_cat=Veicoli.Cod_cat
```

## Join esterno: "outer join"

- Per ogni persona, elencare il padre e, se nota, la madre.

SELECT paternita.figlio, padre, madre

FROM paternita **LEFT JOIN** maternita

ON paternita.figlio = maternita.figlio

## Maternità

MADRE	FIGLIO
Luisa	Maria
Luisa	Luigi
Anna	Olga
Anna	Filippo
Maria	Andrea
Maria	Aldo

## Paternità

SELECT paternita.figlio, padre, madre

FROM paternita **LEFT OUTER JOIN** maternita

ON paternita.figlio = maternita.figlio

PADRE	FIGLIO
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo

## Join Esterno/Outer Join

---

L'operatore di OUTER JOIN può essere applicato usando la seguente sintassi:

- {LEFT | RIGHT | FULL} [OUTER] JOIN + ON <predicato>
- { LEFT | RIGHT | FULL} [OUTER] JOIN + USING <colonne>
- dove "outer" è opzionale

## Esercizi

---

Libro(CodiceLibro, ISBN, Titolo, NomeAutore, CognomeAutore, Pagine, anno)

1. Il numero di libri scritti da Andrea Camilleri prima del 2010
2. Massimo numero di pagine fra i libri presenti nella tabella
3. Il massimo numero di pagine fra quelle dei libri scritti da Andrea Camilleri
4. Per ogni autore, il massimo numero di pagine dei suoi libri
5. I nomi e cognomi degli autori che hanno scritto più di 8 libri
6. Ordinare i libri scritti da Camilleri in ordine di anno decrescente e quelli dello stesso anno in ordine lessicografico per titolo
7. Per ogni anno, il numero di libri pubblicati
8. Per ogni autore il numero dei libri scritti ogni anno da quell'autore

---

# SUBQUERY

# Subquery

---

Una **subquery** è un comando Select, racchiuso tra parentesi tonde, inserito all'interno di un comando SQL, per esempio un'altra Select.

Le subquery possono essere utilizzate nei seguenti casi:

- In espressioni di confronto
- In espressioni di confronto quantificato
- In espressioni IN
- In espressioni EXISTS
- Nel calcolo di espressioni

## Tipi di subquery

---

Targa	Cod_mod	Categoria	Cilindrata	Cod_comb.	cav.Fisc	Velocita	Posti	Imm
-------	---------	-----------	------------	-----------	----------	----------	-------	-----

- Tre tipologie: **scalare**, **di colonna**, **di tabella**
- **Subquery Scalare**: è un comando Select che restituisce un solo valore.
- **SELECT Max(Cilindrata) FROM Veicoli**
- **SELECT Cod\_Categoria FROM Veicoli Where Targa="123456"**
- **Subquery di Colonna**: è un comando Select che restituisce una colonna
- **SELECT cod\_categoria FROM Veicoli**
- **Subquery di Tabella**: è un comando Select che restituisce una tabella
- **SELECT Targa, Cod\_mod, Posti FROM Veicoli**

## Subquery in espressioni di confronto

Categorie

	Cod_cat	Nome_cat
--	---------	----------

Veicoli

Targa	Cod_mod	Categoria	Cilindrata	Cod_comb.	cav.Fisc	Velocita	Posti	Imm
-------	---------	-----------	------------	-----------	----------	----------	-------	-----

Vogliamo trovare tutti i veicoli della categoria "Autovettura"

```
Select Veicoli.*  
From Categorie, Veicoli  
Where Categoria=Cod_cat  
and Nome_Cat = 'Autovettura'
```

Mediane Join

```
Select * From Veicoli  
Where Categoria = (Select cod_cat  
From Categorie  
Where nome_cat='Autovettura')
```

Mediane Subquery

Questa subquery  
restituisce un tipo  
scalare, ossia  
restituisce un singolo  
valore

## Uso di subquery con funzioni di gruppo

---

Ricordiamo che nelle select semplici non è possibile utilizzare contemporaneamente funzioni di gruppo e funzioni su singole righe. Questo viene reso possibile mediante l'uso delle subquery.

Esempio: tutti i veicoli di cilindrata superiore alla media delle cilindrate.

```
Select *  
From Veicoli  
Where Cilindrata > (select AVG(Cilindrata)  
From Veicoli)
```

Esempio: La targa dei veicoli di cilindrata massima.

```
Select targa  
From Veicoli  
Where Cilindrata = (Select max(cilindrata)  
From Veicoli)
```

Veicoli

Targa	Cod_mod	Categoria	Cilindrata	Cod_comb.	cav.Fisc	Velocita	Posti	Imm
-------	---------	-----------	------------	-----------	----------	----------	-------	-----

## Subquery + Join

---

### Veicoli

Targa	Cod_mod*	Categoria	Cilindrata	Cod_combust.	cav.Fis	Velocita	Posti	Imm
-------	----------	-----------	------------	--------------	---------	----------	-------	-----

### Modelli

Cod_Mod	Nome_Mod	Cod_Fab	Num_versioni
---------	----------	---------	--------------

Selezionare i modelli che presentano più versioni del numero minimo di versioni dei veicoli a benzina (Cod\_combustibile='01').

```
Select * From Modelli  
Where num_versioni >  
      (Select min (num_versioni)  
       From Veicoli, Modelli  
       Where Veicoli.cod_mod=Modelli.cod_mod  
             And cod_combust='01')
```

# Subquery Annidate

Veicolic

Targa	Cod_mod*	Categoria	Cilindrata	Cod_comb.	cav.Fisc	Velocita	Posti	Imm
-------	----------	-----------	------------	-----------	----------	----------	-------	-----

Modelli

Cod_Mod	Nome_Mod	Cod_Fab*	Cilind_Media
---------	----------	----------	--------------

Fabbrica

Cod_Fab	Nome_Fab
---------	----------

Selezionare targa e velocità dei veicoli che appartengono a Modelli prodotti nella Fabbrica FIAT

Select targa, velocita  
From Veicolic

Where cod\_mod in (select cod\_mod From Modelli

Where cod\_fab = (select cod\_fab From Fabbrica  
Where Nome\_Fab='FIAT' )

Questo subquery restituisce un tipo scalare, ossia restituisce un singolo valore

## Esempio

---

### Veicoli

Targa	Cod_mod*	Categoria	Cilindrata	Cod_comb.	cav.Fisc	Velocita	Posti	Imm
-------	----------	-----------	------------	-----------	----------	----------	-------	-----

### Modelli

Cod_Mod	Nome_Mod	Cod_Fab	Num_versioni
---------	----------	---------	--------------

Selezionare i modelli che presentano più versioni del numero minimo di versioni dei veicoli a benzina (`Cod_combustibile='01'`). Già realizzata mediante subselect + join

Select \* From Modelli

Where num\_versioni > (Select min (num\_versioni)

From Modelli

Where cod\_mod IN (select cod\_mod

From veicoli

Where cod\_combust='01' )

## Esempio di SELECT nidificate

nome e reddito del padre di Franco

Mediate join:

SELECT Nome, Reddito

FROM Persone, Paternita

WHERE Nome = Padre AND Figlio = 'Franco'

Mediate subquery:

SELECT Nome, Reddito  
FROM Persone

WHERE Nome = (SELECT Padre  
FROM Paternita

WHERE Figlio = 'Franco')

## Paternità

PADRE	FIGLIO
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo

## Persone

NOME	ETA	REDDITO
Andrea	27	21
Aldo	25	15
Maria	55	42
Anna	50	35
Filippo	26	29
Luigi	50	40
Franco	60	20
Olga	30	41
Sergio	85	35
Luisa	75	87

Si può usare = poiché la query interna restituisce un solo valore

## Esempio

Dobbiamo estrapolare i redditi dei padri

Dobbiamo estrarre i redditi dei figli

Nome e reddito dei padri di persone che guadagnano più di 20

SELECT distinct P.Nome, P.Reddito

FROM Persone P, Paternita, Persone F

WHERE P.Nome = Padre AND Figlio = F.Nome

AND F.Reddito > 20

Mediante Join

Persone P

NOME	ETA	REDDITO
Andrea	27	21
Aldo	25	15
Maria	55	42
Anna	50	35
Filippo	26	29
Luigi	50	40
Franco	60	20
Olga	30	41
Sergio	85	35
Luisa	75	87

Paternità

join

PADRE	FIGLIO
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo

Persone F

NOME	ETA	REDDITO
Andrea	27	21
Aldo	25	15
Maria	55	42
Anna	50	35
Filippo	26	29
Luigi	50	40
Franco	60	20
Olga	30	41
Sergio	85	35
Luisa	75	87

## Esempio =ANY

Nome e reddito dei padri di persone che guadagnano più di 20

SELECT distinct P.Nome, P.Reddito

Mediante Join

FROM Persone P, Paternita, Persone F

WHERE P.Nome = Padre AND Figlio = F.Nome

AND F.Reddito > 20

SELECT Nome, Reddito

FROM Persone

WHERE Nome IN (SELECT Padre

FROM Paternita

WHERE Figlio = ANY (SELECT Nome

FROM Persone

WHERE Reddito > 20))

Paternità

PADRE	FIGLIO
Sergio	Franco
Luigi	Olga
Luigi	Filippo
Franco	Andrea
Franco	Aldo

Persone

NOME	ETA	REDDITO
Andrea	27	21
Aldo	25	15
Maria	55	42
Anna	50	35
Filippo	26	29
Luigi	50	40
Franco	60	20
Olga	30	41
Sergio	85	35
Luisa	75	87

## Interrogazioni nidificate, commenti

---

- La forma nidificata è meno dichiarativa, ma talvolta più leggibile (richiede meno variabili)
- La forma piana e quella nidificata possono essere combinate
- Le subquery non possono contenere operatori insiemistici
- Il problema si supera facilmente con intersezione e differenza (per cui esiste una forma alternativa) ma non sempre per l'unione

## Esempio

---

- Nome e reddito dei padri di persone che guadagnano più di 20, con indicazione del reddito del figlio

```
SELECT distinct P.Nome, P.Reddito, F.Reddito
```

```
FROM Persone P, Paternita, Persone F
```

```
WHERE P.Nome = Padre AND Figlio = F.Nome
```

```
AND F.Reddito > 20
```



Mediante Join

```
SELECT Nome, Reddito, ????
```

```
FROM Persone
```

```
WHERE Nome in (SELECT Padre
```

```
FROM Paternita
```

```
WHERE Figlio = any (SELECT Nome
```

```
FROM Persone
```

```
WHERE Reddito > 20) )
```

Mediante Subquery (?)



## Interrogazioni nidificate, commenti, 3

---

- regole di visibilità simili a quelle delle procedure nei linguaggi di programmazione:
  - non è possibile fare riferimenti a variabili definite in blocchi più interni
  - se un nome di variabile è omesso, si assume riferimento alla variabile più "vicina"
- in un blocco si può fare riferimento a variabili definite in blocchi più esterni

## Regole di Visibilità

---

- La seguente query è scorretta:

```
SELECT *
FROM Impiegato
WHERE Dipart in (SELECT Nome
                  FROM Dipartimento D1
                  WHERE Nome = 'Produzione') OR
      Dipart in (SELECT Nome
                  FROM Dipartimento D2
                  WHERE D2.Citta = D1.Citta)
```

- D1 non è visibile nella seconda query nidificata in quanto le due subquery sono allo stesso livello

## Confronto su più attributi

---

- Il confronto con il risultato di una query nidificata può essere basato su più attributi
- Esempio: Trovare tutti gli studenti che hanno un omonimo:

```
SELECT *  
FROM Studenti S  
WHERE (Nome, Cognome) IN (SELECT Nome, Cognome  
                           FROM Studenti S2  
                           WHERE S2.Matricola <> S.Matricola)°
```

Condizione  
importante!

## Commenti finali sulle query nidificate

---

- Query nidificate possono essere "meno dichiarative" in un certo senso ma spesso sono più facilmente interpretabili, in quanto si possono suddividere in blocchi più semplici da interpretare
- L'utilizzo di variabili deve rispettare le regole di visibilità cioè, una variabile può essere usata solo all'interno dello stesso blocco e in un blocco più interno
- Comunque, query nidificate complesse possono essere di difficile comprensione, soprattutto quando si usano molte variabili comuni tra blocchi diversi

---

# QUANTIFICAZIONE

# LA QUANTIFICAZIONE

---

- Tutte le interrogazioni su di una associazione multivalue vanno quantificate



- Non: gli studenti che hanno preso 30 (ambiguo!)  
ma:
  - Gli studenti che hanno preso **sempre** (o solo) 30: **universale**
  - Gli studenti che hanno preso qualche (**almeno un**) 30: **esistenziale**
  - Gli studenti che **non** hanno preso qualche 30 (senza nessun 30): **universale**
  - Gli studenti che **non** hanno preso **sempre** 30: **esistenziale**

# LA QUANTIFICAZIONE

---

- Universale negata = esistenziale:
  - Non tutti i voti sono  $\leq 24$  = Almeno un voto  $> 24$  (esistenziale)
- Esistenziale negata = universale:
  - Non esiste voto diverso da 30 = Tutti i voti sono uguali a 30 (universale)

## Any, All ed Exists

---

- le condizioni in SQL permettono anche il confronto fra un attributo e il risultato di una subquery che restituisce una colonna o una tabella
  - Operatore **Scalare (ANY | ALL) SelectQuery**
    - **ANY**: il predicato è vero se almeno uno dei valori restituiti da Query soddisfano la condizione
    - **ALL**: il predicato è vero se tutti i valori restituiti dalla Query soddisfano la condizione
  - quantificatore **esistenziale**
    - **EXISTS SelectQuery**
    - Il predicato è vero se la SelectQuery restituisce almeno una tupla

## Subquery di confronto quantificato, regole

---

- La subquery deve essere una **subquery di colonna**
- La subquery deve essere inserita **DOPO** l'operatore di confronto **quantificato**
- Non è ammesso il confronto fra due subquery
- Nella subquery non è possibile utilizzare le clausole having e group by

## Subquery in espressioni EXISTS

---

Mediante **EXISTS** (SELECT \* ...) è possibile verificare se il risultato di una subquery restituisce almeno una tupla

IMPIEGATI(Matricola, Cognome, Nome, Mansione, IdReparto\*,

StipendioAnnuale, PremioProduzione, DataAssunzione)

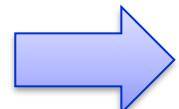
REPARTI(IdReparto, Nomereparto, Indirizzo, Città)

SELECT IdReparto FROM Reparti WHERE EXISTS ( . . . )

SELECT \* FROM Impiegati WHERE Mansione = 'Programmatore');

Facendo uso di **NOT EXISTS** il predicato è vero se la subquery non restituisce alcuna Tupla.

Quale è il  
risultato?  
Ha senso?



## Subquery in espressioni EXISTS

---

IMPIEGATI(Matricola, Cognome, Nome, Mansione, IdReparto\*, StipendioAnnuale, PremioProduzione,  
DataAssunzione)

REPARTI(IdReparto, NomeReparto, Indirizzo, Città)

**SELECT IdReparto FROM Reparti WHERE EXISTS (**

**SELECT \* FROM Impiegati WHERE Mansione = 'Programmatore');**

- Occorre un meccanismo per rendere più flessibile la clausola exists, rendendo le condizioni della subquery dipendenti dalla tupla della query principale (query esterna):
  - Si introduce un **legame fra la query e la subquery** (query correlate)
  - Si definisce una variabile (un alias) nella query esterna, che si utilizza nella subquery

## Subquery in espressioni EXISTS (query correlate)

- Tramite il predicato **EXISTS** è possibile effettuare il controllo sull'esistenza di righe che soddisfano specifiche condizioni. In questo caso la subquery è una subquery di tabella.

Categorie	<u>Cod_cat</u>	Nome_cat
-----------	----------------	----------

Veicoli	Targa	Cod_mod	Categoria*	Cilindrata	Cod_comb.	cav.Fisc	Velocita	Posti	Imm
---------	-------	---------	------------	------------	-----------	----------	----------	-------	-----

Esempio: trovare i nomi di tutte le categorie per cui è presente almeno un veicolo

```
Select Nome_cat  
From Categorie  
Where EXISTS (Select *  
               From Veicoli  
               Where Categorie.cod_cat= Veicoli.categoria)
```



## Clausola NOT EXISTS

---

- Per verificare l'assenza di righe che rispondono a una determinata condizione è definita la forma negativa **NOT EXISTS**

Categorie	<u>Cod_cat</u>	Nome_cat
-----------	----------------	----------

Veicoli	Targa	Cod_mod	Categoria*	Cilindrata	Cod_comb.	cav.Fisc	Velocita	Posti	Imm
---------	-------	---------	------------	------------	-----------	----------	----------	-------	-----

Esempio:

tutte le categorie per cui non è presente nessun veicolo

Select Cat\_nome

From Categorie

Where NOT EXISTS (Select \*

From Veicoli

where Categorie.Cod\_cat= Veicoli. categoria)

## RICORDIAMO LA SINTASSI DEL WHERE

---

- Combinazione booleana di predicati tra cui:
  - Expr Comp Expr
  - Expr Comp ( Sottoselect che torna un valore)
  - [NOT] EXISTS (Sottoselect)
- Inoltre:
  - Espr Comp (ANY | ALL) (Sottoselect)
  - Expr [NOT] IN ( Sottoselect) (oppure IN (v1,..,vn))
- Comp: <, =, >, <>, <=, >=

## LA QUANTIFICAZIONE ESISTENZIALE: EXISTS

---

- Gli studenti con almeno un voto sopra 27
- In SQL:

```
SELECT s.Nome  
FROM Studenti s  
WHERE EXISTS (SELECT *  
               FROM Esami e  
               WHERE e.Matricola = s.Matricola AND e.Voto > 27)
```

# LA QUANTIFICAZIONE ESISTENZIALE: GIUNZIONE

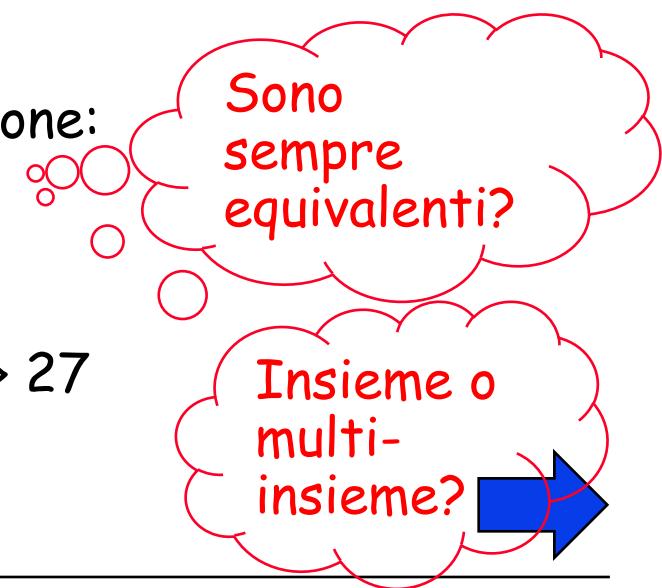
---

- Gli studenti con almeno un voto sopra 27, tramite EXISTS:

```
SELECT s.Nome  
FROM Studenti s  
WHERE EXISTS (SELECT *  
               FROM Esami e  
               WHERE e.Matricola = s.Matricola AND e.Voto > 27)
```

- Stessa quantificazione esistenziale, tramite giunzione:

```
SELECT s.Nome  
FROM Studenti s, Esami e  
WHERE e.Matricola = s.Matricola AND e.Voto > 27
```



# LA QUANTIFICAZIONE ESISTENZIALE: GIUNZIONE

## Studenti

MATRIC	NOME	COGNOME	DATAISCRIT
282320	Gianna	Neri	04-MAG-20
369871	Mario	Rossi	04-MAG-20
515140	Mario	Verdi	04-MAG-20
090456	Mario	Bianchi	04-MAG-20
579555	Luigi	Rossi	04-MAG-20
018701	Luca	Bianchi	04-MAG-20

## Esami

CODICE	MATRIC	VOTO	DATAESAME
A	369871	30	04-MAG-20
B	369871	29	10-MAG-21
C	369871	30	10-GIU-21
D	369871	27	20-GIU-21
A	515140	30	15-MAG-21
B	515140	28	12-MAG-21
C	090456	27	13-MAG-21
D	090456	26	14-GIU-21
A	018701	30	12-LUG-21
D	282320	20	12-GEN-21

SELECT s.Nome

FROM Studenti s

WHERE EXISTS (SELECT \*

FROM Esami e

WHERE e.Matricola = s.Matricola AND e.Voto > 27)



SELECT s.Nome

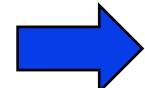
FROM Studenti s, Esami e

WHERE e.Matricola = s.Matricola AND e.Voto > 27



SELECT distinct s.Nome FROM Studenti s , Esami e

WHERE e.Matricola = s.Matricola AND e.Voto > 27



# LA QUANTIFICAZIONE ESISTENZIALE: GIUNZIONE

## Studenti

MATRIC	NOME	COGNOME	DATAISCRIZ
282320	Gianna	Neri	04-MAG-20
369871	Mario	Rossi	04-MAG-20
515140	Mario	Verdi	04-MAG-20
090456	Mario	Bianchi	04-MAG-20
579555	Luigi	Rossi	04-MAG-20
018701	Luca	Bianchi	04-MAG-20

## Esami

CODICE	MATRIC	VOTO	DATAESAME
A	369871	30	04-MAG-20
B	369871	29	10-MAG-21
C	369871	30	10-GIU-21
D	369871	27	20-GIU-21
A	515140	30	15-MAG-21
B	515140	28	12-MAG-21
C	090456	27	13-MAG-21
D	090456	26	14-GIU-21
A	018701	30	12-LUG-21
D	282320	20	12-GEN-21

```
SELECT s.Nome
FROM Studenti s
```

```
WHERE EXISTS (SELECT *
  FROM Esami e
  WHERE e.Matricola = s.Matricola AND e.Voto > 27)
```

```
SELECT s.Nome
FROM Studenti s, Esami e
WHERE e.Matricola = s.Matricola AND e.Voto > 27
```

```
SELECT distinct s.Nome
FROM Studenti s , Esami e
WHERE e.Matricola = s.Matricola AND e.Voto > 27
```

NOME
Mario
Mario
Luca

NOME
Mario
Luca

NOME
Mario
Luca

# LA QUANTIFICAZIONE ESISTENZIALE: GIUNZIONE

## Studenti

MATRIC	NOME	COGNOME	DATAISCRIT
282320	Gianna	Neri	04-MAG-20
369871	Mario	Rossi	04-MAG-20
515140	Mario	Verdi	04-MAG-20
090456	Mario	Bianchi	04-MAG-20
579555	Luigi	Rossi	04-MAG-20
018701	Luca	Bianchi	04-MAG-20

## Esami

CODICE	MATRIC	VOTO	DATAESAME
A	369871	30	04-MAG-20
B	369871	29	10-MAG-21
C	369871	30	10-GIU-21
D	369871	27	20-GIU-21
A	515140	30	15-MAG-21
B	515140	28	12-MAG-21
C	090456	27	13-MAG-21
D	090456	26	14-GIU-21
A	018701	30	12-LUG-21
D	282320	20	12-GEN-21

SELECT s.Nome, s.cognome, s.matricola

FROM Studenti s

WHERE EXISTS (SELECT \*

FROM Esami e WHERE

e.Matricola = s.Matricola AND e.Voto > 27)



NOME	COGNOME	MATRIC
Mario	Rossi	369871
Mario	Verdi	515140
Luca	Bianchi	018701

SELECT s.Nome, s.cognome, s.matricola

FROM Studenti s, Esami e

WHERE e.MatricolaStudente = s.Matricola AND e.Voto > 27



NOME	COGNOME	MATRIC
Mario	Rossi	369871
Mario	Rossi	369871
Mario	Rossi	369871
Mario	Verdi	515140
Mario	Verdi	515140
Luca	Bianchi	018701

# LA QUANTIFICAZIONE ESISTENZIALE: GIUNZIONE

Studenti

MATRIC	NOME	COGNOME	DATAISCRIT
282320	Gianna	Neri	04-MAG-20
369871	Mario	Rossi	04-MAG-20
515140	Mario	Verdi	04-MAG-20
090456	Mario	Bianchi	04-MAG-20
579555	Luigi	Rossi	04-MAG-20
018701	Luca	Bianchi	04-MAG-20

Esami

CODICE	MATRIC	VOTO	DATAESAME
A	369871	30	04-MAG-20
B	369871	29	10-MAG-21
C	369871	30	10-GIU-21
D	369871	27	20-GIU-21
A	515140	30	15-MAG-21
B	515140	28	12-MAG-21
C	090456	27	13-MAG-21
D	090456	26	14-GIU-21
A	018701	30	12-LUG-21
D	282320	20	12-GEN-21

```
SELECT s.Nome, s.cognome, s.matricola
FROM Studenti s, Esami e
WHERE e.MatricolaStudente = s.Matricola AND e.Voto > 27
```

NOME	COGNOME	MATRIC
Mario	Rossi	369871
Mario	Rossi	369871
Mario	Rossi	369871
Mario	Verdi	515140
Mario	Verdi	515140
Luca	Bianchi	018701

```
SELECT distinct s.Nome, s.cognome, s.matricola
FROM Studenti s , Esami e
WHERE e.MatricolaStudente = s.Matricola AND e.Voto > 27
```

NOME	COGNOME	MATRIC
Mario	Verdi	515140
Mario	Rossi	369871
Luca	Bianchi	018701

## Subquery di confronto quantificato, esempio 1

---

### Veicoli

Targa	Cod_mod	Categoria	Cilindrata	Cod_combust.	cav.Fisc	Velocita	Posti	Imm
-------	---------	-----------	------------	--------------	----------	----------	-------	-----

Selezionare tutti i veicoli con cilindrata **inferiore ad almeno** una delle cilindrate dei veicoli con combustibile 02.

```
Select * FROM Veicoli  
Where cilindrata < Any (select cilindrata From Veicoli  
Where cod_combust='02')
```

```
Select * From Veicoli  
Where cilindrata< (select max(cilindrata) From Veicoli  
Where cod_combust='02' )
```

## Subquery di confronto quantificato, esempio

---

### Veicoli

Targa	Cod_mod	Categoria	Cilindrata	Cod_combust.	cav.Fisc	Velocita	Posti	Imm
-------	---------	-----------	------------	--------------	----------	----------	-------	-----

Selezionare **tutti** i veicoli con cilindrata inferiore a **tutte** le cilindrate dei veicoli con combustibile O2 (quindi inferiore alla più bassa di queste cilindrate)

```
Select * FROM Veicoli  
Where cilindrata < All (select cilindrata From Veicoli  
Where cod_combust='O2')
```

```
Select * From Veicoli  
Where cilindrata< (select min(cilindrata) From Veicoli  
Where cod_combust='O2')
```

=ANY e <>ALL

---

In particolare le forme =ANY (equivalentemente IN) e <>ALL (equivalentemente NOT IN), forniscono un modo alternativo per realizzare intersezione e differenza dell'algebra relazionale.

Esempio: intersezione dei modelli con cilindrata inferiore a 1400 e quelli con codice fabbrica uguale a 001

```
Select cod_modello  
From Veicoli  
Where cilindrata < 1400 and  
Cod_modello = ANY [IN] (select cod_modello  
From Veicoli  
Where cod_fab='001')
```

## <> ALL (NOT IN)

---

Differenza fra le due tavelli precedenti: modelli con cilindrata inferiore a 1400 che non sono prodotti dalla fabbrica dal codice 001

```
Select cod_modello  
From Veicoli  
Where cilindrata<1400  
and cod_modello <> ALL [NOT IN] (Select cod_modello  
From Veicoli  
Where cod_fabbrica='001')
```

## LA QUANTIFICAZIONE ESISTENZIALE: ANY

---

- ANY equivale ad EXISTS

- La solita query:

```
SELECT s.Nome FROM Studenti s  
WHERE EXISTS (SELECT * FROM Esami e  
              WHERE e.Matricola = s.Matricola AND e.Voto > 27)
```

- Si può esprimere anche tramite ANY:

```
SELECT s.Nome FROM Studenti s  
WHERE s.Matricola = ANY (SELECT e.Matricola FROM Esami e  
                          WHERE e.Voto > 27)
```

```
SELECT s.Nome FROM Studenti s  
WHERE 27 < ANY (SELECT e.Voto FROM Esami e  
                  WHERE e.Matricola = s.Matricola)
```

## LA QUANTIFICAZIONE ESISTENZIALE: IN

---

- IN è solo un'abbreviazione di =ANY

- La solita query:

```
SELECT s.Nome FROM Studenti s  
WHERE s.Matricola =ANY (SELECT e.Matricola FROM Esami e  
WHERE e.Voto >27)
```

- Si può esprimere anche tramite IN:

```
SELECT s.Nome FROM Studenti s  
WHERE s.Matricola IN (SELECT e.Matricola FROM Esami e  
WHERE e.Voto >27)
```

## Semantica delle espressioni "correlate"

---

- La query più interna può usare variabili della query esterna
- L'interrogazione interna viene eseguita una volta per ciascuna tupla dell'interrogazione esterna

Esempio: trovare tutti gli studenti che hanno un omonimo (usando EXISTS):

- ```
SELECT * FROM Studenti S
WHERE EXISTS (SELECT *
               FROM Studenti S2
              WHERE S2.Nome = S.Nome
                AND S2.Cognome = S.Cognome
                AND S2.Matricola <> S.Matricola)
```

## Semantica delle espressioni "correlate", 2

---

- Esempio: trovare tutti gli studenti che NON hanno un omonimo:

```
SELECT * FROM Studenti S
WHERE NOT EXISTS (SELECT *
                   FROM Studenti S2
                   WHERE S2.Nome = S.Nome
                     AND S2.Cognome = S.Cognome
                     AND S2.Matricola <> S.Matricola)
```

# Persone

| NOME    | ETA | REDDITO |
|---------|-----|---------|
| Andrea  | 27  | 21      |
| Aldo    | 25  | 15      |
| Maria   | 55  | 42      |
| Anna    | 50  | 35      |
| Filippo | 26  | 29      |
| Luigi   | 50  | 40      |
| Franco  | 60  | 20      |
| Olga    | 30  | 41      |
| Sergio  | 85  | 35      |
| Luisa   | 75  | 87      |

# Maternità

| MADRE | FIGLIO  |
|-------|---------|
| Luisa | Maria   |
| Luisa | Luigi   |
| Anna  | Olga    |
| Anna  | Filippo |
| Maria | Andrea  |
| Maria | Aldo    |



# Paternità

| PADRE  | FIGLIO  |
|--------|---------|
| Sergio | Franco  |
| Luigi  | Olga    |
| Luigi  | Filippo |
| Franco | Andrea  |
| Franco | Aldo    |

# Esempio

- Le persone che hanno almeno un figlio

```
SELECT *  
FROM Persone  
WHERE
```

```
EXISTS (SELECT *  
        FROM Paternita  
        WHERE Padre = Nome) OR  
EXISTS (SELECT *  
        FROM Maternita  
        WHERE Madre = Nome)
```

## Esempio

Paternità Z

| PADRE  | FIGLIO  |
|--------|---------|
| Sergio | Franco  |
| Luigi  | Olga    |
| Luigi  | Filippo |
| Franco | Andrea  |
| Franco | Aldo    |

join

Paternità W

| PADRE  | FIGLIO  |
|--------|---------|
| Sergio | Franco  |
| Luigi  | Olga    |
| Luigi  | Filippo |
| Franco | Andrea  |
| Franco | Aldo    |

- I padri i cui figli guadagnano **tutti** più di 20

SELECT distinct Padre

FROM Paternità Z

WHERE NOT EXISTS (

SELECT \*

FROM Paternità W, Persone

WHERE W.Padre = Z.Padre

AND W.Figlio = Nome

AND Reddito <= 20)

Persone

| NOME    | ETA | REDDITO |
|---------|-----|---------|
| Andrea  | 27  | 21      |
| Aldo    | 25  | 15      |
| Maria   | 55  | 42      |
| Anna    | 50  | 35      |
| Filippo | 26  | 29      |
| Luigi   | 50  | 40      |
| Franco  | 60  | 20      |
| Olga    | 30  | 41      |
| Sergio  | 85  | 35      |
| Luisa   | 75  | 87      |

## RIASSUMENDO

---

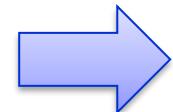
- La quantificazione esistenziale si fa con:
  - Exists (il più espressivo)
  - Giunzione
  - =Any, >Any, <Any...
  - IN
- =Any, >Any, <Any, IN,... non aggiungono potere espressivo
- Il problema vero è: **non confondere esistenziale con universale!**

# LA QUANTIFICAZIONE UNIVERSALE

---

- Gli studenti che hanno preso **solo** 30
- Errore comune (e grave):

```
SELECT s.Nome  
FROM Studenti s, Esami e  
WHERE e.Matricola = s.Matricola AND e.Voto = 30
```



# LA QUANTIFICAZIONE UNIVERSALE CON ALL

---

- la query (studenti con tutti 30):

```
SELECT s.Nome FROM Studenti s  
WHERE NOT EXISTS (SELECT * FROM Esami e  
                    WHERE e.Matricola = s.Matricola  
                    AND e.Voto <> 30)
```

- Sostituendo EXISTS con =ANY, diventa:

```
SELECT s.Nome FROM Studenti s  
WHERE NOT (s.Matricola = ANY (SELECT e.Matricola FROM Esami e  
                                WHERE e.Voto <> 30))
```

- Ovvero:

```
SELECT s.Nome FROM Studenti s  
WHERE s.Matricola <> ALL (SELECT e.Matricola FROM Esami e  
                            WHERE e.Voto <> 30)
```

- Naturalmente, <>ALL è lo stesso di NOT IN...



# LA QUANTIFICAZIONE UNIVERSALE E GLI INSIEMI VUOTI

- Trovare gli studenti che hanno preso **solo** trenta:

```
SELECT s.Nome  
FROM Studenti s  
WHERE NOT EXISTS (SELECT *  
                   FROM Esami e  
                   WHERE e.Matricola = s.Matricola AND e.Voto <> 30)
```



- Perché trovo anche Rossi?



| Nome   | Matricola | Provincia | AnnoNascita |
|--------|-----------|-----------|-------------|
| Bianco | 1         | PI        | 1996        |
| Verdi  | 2         | PI        | 1992        |
| Rossi  | 3         | PI        | 1992        |

| Mater. | Matricola | Voto |
|--------|-----------|------|
| RC     | 1         | 30   |
| IS     | 2         | 30   |
| RC     | 2         | 20   |

# LA QUANTIFICAZIONE UNIVERSALE E GLI INSIEMI VUOTI

Studenti

| MATRICOLA | NOME   | COGNOME | DATAISCRIZIONE |
|-----------|--------|---------|----------------|
| 282320    | Gianna | Neri    | 04-MAG-20      |
| 369871    | Mario  | Rossi   | 04-MAG-20      |
| 515140    | Mario  | Verdi   | (null)         |
| 090456    | Mario  | Bianchi | 04-MAG-20      |
| 579555    | Luigi  | Rossi   | (null)         |
| 018701    | Luca   | Bianchi | 04-MAG-20      |
| 100100    | Sara   | Viola   | 04-GIU-20      |

Esami

| CODICE | MATRIC | VOTO | DATAESAME |
|--------|--------|------|-----------|
| A      | 369871 | 30   | 04-MAG-20 |
| B      | 369871 | 29   | 10-MAG-21 |
| C      | 369871 | 30   | 10-GIU-21 |
| D      | 369871 | 27   | 20-GIU-21 |
| A      | 515140 | 30   | 15-MAG-21 |
| B      | 515140 | 28   | 12-MAG-21 |
| C      | 090456 | 27   | 13-MAG-21 |
| D      | 090456 | 26   | 14-GIU-21 |
| A      | 018701 | 30   | 22-LUG-21 |
| D      | 282320 | 20   | 12-GEN-21 |

Trovare gli studenti che hanno preso **solo** trenta:

```
SELECT s.Nome, s.cognome, s.matricola  
FROM Studenti s  
WHERE NOT EXISTS (SELECT * FROM Esami e  
                   WHERE e.Matricola = s.Matricola  
                     AND e.Voto <> 30)
```

| NOME  | COGNOME | MATRICOLA |
|-------|---------|-----------|
| Luigi | Rossi   | 579555    |
| Sara  | Viola   | 100100    |
| Luca  | Bianchi | 018701    |

Cosa cambia se invece di **NOT EXISTS** uso **<> ALL** oppure **NOT IN?**

# LA QUANTIFICAZIONE UNIVERSALE E GLI INSIEMI VUOTI

Studenti

| MATRICOLA | NOME   | COGNOME | DATAISCRIZIONE |
|-----------|--------|---------|----------------|
| 282320    | Gianna | Neri    | 04-MAG-20      |
| 369871    | Mario  | Rossi   | 04-MAG-20      |
| 515140    | Mario  | Verdi   | (null)         |
| 090456    | Mario  | Bianchi | 04-MAG-20      |
| 579555    | Luigi  | Rossi   | (null)         |
| 018701    | Luca   | Bianchi | 04-MAG-20      |
| 100100    | Sara   | Viola   | 04-GIU-20      |

Esami

| CODICE | MATRIC | VOTO | DATAESAME |
|--------|--------|------|-----------|
| A      | 369871 | 30   | 04-MAG-20 |
| B      | 369871 | 29   | 10-MAG-21 |
| C      | 369871 | 30   | 10-GIU-21 |
| D      | 369871 | 27   | 20-GIU-21 |
| A      | 515140 | 30   | 15-MAG-21 |
| B      | 515140 | 28   | 12-MAG-21 |
| C      | 090456 | 27   | 13-MAG-21 |
| D      | 090456 | 26   | 14-GIU-21 |
| A      | 018701 | 30   | 22-LUG-21 |
| D      | 282320 | 20   | 12-GEN-21 |

Trovare gli studenti che hanno preso **solo** trenta:

```
SELECT s.Nome, s.cognome, s.matricola
FROM Studenti s
WHERE s.Matricola <> ALL (SELECT e.Matricola
                           FROM Esami e
                           WHERE e.Voto <> 30)
```

| NOME  | COGNOME | MATRICOLA |
|-------|---------|-----------|
| Luigi | Rossi   | 579555    |
| Sara  | Viola   | 100100    |
| Luca  | Bianchi | 018701    |

# LA QUANTIFICAZIONE UNIVERSALE E GLI INSIEMI VUOTI

Studenti

| MATRICOLA | NOME   | COGNOME | DATAISCRIZIONE |
|-----------|--------|---------|----------------|
| 282320    | Gianna | Neri    | 04-MAG-20      |
| 369871    | Mario  | Rossi   | 04-MAG-20      |
| 515140    | Mario  | Verdi   | (null)         |
| 090456    | Mario  | Bianchi | 04-MAG-20      |
| 579555    | Luigi  | Rossi   | (null)         |
| 018701    | Luca   | Bianchi | 04-MAG-20      |
| 100100    | Sara   | Viola   | 04-GIU-20      |

Esami

| CODICE | MATRIC | VOTO | DATAESAME |
|--------|--------|------|-----------|
| A      | 369871 | 30   | 04-MAG-20 |
| B      | 369871 | 29   | 10-MAG-21 |
| C      | 369871 | 30   | 10-GIU-21 |
| D      | 369871 | 27   | 20-GIU-21 |
| A      | 515140 | 30   | 15-MAG-21 |
| B      | 515140 | 28   | 12-MAG-21 |
| C      | 090456 | 27   | 13-MAG-21 |
| D      | 090456 | 26   | 14-GIU-21 |
| A      | 018701 | 30   | 22-LUG-21 |
| D      | 282320 | 20   | 12-GEN-21 |

Trovare gli studenti che hanno preso **solo** trenta:

```
SELECT s.Nome, s.cognome, s.matricola  
FROM Studenti s  
WHERE s.Matricola NOT IN (SELECT e.Matricola  
                           FROM Esami e  
                           WHERE e.Voto <> 30)
```

| NOME  | COGNOME | MATRICOLA |
|-------|---------|-----------|
| Luigi | Rossi   | 579555    |
| Sara  | Viola   | 100100    |
| Luca  | Bianchi | 018701    |

# LA QUANTIFICAZIONE UNIVERSALE E GLI INSIEMI VUOTI

Studenti

| MATRICOLA | NOME   | COGNOME | DATAISCRIZIONE |
|-----------|--------|---------|----------------|
| 282320    | Gianna | Neri    | 04-MAG-20      |
| 369871    | Mario  | Rossi   | 04-MAG-20      |
| 515140    | Mario  | Verdi   | (null)         |
| 090456    | Mario  | Bianchi | 04-MAG-20      |
| 579555    | Luigi  | Rossi   | (null)         |
| 018701    | Luca   | Bianchi | 04-MAG-20      |
| 100100    | Sara   | Viola   | 04-GIU-20      |

Esami

| CODICE | MATRIC | VOTO | DATAESAME |
|--------|--------|------|-----------|
| A      | 369871 | 30   | 04-MAG-20 |
| B      | 369871 | 29   | 10-MAG-21 |
| C      | 369871 | 30   | 10-GIU-21 |
| D      | 369871 | 27   | 20-GIU-21 |
| A      | 515140 | 30   | 15-MAG-21 |
| B      | 515140 | 28   | 12-MAG-21 |
| C      | 090456 | 27   | 13-MAG-21 |
| D      | 090456 | 26   | 14-GIU-21 |
| A      | 018701 | 30   | 22-LUG-21 |
| D      | 282320 | 20   | 12-GEN-21 |

Trovare gli studenti che hanno preso **solo** trenta:

SELECT s.Nome, s.cognome, s.matricola

FROM Studenti s

WHERE NOT EXISTS (SELECT \* FROM Esami e

WHERE e. Matricola = s. Matricola

AND e.Voto <> 30)

AND EXISTS (SELECT \* FROM Esami e

WHERE e. Matricola = s. Matricola)

| NOME | COGNOME | MATRICOLA |
|------|---------|-----------|
| Luca | Bianchi | 018701    |

## GLI INSIEMI VUOTI

---

- Se voglio gli studenti che hanno preso **solo più di 27**, e hanno superato **qualche esame**:

```
SELECT s.Nome  
FROM Studenti s  
WHERE NOT EXISTS (SELECT *  
                   FROM Esami e  
                   WHERE e.Matricola = s.Matricola AND e.Voto < 27)  
AND EXISTS (SELECT *  
            FROM Esami e  
            WHERE e.Matricola = s.Matricola)
```

- Oppure:

```
SELECT s.Nome  
FROM Studenti s, Esami e  
WHERE s.Matricola = e.Matricola  
GROUP BY s.Matricola, s.Nome  
HAVING Min(e.Voto) >= 27
```

## OTTIMIZZARE IL NOT EXISTS

---

- Loop interno valutato una volta per ogni studente:

```
SELECT s.Nome  
FROM Studenti s  
WHERE NOT EXISTS (SELECT *  
                   FROM Esami e  
                   WHERE e.Matricola = s.Matricola AND e.Voto >  
                     21)
```

- Loop interno valutato una sola volta (loop interno decorrelato):

```
SELECT s.Nome  
FROM Studenti s  
WHERE s.Matricola NOT IN (SELECT e.Matricola  
                           FROM Esami e  
                           WHERE e.Voto > 21)
```

## NOT IN ED OUTER JOIN

---

- Loop interno valutato una sola volta (loop interno decorrelato):

```
SELECT s.Nome  
FROM Studenti s  
WHERE s.Matricola NOT IN (SELECT e.Matricola  
                           FROM Esami e  
                           WHERE e.Voto > 21)
```

- Outer join:

```
SELECT s.Nome  
FROM Studenti s LEFT JOIN (SELECT *  
                           FROM Esami e  
                           WHERE e.Voto > 21) USING Matricola  
WHERE e.Voto IS NULL
```

## Subquery nel calcolo di espressioni

---

Le subquery oltre che essere usate all'interno della clausola WHERE, possono anche essere utilizzate nel **calcolo di espressioni**, dunque per definire colonne.

**Esempio:** Per ogni veicolo rappresentare la targa, la cilindrata e la differenza fra la cilindrata e la cilindrata minima

### Veicoli

| Targa | Cod_mod | Categoria | Cilindrata | Cod_comb. | cav.Fisc | Velocita | Posti | Imm |
|-------|---------|-----------|------------|-----------|----------|----------|-------|-----|
|-------|---------|-----------|------------|-----------|----------|----------|-------|-----|

Select Targa, Cilindrata, Cilindrata - (Select min (Cilindrata)  
From Veicoli) Differenza  
From Veicoli

## Esempio

### Veicoli

| Targa | Cod_mod* | Categoria | Cilindrata | Cod_comb. | cav.Fisc | Velocita | Posti | Imm |
|-------|----------|-----------|------------|-----------|----------|----------|-------|-----|
|-------|----------|-----------|------------|-----------|----------|----------|-------|-----|

### Modelli

| Cod_Mod | Nome_Mod | Cod_Fab | Cilind_Media |
|---------|----------|---------|--------------|
|---------|----------|---------|--------------|

Per ciascun veicolo rappresentare la targa e la differenza fra la cilindrata e la cilindrata media del proprio modello

Select targa, Cilindrata - (Select cilind\_media  
From Modelli  
Where Veicoli.cod\_mod=Modelli.cod\_mod)  
From Veicoli



Tabella select principale

---

# **UNIONE, INTERSEZIONE, DIFFERENZA**

## Operazioni booleane su tabelle

---

A volte può essere utile poter ottenere un'unica tabella contenente alcuni dei dati contenuti in due tabelle omogenee, ossia con attributi definiti sullo stesso dominio. Per esempio date le tabelle:

Paternità

| PADRE  | FIGLIO  |
|--------|---------|
| Sergio | Franco  |
| Luigi  | Olga    |
| Luigi  | Filippo |
| Franco | Andrea  |
| Franco | Aldo    |

Maternità

| MADRE | FIGLIO  |
|-------|---------|
| Luisa | Maria   |
| Luisa | Luigi   |
| Anna  | Olga    |
| Anna  | Filippo |
| Maria | Andrea  |
| Maria | Aldo    |

Operazioni booleane,  
calcolando  
unione,  
intersezione e  
differenza.

## Unione, intersezione e differenza

---

In SQL la **SELECT** da sola non permette di fare questo tipo di operazioni su tabelle. Esistono per questo dei costrutti esplicativi che utilizzano le parole chiave

**UNION**

**INTERSECT**

**EXCEPT** (in oracle **MINUS**)

Tali operatori lavorano sulle tabelle come se fossero insiemi di righe, dunque i duplicati vengono eliminati (a meno che si usi la specifica **ALL**) anche dalle proiezioni!

## Unione

---

L'operatore **UNION** realizza l'operazione di unione definita nell'algebra relazionale. Utilizza come operandi le due tavole risultanti da comandi **SELECT** e restituisce una terza tabella che contiene **tutte le righe della prima e della seconda tabella**. Nel caso in cui dall'unione e dalla proiezione risultassero delle righe duplicate, l'operatore UNION ne mantiene una sola copia, a meno che non sia specificata l'opzione **ALL** che indica la volontà di mantenere i duplicati

**SELECT ...**

**UNION [all]**

**SELECT ...**

## Notazione posizionale!

---

```
SELECT padre  
FROM paternita  
UNION  
SELECT madre  
FROM maternita
```

- quali nomi per gli attributi del risultato?
  - nessuno
  - quelli del primo operando
  - ...

In SQL quelli del primo operando

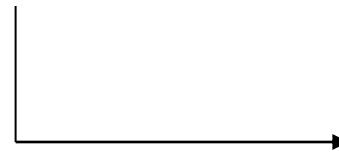
# Paternità

| PADRE  | FIGLIO  |
|--------|---------|
| Sergio | Franco  |
| Luigi  | Olga    |
| Luigi  | Filippo |
| Franco | Andrea  |
| Franco | Aldo    |

# Maternità

| MADRE | FIGLIO  |
|-------|---------|
| Luisa | Maria   |
| Luisa | Luigi   |
| Anna  | Olga    |
| Anna  | Filippo |
| Maria | Andrea  |
| Maria | Aldo    |

```
SELECT padre, figlio  
FROM paternita  
UNION  
SELECT madre, figlio  
FROM maternita
```



L'attributo Padre della prima Select viene messo nella stessa colonna con l'attributo Madre della seconda select

| PADRE  | FIGLIO  |
|--------|---------|
| Sergio | Franco  |
| Luigi  | Olga    |
| Luigi  | Filippo |
| Franco | Andrea  |
| Franco | Aldo    |
| Luisa  | Maria   |
| Luisa  | Luigi   |
| Anna   | Olga    |
| Anna   | Filippo |
| Maria  | Andrea  |
| Maria  | Aldo    |

# Paternità

| PADRE  | FIGLIO  |
|--------|---------|
| Sergio | Franco  |
| Luigi  | Olga    |
| Luigi  | Filippo |
| Franco | Andrea  |
| Franco | Aldo    |

# Maternità

| MADRE | FIGLIO  |
|-------|---------|
| Luisa | Maria   |
| Luisa | Luigi   |
| Anna  | Olga    |
| Anna  | Filippo |
| Maria | Andrea  |
| Maria | Aldo    |

```
SELECT Padre, Figlio  
FROM paternita  
UNION  
SELECT Figlio, Madre  
FROM maternita
```

```
SELECT padre as genitore, figlio  
FROM paternita  
UNION  
SELECT figlio, madre as genitore  
FROM maternita
```



L'attributo Padre della prima Select viene messo nella stessa colonna con l'attributo Figlio della seconda select

| PADRE   | FIGLIO  |
|---------|---------|
| Sergio  | Franco  |
| Luigi   | Olga    |
| Luigi   | Filippo |
| Franco  | Andrea  |
| Franco  | Aldo    |
| Maria   | Luisa   |
| Luigi   | Luisa   |
| Olga    | Anna    |
| Filippo | Anna    |
| Andrea  | Maria   |
| Aldo    | Maria   |

## Riepilogo: Notazione posizionale

---

Sbagliata

```
SELECT padre as genitore, figlio  
FROM paternita  
UNION  
SELECT figlio, madre as genitore  
FROM maternita
```

Corretta:

```
SELECT padre as genitore, figlio  
FROM paternita  
UNION  
SELECT madre as genitore, figlio  
FROM maternita
```

Quanto detto  
riguardo alla  
notazione posizionale  
dell'operatore **UNION**  
vale equivalentemente  
per gli altri operatori  
booleani **EXCEPT**  
**(Minus in Oracle)** e  
**INTERSECT**.

## Differenza

---

L'operatore **EXCEPT** utilizza come operandi due tavole ottenute mediante due **select** e ha come risultato una nuova tabella che contiene **tutte le righe della prima che non si trovano nella seconda**.

Realizza la differenza dell'algebra relazionale. Anche qui si può specificare l'opzione **ALL** per indicare la volontà di mantenere i duplicati.

Sintassi:

**SELECT ...**

**EXCEPT [ALL]**

**SELECT ...**



## Except, esempio

---

Impiegati il cui nome  
che non coincide col  
cognome di qualche  
altro impiegato

```
SELECT Nome  
FROM Impiegato  
EXCEPT  
SELECT Cognome as Nome  
FROM Impiegato
```

Nota che la differenza  
può essere effettuata  
da un'unica select che  
utilizza subselect

```
SELECT nome  
FROM Impiegato  
WHERE nome<>All[not in]  
(SELECT cognome  
FROM Impiegato)
```

## Intersezione

---

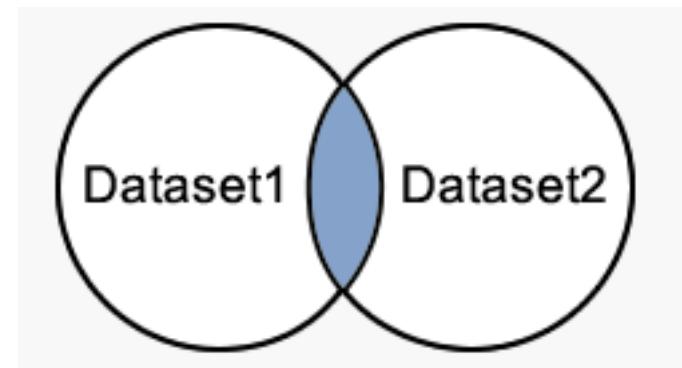
L'operatore **INTERSECT** utilizza come operandi due tabelle risultanti dai comandi **SELECT** e restituisce una tabella che contiene **le righe comuni alle due tabelle iniziali**. Realizza l'intersezione dell'algebra relazionale.

Sintassi

**SELECT ...**

**INTERSECT [ALL]**

**SELECT ...**



L'opzione **ALL** serve a mantenere gli eventuali duplicati di righe.

In sua assenza, si mantiene una sola copia delle righe duplicate.

---

# **SQL PER LA MODIFICA DI BASI DI DATI**

# Data Manipulation Language

---

Introduciamo ora il **Data Manipulation Language (DML)** ossia il linguaggio SQL che serve per **inserire, modificare e cancellare** i dati del database, ma anche per **interrogare il database**, ossia estrarre i dati dal database.

Inizialmente descriveremo le istruzioni che servono a inserire, cancellare e modificare i dati.

In seguito introdurremo le istruzioni per estrarre dal database le informazioni che ci interessano.

# SQL PER MODIFICARE I DATI

---

- `INSERT INTO Tabella [ (A1,..,An)]  
( VALUES (V1,..,Vn) | AS Select )`
- `UPDATE Tabella  
SET Attributo = Expr, ..., Attributo = Expr  
WHERE Condizione`
- `DELETE FROM Tabella  
WHERE Condizione`

## Insert semplice

---

- Supponiamo di volere inserire un nuovo dato in una tabella.
- Tale operazione si realizza mediante l'istruzione

INSERT INTO... VALUES

Sintassi:

INSERT INTO nome tabella  
[(ListaAttributi)] **VALUES** (ListaDiValori) |

SQLSelect

## Insert, Esempio 1

---

Supponiamo di avere definito la tabella:

Esami

| Corso | Insegnante | Matricola | Voto |
|-------|------------|-----------|------|
|-------|------------|-----------|------|

**INSERT INTO Esami (Corso, Matricola, Voto)  
VALUES ('DB1', '123456', 27)**

| Corso | Insegnante | Matricola | Voto |
|-------|------------|-----------|------|
| DB1   | NULL       | 123456    | 27   |

Ai valori non attribuiti viene assegnato NULL, a meno che non sia specificato un diverso valore di default.  
L'inserimento fallisce se NULL non è permesso per gli attributi mancanti.

## Insert, Esempio 2

---

```
INSERT INTO Esami  
VALUES ('DB2', 'Verdi', '123123', 30)
```

| Corso | Insegnante | Matricola | Voto |
|-------|------------|-----------|------|
| DB1   | NULL       | 123456    | 27   |
| DB2   | Verdi      | 123123    | 30   |

Non specificare gli attributi equivale a specificare tutte le colonne della tabella.

NB: Si deve rispettare l'ordine degli attributi

## Insert mediante SELECT

---

E' possibile effettuare un insert prendendo i dati da un'altra tabella. Questo è possibile mediante il comando di interrogazione del database **SELECT**, che vedremo nelle prossime lezioni. Parleremo in seguito di questo tipo di inserimento di dati.

Con questo tipo di insert si possono effettuare tanti inserimenti simultaneamente.

**Esempio:**

Tabella: Indirizzi\_Studenti( Indirizzo, Telefono, Email)

```
INSERT INTO Indirizzi_Studenti (Indirizzo, Telefono)
SELECT Indirizzo, Telefono
FROM Studenti
```

# Delete

---

Cancellazione di righe da tavole

Sintassi:

```
DELETE FROM nome_tabella  
[WHERE Condizione]
```

Per eliminare un elemento bisogna individuare quale.  
Questo si può stabilire mediante la clausola **WHERE**, dove viene stabilita una condizione che individua l'elemento (o gli elementi) da cancellare.  
Spesso un particolare elemento può essere individuato mediante il suo valore nella chiave primaria.

## Delete, esempio

---

Cancellare dalla tabella Esami i dati relativi allo studente il cui numero di matricola è '123456'

```
DELETE FROM Esami  
WHERE Matricola = '123456'
```

Operazione non reversibile

Se la condizione è omessa questa istruzione cancella l'intero contenuto della tabella

(!!! Attenzione quindi !!!)

Lo schema invece non viene modificato. L'istruzione

```
DELETE FROM Esami
```

Restituisce la tabella Esami con l'istanza vuota

## Delete

---

- La condizione del delete può essere una normale condizione di SELECT (vedremo dopo)
- Questa modalità di delete permette di cancellare più righe con un'unica istruzione, purchè le righe soddisfino la condizione.
- Esempio:
  - Eliminare tutte le righe della tabella esami in cui il numero di matricola non si trova nella tabella Studenti

```
DELETE FROM Esami  
WHERE Matricola NOT IN  
(SELECT Matricola FROM Studenti)
```

## Update

---

Inoltre è possibile aggiornare alcuni dati seguendo la  
seguente sintassi:

UPDATE Tabella  
SET Attributo = Espr  
WHERE Condizione

## Update, esempi

---

Esempio: Dalla tabella Aule modificare il numero dell'aula da 3 a 7.

```
UPDATE Aule  
SET Aula = 7  
WHERE Aula = 3
```

Esempio: modificare il valore del reddito delle persone più giovani di 30 anni attribuendo loro un aumento del 10%.

```
UPDATE Persone  
SET Reddito = Reddito * 1.1  
WHERE Eta < 30
```

---

# DATA DEFINITION LANGUAGE (DDL) SQL

Materiale adattato dal libro Albano et al e  
dal libro Atzeni-et al., Basi di dati

## Definizione degli oggetti in SQL

---

Introduciamo il **Data Definition Language (DDL)** SQL, che consiste nell'insieme delle istruzioni SQL che permettono la creazione, modifica e cancellazione delle tabelle, dei domini e degli altri oggetti del database, al fine di definire il suo schema logico.

## Definizione delle tabelle

---

- Le tabelle (corrispondenti alle relazioni dell'algebra relazionale) vengono definite in
  - SQL mediante l'Istruzione **CREATE TABLE**.
- 
- Questa istruzione
    - definisce uno schema di relazione e ne crea un'istanza vuota
    - specifica attributi, domini e vincoli

## CREATE TABLE, sintassi

---

CREATE TABLE <nome\_tabella>

```
( nome_colonna_1 tipo_colonna_1 clausola_default_1 vincolo_di_colonna_1,  
  nome_colonna_2 tipo_colonna_2 clausola_default_2 vincolo_di_colonna_2,  
  ....  
  nome_colonna_k tipo_colonna_k clausola_default_k vincolo_di_colonna_k,  
  vincoli di tabella  
)
```

L'istruzione che crea la tabella è **CREATE TABLE**, seguito da un nome che la caratterizza, e dalla lista delle colonne (attributi), di cui si specificano le caratteristiche. Alla fine si possono anche specificare eventuali vincoli di tabella, di cui parleremo in seguito

## CREATE TABLE, esempio

---

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

## **CREATE TABLE**, effetto

---

L'effetto del comando **CREATE TABLE** definisce uno schema di relazione e ne crea un'istanza vuota, specificandone attributi, domini e vincoli.

Una volta creata, la tabella è pronta per l'inserimento dei dati che dovranno soddisfare i vincoli imposti.

Nel caso dell'esempio dato nella slide precedente, si ottiene

### **IMPIEGATO**

| Matricola | Nome | Cognome | Dipart | Stipendio |
|-----------|------|---------|--------|-----------|
|-----------|------|---------|--------|-----------|

La visualizzazione dello schema di una tabella, dopo che è stata creata, può essere ottenuta mediante il comando SQL **DESCRIBE**. Nel caso specifico:

**DESCRIBE impiegato**

## SQL PER LA DEFINIZIONE DI BASI DI DATI

---

- SQL non è solo un linguaggio di interrogazione (Query Language), ma
- Un linguaggio per la definizione di basi di dati (Data-definition language (DDL))
  - CREATE SCHEMA Nome AUTHORIZATION Utente
  - CREATE TABLE o VIEW, con vincoli
  - CREATE INDEX
  - CREATE PROCEDURE
  - CREATE TRIGGER
- Un linguaggio per stabilire controlli sull'uso dei dati: GRANT
- Un linguaggio per modificare i dati.

## I tipi

---

I tipi più comuni per i valori degli attributi sono:

- **CHAR(n)** per stringhe di caratteri di lunghezza fissa n;
- **VARCHAR(n)** per stringhe di caratteri di lunghezza variabile di al massimo n caratteri;
- **INTEGER** per interi con la dimensione uguale alla parola di memoria standard dell'elaboratore;
- **REAL** per numeri reali con dimensione uguale alla parola di memoria standard dell'elaboratore;
- **NUMBER(p,s)** per numeri con p cifre, di cui s decimali;
- **FLOAT(p)** per numeri binari in virgola mobile, con almeno p cifre significative;
- **DATE** per valori che rappresentano istanti di tempo (in alcuni sistemi, come Oracle), oppure solo date (e quindi insieme ad un tipo TIME per indicare ora, minuti e secondi).

## DEFINIZIONE DI TABELLE: ESEMPIO

---

```
CREATE TABLE Impiegati
(
    Codice CHAR(8) NOT NULL,
    Nome CHAR(20),
    AnnoNascita INTEGER CHECK (AnnoNascita < 2000),
    Qualifica CHAR(20) DEFAULT 'Impiegato',
    Supervisore CHAR(8),
    PRIMARY KEY pk_impiagato (Codice),
    FOREIGN KEY fk_ Impiegati (Supervisore)
    REFERENCES Impiegati )
```



```
CREATE TABLE FamiliariACarico
(
    Nome CHAR(20),
    AnnoNascita INTEGER,
    GradoParentela CHAR(10),
    CapoFamiglia CHAR(8),
    FOREIGN KEY fk_ FamiliariACarico (CapoFamiglia)
    REFERENCES Impiegati )
```

## DEFINIZIONE DI TABELLE

---

- Ciò che si crea con un *CREATE* si può eliminare con il comando *DROP* o cambiare con il comando *ALTER*.

*CREATE TABLE Nome*

( Attributo Tipo [ValoreDefault] [VincoloAttributo]  
{, Attributo Tipo [Default] [VincoloAttributo]}  
{, VincoloTabella})

Default := DEFAULT {valore | null | username}

- Nuovi attributi si possono aggiungere con:

*ALTER TABLE Nome ADD COLUMN NuovoAttr Tipo*

## Modificare una tabella

---

Con il comando **ALTER TABLE** è possibile (standard SQL):

1. Aggiungere una colonna (**ADD [COLUMN]**)
2. Eliminare una colonna (**DROP [COLUMN]**)
3. Modificare la colonna (**MODIFY**)
4. Aggiungere l'assegnazione di valori di default (**SET DEFAULT**)
5. Eliminare l'assegnazione di valori di default (**DROP DEFAULT**)
6. Aggiungere vincoli di tabella (**ADD CONSTRAINT**)
7. Eliminare vincoli di tabella (**DROP CONSTRAINT**)
8. Altre opzioni sono possibili nei linguaggi specifici (vedi manuali)

## Aggiungere una colonna

---

Sintassi:

`ALTER TABLE nome_tabella`

`ADD [COLUMN] nome_col tipo_col default_col vincolo_col`

In mancanza di altre specifiche, la nuova colonna viene inserita come ultima colonna della tabella.

Altrimenti è possibile dare questa specifica:

`ADD COLUMN <CREA_DEFINIZIONE> [FIRST/AFTER <nome_colonna>]`

`FIRST` permette di aggiungerla come prima colonna/`AFTER` colonna subito dopo la colonna indicata

**ESEMPIO:** Aggiungere alla tabella Impiegato la colonna nomecapo.

`ALTER TABLE impiegato`

`ADD COLUMN nomecapo varchar(20) default 'Rossi' not null`

## Regole per aggiungere una colonna

---

- Si può aggiungere una colonna in qualsiasi momento se **non viene specificato NOT NULL**.
- Come si può aggiungere una colonna **NOT NULL** con tre passaggi?

## Eliminare una colonna

---

**ALTER TABLE nome\_tabella**

**DROP COLUMN nome\_colonna {RESTRICT/CASCADE}**

In SQL standard le opzioni **RESTRICT/CASCADE** sono alternative ed è obbligatorio specificare l'una o l'altra

**RESTRICT**: se un'altra tabella si ha un vincolo di integrità referenziale con questa colonna, l'esecuzione del comando drop fallisce.

**CASCADE**: eliminando la colonna, vengono eliminate tutte le dipendenze logiche di altre colonne dello schema da questa.

**ALTER TABLE Impiegato**  
**Drop column dipart restrict**

**ALTER TABLE impiegato**  
**Drop column dipart cascade**

## Modificare una colonna

---

Se si vogliono modificare le caratteristiche di una colonna dopo averla definita, occorre eseguire l'istruzione:

```
ALTER TABLE nome_tabella MODIFY  
nome_colonna tipo_col default_col vincoli_col
```

**ESEMPIO:** Supponendo che nella tabella Impiegato ci sia una colonna 'nome' definita come varchar(20), modificarla in modo che diventi un varchar(30) e sia definito su di essa il vincolo not null.

```
ALTER TABLE impiegato MODIFY  
nome varchar(30) not null
```

## Assegnare un valore di default

---

Nell'SQL standard è possibile impostare un valore di default col comando specifico SET DEFAULT, con la seguente sintassi

```
ALTER TABLE nome_tabella  
ALTER [COLUMN] nome_colonna  
SET DEFAULT valore_default
```

**ESEMPIO:** Imporre il valore di default 'Direzione Generale' ai valori della colonna Dipart in cui tale valore non è assegnato esplicitamente

```
ALTER TABLE Impiegato  
Alter [column] Dipart  
SET DEFAULT 'Direzione Generale'
```

## Eliminare un valore di default

---

In SQL standard è possibile eliminare un vincolo di default da una colonna mediante l'istruzione

```
ALTER TABLE nome_tabella  
ALTER [COLUMN] nome_colonna  
DROP DEFAULT
```

Eseguendo questa istruzione il valore di default diventa automaticamente NULL

**Esempio:** Eliminare il default introdotto nell'esercizio precedente

```
ALTER TABLE Impiegato  
ALTER [COLUMN] Dipart  
DROP DEFAULT
```

## Aggiungere vincoli di tabella (che vedremo in dettaglio dopo)

---

Se si vuole aggiungere un vincolo di tabella, si esegue il comando

```
ALTER TABLE nome_tabella  
ADD CONSTRAINT nome_vincolo vincolo_di_tabella
```

**ESEMPIO:** Nella tabella Impiegato, aggiungere un vincolo di unicità alla coppia (nome, cognome)

```
ALTER TABLE impiegato  
ADD CONSTRAINT unique_const unique(nome, cognome)
```

N.B.: Occorre assegnare un nome al vincolo

## Esempi

---

Aggiungere un vincolo di chiave primaria

```
ALTER TABLE Info_Personali ADD CONSTRAINT  
Pkey PRIMARY KEY (id_impianto);
```

Aggiungere un vincolo di chiave esterna

```
ALTER TABLE Info_Personali ADD CONSTRAINT  
Fkey FOREIGN KEY (id_Impiegato) REFERENCES Impiegati (id_impianto)
```

Aggiungere un vincolo di unicità

```
ALTER TABLE Info_Personali ADD CONSTRAINT unique_con UNIQUE  
(codice_fiscale)
```

Aggiungere un vincolo CHECK

```
ALTER TABLE Info_Personali ADD CONSTRAINT check_con CHECK  
(stipendio > 0)
```

## Eliminare vincoli di tabella

---

Nello standard SQL, se si vuole eliminare un vincolo di tabella si esegue l'istruzione

`ALTER TABLE nome_tabella`

`DROP CONSTRAINT nome_vincolo{RESTRICT/CASCADE}`

L'opzione **RESTRICT** non permette di eliminare vincoli di unicità e di chiave primaria su una colonna se esistono vincoli di chiave esterna che si riferiscono a tale colonna.

L'opzione **CASCADE** non opera questa restrizione.

Da notare che per eliminare un vincolo, esso deve essere definito mediante un identificatore

## Drop Table

---

Si può eliminare una tabella mediante l'istruzione DROP TABLE

Nello standard SQL si possono anche specificare le opzioni

**RESTRICT/CASCADE**

**RESTRICT:** se la tabella è utilizzata nella definizione di altri oggetti dello schema, la sua eliminazione viene impedita.

**CASCADE:** vengono eliminate tutte le dipendenze degli altri oggetti dello schema da questa tabella

## Esercizio

---

1. Creare una tabella studenti che contiene matricola, nome cognome data di nascita e numero di esami effettuati, senza specificare alcun vincolo.
2. Dopo aver creato la tabella, aggiungere una colonna con la media dei voti.
3. Aggiungere quindi le colonne telefono ed email.
4. Quindi modificare la tabella in modo tale da rendere il numero di matricola chiave primaria.
5. Aggiungere un vincolo di tabella, specificando che la tripla nome cognome e data di nascita non può essere uguale per diversi studenti.
6. Cancellare la colonna relativa al numero di esami effettuati
7. Eliminare il vincolo creato al punto 5.
8. Eliminare le colonne email e numero di telefono.

## Domanda 1

---

- Creare una tabella studenti che contiene matricola, nome cognome data di nascita e numero di esami effettuati, senza specificare alcun vincolo.

```
CREATE TABLE studenti
(
    matricola char(6),
    nome varchar(20),
    cognome varchar(20),
    nascita date,
    n_esami number(3)
)
```

## Domanda 2 e 3

---

2. Dopo aver creato la tabella, aggiungere una colonna con la media dei voti.

Alter table studenti add  
media\_voti number(5,2) check (media\_voti>=0)

3. Aggiungere quindi le colonne telefono ed email.

Alter table studenti add  
(telefono varchar(15),  
Email varchar(20))

---

4. Quindi modificare la tabella in modo tale da rendere il numero di matricola chiave primaria.

Alter table studenti modify  
Matricola char(6) primary key

5. Aggiungere un vincolo di tabella, specificando che la tripla nome cognome e data di nascita non può essere uguale per diversi studenti.

Alter table studenti add constraint  
ncn\_unique unique(nome,cognome, nascita)

---

6. Cancellare la colonna relativa al numero di esami effettuati

Alter table studenti  
Drop column n\_esami

7. Eliminare il vincolo creato al punto 5.

Alter Table studenti drop constraint ncn\_unique

8. Eliminare le colonne email e numero di telefono.

Alter table studenti  
Drop email, drop telefono

---

# I vincoli

## Vincoli intrarelazionali

---

- I **vincoli di integrità** consentono di limitare i valori ammissibili per una determinata colonna della tabella in base a specifici criteri.
- I vincoli di integrità intrarelazionali (ossia che non fanno riferimento ad altre relazioni) sono:
  - NOT NULL
  - UNIQUE definisce chiavi
  - PRIMARY KEY: chiave primaria (una sola, implica NOT NULL)
  - CHECK, vedremo più avanti

## UNIQUE

---

- Può essere espresso in due forme:
  - nella definizione di un attributo, se forma da solo la chiave
  - come elemento separato
- Il vincolo **unique** utilizzato nella definizione dell'attributo indica che non ci possono essere due valori uguali in quella colonna.
- È una chiave della relazione, ma non una chiave primaria.

## Esempio di vincolo unique

---

```
CREATE TABLE Impiegato(  
    Matricola CHAR(6) PRIMARY KEY,  
    Codice_fiscale CHAR(16) UNIQUE,  
    Nome VARCHAR(20) NOT NULL,  
    Cognome VARCHAR(20) NOT NULL,  
    Dipart VARCHAR(15),  
    Stipendio NUMBER(9) DEFAULT 0,  
    FOREIGN KEY(Dipart) REFERENCES Dipartimento(NomeDip),  
    UNIQUE (Cognome,Nome)  
)
```

## Vincolo unique per insiemi di attributi

---

- Il **vincolo di unicità** può anche essere riferito a coppie o insiemi di attributi. Ciò non significa che per gli attributi dell'insieme considerato ogni singolo valore deve apparire una sola volta, ma che non ci siano due dati (righe) per cui l'insieme dei valori corrispondenti a quegli attributi siano uguali.
- In questo caso il vincolo viene dichiarato dopo aver dichiarato tutte le colonne mediante un vincolo di tabella, utilizzando il comando

Unique (lista attributi)

## Esempio vincolo unique per insiemi di attributi

---

```
CREATE TABLE Impiegato(
    Matricola CHAR(6) PRIMARY KEY,
    codice_fiscale CHAR(16) UNIQUE,
    Nome VARCHAR(20) NOT NULL,
    Cognome VARCHAR(20) NOT NULL,
    Dipart VARCHAR(15),
    Stipendio NUMBER(9) DEFAULT 0,
    FOREIGN KEY(Dipart) REFERENCES Dipartimento(NomeDip),
    UNIQUE (Cognome,Nome)
)
```

Scrivere:

Nome VARCHAR(20) NOT NULL UNIQUE,  
Cognome VARCHAR(20) NOT NULL UNIQUE,  
Sarebbe stato equivalente?

## Primary key

---

Due forme:

- nella definizione di un attributo, se formato da solo la chiave
- come elemento separato

Il vincolo **PRIMARY KEY** è simile a unique, ma definisce la chiave primaria della relazione, ossia un attributo che individua univocamente un dato.

Implica sia il vincolo **UNIQUE** che il vincolo **NOT NULL** (non è ammesso che per uno degli elementi della tabella questo valore sia non definito). Serve ad identificare univocamente i soggetti del dominio. Questo vincolo permette spesso il collegamento fra due tabelle (vedremo in seguito)

## Esempio chiave primaria

---

```
CREATE TABLE Impiegato(  
    Matricola CHAR(6) PRIMARY KEY,  
    Codice_fiscale CHAR(16) UNIQUE,  
    Nome VARCHAR(20) NOT NULL,  
    Cognome VARCHAR(20) NOT NULL,  
    Dipart VARCHAR(15),  
    Stipendio NUMBER(9) DEFAULT 0,  
    FOREIGN KEY(Dipart) REFERENCES Dipartimento(NomeDip),  
    UNIQUE (Cognome,Nome)  
)
```

## Chiave primaria con insiemi di attributi

---

Analogamente al vincolo unique, anche il vincolo di chiave primaria può essere definito su un insieme di elementi. In tal caso la sintassi è simile a quella di unique

Primary key (lista di attributi)

```
CREATE TABLE Studente(  
    Nome VARCHAR(20),  
    Cognome VARCHAR(20),  
    nascita DATE,  
    Corso_Laurea VARCHAR(15),  
    Facolta VARCHAR (20)  
    PRIMARY KEY(Cognome, Nome, Nascita)  
)
```

## Vincoli interrelazionali

---

- I vincoli interrelazionali sono quei vincoli che vengono imposti quando gli attributi di due diverse tabelle devono essere messi in relazione.
- Questo è fatto per soddisfare l'esigenza di un database di **non essere ridondante** e di avere i **dati sincronizzati**.
- Se due tabelle gestiscono gli stessi dati, è bene che di essi non ce ne siano più copie, sia allo scopo di non occupare troppa memoria, sia affinché le modifiche fatte su dati uguali utilizzati da due tabelle siano coerenti.

## Vincoli interrelazionali

---

- **REFERENCES** e **FOREIGN KEY** permettono di definire vincoli di integrità referenziale
- di nuovo **due sintassi**
  - per singoli attributi (come vincolo di colonna)
  - su più attributi (come vincolo di tabella)
- E' possibile definire politiche di **reazione alla violazione** (ossia stabilire l'azione che il DBMS deve compiere quando si viola il vincolo)

## Vincolo di chiave esterna come vincolo di colonna

---

```
CREATE <nome tabella>
(attributo_1...,  

 attributo_2...,  

 ...  

attributo_k REFERENCES tabella_riferita(colonna_riferita)  

...  

)
```

## Vincolo di chiave esterna come vincolo di tabella

---

```
CREATE <nome_tabella>
```

```
(attributo_1 ...,
```

```
    attributo_2 ...,
```

```
...
```

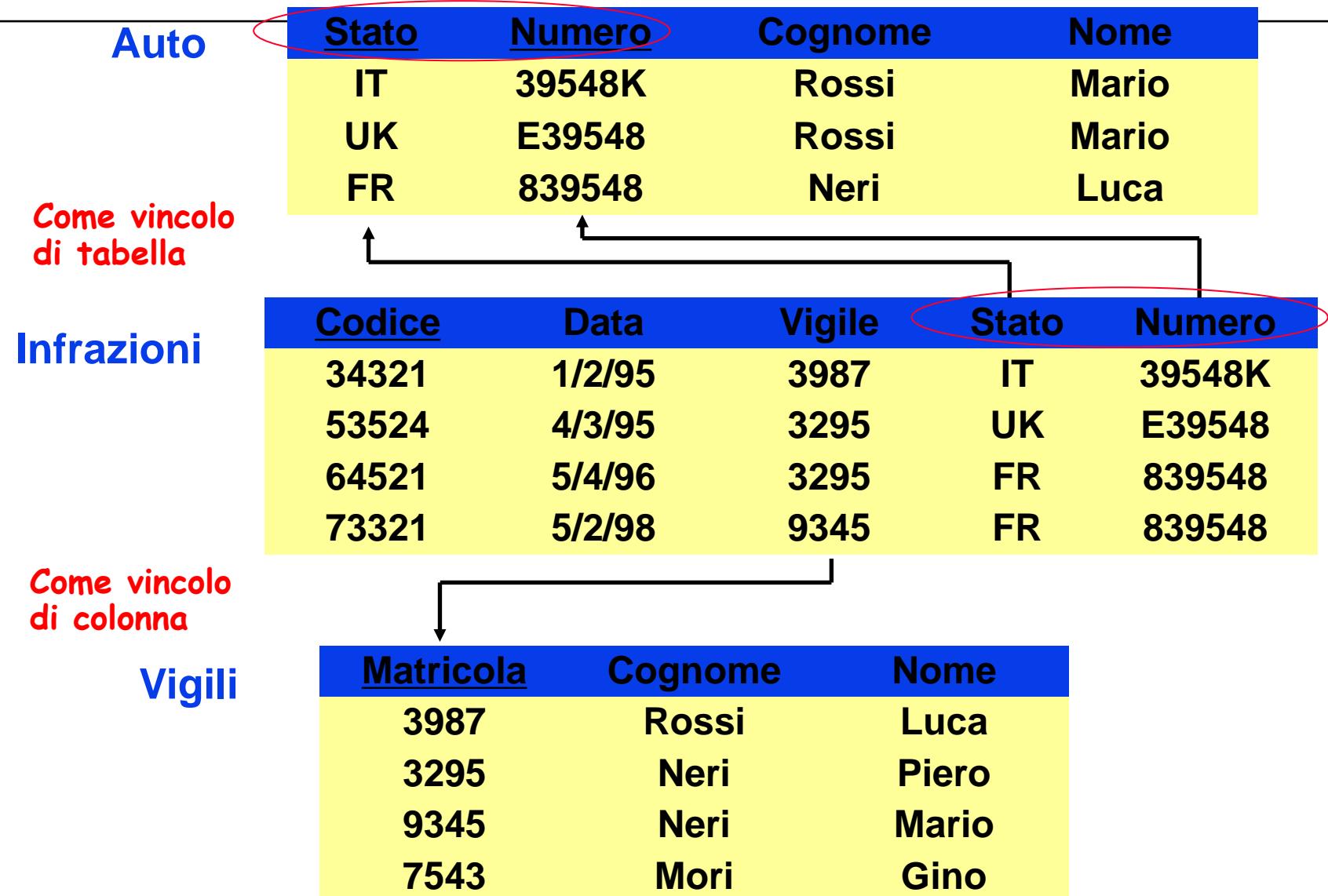
```
    attributo_n ...,
```

```
FOREIGN KEY (col_referenti) REFERENCES  
    tab_riferita(col_riferite)
```

```
...
```

```
)
```

## Vincolo referenziale, esempio



# Vincolo references, foreign key, esempio

---

```
Create TABLE Vigili (
    Matricola INTEGER PRIMARY KEY,
    Nome VARCHAR (15),
    Cognome VARCHAR (15))
```

```
CREATE TABLE Auto (
    Stato CHAR(2),
    Numero CHAR (6),
    Cognome VARCHAR(15),
    Nome VARCHAR(15),
    Primary key (stato, numero))
```

```
CREATE TABLE Infrazioni(
    Codice CHAR(6) PRIMARY KEY,
    Data DATE NOT NULL,
    Vigile INTEGER NOT NULL
    REFERENCES Vigili(Matricola),
    Stato CHAR(2),
    Numero CHAR(6),
    FOREIGN KEY(Stato, Numero)
    REFERENCES Auto(Stato,
    Numero)
)
```

## Vincolo: **CHECK**

---

Un vincolo di **CHECK** richiede che una colonna, o una combinazione di colonne, soddisfi una condizione per ogni riga della tabella.

Il vincolo **CHECK** deve essere una espressione booleana che è valutata usando i valori della colonna che vengono inseriti o aggiornati nella riga.

Può essere espresso sia come **vincolo di riga** che come **vincolo di tabella**.

Se è espresso come **vincolo di riga**, può coinvolgere solo l'attributo su cui è definito, mentre se serve eseguire un check che coinvolge due o più attributi, si deve definire come **vincolo di tabella**

## CHECK, esempio

---

Nessuno stipendio degli impiegati può avere valore minore di 0 (espresso come vincolo di riga).

```
CREATE TABLE IMPIEGATO (
    Matricola CHAR(6) PRIMARY KEY,
    Nome CHAR(20) NOT NULL,
    Cognome CHAR(20) NOT NULL,
    Dipart CHAR(15),
    Stipendio NUMERIC(9) DEFAULT 0 CHECK (Stipendio>=0),
    FOREIGN KEY(Dipart) REFERENCES
        Dipartimento(NomeDip),
    UNIQUE (Cognome,Nome)
)
```

## CHECK, esempio

---

Nessuno stipendio degli impiegati può avere valore minore di 0 (espresso come vincolo di tabella).

```
CREATE TABLE IMPIEGATO (
    Matricola CHAR(6) PRIMARY KEY,
    Nome CHAR(20) NOT NULL,
    Cognome CHAR(20) NOT NULL,
    Dipart CHAR(15),
    Stipendio NUMERIC(9) DEFAULT 0,
    FOREIGN KEY(Dipart) REFERENCES
        Dipartimento(NomeDip),
    UNIQUE (Cognome,Nome) ,
    CHECK (Stipendio>=0)
)
```

## Check, esempio

---

I dipartimenti si possono trovare solo nelle locazioni di Boston, New York e Dallas.

```
CREATE TABLE Dipartimenti  
(dip_cod char(4) primary key,  
dip_nome varchar2(20) not null,  
dip_citta varchar2(15) not null,  
CHECK (dip_citta = 'Boston'  
       or dip_citta='New York'  
       or dip_citta='Dallas')  
)
```

## Esempio Check

---

Un vincolo check sullo stipendio e la commissione per evitare che la commissione sia più alta del salario.

```
Create table pagamenti(  
pag_cod char(6),  
pag_codicef char(16) REFERENCES dipendenti(dipe_codicef),  
pag_stipendio number(8,2),  
pag_commissione number (8,2),  
check (pag_stipendio>pag_commissione) )
```

## VINCOLI D'INTEGRITÀ: CHIAVI E GENERALI

---

- Vincoli su attributi
  - VincoloAttributo :=  
[NOT NULL [UNIQUE] ] | [CHECK (Condizione) ]  
[REFERENCES Tabella [(Attributo {, Attributo})]]
- Vincoli su tabella
  - VincoloTabella := UNIQUE (Attributo {, Attributo})  
| CHECK (Condizione) |  
| PRIMARY KEY [Nome] (Attributo {, Attributo})  
| FOREIGN KEY [Nome] (Attributo {, Attributo})  
  REFERENCES Tabella [(Attributo {, Attributo})]  
  [ON DELETE {NO ACTION} | CASCADE | SET NULL] ]

## Reazione alla violazione

---

Quando si crea un vincolo foreign key in una tabella, in SQL standard si può specificare l'azione da intraprendere quando delle righe nella tabella riferita vengono cancellate o modificate.

Tali reazioni alla violazione vengono dichiarate al momento della definizione dei vincoli di foreign key rispettivamente mediante i comandi

ON DELETE

ON UPDATE

## Sintassi

---

Per vincoli foreign key **di colonna**:

**CREATE TABLE nome\_tabella**

( attr\_1...,  
attr\_2...,  
...

attr\_k ... REFERENCES tab\_riferita(attr\_riferito)  
ON DELETE/ON UPDATE reazione

...  
attr\_n ...  
vincoli tabella)

Per vincoli foreign key **di tabella**:

**CREATE TABLE nome\_tabella**

( attr\_1...,  
attr\_2...,  
...

attr\_n ...,

**FOREIGN KEY tab\_referente(attr\_referenti) REFERENCES  
tab\_riferita(attr\_riferito) ON DELETE/ON UPDATE reazione**

## Reazioni alla violazione on delete

---

Impedire il delete (**NO ACTION**): Blocca il delete delle righe dalla tabella riferita quando ci sono righe che dipendono da essa.

Questa è l'azione che viene attivata per **default**.

Generare un delete a catena (**CASCADE**): Cancella tutte le righe dipendenti dalla tabella quando la corrispondente riga è cancellate dalla tabella riferita.

Assegnare valore NULL (**SET NULL**): Assegna NULL ai valori della colonna che ha il vincolo foreign key nella tabella quando la riga corrispondente viene cancellata dalla tabella riferita.

Assegnare il valore di default (**SET DEFAULT**): Assegna il valore di default ai valori della colonna che ha il vincolo foreign key nella tabella quando la riga corrispondente viene cancellata dalla tabella riferita.

## Reazioni alla violazione on update

---

Nello standard SQL la reazione alla violazione può anche essere attivata quando i dati della tabella riferita vengono aggiornati.

Viene attivato mediante il comando **ON UPDATE** seguito da:

**CASCADE** : Le righe della tabella referente vengono impostati ai valori della tabella riferita

**SET NULL** : i valori della tabella referente vengono impostati a NULL

**SET DEFAULT** : i valori della tabella referente vengono impostati al valore di default

**NO ACTION** : rifiuta gli aggiornamenti che violino l'integrità referenziale

## ESEMPIO

---

```
CREATE TABLE Impiegati
( Codice CHAR(8) NOT NULL,
  Nome CHAR(20) NOT NULL,
  AnnoNascita INTEGER NOT NULL,
  Dipartimento CHAR(20),
  Stipendio FLOAT NOT NULL,
  Supervisore CHAR(8),
  PRIMARY KEY pk_impiagato (Codice),
  FOREIGN KEY fk_ Impiegati (Supervisore)
    REFERENCES Impiegati
    ON DELETE SET NULL
)
```

## ESEMPIO

---

```
CREATE TABLE FamiliariACarico
( Nome CHAR(20) NOT NULL,
  AnnoNascita INTEGER NOT NULL,
  GradoParentela CHAR(10) NOT NULL,
  CapoFamiglia CHAR(8) NOT NULL,
  PRIMARY KEY pk_FamiliariACarico (CapoFamiglia, Nome)
  FOREIGN KEY fk_FamiliariACarico (CapoFamiglia)
  REFERENCES Impiegati
  ON DELETE CASCADE )
```

---

# viste

## Viste

---

Le **Viste Logiche** o **Viste** o **View** possono essere definite come delle tabelle virtuali, i cui dati sono riaggregazioni dei dati contenuti nelle tabelle "fisiche" presenti nel database.

Le tabelle fisiche sono gli unici veri contenitori di dati.

Le viste non contengono dati fisicamente diversi dai dati presenti nelle tabelle, ma forniscono una diversa visione, dinamicamente aggiornata, di quegli stessi dati.

La vista appare all'utente come una normale tabella, in cui può effettuare **interrogazioni** e, limitatamente ai suoi privilegi, anche **modifiche dei dati**.

## Viste, Vantaggi

---

- Le viste **semplificano la rappresentazione dei dati**. Oltre ad assegnare un nome alla vista, la sintassi dell'istruzione **CREATE VIEW** consente di cambiare i nomi delle colonne
- Le viste possono essere anche estremamente **convenienti per svolgere una serie di query molto complesse**
- Le viste consentono di **proteggere i database**: le view ad accesso limitato possono essere utilizzate per controllare le informazioni alle quali accede un certo utente del database
- Le viste consentono inoltre di **convertire le unità di misura e creare nuovi formati**

## Viste, Limitazioni

---

- non è possibile utilizzare gli operatori booleani UNION, INTERSECT ED EXCEPT;
- Gli operatori intersect ed except possono essere realizzati mediante una select semplice. La stessa cosa non si può dire dell'operatore Union
- Non è possibile utilizzare la clausola ORDER BY

## Viste, sintassi

---

Il comando DDL che consente di definire una vista ha la seguente sintassi

```
CREATE VIEW NomeVista [ ( ListaAttributi ) ] AS SelectSQL  
[ with [ local | cascaded ] check option ]
```

I nomi delle colonne indicati nella lista attributi sono i nomi assegnati alle colonne della vista, che corrispondono ordinatamente alle colonne elencate nella select.

Se questi non sono specificati, le colonne della vista assumono gli stessi nomi di quelli della/e tabella/e a cui si riferisce.

## Create View, Esempio

---

- Creare una vista contenente la matricola, il nome, il cognome e lo stipendio degli impiegati del dipartimento di Amministrazione il cui stipendio è maggiore di 1000 euro

```
Create view ImpiegatiAmmin  
    (Matricola, Nome, Cognome, Stipendio) AS  
        Select Matr, Nome, Cognome, Stip  
        From Impiegato  
        Where Dipart = 'Amministrazione' and  
              Stipendio > 1000
```

## Create View, Esempio

---

| Veicoli | Targa | Cod_mod | Cod_cat | Cilindrata | Cod_comb. | cav.Fisc | Velocita | Posti | Imm |
|---------|-------|---------|---------|------------|-----------|----------|----------|-------|-----|
|---------|-------|---------|---------|------------|-----------|----------|----------|-------|-----|

- Creare una vista che contiene la targa e la cilindrata delle macchine con cilindrata <1500

```
Create view PiccolaCilindrata  
(PC_Targa, PC_cilindrata)  
As Select targa, cilindrata  
From Veicoli  
Where cilindrata<1500
```

- La vista ottenuta è composta da due colonne denominate PC\_targa e PC\_cilindrata corrispondenti a Targa e Cilindrata di Veicoli e alle righe della stessa tabella in cui la cilindrata è minore di 1500

## Modifica di una vista

---

Sebbene il **contenuto** di una vista sia **dinamico**, la sua **struttura** non lo è.

Se una vista è definita su una subquery

Select \* From T1

E in seguito alla tabella T1 viene aggiunta una colonna, questa nuova definizione non si estende alla vista. Ossia la vista conterrà sempre le stesse colonne che aveva prima dell'inserimento della nuova colonna in T1.

## Vista basata su due tabelle

---

### Categorie

| <u>Cod_cat</u> | Nome_cat |
|----------------|----------|
|----------------|----------|

### Veicoli

| Targa | Cod_mod | Categoria* | Cilindrata | Cod_comb | cav.Fisc | Velocita | Posti | Imm |
|-------|---------|------------|------------|----------|----------|----------|-------|-----|
|-------|---------|------------|------------|----------|----------|----------|-------|-----|

Creare una vista che descrive la targa, il codice del modello e il nome della categoria dei veicoli.

Create view A2 as

Select targa, Cod\_Modello, Nome\_Categoria

From Veicoli, Categorie

Where Categorie.cod\_cat=Veicoli.Cod\_cat

Si noti che manca la specifica dei nomi delle colonne della vista. In tal caso vengono acquisiti i nomi delle colonne della tabella madre.

## Viste di gruppo

---

Una **vista di gruppo** è una vista in cui una delle colonne è una funzione di gruppo.

In questo caso è obbligatorio assegnare un nome alla colonna della vista corrispondente alla funzione di gruppo

Modelli

| <u>Cod_Mod</u> | Nome_Mod | Cod_Fab | Cilind_Media | num_versioni |
|----------------|----------|---------|--------------|--------------|
|----------------|----------|---------|--------------|--------------|

Creare una vista che per ciascuna fabbrica riporti il numero globale delle versioni dei modelli prodotti

```
Create view A3 (cod_Fabbrica, numero_versioni) AS  
Select cod_Fabbrica, sum(num_versioni)  
From Modelli  
Group by Cod_Fabbrica
```

Oss: nelle viste di gruppo  
è necessario ridenominare  
la colonna relativa  
all'operatore aggregato



## Viste di gruppo

---

E' una vista di gruppo anche una vista che è definita in base ad una vista di gruppo

Esempio:

```
Create view A3 (cod_Fabbrica, numero_versioni) AS  
Select cod_Fabbrica, sum(num_versioni)  
From Modelli  
Group by Cod_Fabbrica
```

```
Create view A4 AS  
Select num_versioni  
From A3
```

## Eliminazione delle viste

---

- Le viste si eliminano col comando Drop View.
- Sintassi:

Drop View nome\_view {Restrict/Cascade}

**Restrict:** la vista viene eliminata solo se non è riferita nella definizione di altri oggetti

**Cascade:** oltre che essere eliminata la vista, vengono eliminate tutte le dipendenza da tale vista di altre definizioni dello schema

## Esempio

---

```
Create view A3 (cod_Fabbrica, num_versioni) AS  
Select cod_Fabbrica, sum(numero_versioni)  
From Modelli  
Group by Cod_Fabbrica
```

```
Create view A4 AS  
Select num_versioni  
From A3
```

L'istruzione **Drop View A3 Cascade** elimina oltre che la vista A3, anche la vista A4 che dipende da essa.

Invece **Drop View A3 Restrict** impedisce la cancellazione di A3, finchè è presente A4, che dipende da essa

# TABELLE INIZIALIZZATE E TABELLE CALCOLATE

---

- Tabelle inizializzate:

```
CREATE TABLE Nome EspressioneSELECT  
CREATE TABLE Supervisori  
    SELECT Codice, Nome, Qualifica, Stipendio  
    FROM Impiegati  
    WHERE Supervisore IS NULL
```

- Tabelle calcolate (viste):

```
CREATE VIEW Nome [(Attributo {, Attributo})]  
    AS EspressioneSELECT [WITH CHECK OPTION];  
CREATE VIEW Supervisori  
    AS      SELECT Codice, Nome, Qual., Stip.  
            FROM Impiegati  
            WHERE Supervisore IS NULL
```

## VISTE MODIFICABILI

---

- Le tabelle delle viste si interrogano come le altre, ma in generale non si possono modificare.
- Deve esistere una corrispondenza biunivoca fra le righe della vista e le righe di una tabella di base, ovvero:
  - 1) SELECT senza DISTINCT e solo di attributi
  - 2) FROM una sola tabella modificabile
  - 3) WHERE senza SottoSelect
  - 4) GROUP BY e HAVING non sono presenti nella definizione.
- Possono esistere anche delle restrizioni su SELECT su viste definite usando GROUP BY.

## Aggiornamento delle VIEW

---

Le operazioni INSERT/UPDATE/DELETE sulle VIEW non erano  
permesse nelle prime edizioni di SQL

- I nuovi DBMS permettono di farlo con certe limitazioni dovute alla definizione della VIEW stessa
- Ha senso aggiornare una VIEW?

Dopotutto si potrebbe aggiornare la tabella di base direttamente ...

## Aggiornamento delle VIEW

---

- ... utile nel caso di accesso dati controllato
- Esempio:  
*Impiegato( Nome, Cognome, Dipart, Ufficio, Stipendio)*
- Il personale della segreteria non può accedere ai dati sullo stipendio ma può modificare gli altri campi della tabella, aggiungere e/o cancellare tuple
- Si può controllare l'accesso tramite la definizione della VIEW:  
*CREATE VIEW Impiegato2 AS  
SELECT Nome, Cognome, Dipart, Ufficio  
FROM Impiegato  
INSERT INTO Impiegato2 VALUES (...)*
- Stipendio verrà inizializzato a Null
- Se Null non è permesso per Stipendio l'operazione fallisce

## Aggiornamento VIEW (2)

---

- Immaginiamo la seguente VIEW:

```
CREATE VIEW ImpiegatoRossi
```

```
AS
```

```
SELECT *
```

```
FROM Impiegato
```

```
WHERE Cognome='Rossi'
```

- La seguente operazione ha senso:

```
INSERT INTO ImpiegatoRossi (...'Rossi',...)
```

## Aggiornamento VIEW (2)

---

```
CREATE VIEW ImpiegatoRossi  
AS  
SELECT *  
FROM Impiegato  
WHERE Cognome='Rossi'
```

- Ma che succede nel caso di:

```
INSERT INTO ImpiegatoRossi (...'Bianchi',...)
```

- In genere è permesso, finisce nella tabella base ma non è visibile dalla VIEW

## With check option 1

---

- L'opzione **With Check Option** messa alla fine della definizione della vista assicura che le operazioni di inserimento e di modifica dei dati effettuate utilizzando la vista soddisfino la clausola Where della subquery.

```
CREATE VIEW ImpiegatoRossi AS  
    SELECT * FROM Impiegato  
    WHERE Cognome='Rossi'  
    WITH CHECK OPTION
```

- Adesso l'insert con 'Bianchi' fallisce, quella con 'Rossi' viene invece eseguita.

## Local, Cascaded

---

Supponiamo che una **vista V1** sia definita in termini di un'altra vista **V2**. Se si crea V1 specificando la clausola **WITH CHECK OPTION**, il DBMS verifica che la nuova tupla t inserita soddisfi **sia la definizione di V1 che quella di V2** (e di tutte le altre eventuali viste da cui V1 dipende), **indipendentemente** dal fatto che V2 sia stata a sua volta definita **WITH CHECK OPTION**

Questo comportamento di default è equivalente a definire V1  
**WITH CASCADED CHECK OPTION**

Lo si può alterare definendo V1  
**WITH LOCAL CHECK OPTION**

Ora il DBMS verifica solo che t soddisfi la specifica di V1 e quelle di tutte e **sole le viste da cui V1 dipende per cui è stata specificata la clausola WITH CHECK OPTION**

## Esempio

---

| Veicoli | Targa | Cod_mod | Categoria | Cilindrata | Cod_comb. | cav.Fisc | Velocita | Posti | Imm |
|---------|-------|---------|-----------|------------|-----------|----------|----------|-------|-----|
|---------|-------|---------|-----------|------------|-----------|----------|----------|-------|-----|

| Categorie | Cod_cat | Nome_cat |
|-----------|---------|----------|
|-----------|---------|----------|

La seguente vista è aggiornabile

```
Create view A1  
(A1_Targa, A1_cilindrata)  
As Select targa, cilindrata  
From Veicoli  
Where cilindrata<1500
```

Quest'altra (in generale) invece non lo è

```
Create view A2 as  
Select targa, Cod_Modello, Nome_Categoria  
From Veicoli, Categorie  
Where Categorie.cod_cat=Veicoli.Cod_cat
```

Due tavelle

## Vantaggi delle viste: facilitazione nell'accesso ai dati

---

- In generale uno dei requisiti per la progettazione di un database relazionale è la **normalizzazione dei dati**.
- Sebbene la forma normalizzata del database permette una corretta modellazione della realtà che il DB rappresenta, a volte dal punto di vista dell'utente comporta una maggior<sup>e</sup> difficoltà di comprensione rispetto a una rappresentazione non normalizzata.
- Le viste permettono di fornire all'utente i dati in una forma più intuitiva.

## Vantaggi delle viste: diverse visioni dei dati

---

Esistono dei dati che sono presenti nelle tabelle del database, che sono **poco significativi per l'utente**, e altri che **devono essere nascosti all'utente** (esempio: lo stipendio di un dipendente, la password di un account etc.).

L'uso delle viste da parte dell'utente permette di **limitare il suo accesso ai dati del database**, eliminando quelli non interessanti per lui e quelli che devono essere tenuti nascosti.

L'uso delle viste può essere considerato come una **tecnica per assicurare la sicurezza dei dati**

## Vantaggi delle Viste: Indipendenza Logica

---

Un vantaggio delle viste riguarda **l'indipendenza logica** delle applicazioni e delle operazioni eseguite dagli utenti rispetto alla struttura logica dei dati.

Ciò significa che è possibile poter operare modifiche allo schema senza dover apportare modifiche alle applicazioni che utilizzano il database.

## UTILITÀ DELLE VISTE

---

- Per nascondere certe modifiche all'organizzazione logica dei dati (indipendenza logica)
- Per offrire visioni diverse degli stessi dati senza ricorrere a duplicazioni
- Per rendere più semplici, o per rendere possibili, alcune interrogazioni

## Un'interrogazione non standard

---

Il dipartimento che impiega il massimo budget in stipendi dei dipendenti

Select Dipart

from Impiegato

group by Dipart

having sum(Stipendio) >= all

(select sum(Stipendio)

from Impiegato

group by Dipart)

## Soluzione con le viste

---

```
CREATE VIEW BudgetStipendi(Dipartimento,TotaleStipendi) AS  
SELECT Dipart, sum(Stipendio)  
FROM Impiegato  
GROUP BY Dipart
```

```
SELECT Dipartimento  
FROM BudgetStipendi  
WHERE  
    TotaleStipendi =(SELECT max(TotaleStipendi)  
                    FROM BudgetStipendi)
```

La vista è utilizzata come una normale tabella. Di solito vengono utilizzate in questo modo quando si devono usare a catena due diversi operatori aggregati (la massima somma, il minimo numero...)

## Ancora sulle viste

---

Calcolare la media del numero degli uffici distinti presenti in ogni dipartimento

Interrogazione scorretta

```
select avg(count(distinct Ufficio))  
from Impiegato  
group by Dipart
```

Due operatori  
aggregati annidati

Con una vista

```
create view UfficiDipart (NomeDip,NroUffici) as  
select Dipart, count(distinct Ufficio)  
from Impiegato  
group by Dipart
```

```
select avg(NroUffici)  
from UfficiDipart
```

---

# Procedure e trigger

## CREATE PROCEDURE/FUNCTION

---

```
CREATE FUNCTION contaStudenti IS
    DECLARE
        tot INTEGER;
    BEGIN
        SELECT COUNT(*) INTO tot FROM STUDENTI;
        RETURN (tot);
    END
```

# Trigger

---

- Un **trigger** definisce un'azione che il database deve attivare automaticamente quando si verifica (nel database) un determinato evento.
- Possono essere utilizzati:
  - per **migliorare l'integrità referenziale dichiarativa**
  - per **imporre regole complesse** legate all'attività del database
  - per **effettuare revisioni** sulle modifiche dei dati.

# Trigger

---

- L'esecuzione dei trigger è quindi trasparente all'utente.
- I trigger vengono eseguiti automaticamente dal database quando specifici tipi di comandi (**Eventi**) di manipolazione dei dati vengono eseguiti su specifiche tabelle.
- Tali comandi comprendono i comandi DML **insert, update e delete**, ma gli ultimi DBMS prevedono anche trigger su istruzioni DDL come **Create View** ecc.
- Anche gli aggiornamenti di specifiche colonne possono essere utilizzati come trigger di eventi.

## Trigger a livello di riga

---

- I trigger a livello di riga vengono eseguiti una volta per ciascuna riga modificata in una transazione; vengono spesso utilizzati in applicazioni di revisione dei dati e si rivelano utili per **operazioni di audit dei dati** e per **mantenere sincronizzati i dati distribuiti**.
- Per creare un trigger a livello di riga occorre specificare la clausola  
**FOR EACH ROW**
- nell'istruzione `create trigger`.

## Trigger a livello di istruzione

---

- I trigger a livello di istruzione vengono eseguiti **una sola volta per ciascuna transazione**, indipendentemente dal numero di righe che vengono modificate (quindi anche se, ad esempio, in una tabella vengono inserite 100 righe, il trigger verrà eseguito solo una volta).
- Vengono pertanto utilizzati per **attività correlate ai dati**; vengono utilizzati di solito per **imporre misure aggiuntive di sicurezza sui tipi di transazione che possono essere eseguiti su una tabella**.
- E' il tipo di trigger **predefinito** nel comando create trigger (ossia non occorre specificare che è un trigger al livello di istruzione).

## Trigger, struttura

---

- I trigger si basano sul paradigma evento-condizione-azione (ECA).
- L'istruzione **Create Trigger** seguita dal **nome** assegnato al trigger
- **Tipo di trigger**, Before/After
- **Evento che scatena il trigger** Insert/Delete/Update
- **[For each row]**, Se si vuole specificare trigger al livello di riga (altrimenti nulla per trigger al livello di istruzione)
- Specificare a quale **tabella** si applica
- **Condizione** che si deve verificare perché il trigger sia eseguito
- **Azione**, definita dal codice da eseguire se si verifica la condizione

## Trigger: sintassi

---

```
create trigger <NomeTrigger>
  Tipo di trigger Evento {, Evento}
  ON <TabellaTarget>
  for each row
  [when <Predicato SQL>]
  Blocco PL/SQL
```

Tipo di trigger Evento: before o after

Evento: insert, update, delete

for each row specifica la granularità.

In assenza di questa clausola si intende per ogni istruzione

## ESEMPIO DI TRIGGER

---

```
CREATE TRIGGER ControlloStipendio
  BEFORE INSERT ON Impiegati
DECLARE
  StipendioMedio FLOAT
BEGIN
  SELECT avg(Stipendio) INTO StipendioMedio
    FROM Impiegati
   WHERE Dipartimento = :new.Dipartimento;
  IF :new.Stipendio > 2 * StipendioMedio
  THEN RAISE_APPL_ERR.(-2061, 'Stipendio alto')
  END IF;
END;
```

## Tipi di Trigger

---

- **BEFORE e AFTER**: i trigger possono essere eseguiti prima o dopo l'utilizzo dei comandi `insert`, `update` e `delete`; all'interno del trigger è possibile fare riferimento ai vecchi e nuovi valori coinvolti nella transazione.
- Occorre utilizzare la clausola  
**BEFORE/AFTER <tipo di evento> (insert, delete, update).**
- Se si tratta di un trigger **BEFORE UPDATE**:
  - per valori **vecchi** intendiamo i valori che sono nella tabella e che vogliamo modificare
  - per **nuovi** quelli che vogliamo inserire al posto dei vecchi.
- Se si tratta di un trigger **AFTER UPDATE**:
  - per **vecchi** intendiamo quelli che c'erano prima dell'update
  - Per **nuovi** quelli presenti nella tabella alla fine della modifica.

## Trigger Attivi e Passivi

---

- Un **trigger** è attivo quando, in corrispondenza di certi eventi, modifica lo stato della base di dati.
- Un **trigger** è passivo se serve a provocare il fallimento della transazione corrente sotto certe condizioni.

## Tipi di Trigger

---

**INSTEAD OF:** per specificare che cosa fare invece di eseguire le azioni che hanno attivato il trigger.

- Ad esempio, è possibile utilizzare un trigger **INSTEAD OF** per reindirizzare le INSERT in una tabella verso una tabella differente o per aggiornare con update più tabelle che siano parte di una vista.
- I trigger **instead-of** possono essere definiti su viste (relazionali od oggetto).
- I trigger instead-of devono essere a livello di riga.

# I TRIGGER

---

- Proprietà essenziale dei trigger: terminazione
- Utilità dei trigger
  - Trattare vincoli non esprimibili nello schema
  - Attivare automaticamente azioni sulla base di dati quando si verificano certe condizioni

---

# Controllo degli accessi

## Controllo degli accessi

---

- Ogni componente dello schema (risorsa) può essere protetta (tabelle, attributi, viste, domini, ecc.)
- Il possessore della risorsa (colui che la crea) assegna dei privilegi agli altri utenti
- Un utente predefinito (\_system) rappresenta l'amministratore della base di dati ed ha completo accesso alle risorse
- Ogni privilegio è caratterizzato da:
  - la risorsa a cui si riferisce
  - l'utente che concede il privilegio
  - l'utente che riceve il privilegio
  - l'azione che viene permessa sulla risorsa
  - se il privilegio può esser trasmesso o meno ad altri utenti

## CONTROLLO DEGLI ACCESSI - Tipi di privilegi

---

- Tipi di privilegi:
  - SELECT: lettura di dati
  - INSERT [(Attributi)]: inserire record (con valori non nulli per gli attributi)
  - DELETE: cancellazione di record
  - UPDATE [(Attributi)]: modificare record (o solo gli attributi)
  - REFERENCES [(Attributi)]: definire chiavi esterne in altre tavelle che riferiscono gli attributi.
- WITH GRANT OPTION: si possono trasferire i privilegi ad altri utenti.

## CONTROLLO DEGLI ACCESSI

---

- Chi crea lo schema della BD è l'unico che può fare CREATE, ALTER e DROP
- Chi crea una tabella stabilisce i modi in cui altri possono farne uso:
  - GRANT Privilegi ON Oggetto TO Utenti [ WITH GRANT OPTION ]

## grant e revoke

---

- Per concedere un privilegio ad un utente:

grant < Privileges | all privileges > on Resource to  
Users [ with grant option ]

grant option specifica se ha il privilegio di propagare il privilegio ad altri utenti

Es.: grant select on Department to Stefano

Per revocare il privilegio:

revoke Privileges on Resource from Users [ restrict | cascade ]

## Opzioni di grant e revoke

---

- La revoca deve essere fatta dall'utente che aveva concesso I privilegi
  - restrict (di default) specifica che il comando non deve essere eseguito qualora la revoca dei privilegi all'utente comporti qualche altra revoca (dovuta ad un precedente grant option)
  - cascade invece forza l'esecuzione del comando
- Attenzione alle reazioni a catena

## CONTROLLO DEGLI ACCESSI (cont.)

---

- Chi definisce una tabella o una VIEW ottiene automaticamente tutti i privilegi su di esse, ed è l'unico che può fare un DROP e può autorizzare altri ad usarla con GRANT.
- Nel caso di viste, il "creatore" ha i privilegi che ha sulle tabelle usate nella definizione.
- Le autorizzazioni si annullano con il comando:
  - REVOKE [ GRANT OPTION FOR ] Privilegi ON Oggetto FROM Utenti [ CASCADE ]
  - Quando si toglie un privilegio a U, lo si toglie anche a tutti coloro che lo hanno avuto solo da U.

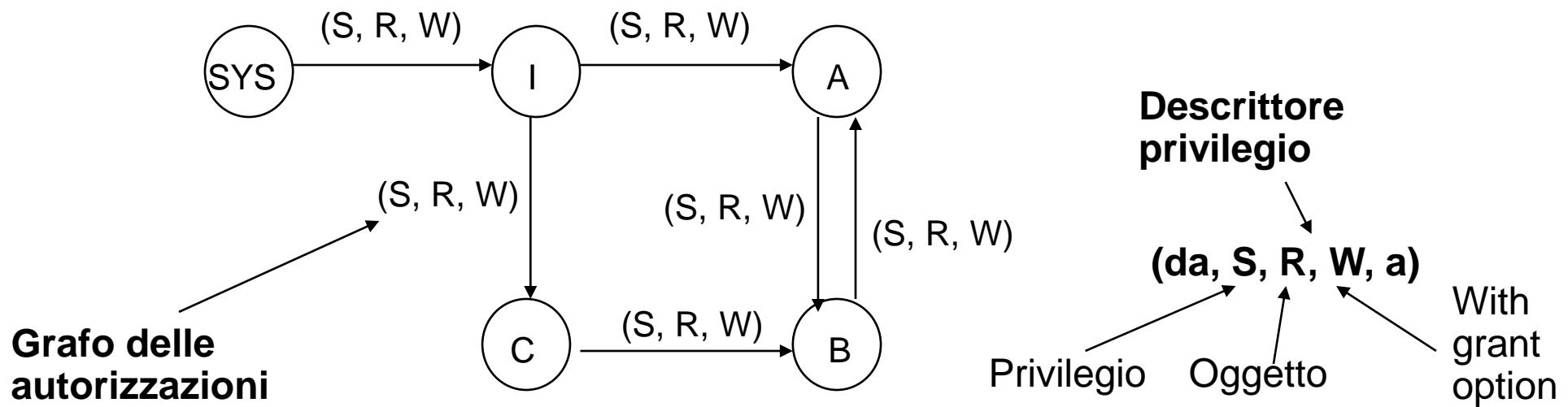
## ESEMPI DI GRANT

---

- GRANT INSERT, SELECT ON Esami TO Tizio .
- GRANT DELETE ON Esami TO Capo WITH GRANT OPTION
  - Capo può cancellare record e autorizzare altri a farlo.
- GRANT UPDATE (voto) ON Esami TO Sicuro
  - Sicuro può modificare solo il voto degli esami.
- GRANT SELECT, INSERT ON VistaEsamiBD1 TO Albano
  - Albano può interrogare e modificare solo i suoi esami.

# GRAFO DELLE AUTORIZZAZIONI

- L'utente I ha creato la tabella R e innesca la seguente successione di eventi:
  - I: GRANT SELECT ON R TO A WITH GRANT OPTION
  - A: GRANT SELECT ON R TO B WITH GRANT OPTION
  - B: GRANT SELECT ON R TO A WITH GRANT OPTION
  - I: GRANT SELECT ON R TO C WITH GRANT OPTION
  - C: GRANT SELECT ON R TO B WITH GRANT OPTION



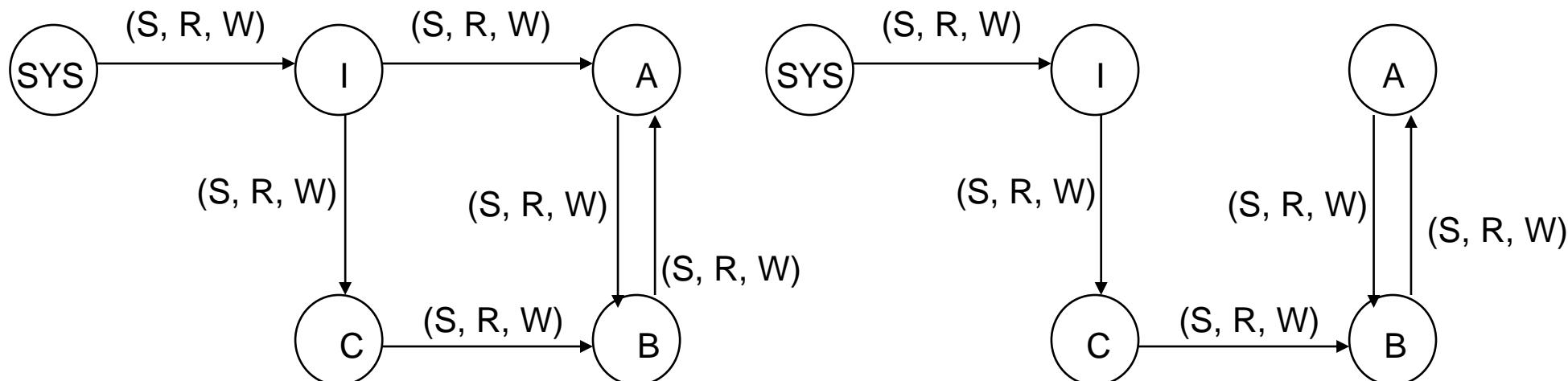
# GRAFO DELLE AUTORIZZAZIONI: PROPRIETA'

---

- Se un nodo N ha un arco uscente con un privilegio, allora esiste un cammino da SYSTEM a N con ogni arco etichettato dallo stesso privilegio + WGO.
- Effetto del REVOKE, ad es.

I: REVOKE SELECT ON R FROM A CASCADE

- e poi I: REVOKE SELECT ON R FROM C CASCADE



---

# Indice e catalogo

## CREAZIONE DI INDICI

---

- Cosa sono e a cosa servono
- Non è un comando standard dell'SQL e quindi ci sono differenze nei vari sistemi
  - CREATE INDEX NomeIdx ON Tabella(Attributi)
  - CREATE INDEX NomeIdx ON Tabella  
WITH STRUCTURE = BTREE, KEY = (Attributi)
  - DROP INDEX NomeIdx

## CATALOGO (DEI METADATI)

---

- Alcuni esempi di tavole, delle quali si mostrano solo alcuni attributi, sono:
  - Tabella delle password:
    - PASSWORD(username, password)
  - Tabella delle basi di dati:
    - SYSDB(dbname, creator, dbpath, remarks)
  - Tabella delle tavole (type = view or table):
    - SYSTABLES(name, creator, type, colcount, filename, remarks)

## CATALOGO (cont.)

---

- Alcuni esempi di tavole, delle quali si mostrano solo alcuni attributi, sono:
  - Tabella degli attributi:
    - `SYSCOLUMNS(name, tbname, tbcreator, colno, coltype, lenght, default, remarks)`
  - Tabella degli indici:
    - `SYSINDEXES(name, tbname, creator, uniquerule, colcount)`
  - e altre ancora sulle viste, vincoli, autorizzazioni, etc. (una decina).

## RIEPILOGO

---

- DDL consente la definizione di tabelle, viste e indici. Le tabelle si possono modificare aggiungendo o togliendo attributi e vincoli.
- Le viste si possono interrogare come ogni altra tabella, ma in generale non consentono modifiche dei dati.
- I comandi GRANT / REVOKE + viste offrono ampie possibilità di controllo degli usi dei dati.

## RIEPILOGO

---

- SQL consente di dichiarare molti tipi di vincoli, oltre a quelli fondamentali di chiave e referenziale.
- Oltre alle tabelle fanno parte dello schema le procedure e i trigger.
- La padronanza di tutti questi meccanismi -- e di altri che riguardano aspetti fisici, affidabilità, sicurezza -- richiede una professionalità specifica (DBA).

---

# DBMS

---

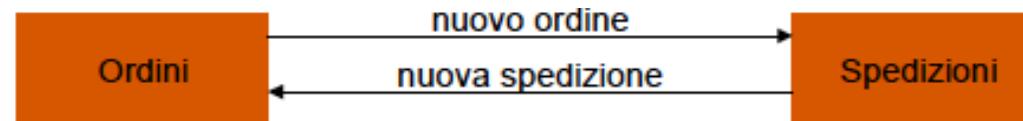
# **PARTE I**

- un DBMS è un sistema software che gestisce grandi quantità di dati persistenti e condivisi
- La gestione di **grandi quantità di dati** richiede particolare attenzione ai problemi di **efficienza** (ottimizzazione delle richieste, ma non solo!)
- La **persistenza** e la **condivisione** richiedono che un DBMS fornisca dei meccanismi per garantire l'**affidabilità** dei dati (fault tolerance), per il **controllo degli accessi** e per il **controllo della concorrenza**
- Diverse altre funzionalità vengono messe a disposizione per motivi di **efficacia**, ovvero per semplificare la descrizione dei dati, lo sviluppo delle applicazioni, l'amministrazione di un DB, ecc.

## Condivisione dei dati

---

- La **gestione integrata** e la **condivisione dei dati** permettono di evitare ripetizioni (ridondanza dovuta a copie multiple dello stesso dato), e quindi un inutile spreco di risorse (memoria)
- Inoltre, la **ridondanza** può dar luogo a problemi di **inconsistenza** delle copie e, in ogni caso, comporta la necessità di **propagare** le modifiche, con un ulteriore spreco di risorse (CPU e rete)
- Esempio: il settore Ordini di un'azienda manifatturiera memorizza i propri dati in un file, non condiviso con gli altri settori aziendali. Ogni volta che arriva un ordine, i dati relativi devono essere trasmessi al settore Spedizioni, affinché l'ordine possa essere evaso. A spedizione eseguita, i dati relativi devono essere ritrasmessi al settore Ordini



## Il modello dei dati

---

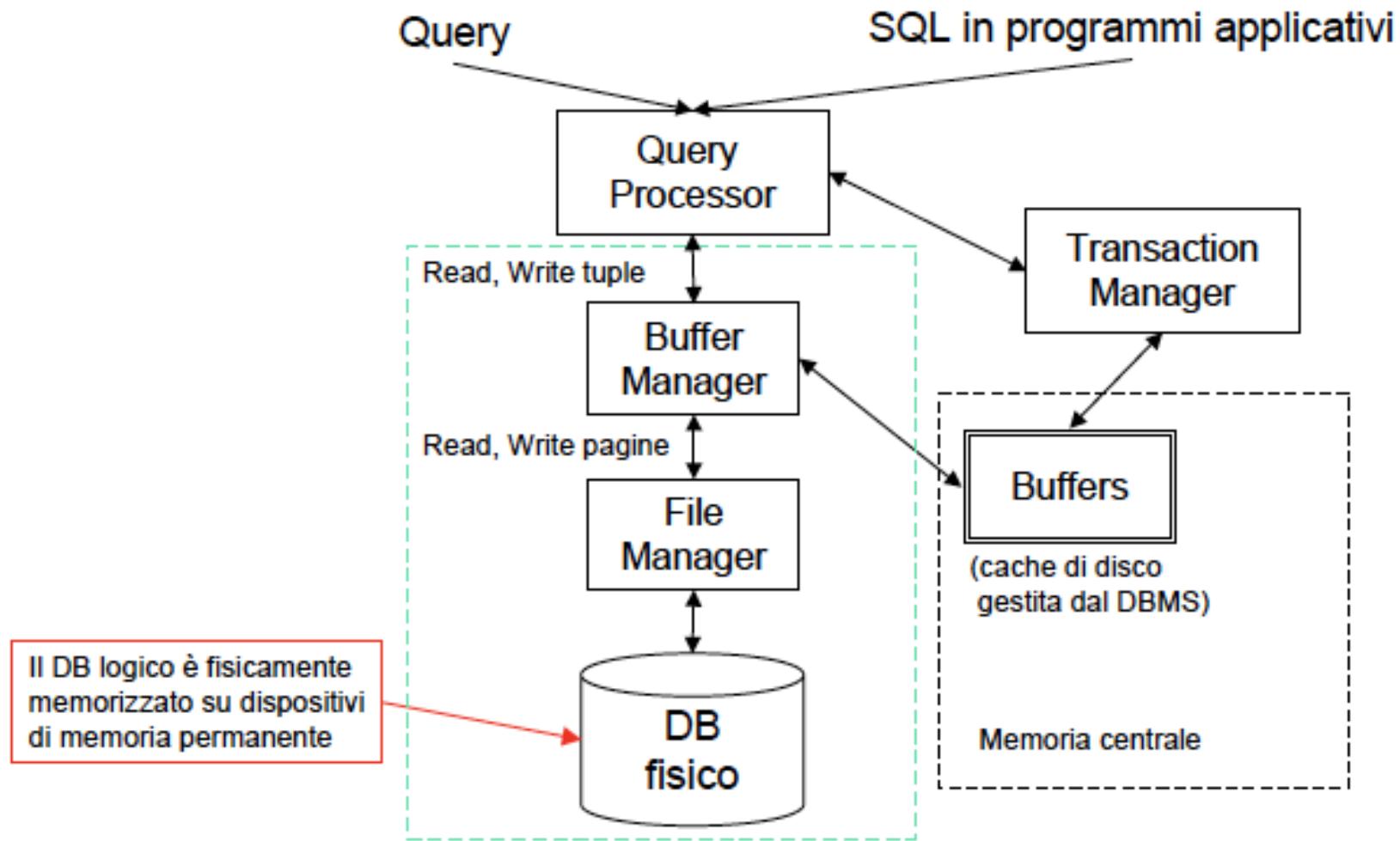
- Dal punto di vista utente un DB è visto come una collezione di dati che modellano una certa porzione della realtà di interesse
- L'**astrazione logica** con cui i dati vengono resi disponibili all'utente definisce un **modello dei dati**; più precisamente:
  - un modello dei dati è una collezione di concetti che vengono utilizzati per descrivere i dati, le loro associazioni/relazioni, e i vincoli che questi devono rispettare
- Un ruolo di primaria importanza nella definizione di un modello dei dati è svolto dai **meccanismi che possono essere usati per strutturare i dati** (cfr. i costruttori di tipo in un linguaggio di programmazione)
- Ad es. esistono modelli in cui i dati sono descritti (solo) sotto forma di alberi (modello **gerarchico**), di grafi (modello **reticolare**), di oggetti complessi (modello **a oggetti**), di relazioni (modello **relazionale**)

## Indipendenza fisica e logica

---

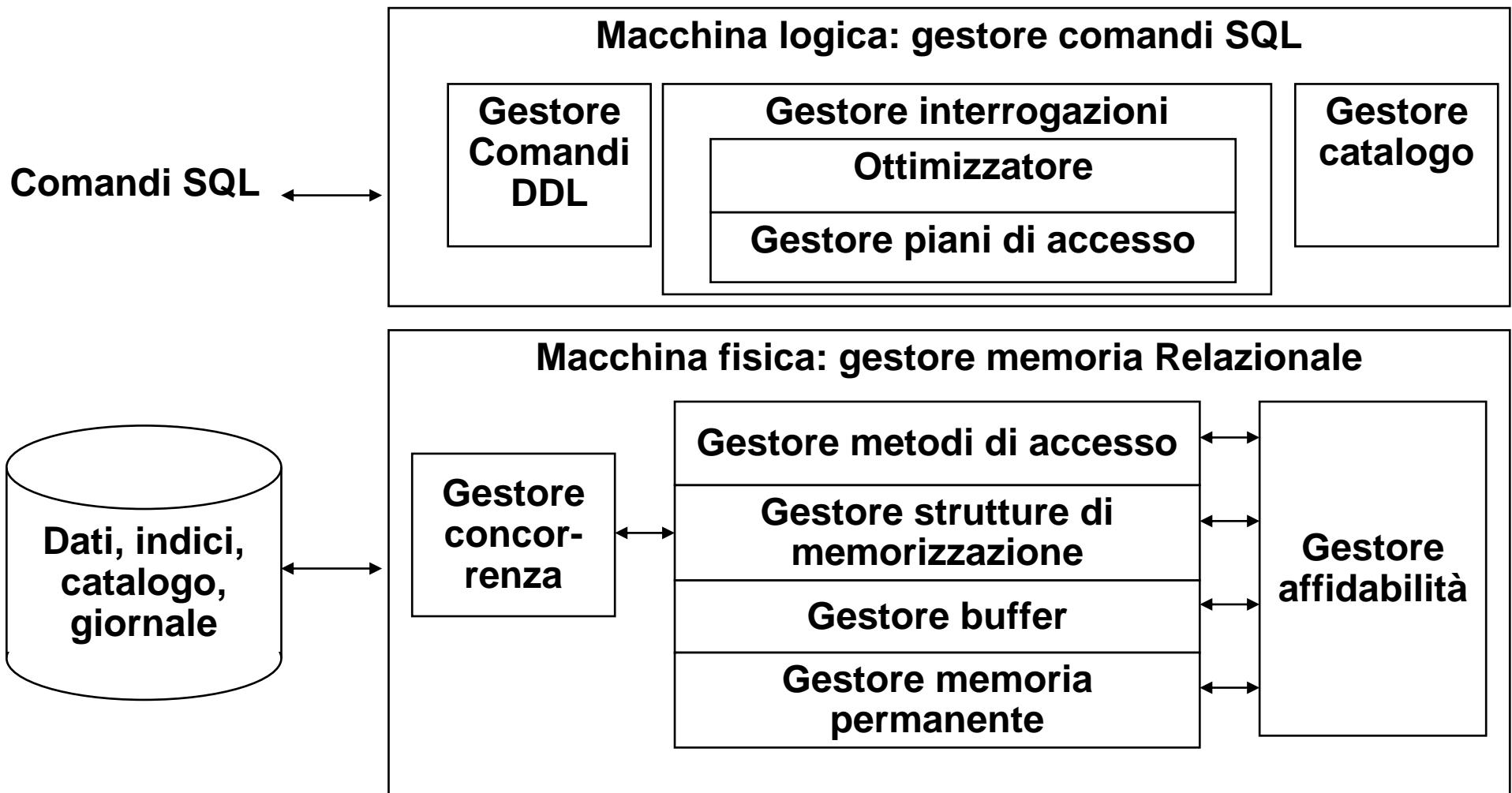
- Tra gli obiettivi di un DBMS vi sono quelli di fornire caratteristiche di:
- **Indipendenza fisica**
  - L'**organizzazione fisica** dei dati dipende da considerazioni legate all'efficienza delle organizzazioni adottate. La riorganizzazione fisica dei dati **non deve comportare effetti collaterali sui programmi applicativi**
- **Indipendenza logica**
  - permette di accedere ai dati logici indipendentemente dalla loro rappresentazione fisica

# L'architettura (semplificata) di un DBMS



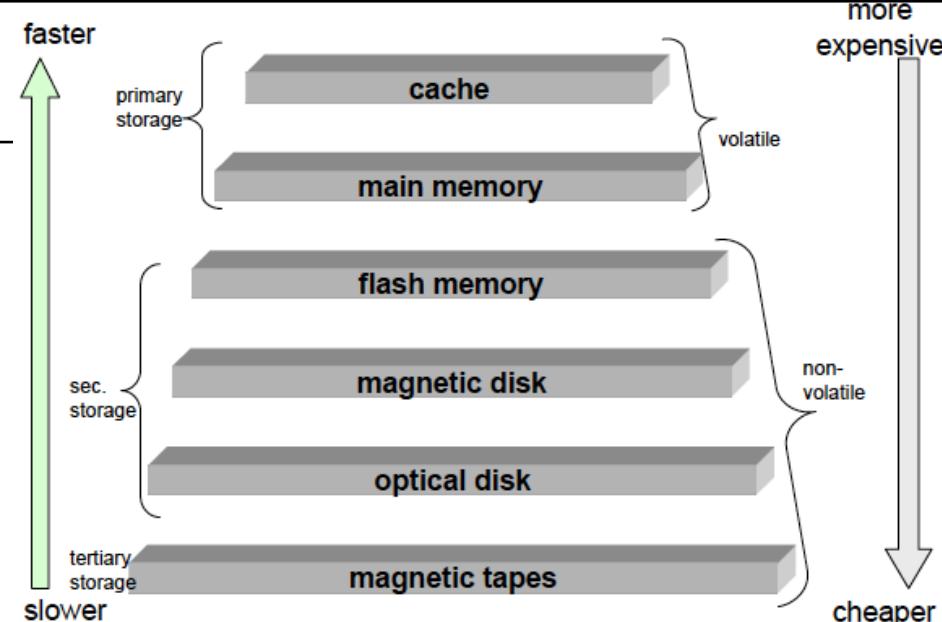
# ARCHITETTURA DEI DBMS

---



# Gerarchia delle memorie

- La memoria di un sistema di calcolo è organizzata in una gerarchia.
- Al livello più alto memorie di piccola dimensione, molto veloci, costose; scendendo lungo la gerarchia la dimensione aumenta, diminuiscono la velocità e il costo
- Prestazioni di una memoria:
  - dato un indirizzo di memoria, le prestazioni si misurano in termini di tempo di accesso, determinato dalla somma della
    - latenza (tempo necessario per accedere al primo byte), e del
    - tempo di trasferimento (tempo necessario per muovere i dati)



$$\text{Tempo di accesso} = \text{lazienza} + \frac{\text{dimensione dati da trasferire}}{\text{velocità di trasferimento}}$$

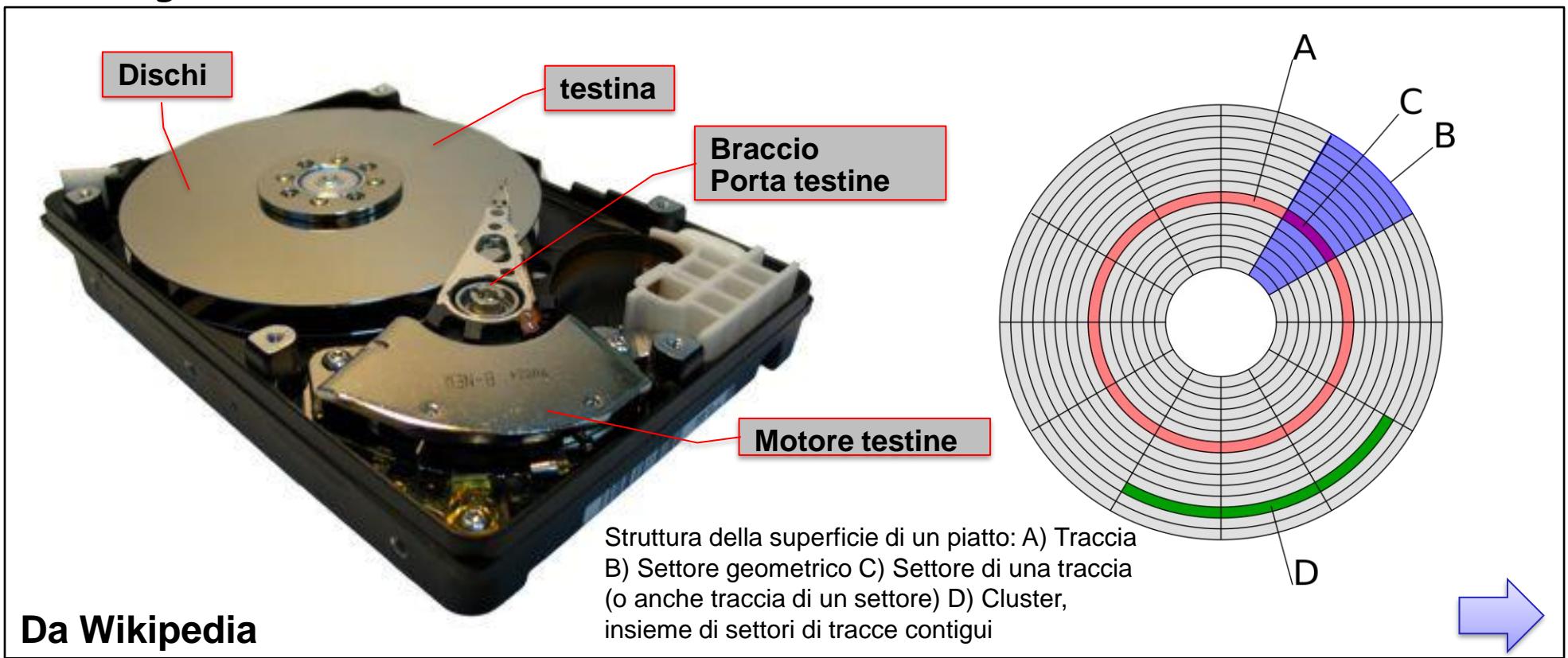
## Implicazioni per i DBMS

---

- Un DB, a causa della sua dimensione, risiede normalmente su dischi (ed eventualmente anche su altri tipi di dispositivi)
- I dati devono essere trasferiti in memoria centrale per essere elaborati dal DBMS
- Il trasferimento non avviene in termini di singole tuple, bensì di blocchi (o pagine, termine comunemente usato quando i dati sono in memoria)
- Poiché spesso le operazioni di I/O costituiscono il collo di bottiglia del sistema, si rende necessario ottimizzare l'implementazione fisica del DB, attraverso:
  - Organizzazione efficiente delle tuple su disco
  - Strutture di accesso efficienti
  - Gestione efficiente dei buffer in memoria
  - Strategie di esecuzione efficienti per le query

# Gli hard disk

Un hard disk (HD) è un dispositivo elettro-meccanico per la conservazione di informazioni sotto forma magnetica, su supporto rotante a forma di piatto su cui agiscono delle testine di lettura/scrittura



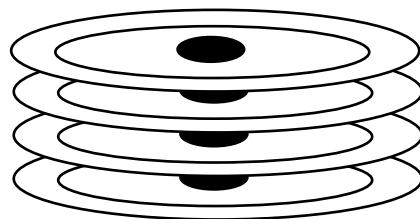
Da Wikipedia

# Il meccanismo del disk drive

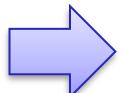
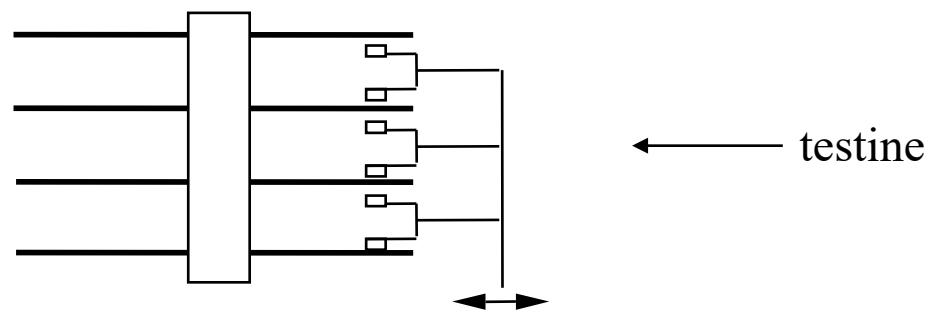
---

Include organi di registrazione, di posizionamento e di rotazione.

Un'unità a dischi contiene una pila di dischi metallici che ruota a velocità costante ed alcune testine di lettura che si muovono radialmente al disco



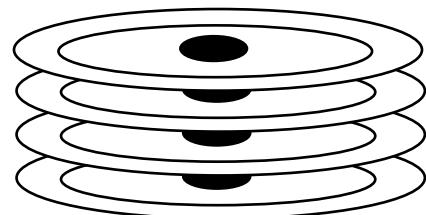
Pacco di dischi  
Cilindro  
Traccia



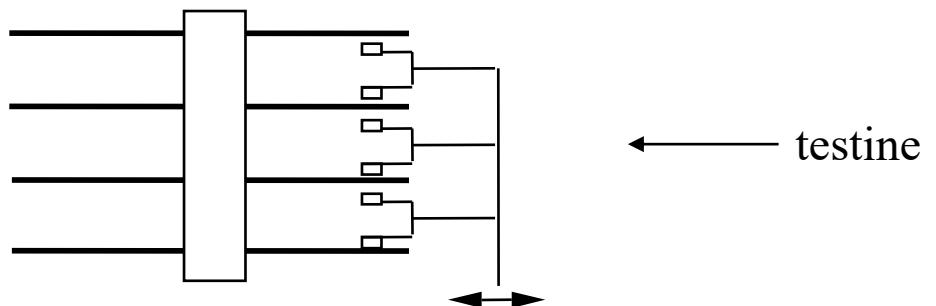
# MEMORIE A DISCO

---

- Un'unità a dischi contiene una pila di dischi metallici che ruota a velocità costante ed alcune testine di lettura che si muovono radialmente al disco



Pacco di dischi  
Cilindro  
Traccia



- Una **traccia** è organizzata in settori di dimensione fissa; i settori sono raggruppati logicamente in **blocchi**, che sono l'unità di trasferimento.
- Trasferire un blocco richiede un tempo di posizionamento delle testine, un tempo di latenza rotazionale e tempo per il trasferimento (trascurabile)
  - IBM 3380 (1980) : 2Gb, 16 ms, 8.3 ms, 0.8 ms/2.4Kb
  - IBM Ultrastar 36Z15 (2001): 36GB, 4.2 ms, 2 ms, 0.02 ms/Kb

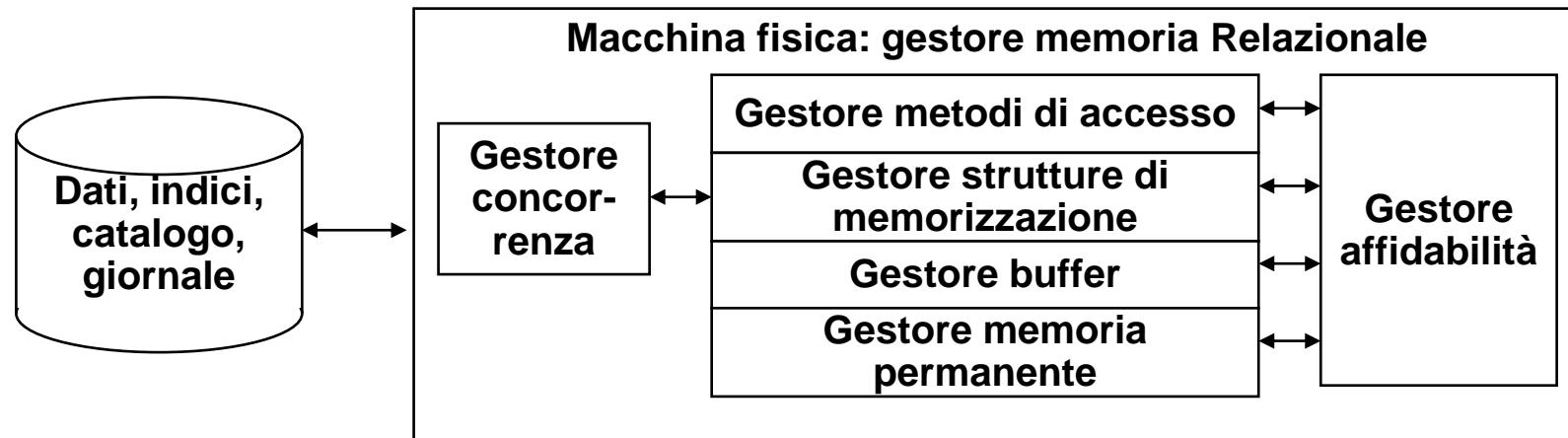
- Un blocco (o **pagina**) è una sequenza contigua di settori su una traccia, e costituisce l'unità di I/O per il trasferimento di dati da/per la memoria principale
- La dimensione tipica di una pagina è di qualche KB (4 -64 KB)
- Pagine piccole comportano un maggior numero di operazioni di I/O
- Pagine grandi tendono ad aumentare la frammentazione interna (pagine parzialmente riempite) e richiedono più spazio in memoria per essere caricate
- Il tempo di trasferimento di una pagina ( $T_t$ ) da disco a memoria centrale dipende dalla dimensione della pagina ( $P$ ) e dal transfer rate ( $Tr$ )
- Esempio: con un transfer rate di 21.56 MB/sec e  $P = 4$  KB si ha  
 $T_t = 0.18$  ms, con  $P = 64$  KB si ha  $T_t = 2.9$  ms

# L'IMPORTANZA DELLA LOCALITÀ

---

- Per leggere un file da un MB servono (IBM Ultrastar 36Z15 ):
  - 0,027 secondi se memorizzato in settori consecutivi
  - 0,8 secondi se memorizzato in blocchi da 16 settori ciascuno (8KB) distribuiti a caso ( $128 * (4,2 + 2 + 0,16)$ )
  - 12,7 secondi se memorizzato in blocchi da 1 settore ciascuno, distribuiti a caso ( $2048 * (4,2 + 2 + 0,01)$ )

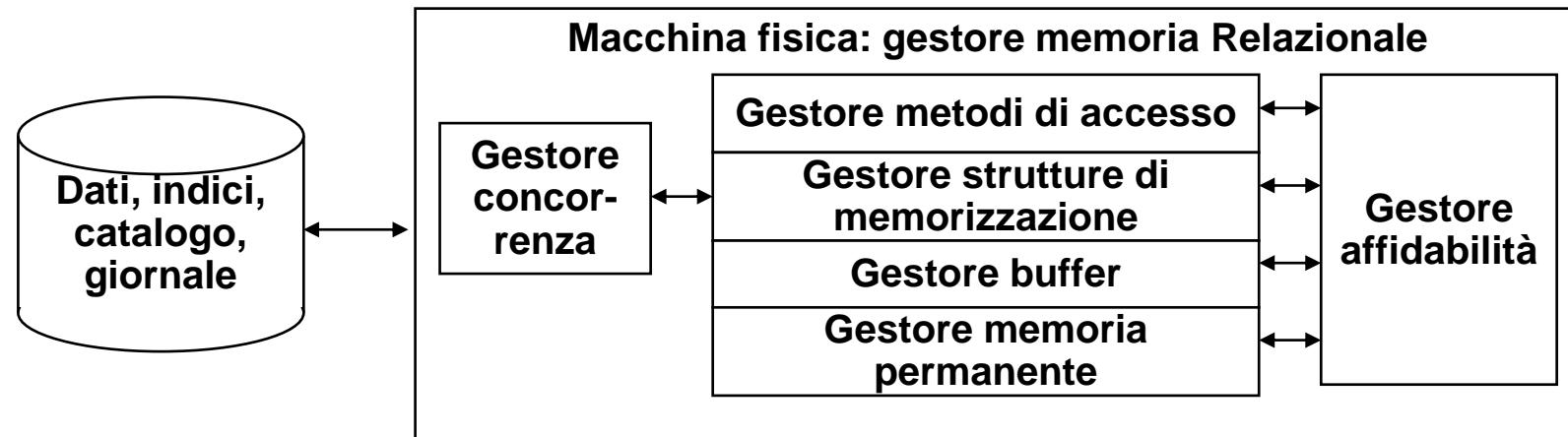
# GESTORE MEMORIA PERMANENTE E GESTORE DEL BUFFER



## • GESTORE MEMORIA PERMANENTE

- Fornisce un'astrazione della memoria permanente in termini di insiemi di file logici di pagine fisiche di registrazioni (blocchi), nascondendo le caratteristiche dei dischi e del sistema operativo.

# GESTORE MEMORIA PERMANENTE E GESTORE DEL BUFFER



## GESTORE DEL BUFFER

- Si preoccupa del trasferimento delle pagine tra la memoria temporanea e la memoria permanente, offrendo agli altri livelli una visione della memoria permanente come un insieme di pagine utilizzabili in memoria temporanea, astraendo da quando esse vengano trasferite dalla memoria permanente al buffer e viceversa

# GESTORE DEL BUFFER: AREA DELLE PAGINE

- Pin count: quando una pagina viene richiesta/rilasciata il contatore viene incrementato/decrementato,
- dirty/non dirty: la componente che ha eseguito la modifica chiede al buffer di aggiornare la proprietà dirty/nondirty della pagina modificata

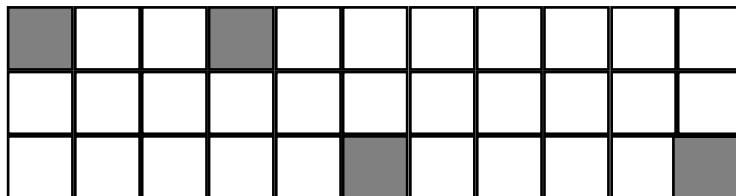
**getAndPinPage(205):**  
richiede la pagina al Buffer Manager e vi pone un pin ("spillo"), ad indicarne l'uso

tabella associativa

|     |   |
|-----|---|
| 127 | 4 |
| 100 | 1 |
| 205 | 3 |
| 35  | 2 |

<idPage, posBuffer>

Buffer Pool (Memoria Temporanea)



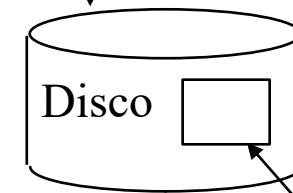
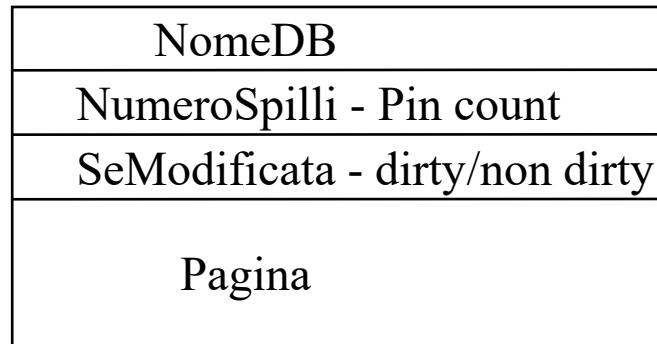
Area libera

Area occupata

**unPinPage:** rilascia la pagina e elimina un pin

**setDirty:** indica che la pagina è stata modificata, ovvero è dirty ("sporca")

**flushPage:** forza la scrittura della pagina su disco, rendendola così "pulita"



Pagina fisica

## Politiche di rimpiazzamento

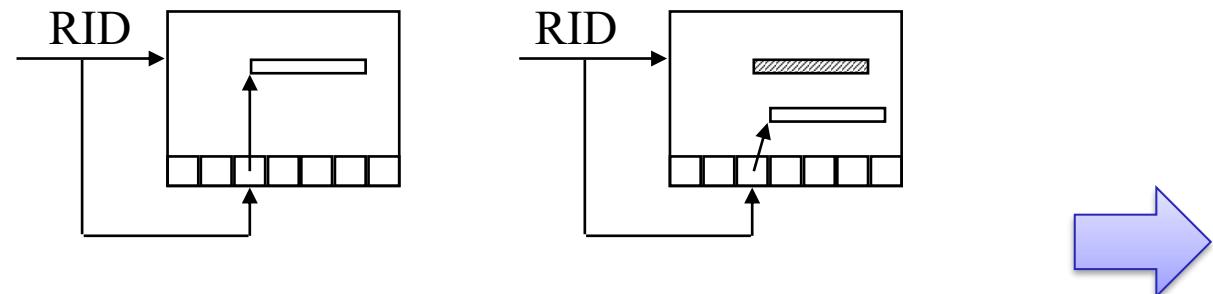
---

- Nei sistemi operativi una comune politica adottata per decidere quale pagina rimpiazzare è la LRU (Least Recently Used), ovvero si rimpiazza la pagina che da più tempo non è in uso
- Nei DBMS, LRU non è sempre una buona scelta, in quanto per alcune query il "pattern di accesso" ai dati è noto, e può quindi essere utilizzato per operare scelte più accurate, in grado di migliorare anche di molto le prestazioni
- L'hit ratio, ovvero la frazione di richieste che non provocano una operazione di I/O, indica sinteticamente quanto buona è una politica di rimpiazzamento
- Esempio: esistono algoritmi di join che scandiscono N volte le tuple di una relazione. In questo caso la politica migliore sarebbe la MRU (Most Recently Used), ovvero rimpiazzare la pagina usata più di recente!
- ... altro motivo per cui i DBMS non usano (tutti) i servizi offerti dai sistemi operativi...

# STRUTTURA DI UNA PAGINA

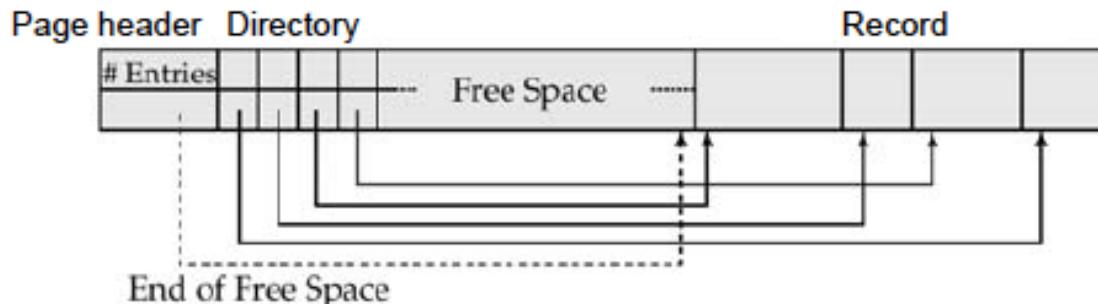
---

- Struttura fisica: un insieme, di dimensione fissa, di caratteri.
- Struttura logica:
  - informazioni di servizio;
  - un'area che contiene le stringhe che rappresentano i record;
- Il problema dei riferimenti ai record: coppia (PID della pagina, posizione nella pagina) (RID).



# Organizzazione a slot delle pagine

- Il formato tipico di una pagina in un DBMS è descritto in figura

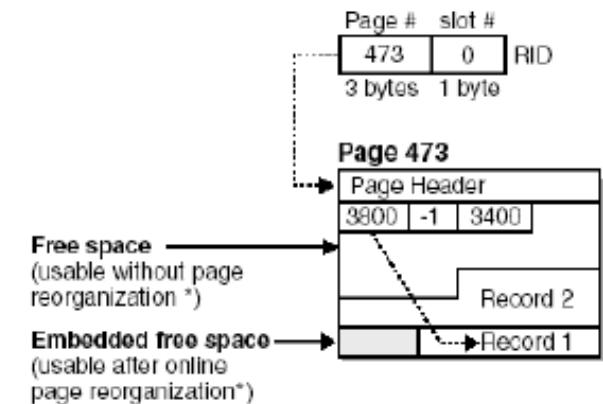
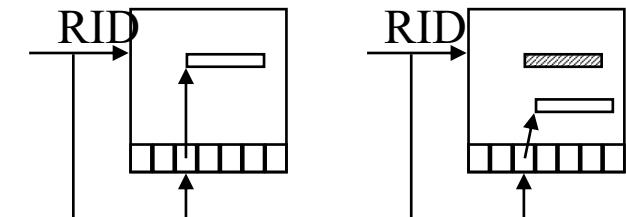


La **directory** contiene un puntatore per ogni record nella pagina

Con questa soluzione l'**identificatore di un record (RID)** nel DB è formato da una coppia:

- **PID**: identificatore della pagina
- **Slot**: posizione all'interno della pagina

È possibile sia individuare velocemente un record, sia permettere la sua riallocazione nella pagina senza modificare il RID



---

# **PARTE II**

# GESTORE STRUTTURE DI MEMORIZZAZIONE

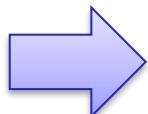
---

- Tipi di organizzazioni:
  - Seriali o Sequenziali
  - Per chiave
  - Per attributi non chiave
- Parametri che caratterizzano un'organizzazione:
  - Occupazione di memoria
  - Costo delle operazioni di:
    - Ricerca per valore o intervallo
    - Modifica
    - Inserzione
    - Cancellazione

# ORGANIZZAZIONI SERIALE E SEQUENZIALE

---

- Organizzazione **seriale (heap file)**: i dati sono memorizzati in modo disordinato uno dopo l'altro:
  - Semplice e a basso costo di memoria
  - Poco efficiente
  - Va bene per pochi dati
- E' l'organizzazione standard di ogni DBMS.



# ORGANIZZAZIONI SERIALE E SEQUENZIALE

---

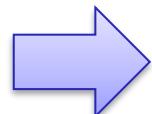
- Organizzazione **sequenziale**: i dati sono ordinati sul valore di uno o più attributi:
  - Ricerche più veloci
  - Nuove inserzioni fanno perdere l'ordinamento

## Costo di ricerca

Ricerca binaria su file di dati ordinato richiede  $\log_2 b$  accessi a blocco/pagina

Se il file contiene  $b_i$  blocchi, la localizzazione di un record richiede:

- ricerca binaria nel file
- accesso al blocco che contiene il record
- Quindi richiede  $\log_2 b_i + 1$  accessi a blocco/pagina



## Esempio

---

- File ordinato con  $r = 30.000$  record
- Dimensioni dei blocchi/pagine su disco  $B = 1024$  byte
- I record hanno dimensione fissa e sono indivisibili, lunghezza  $R = 100$  byte
- Il fattore di blocco/pagine è

$$bfr = \lfloor (B/R) \rfloor = \lfloor (1024/100) \rfloor = 10 \text{ record per blocco/pagine}$$

- Numero blocchi necessari  $b = \lceil (r/bfr) \rceil = \lceil (30000/10) \rceil = 3000$  blocchi
- Una ricerca binaria su file dati richiede circa  $\lceil \log_2 b \rceil = \lceil \log_2 3000 \rceil = 12$  accessi ai blocchi/pagine

# ORGANIZZAZIONI PER CHIAVE

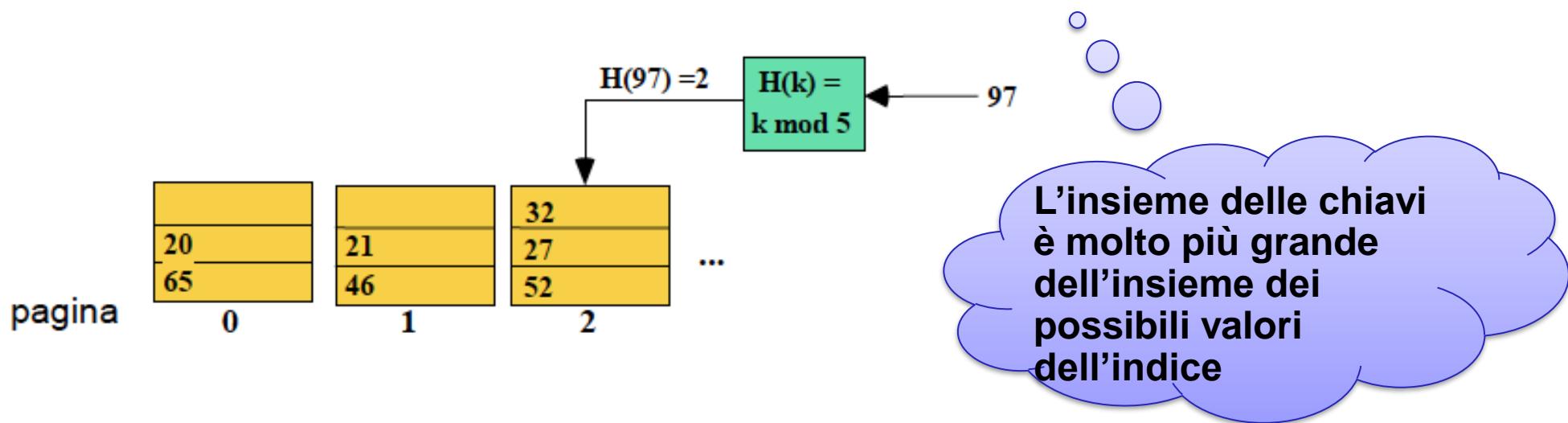
---

- **Obiettivo:** noto il valore di una chiave, trovare il record di una tabella con qualche accesso al disco (ideale: 1 accesso).
- **Alternative:**
  - Metodo procedurale (hash) o tabellare (indice)
  - Organizzazione statica o dinamica.



# HASH FILE

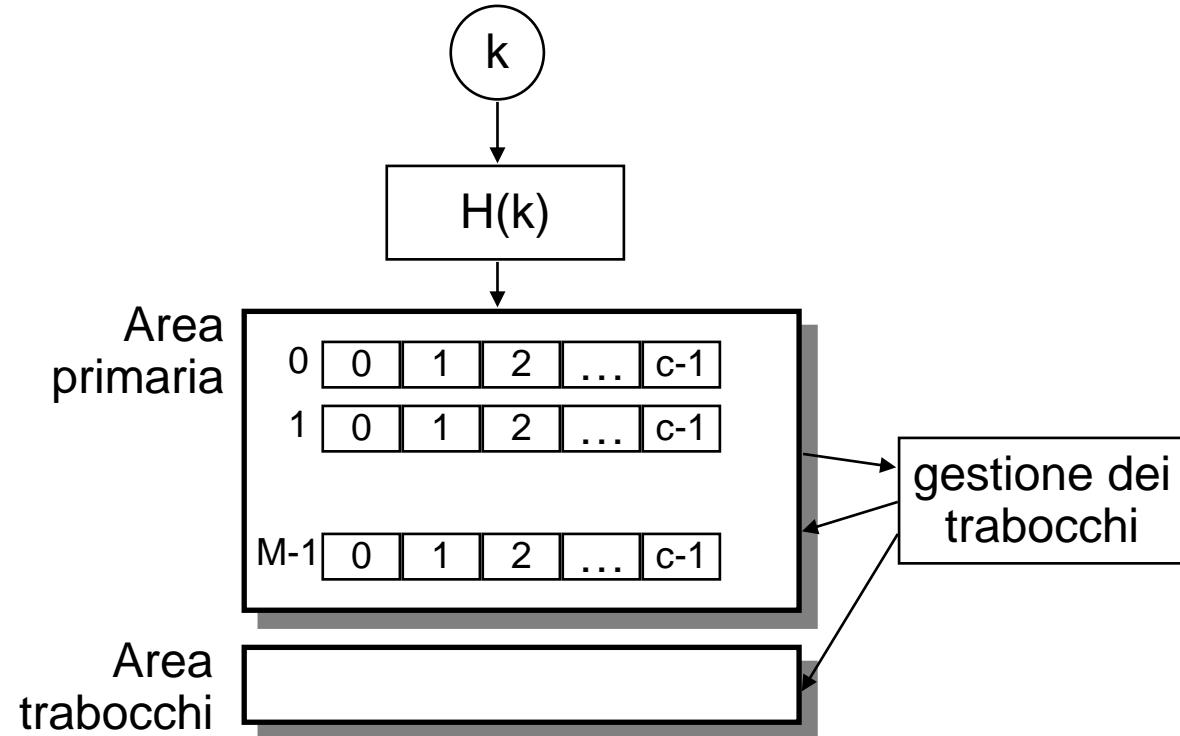
- In un file hash i record vengono allocati in una pagina il cui indirizzo dipende dal valore di chiave del record:  
 $\text{key} \rightarrow H(\text{key}) \rightarrow \text{page address}$
- Una comune funzione hash è il resto della divisione intera:  
$$H(k) = k \bmod N_p$$
- Si può applicare anche a chiavi alfanumeri che dopo averle convertite



# ORGANIZZAZIONE PER CHIAVE: METODO PROCEDURALE STATICO

Parametri di progetto:

- La funzione per la trasformazione della chiave
- Il fattore di caricamento  $d=N/(M*c)$
- La capacità  $c$  delle pagine
- Il metodo per la gestione dei trabocchi



Frazione dello spazio fisico disponibile mediamente utilizzata.

Se  $N$  è il numero di tuple previsto per il file,  $M$  il fattore di pagine e  $c$  il fattore di caricamento, il file può prevedere un numero di blocchi  $B$  pari al numero intero immediatamente superiore a  $d$ .

## Strutture Hash

---

- Le collisioni (overflow) sono di solito gestite con liste linked.
- E' l'organizzazione più efficiente per l'accesso diretto basato su valori della chiave con condizioni di uguaglianza (**accesso puntuale**)
- Non è efficiente per **ricerche basate su intervalli** (n. per ricerche basate su altri attributi)
- Funzionano solo con file la cui dimensione non varia molto nel tempo  
**(PROCEDURALE STATICO)**

# Strutture Hash

---

Svantaggi:

Trade-off:

- Se il numero di blocchi è troppo piccolo rispetto al Database si hanno frequenti collisioni (con catene di overflow, etc).
- Se il numero di blocchi è troppo grande rispetto al Database si ha un fattore di riempimento dei blocchi molto basso.
- Struttura Hash non è efficiente per le query su un range di valori:  
SELECT \*  
FROM STUDENTI  
WHERE (MATRICOLA>10000) AND (MATRICOLA<20000)
- Struttura Hash non è efficiente per le operazioni che coinvolgono attributi che non sono chiave.

# ORGANIZZAZIONE PER CHIAVE: METODO TABELLARE

---

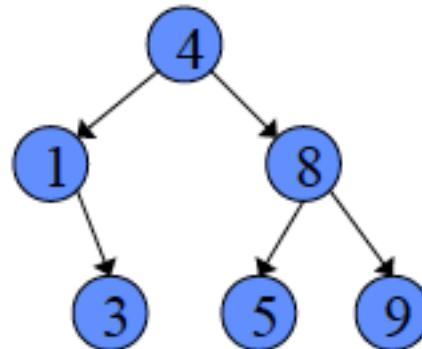
- Il metodo procedurale (hash) è utile per ricerche per chiave ma non per intervallo. Per entrambi i tipi di ricerche è utile invece il **metodo tabellare**:
  - si usa un **indice**, ovvero di un **insieme ordinato** di coppie  $(k, r(k))$ , dove  $k$  è un valore della chiave ed  $r(k)$  è un riferimento al record con chiave  $k$ .
- L'indice è gestito di solito con un'opportuna struttura albero detta **B<sup>+</sup>-albero**, la struttura più usata e ottimizzata dai DBMS.
- Gli indici possono essere multi-attributi.
- Indice: struttura che contiene **informazioni sulla posizione di memorizzazione delle tuple** sulla base del valore del campo chiave.
- La **realizzazione di indici** avviene tipicamente attraverso l'utilizzo di **strutture ad albero multi-livello**.

## Albero binario di ricerca (ripasso)

---

Albero binario etichettato, in cui per ogni nodo,

- il sottoalbero sinistro contiene solo etichette minori di quella del nodo
- il sottoalbero destro etichette maggiori



Tempo di ricerca (e inserimento), pari alla profondit.:

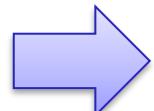
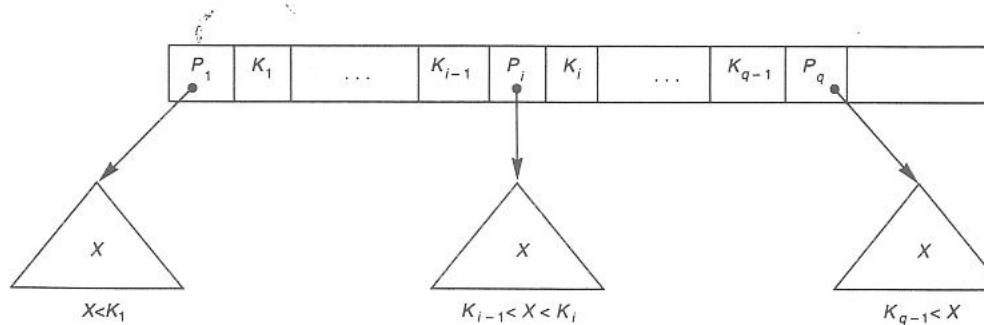
- logaritmico nel caso "medio" (assumendo un ordine di inserimento casuale)

# Albero di ricerca di ordine P - Parte I (ripasso)

- Ogni nodo ha (fino a) P figli e (fino a) P-1 etichette, ordinate
- Un albero di ricerca di ordine p è un albero i cui nodi contengono al più p-1 search value e p puntatori nel seguente ordine

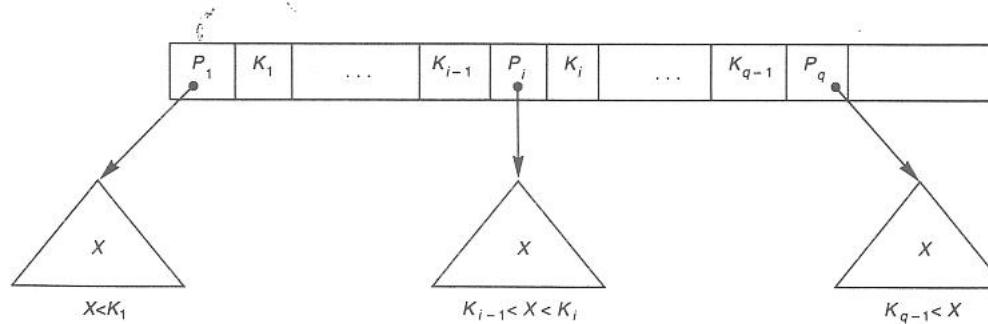
$$\langle P_1, K_1, P_2, K_2, \dots, P_{q-1}, K_{q-1}, P_q \rangle \quad \text{con } q \leq p$$

- Ogni  $P_i$  è un puntatore ad un nodo figlio (o un puntatore nullo) e ogni  $K_i$  è un search value appartenente ad un insieme totalmente ordinato.

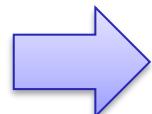


## Albero di ricerca di ordine P - Parte II (ripasso)

---

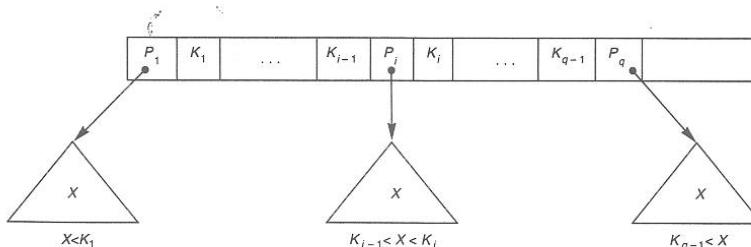


- Ogni albero di ricerca deve soddisfare due vincoli fondamentali:
  - in ogni nodo  $K_1 < K_2 < \dots < K_{q-1}$ ;
  - per tutti i valori di  $X$  presenti nel sottoalbero puntato da  $P_i$ , vale la seguente relazione:
    - $K_{i-1} < X < K_i$  per  $1 < i < q$ ;
    - $X < K_i$  per  $i = 1$ ;
    - $K_{i-1} < X$  per  $i = q$ .



## Albero di ricerca di ordine P - Parte III (ripasso)

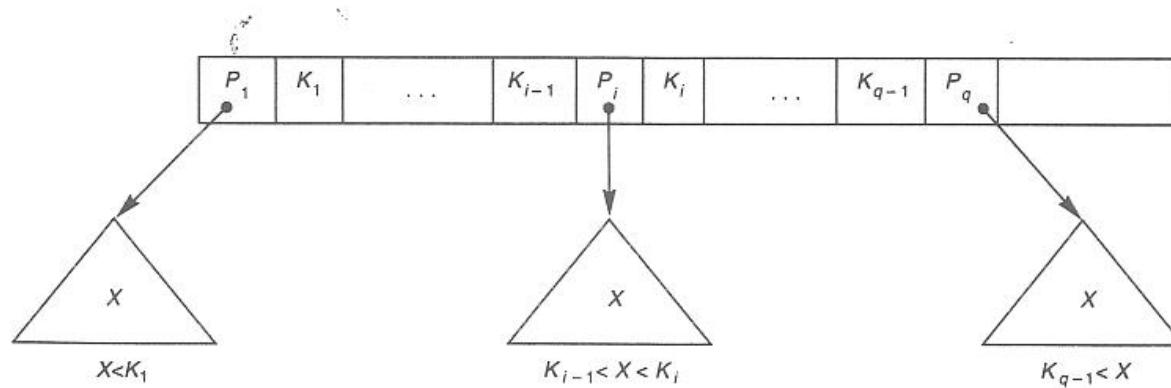
---



- Un albero di ricerca può essere utilizzato per cercare record memorizzati su disco. **Ogni ricerca/modifica comporta la visita di un cammino radice foglia**
- I valori di ricerca (search value) possono essere i valori di uno dei campi del file (search field).
- Ad ogni valore di ricerca è associato un puntatore al record avente quel valore (oppure al blocco contenente quel record) nel file di dati.
- L'albero stesso può essere memorizzato su disco, assegnando ad ogni nodo dell'albero una **pagina**. Quando un nuovo record deve essere inserito, l'albero di ricerca deve essere aggiornato includendo il valore del campo di ricerca del nuovo record, col relativo puntatore al record (o alla pagina che lo contiene), nell'albero di ricerca.

## Albero di ricerca di ordine P - Parte IV (ripasso)

---

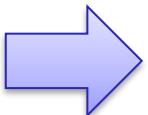


- Per inserire (risp. cancellare) valori di ricerca nell' (risp. dall') albero di ricerca sono necessari specifici algoritmi che garantiscano il rispetto dei due vincoli fondamentali.
- In generale, tali algoritmi non garantiscono che un albero di ricerca risulti sempre bilanciato (nodi foglia tutti allo stesso livello).
- Soluzione: B-alberi e B+ -alberi.

## B-tree - Parte I

---

- un **B-tree di ordine p**, se usato come struttura di accesso su un campo chiave per la ricerca di record in un file di dati, deve soddisfare le seguenti condizioni:
  - ogni nodo interno del B-albero ha la forma:
$$\langle P_1, \langle K_1, Pr_1 \rangle, P_2, \langle K_2, Pr_2 \rangle, \dots, \langle K_{q-1}, Pr_{q-1} \rangle, P_q \rangle \text{ con } q \leq p$$
    1.  $P_i$  è un **tree pointer** (puntatore ad un altro nodo del B-albero),
    2.  $K_i$  è la chiave di ricerca
    3.  $Pr_i$  è un **data pointer** (puntatore ad un record il cui campo **chiave di ricerca** è uguale a  $K_i$  o alla pagina che contiene tale record);
  - per ogni nodo, si ha che:
$$K_1 < K_2 < \dots < K_{q-1};$$
  - ogni nodo ha al più p tree pointer;



## B-tree - Parte II

---

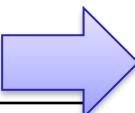
- per tutti i valori  $X$  della chiave di ricerca appartenenti al sottoalbero puntato da  $P_i$ , si ha che:

$$K_{i-1} < X < K_i \quad \text{per } 1 < i < q;$$

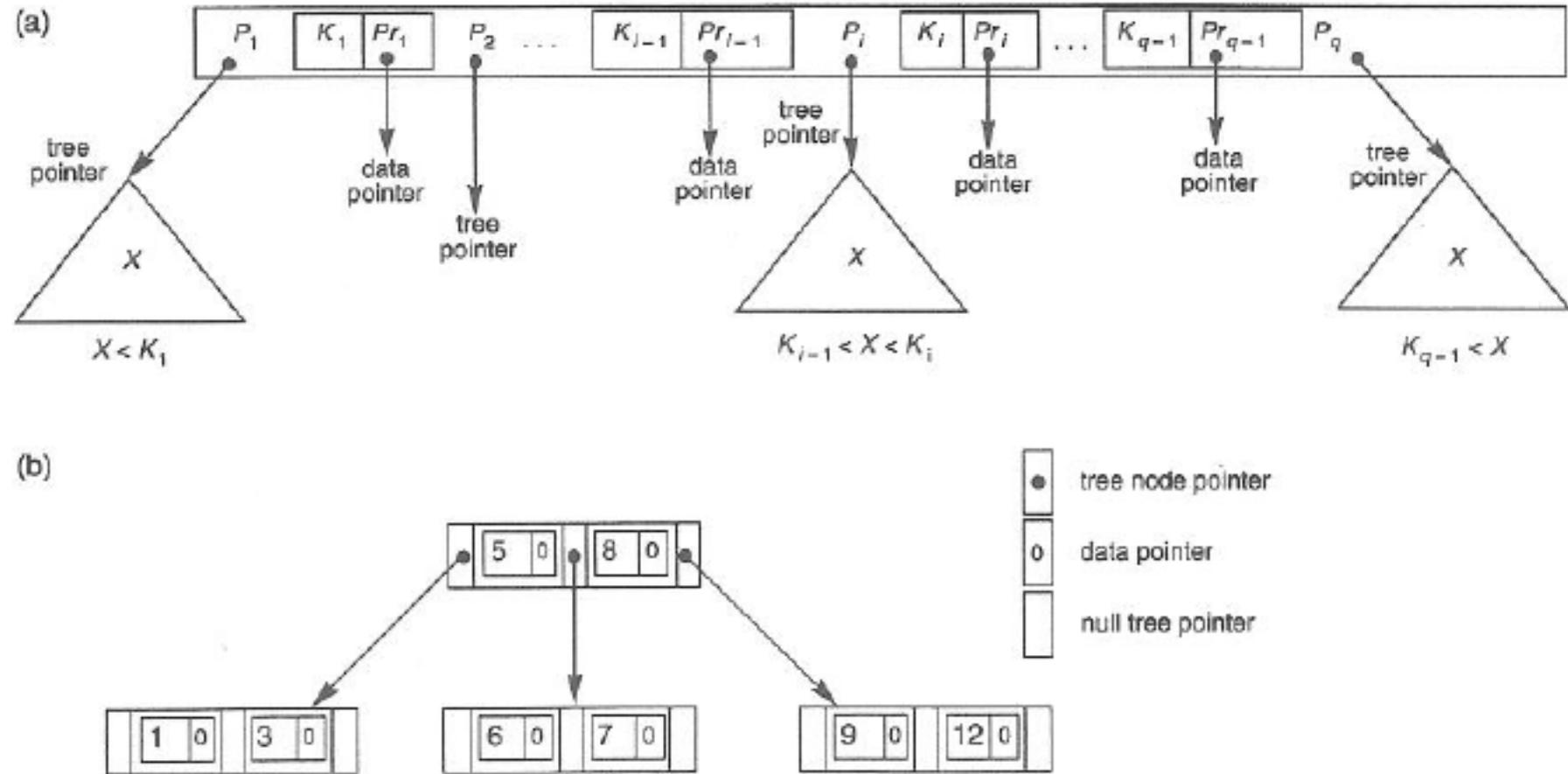
$$X < K_i \quad \text{per } i = 1;$$

$$K_{i-1} < X \quad \text{per } i = q;$$

- la **radice** ha almeno due tree pointer, a meno che non sia l'unico nodo dell'albero;
- ogni **nodo**, esclusa la radice, ha almeno  $\lceil p/2 \rceil$  tree pointer;
  - un nodo con  $q$  tree pointer,  $q \leq p$ , ha  $q-1$  campi chiave di ricerca (e  $q-1$  data pointer);
- tutti i nodi **foglia** sono posti allo stesso livello (i nodi foglia hanno la stessa struttura dei nodi interni, ad eccezione del fatto che tutti i loro tree pointer  $P_i$  sono nulli)



# B-alberi: struttura



## B<sup>+</sup>-tree

---

- Un B<sup>+</sup>-tree è un B-albero in cui i data pointer sono memorizzati solo nei nodi foglia dell'albero. La struttura dei nodi foglia differisce dal B-tree quindi da quella dei nodi interni.
- Se il campo di ricerca è un campo chiave, i nodi foglia hanno per ogni valore del campo di ricerca una entry e un puntatore ad un record.
- Se un campo di ricerca non è un campo chiave, i puntatori indirizzano un blocco che contiene i puntatori ai record del file di dati, rendendo così necessario un passo aggiuntivo per l'accesso ai dati.
- I nodi foglia di un B<sup>+</sup>-tree sono generalmente messi fra loro in relazione (ciò viene sfruttato nel caso di range query). Essi corrispondono al primo livello di un indice. I nodi interni corrispondono agli altri livelli di un indice.
- Alcuni valori del campo di ricerca presenti nei nodi foglia sono ripetuti nei nodi interni per guidare la ricerca.



## B<sup>+</sup>-tree - Parte I

---

La struttura dei **nodi interni** (di ordine p) di un B<sup>+</sup>-tree è la seguente:

(1) ogni nodo interno del B<sup>+</sup>-tree ha la forma:

$$\langle P_1, K_1, P_2, K_2, \dots, P_{q-1}, K_{q-1}, P_q \rangle \text{ con } q \leq p$$

ogni **P<sub>i</sub>** è un **tree pointer** (puntatore ad un altro nodo del B<sup>+</sup>-tree)

(2) per ogni nodo interno, si ha che:

$$K_1 < K_2 < \dots < K_{q-1} ;$$

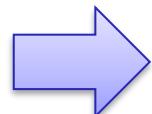
(3) ogni nodo interno ha al più p tree pointer

(4) per tutti i valori X della search key nel sottoalbero puntato da P<sub>i</sub>, si ha che

$$X \leq K_i \quad \text{per } i = 1;$$

$$K_{i-1} < X \leq K_i \quad \text{per } 1 < i < q;$$

$$K_{i-1} < X \quad \text{per } i = q;$$

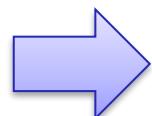
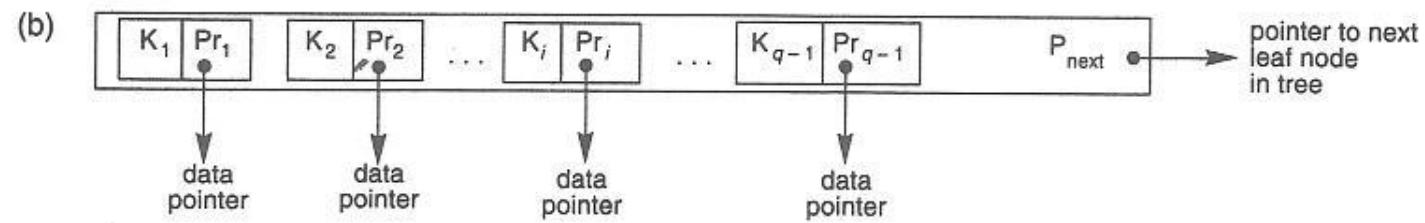
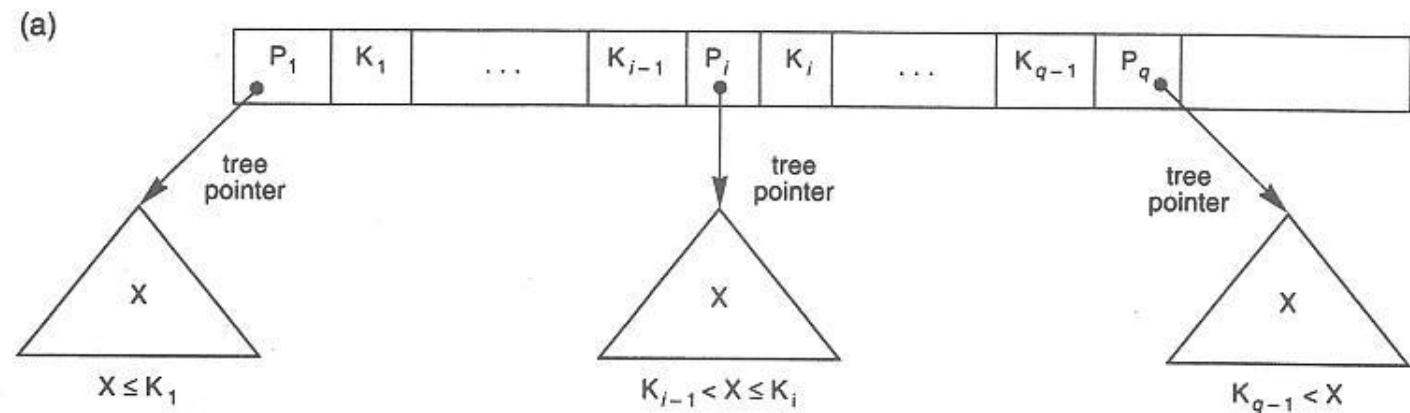


## B<sup>+</sup>-tree - Parte II

(5) ogni nodo interno, esclusa la radice, ha almeno  $\lceil p/2 \rceil$  tree pointer.

La radice ha almeno 2 tree pointer se è un nodo interno.

(6) un nodo interno con  $q$  tree pointer, con  $q \leq p$ , ha  $q-1$  campi di ricerca.



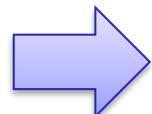
La struttura dei nodi **foglia** (di ordine  $p_{leaf}$ ) di un B<sup>+</sup>-albero è la seguente:

(1) ogni nodo foglia è della forma:  $\langle \langle K_1, Pr_1 \rangle, \langle K_2, Pr_2 \rangle, \dots, \langle K_q, Pr_q \rangle, P_{next} \rangle$ , dove  $q \leq p_{leaf}$  e per ogni nodo, si ha che:  $K_1 < K_2 < \dots < K_q$ , e

- $P_{next}$  è un tree pointer punta al successivo nodo foglia del B<sup>+</sup>-tree
- ogni  $Pr_i$  è un data pointer che punta al record con valore del campo di ricerca uguale a  $K_i$  o ad un blocco contenente tale record (o ad un blocco di puntatori ai record con valore del campo di ricerca uguale a  $K_i$ , se il campo di ricerca non è una chiave)

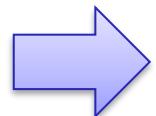
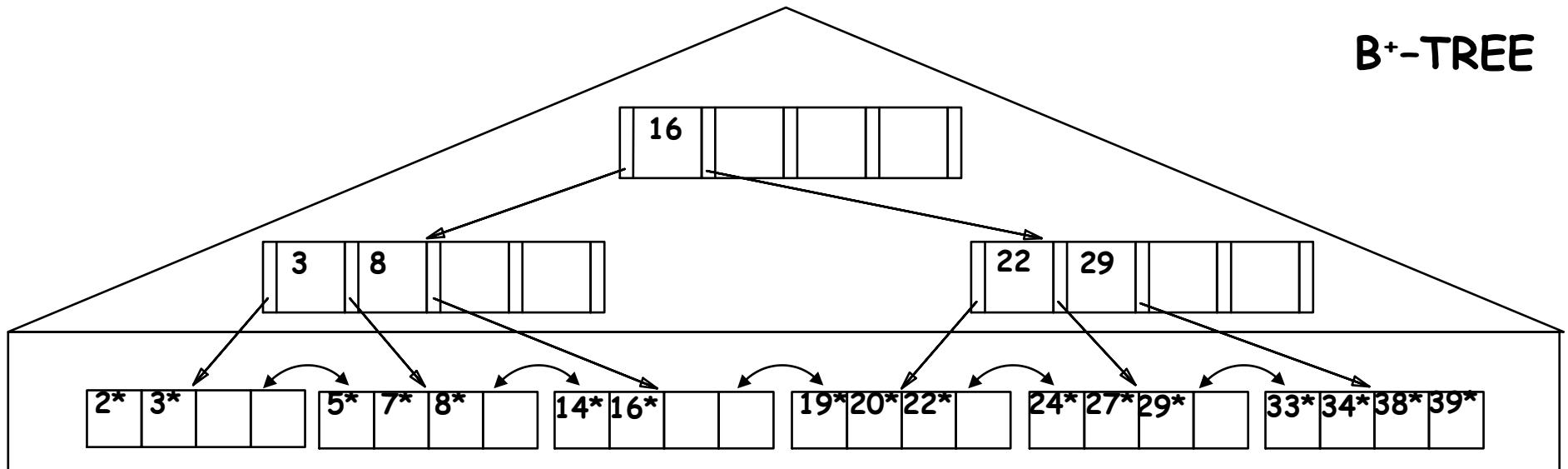
(4) ogni nodo foglia ha almeno  $\lceil p_{leaf}/2 \rceil$  valori

(5) tutti i nodi foglia sono dello stesso livello



## B<sup>+</sup>-tree - Parte IV

---



- Di solito un indice è organizzato a B<sup>+</sup>- albero per permettere di trovare con pochi accessi, a partire da un valore v, i record di R in cui il valore di A è in una relazione specificata con v.
- Due tipologie di indici ad albero:
  - Indici **statici**: La struttura ad albero viene creata sulla base dei dati presenti nel DB, e non più modificata (o modificata periodicamente).
  - Indici **dinamici**: La struttura ad albero viene aggiornata ad ogni operazione sulla base di dati di inserimento o di cancellazione, memorizzati, preservando le prestazioni senza bisogno di riorganizzazioni.

- **Indice**: struttura che contiene informazioni sulla posizione di memorizzazione delle tuple sulla base del valore del campo chiave.
- Tali strutture velocizzano l'accesso casuale via chiave di ricerca.
- Possiamo distinguere:
  - INDICE **PRIMARIO**: in questo caso la chiave di ordinamento del file sequenziale coincide con la **chiave di ricerca dell'indice**.
  - INDICE **SECONDARIO**: in questo caso invece la **chiave di ordinamento** e la **chiave di ricerca** sono diverse.
- Un **indice** può essere **definito su di un insieme  $A_1, \dots, A_n$  di attributi**.
  - In questo caso, l'indice contiene un record per ogni combinazione di valori assunti dagli attributi  $A_1, \dots, A_n$  nella relazione, e può essere utilizzato per rispondere in modo efficiente ad interrogazioni che specifichino un valore per ciascuno di questi attributi.



## Indice primario

---

- Un indice **primario** è un file ordinato i cui record sono di lunghezza fissa e sono costituiti da due campi
  - Il primo campo è dello stesso tipo del **campo chiave di ordinamento** (Chiave primaria)
  - Il secondo campo è un **puntatore a un blocco del disco**
- Esiste un record nel file dell'indice per ogni blocco nel file di dati

$\langle K(i) , RID(i) \rangle$

| RID | Matr | Prov | An   |
|-----|------|------|------|
| 1   | 106  | MI   | 1972 |
| 2   | 102  | PI   | 1970 |
| 3   | 107  | PI   | 1971 |
| 4   | 104  | FI   | 1968 |
| 5   | 100  | MI   | 1970 |
| 6   | 103  | PI   | 1972 |

## Indice secondario

---

- Un indice **secondario** può essere definito su un **campo non chiave** che è una chiave **candidata** e ha valori univoci, o su un campo non chiave con **valori duplicati**
- Il **primo campo** è dello stesso tipo del campo che non viene usato per ordinare il file ed è chiamato **campo di indicizzazione**
- Il **secondo campo** è un **puntatore al blocco** oppure un **puntatore al record (RID)**

| Matr | RID |
|------|-----|
| 100  | 5   |
| 102  | 2   |
| 103  | 6   |
| 104  | 4   |
| 106  | 1   |
| 107  | 3   |

Indice su Matr



# ESEMPI DI INDICI PER ATTRIBUTO CHIAVE O NON CHIAVE

---

- Tabella:

| RID | Matr | Prov | An   |
|-----|------|------|------|
| 1   | 106  | MI   | 1972 |
| 2   | 102  | PI   | 1970 |
| 3   | 107  | PI   | 1971 |
| 4   | 104  | FI   | 1968 |
| 5   | 100  | MI   | 1970 |
| 6   | 103  | PI   | 1972 |

- Indici

| Matr | RID |
|------|-----|
| 100  | 5   |
| 102  | 2   |
| 103  | 6   |
| 104  | 4   |
| 106  | 1   |
| 107  | 3   |

Indice su Matr

| An   | RID |
|------|-----|
| 1968 | 4   |
| 1970 | 2   |
| 1970 | 5   |
| 1971 | 3   |
| 1972 | 1   |
| 1972 | 6   |

Indice su An

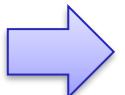
---

# **PARTE III**

# ORDINAMENTO DI ARCHIVI

---

- Perché è importante l'ordinamento di archivi
  - Risultato di interrogazioni ordinato (order by)
  - Per eseguire alcune operazioni relazionali (join, select distinct, group by)
- Algoritmo sort-merge: costo  $N * \log(N)$



## Sort: implementazione

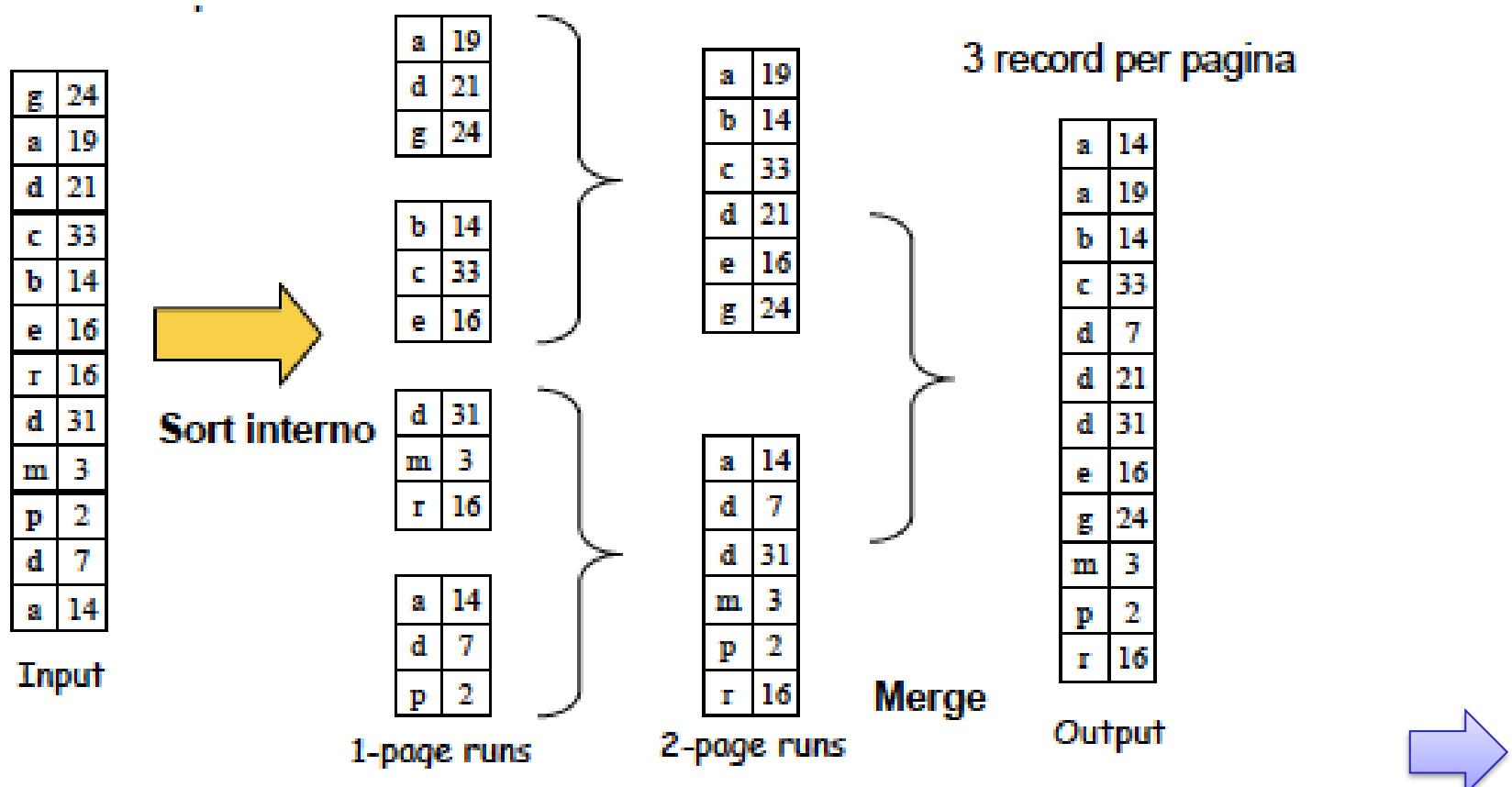
---

- L'algoritmo più comunemente utilizzato dai DBMS è quello detto di **Merge Sort a Z vie (Z-way Sort-Merge)**
- Supponiamo di dover ordinare un input che consiste di un file di **NP pagine** e di avere a disposizione solo **NB < NP buffer** in memoria centrale
- L'algoritmo opera in 2 fasi:
  - **Sort interno**: si leggono una alla volta le pagine del file; i record di ogni pagina vengono ordinati facendo uso di un algoritmo di sort interno (es. Quicksort); **ogni pagina così ordinata, detta anche "run"**, viene scritta su disco in un file temporaneo
  - **Merge**: operando uno o più passi di fusione, le **run vengono fuse**, fino a produrre un'unica run



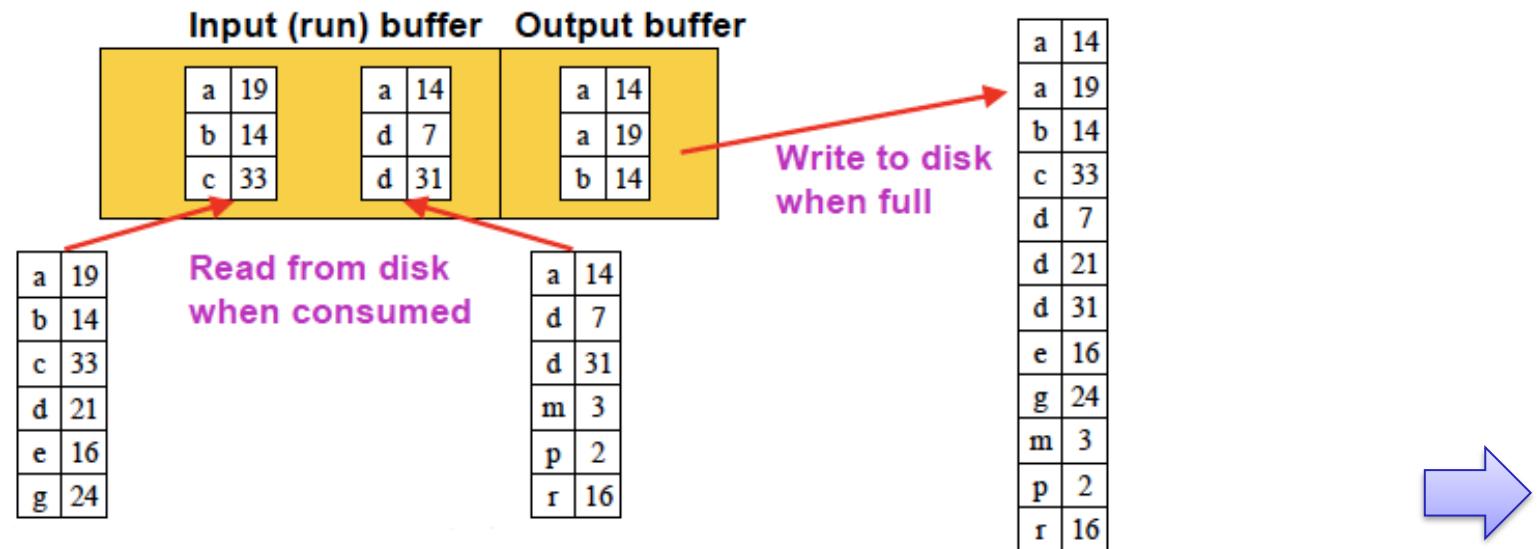
## Z-way Merge Sort: caso base

- Per semplicità consideriamo il caso base a  $Z = 2$  vie, e supponiamo di avere a disposizione solo  $NB = 3$  buffer in memoria centrale



## Fusione delle run

- Nel caso base  $Z = 2$  si fondono 2 run alla volta
- Con  $NB = 3$ , si associa un buffer a ognuna delle run, il terzo buffer serve per produrre l'output, 1 pagina alla volta
- Si legge la prima pagina da ciascuna run e si può quindi determinare la prima pagina dell'output; quando tutti i record di una pagina di run sono stati consumati, si legge un'altra pagina della run



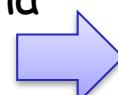
## Caso base - Complessità

---

- Consideriamo per semplicità solo il numero di operazioni di I/O
- Nel caso base  $Z = 2$  (e  $NB = 3$ ) si può osservare che:
  - Nella fase di sort interno si leggono e si riscrivono  $NP$  pagine
  - Ad ogni passo di merge si leggono e si riscrivono  $NP$  pagine ( $2 * NP$ )
    - Il numero di passi fusione è pari a  $\lceil \log_2 NP \rceil$ , in quanto ad ogni passo il numero di run si dimezza
    - Il costo complessivo è pertanto pari a  $2 * NP * (\lceil \log_2 NP \rceil + 1)$

Esempio: per ordinare  $NP = 8000$  pagine sono necessarie circa  $224.000$  ( $2 * 8000 * (\lceil \log_2 8000 \rceil + 1)$ ) operazioni di I/O; se ogni I/O richiede 20 msec, il sort richiede 4.480 secondi, ovvero circa 1h 15 minuti!

- In realtà se  $NP$  non è una potenza di 2 il numero effettivo di I/O è leggermente minore di quello calcolato, in quanto in uno o più passi di fusione può capitare che una run non venga fusa con un'altra



## Z-way Sort-Merge: caso generale

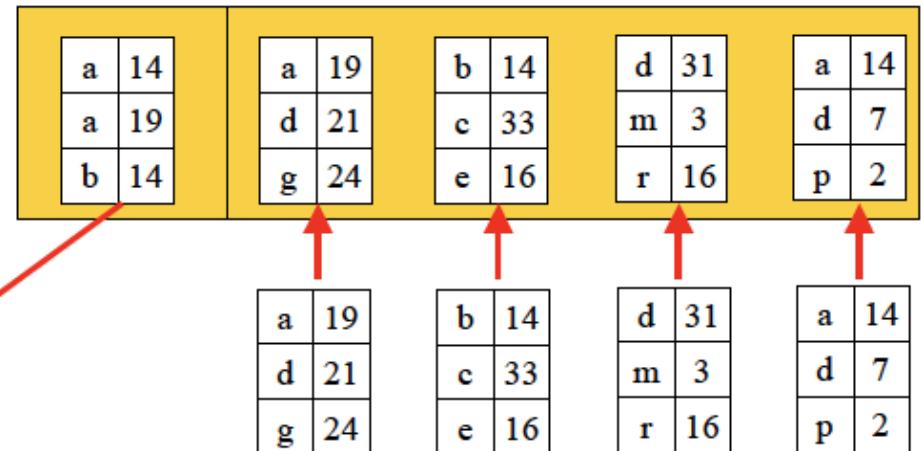
- Una prima osservazione è che nel passo di sort interno, avendo a disposizione **NB buffer**, si possono ordinare **NP** pagine alla volta (anziché una sola), il che porta a un costo di  $2 * NP * (\lceil \log_2 (NP/NB) \rceil + 1)$
- Esempio: con  $NP = 8000$  pagine e  $NB = 3$  si hanno ora  $208.000$  I/O
  - Miglioramenti sostanziali si possono ottenere se, avendo  $NB > 3$  buffer a disposizione, **si fondono  $NB - 1$  run alla volta** (1 buffer è per l'output)
  - In questo caso il numero di passi di fusione è logaritmico in  $NB - 1$ , ovvero è pari a  $2 * NP * (\lceil \log_{NB-1} (NP/NB) \rceil + 1)$

- Esempio:

con  $NP = 8000$  pagine e  $NB = 11$

si hanno  $64000$  I/O,

per un tempo stimato pari a  $1280$  sec



## Utilità del Sort

---

- Oltre che per ordinare le tuple, il **Sort** può essere utilizzato per:
  - Query in cui compare l'opzione DISTINCT, ovvero per eliminare i duplicati
  - Query che contengono la clausola GROUP BY

# REALIZZAZIONE DEGLI OPERATORI RELAZIONALI

---

- Si considerano i seguenti operatori:
  - *Proiezione*
  - *Selezione*
  - *Raggruppamento*
  - *Join*

```
SELECT *
      Provincia
  FROM Studenti R
```

```
SELECT DISTINCT
      Provincia
  FROM Studenti R
```

- Caso di DISTINCT: approccio basato sull'ordinamento (non è l'unico!):
  - Si legge R e si scrive T che contiene solo gli attributi della SELECT
  - Si ordina T su tutti gli attributi
  - Si eliminano i duplicati

## RESTRIZIONE CON CONDIZIONE SEMPLICE

---

```
SELECT *
FROM   Studenti R
WHERE  R.Provincia = 'PI'
```

- *Senza indice e dati disordinati:*  
*Npag(R).*
- *Con indice (B<sup>+</sup>-albero): CI + CD*

# OPERAZIONI DI AGGREGAZIONE

---

- **Senza GROUP BY**
  - Si visitano i dati e si calcolano le funzioni di aggregazione.
    - `Select count(*) from Persone`
- **Con GROUP BY**
  - Approccio basato sull'ordinamento (non è l'unico!):
    - Si ordinano i dati sugli attributi del GROUP BY, poi si visitano i dati e si calcolano le funzioni di aggregazione per ogni gruppo.
      - `Select cognome, count(cognome) from Persone group by cognome`

# GIUNZIONE

```
SELECT *
FROM   Studenti S, Esami E
WHERE  S.Matricola=E.Matricola
```

- $R \times S$  è grande; pertanto,  $R \times S$  seguito da una restrizione è inefficiente.

| R  | A | B |
|----|---|---|
| 22 | a |   |
| 87 | s |   |
| 45 | h |   |
| 32 | b |   |

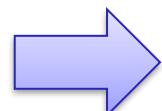
| S  | A | C | D |
|----|---|---|---|
| 22 | z | 8 |   |
| 45 | k | 4 |   |
| 22 | s | 7 |   |
| 87 | s | 9 |   |
| 32 | c | 3 |   |
| 45 | h | 5 |   |
| 32 | g | 6 |   |

PJ: R.A = S.A

➡

|    | A | C | D | B |
|----|---|---|---|---|
| 22 | z | 8 | a |   |
| 22 | s | 7 | a |   |
| 87 | s | 9 | s |   |
| 45 | k | 4 | h |   |
| 45 | h | 5 | h |   |
| 32 | c | 3 | b |   |
| 32 | g | 6 | b |   |

Anche se logicamente il Join sia commutativo, dal punto di vista fisico vi è una chiara distinzione, che influenza anche le prestazioni, tra operando sinistro (o "esterno", "outer") e operando destro (o "interno", "inner")



## NESTED LOOPS

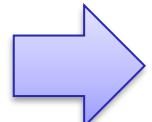
- Il costo di esecuzione dipende dallo spazio a disposizione in memoria centrale

- Nel caso base in cui vi sia 1 buffer per R e 1 buffer per S, bisogna leggere 1 volta R e  $N_{rec}(R)$  volte S, ovvero tante volte quante sono le tuple della relazione esterna, per un totale di

$$N_{pag}(R) + N_{rec}(R) * N_{pag}(S) \text{ I/O}$$

- Se è possibile allocare  $N_{pag}(S)$  buffer per la relazione interna il costo si riduce a

$$N_{pag}(R) + N_{pag}(S)$$



## Nested Loops: esempio (relazione esterna vs interna)

---

- Se si fa il join tra Dipartimenti e Impiegati ( $DipImp = NumDip$ ) e si ha:

$$Npag(Dipartimenti) = 20 \quad Nrec(Dipartimenti) = 100$$

$$Npag(Impiegati) = 1000 \quad Nrec(Impiegati) = 10000$$

e considerando il caso base (1 buffer per ciascuna relazione):

- Dipartimenti esterna:

$$Npag(Dipartimenti) + Nrec(Dipartimenti) * Npag(Impiegati) = 100.020 \text{ I/O}$$

- Impiegati esterna:

- $Npag(Impiegati) + Nrec(Impiegati) * Npag(Dipartimenti) = 201.000 \text{ I/O}$

## NESTED LOOPS

- Per ogni record della relazione esterna R, si visita tutta la relazione interna S.

• Costo:  $Npag(R) + Nrec(R) * Npag(S)$

$$\approx \frac{Npag(R) * Nrec(R)}{Npag(R)} * Npag(S)$$

```
foreach record r in R do  
foreach record s in S do  
if  $r_i = s_j$  then  
aggiungi  $\langle r, s \rangle$  al risultato
```

con S esterna:

• Costo:  $Npag(S) + Nrec(S) * Npag(R)$

$$\approx \frac{Npag(S) * Nrec(S)}{Npag(S)} * Npag(R)$$

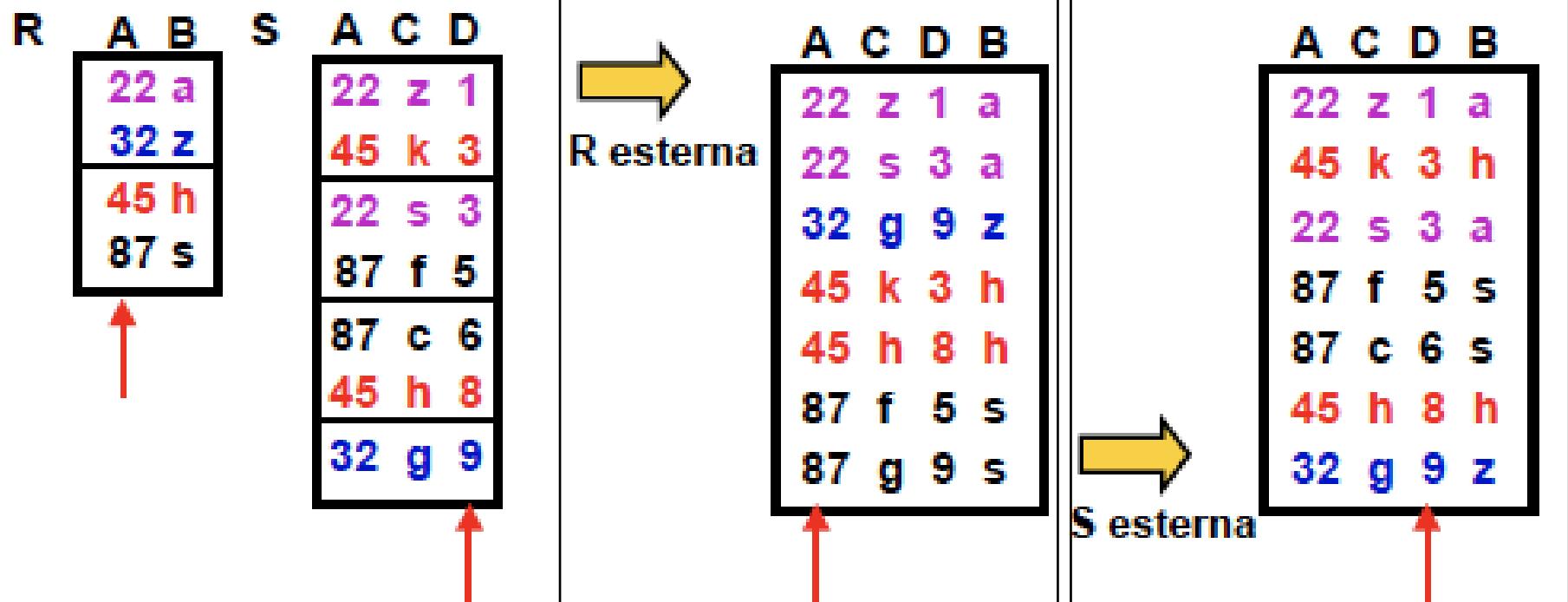
$$\frac{Nrec(R)}{Npag(R)} < \frac{Nrec(S)}{Npag(S)}$$

si sceglierà R come esterna e S come interna se  
 $Nrec(R) * Npag(S) < Nrec(S) * Npag(R)$   
che corrisponde a dire che le tuple di R sono più grandi di quelle di S

• Come esterna conviene la relazione con record più lunghi/grandi

## Nested loops: cammini di accesso

- L'ordine con cui vengono generate le tuple del risultato coincide con l'ordine eventualmente presente nella relazione esterna



Pertanto se l'ordine che si genera è “**interessante**”, ad esempio perché la query contiene ORDER BY R.A, la scelta della relazione esterna può **risultarne influenzata**

## Nested loop a pagine (PageNestedLoop)

- Molti DBMS usano una variante del Nested Loops, detta **Nested loop a pagine**, che, rinunciando a preservare l'ordine della relazione esterna, risulta più efficiente in quanto *esegue il join di tutte le tuple in memoria prima di richiedere nuove pagine della relazione interna*

Per ogni **pagina**  $p_R$  di R:

{ Per ogni **pagina**  $p_S$  in S:

{ esegui il join di tutte le tuple in  $p_R$  e  $p_S$  } }

| R  | A | B |
|----|---|---|
| 22 | a |   |
| 87 | s |   |
| 87 | h |   |
| 92 | b |   |

| S  | A | C | D |
|----|---|---|---|
| 22 | z | 8 |   |
| 87 | c | 8 |   |
| 22 | s | 7 |   |
| 87 | f | 9 |   |



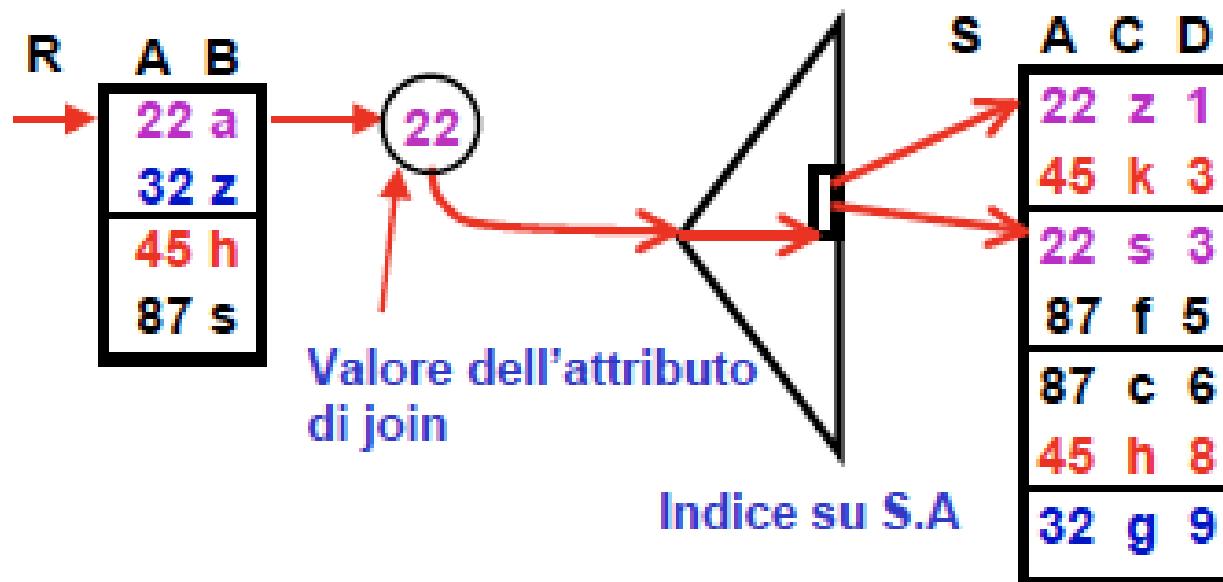
|    | A | C | D | B |
|----|---|---|---|---|
| 22 | z | 8 |   | a |
| 87 | c | 8 |   | s |
| 22 | s | 7 |   | a |
| 87 | f | 9 |   | s |
| 87 | c | 8 |   | h |
| 87 | f | 9 |   | h |

Il costo è ora pari a  
 $NP(R) + NP(R) * NP(S)$  I/O

La strategia si estende anche al caso in cui a R siano assegnati più buffer

## Nested loop con indice (IndexNestedLoop)

- Data una tupla della relazione esterna R, la scansione completa della relazione interna S può essere sostituita da una scansione basata su un indice costruito sugli attributi di join di S, secondo il seguente schema:



IndexNestedLoop  
foreach r in R do  
  foreach s in get-through-index( ISj,=r.i )  
    aggiungi <r,s> al risultato

- L'accesso alla relazione interna mediante indice porta in generale a ridurre di molto i costi di esecuzione del Nested Loops Join



## Sort-merge:

- Il Sort-merge Join è applicabile quando entrambi gli insiemi di tuple in input sono ordinati sugli attributi di join
- Per R (S) ciò è possibile se:
  - R (S) è fisicamente ordinata sugli attributi di join
  - Esiste un indice sugli attributi di join di R (S)

| R             | A  | B | S | A  | C | D |
|---------------|----|---|---|----|---|---|
|               | 22 | a |   | 22 | z | 8 |
|               | 32 | s |   | 22 | s | 7 |
|               | 45 | h |   | 35 | h | 4 |
|               | 87 | b |   | 45 | s | 9 |
| PJ: R.A = S.A |    |   |   | 45 | c | 3 |
|               |    |   |   | 87 | h | 5 |
|               |    |   |   | 87 | g | 6 |



| A  | C | D | B |
|----|---|---|---|
| 22 | z | 8 | a |
| 22 | s | 7 | a |
| 45 | s | 9 | h |
| 45 | c | 3 | h |
| 87 | h | 5 | b |
| 87 | g | 6 | b |

La logica dell'algoritmo (senza considerare il tempo per il sort) sfrutta il fatto che entrambi gli input sono ordinati per evitare di fare inutili confronti, il che fa sì che il numero di letture sia dell'ordine di

$N_{pag}(R) + N_{pag}(S)$   
se si accede sequenzialmente alle due relazioni

### SortMerge

```
r = first(R); s = first(S);
while r in R and s in S do
    if r.i = s.i
        avanza r ed s fino a che r.i ed s.i non
        cambiano entrambe, aggiungendo
        ciascun <r,s> al risultato
    else if r.i < s.i avanza r dentro R
    else if r.i > s.i avanza s dentro S
```

# ALTRI METODI (riepilogo)

---

- Nested loop a **pagina**:

- Per ogni pagina di R, si visitano le pagine di S, e si trovano i record  $\langle r, s \rangle$  della giunzione, con r in R-pagina e s in S-pagina.

- Nested loop con **indice**:

- Si usa quando esiste l'indice  $IS_j$  sull'attributo di giunzione j della relazione interna S

- **Sort-merge**:

- Si usa quando R e S sono ordinate sull'attributo di giunzione: si visitano in R ed S in parallelo

**PageNestedLoop**

```
foreach r in R do
    foreach s in S where r.i = s.j do
        aggiungi  $\langle r, s \rangle$  al risultato
```

**IndexNestedLoop**

```
foreach r in R do
    foreach s in get-through-index( $IS_j = r.i$ )
        aggiungi  $\langle r, s \rangle$  al risultato
```

**SortMerge**

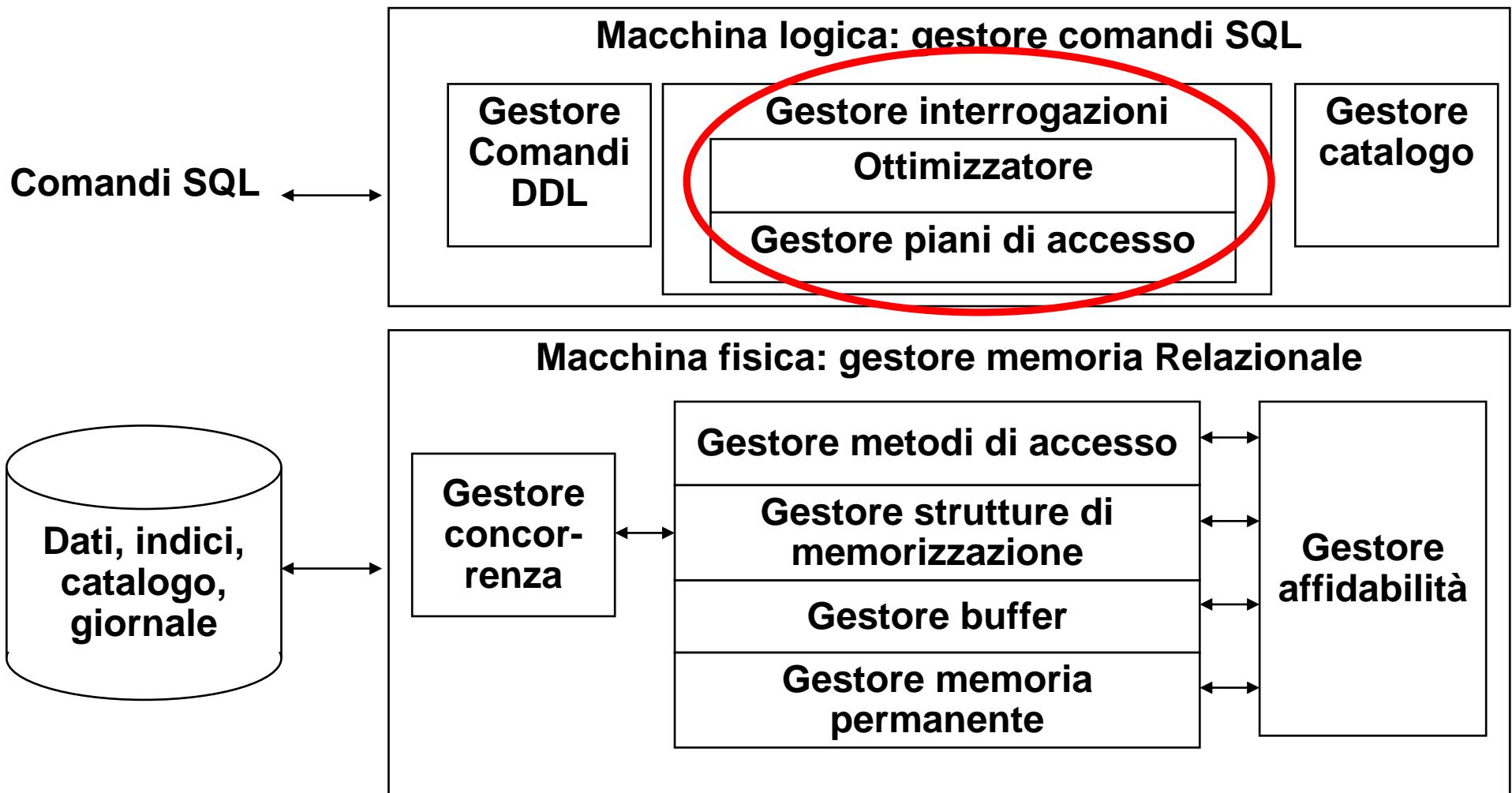
```
r = first(R); s = first(S);
while r in R and s in S do
```

```
if r.i = s.j
    avanza r ed s fino a che r.i ed s.j non
    cambiano entrambe, aggiungendo
    ciascun  $\langle r, s \rangle$  al risultato
else if r.i < s.j avanza r dentro R
else if r.i > s.j avanza s dentro S
```

---

# PIANI DI ACCESSO

# ARCHITETTURA DEI DBMS



# ESECUZIONE DI UN'INTERROGAZIONE

---

```
// analisi lessicale e sintattica del comando SQL Q
SQLCommand parseTree = Parser.parseStatement(Q);

// analisi semantica del comando
Type type = parseTree.check();

// ottimizzazione dell'interrogazione
Value pianoDiAccesso = parseTree.optimize();

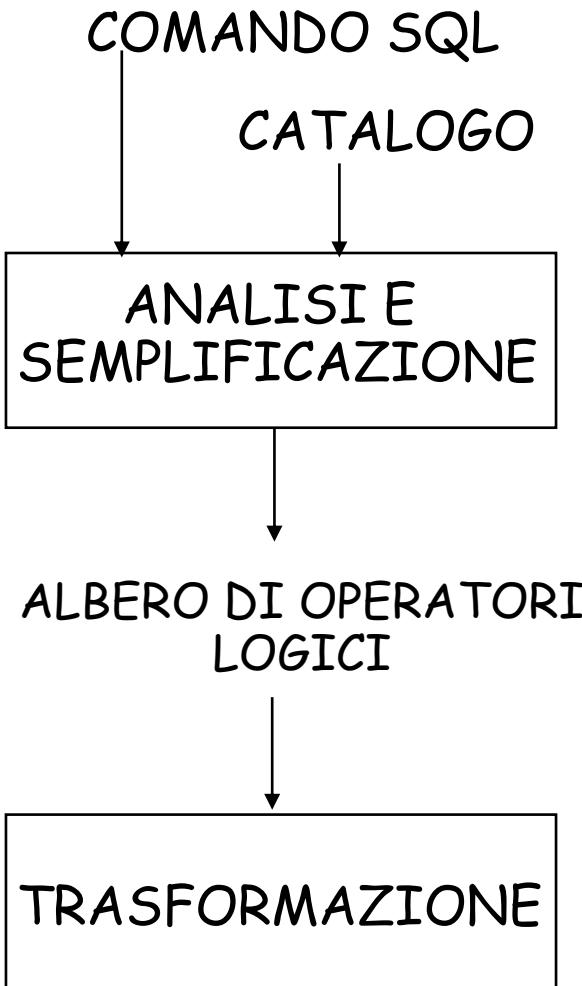
// esecuzione del piano di accesso
pianoDiAccesso.open();
while !pianoDiAccesso.isDone() do
{ Record rec = pianoDiAccesso.next();
  print(rec);
}
pianoDiAccesso.close();
```

# OTTIMIZZATORE DELLE INTERROGAZIONI

---

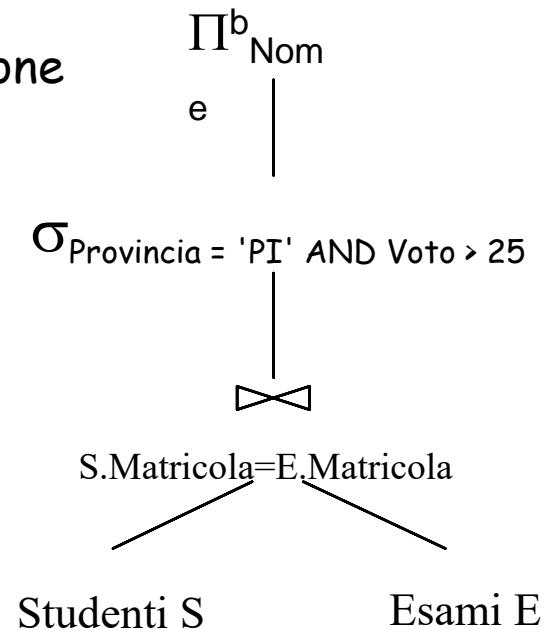
- L'ottimizzazione delle interrogazione è fondamentale nei DBMS.
- E' necessario conoscere il funzionamento dell'ottimizzatore per una buona progettazione fisica.
- Obiettivo dell'ottimizzatore:
  - Scegliere il piano con costo minimo, fra possibili piani alternativi, usando le statistiche presenti nel catalogo.

# FASI DEL PROCESSO DI OTTIMIZZAZIONE



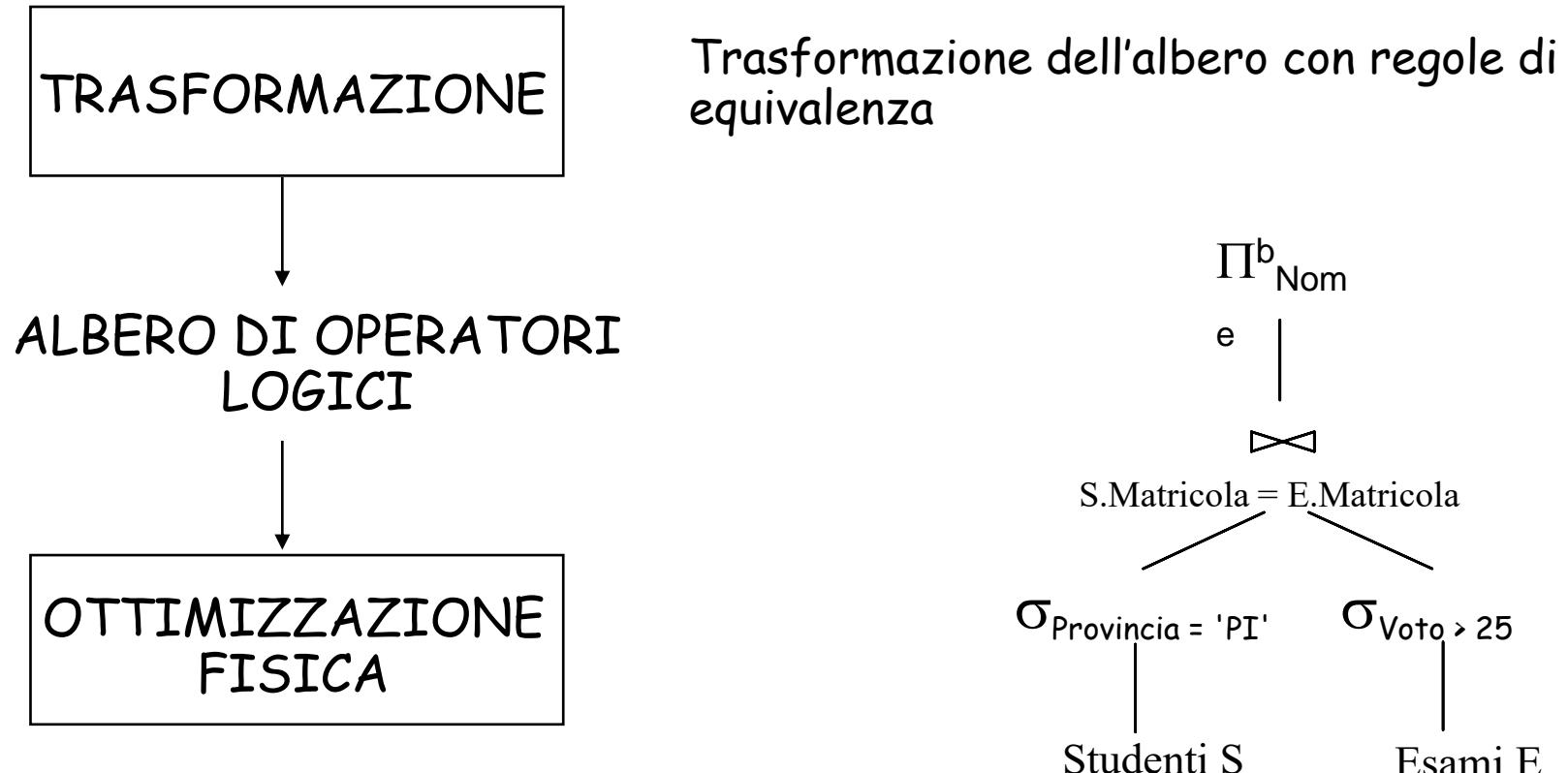
```
SELECT Nome  
FROM Studenti S, Esami E  
WHERE S.Matricola=E.Matricola AND  
Provincia='PI' AND Voto>25
```

Verifica la correttezza del comando, normalizzazione e semplificazione della condizione



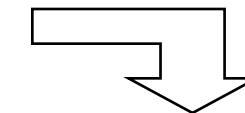
## FASI DEL PROCESSO (cont)

---



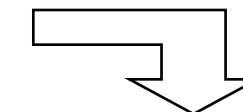
# TRASFORMAZIONI INTERESSANTI

```
SELECT Matricola, Nome  
FROM Studenti  
WHERE Matricola IN ( SELECT Matricola  
                      FROM Esami  
                     WHERE Materia = 'BD');
```



```
SELECT Matricola, Nome  
FROM Studenti S, Esami E  
WHERE S.Matricola = E.Matricola AND Materia = 'BD';
```

```
SELECT Matricola, Nome  
FROM VistaStudentiPisani S, VistaEsamiBD E  
WHERE S.Matricola = E.Matricola ;
```



?

# FASI DEL PROCESSO (cont.)



RISULTATO

| nome  |
|-------|
| Tonio |
| Pino  |

Piano di accesso: **scelta** dell'algoritmo per eseguire ogni operatore.

**Ideale:** Trovare il piano migliore

**Euristica:** evitare i piani peggiori!

Project({“Nome”})

NestedLoop(“S.Matricola=E.Matricola”)

Filter(“Voto >25”)

IndexFilter(Studenti,IdxP, “Provincia = ‘Pi’ ”)

TableScan(“Esami”)

Le foglie sono le tabelle ed i nodi interni specificano le modalità con cui gli accessi alle tabelle e le operazioni relazionali sono effettuate

# REALIZZAZIONE DEGLI OPERATORI RELAZIONALI

---

- Si considerano i seguenti operatori:
  - Proiezione
  - Selezione
  - Raggruppamento
  - Join
- Un operatore può essere realizzato con algoritmi diversi, codificati in opportuni **operatori fisici**.

## OPERATORI FISICI

---

- Gli algoritmi per realizzare gli operatori relazionali si codificano in opportuni operatori fisici.
  - Ad esempio **TableScan (R)**, è l'operatore fisico per la scansione di R.
- Ogni operatore fisico è un **iteratore**, un oggetto con metodi
  - *open*,
  - *next*,
  - *isDone*,
  - *reset*
  - *close*
- realizzati usando gli operatori della macchina fisica, con *next* che ritorna un record.
- Come esempio di operatori fisici prenderemo in considerazione quelli del sistema JRS e poi vedremo come utilizzarli per descrivere un algoritmo per eseguire un'interrogazione SQL (**piano di accesso**). 

## Interfaccia a iteratore

---

- I DBMS definiscono gli operatori mediante un'interfaccia a "iteratore", i cui metodi principali sono:
  - **open**: inizializza lo stato dell'operatore, **alloca buffer** per gli input e l'output, **richiama ricorsivamente open sugli operatori figli**; viene anche usato per **passare argomenti** (ad es. la condizione che un operatore Filter deve applicare)
  - **next**: usato per **richiedere** un'altra tupla del risultato dell'operatore; l'implementazione di questo metodo **include next sugli operatori figli** e codice specifico dell'operatore
  - **close**: usato per **terminare l'esecuzione** dell'operatore, con conseguente **rilascio delle risorse** ad esso allocate
  - **isDone**: indica se vi sono ancora valori da leggere, in generale è booleano.

# PIANI DI ACCESSO

Un piano di accesso è un algoritmo per eseguire un'interrogazione usando gli operatori fisici disponibili.

Interrogazione:

```
SELECT    Nome
FROM      Studenti S, Esami E
WHERE     S.Matricola=E.Matricola AND
          Provincia='PI' AND Voto>25
```

Piano di accesso:

Le foglie sono le tabelle ed i nodi interni specificano le modalità con cui gli accessi alle tabelle e le operazioni relazionali sono effettuate

Project({“Nome”})

NestedLoop(“S.Matricola=E.Matricola”)

Filter(“Provincia = ‘Pi’ ”)

TableScan(“Studenti”)

Filter(“Voto >25”)

TableScan(“Esami”)

# OPERATORI LOGICI E FISICI

| Operatore logico | Operatore fisico                                                                                                  |
|------------------|-------------------------------------------------------------------------------------------------------------------|
| $R$              | <i>TableScan (R)</i><br>per la scansione di R;                                                                    |
|                  | <i>IndexScan (R, Idx)</i><br>per la scansione di R con l'indice Idx;                                              |
|                  | <i>SortScan (R, <math>\{A_i\}</math>)</i><br>per la scansione di R ordinata sugli $\{A_i\}$ ;                     |
| $\pi_{\{A_i\}}$  | <i>Project (O, <math>\{A_i\}</math>)</i><br>per la proiezione dei record di O senza l'eliminazione dei duplicati; |
|                  | <i>Distinct (O)</i><br>per eliminare i duplicati dei record ordinati di O;                                        |

# OPERATORI LOGICI E FISICI (cont.)

Operatore logico

 $\sigma_{\psi}$  $\tau_{\{A_i\}}$ 

Operatore fisico prendono O?

Quale è la differenza fra gli operatori che prendono R e quelli che prendono O?

**Filter (O,  $\psi$ )**

per la restrizione senza indici dei record di O;

**IndexFilter (R, Idx,  $\psi$ )**

per la restrizione con indice dei record di R;

**Sort (O,  $\{A_i\}$ )**

per ordinare i record di O sugli  $\{A_i\}$ , per valori crescenti;

# OPERATORI LOGICI E FISICI (cont.)

---

Operatore logico

Operatore fisico

$\{A_i\} \gamma \{f_i\}$

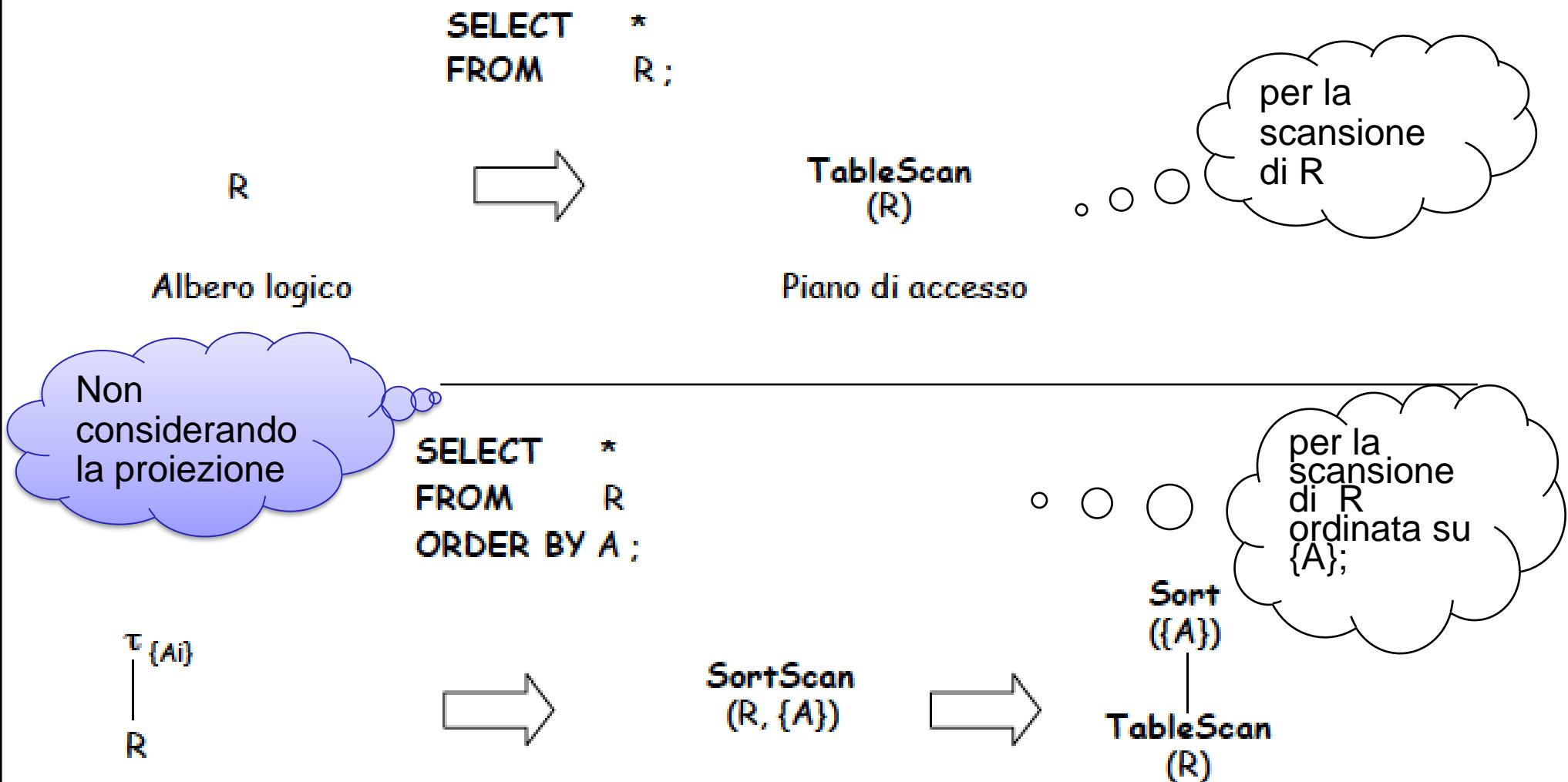
**GroupBy ( $O$ ,  $\{A_i\}$ ,  $\{f_i\}$ )**  
per raggruppare i record di  $O$  sugli  $\{A_i\}$  usando le funzioni di aggregazione in  $\{f_i\}$ .

- Nell'insieme  $\{f_i\}$  vi sono le funzioni di aggregazione presenti nella SELECT e nella HAVING.
- L'operatore restituisce record con attributi gli  $\{A_i\}$  e le funzioni in  $\{f_i\}$ .
- I record di  $O$  sono ordinati sugli  $\{A_i\}$ ;

# OPERATORI LOGICI E FISICI (cont.)

| Operatore logico   | Operatore fisico                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                    | <p><b>NestedLoop</b> (<math>O_E</math>, <math>O_I</math>, <math>\psi_J</math>)<br/>per la giunzione con il <i>nested loop</i> e <math>\psi_J</math> la condizione di giunzione;</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|                    | <p><b>PageNestedLoop</b> (<math>O_E</math>, <math>O_I</math>, <math>\psi_J</math>)<br/>per la giunzione con il <i>page nested loop</i>;</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| $\bowtie_{\psi_J}$ | <p><b>IndexNestedLoop</b> (<math>O_E</math>, <math>O_I</math>, <math>\psi_J</math>)<br/>per la giunzione con il <i>index nested loop</i>. L'operando interno <math>O_I</math> è un <i>IndexFilter</i>(<math>R</math>, <math>Idx</math>, <math>\psi_J</math>) oppure <i>Filter</i> (<math>O, \psi'</math>): con <math>O</math> un <i>IndexFilter</i>(<math>R</math>, <math>Idx</math>, <math>\psi_J</math>);<br/>per ogni record <math>r</math> di <math>O_E</math>, la condizione <math>\psi_J</math> dell'<i>IndexFilter</i> è quella di giunzione con gli attributi di <math>O_E</math> sostituiti dai valori in <math>r</math>.</p> |
|                    | <p><b>SortMerge</b> (<math>O_E</math>, <math>O_I</math>, <math>\psi_J</math>)<br/>per la giunzione con il <i>sort-merge</i>, con i record di <math>O_E</math> e <math>O_I</math> ordinati sugli attributi di giunzione.</p>                                                                                                                                                                                                                                                                                                                                                                                                            |

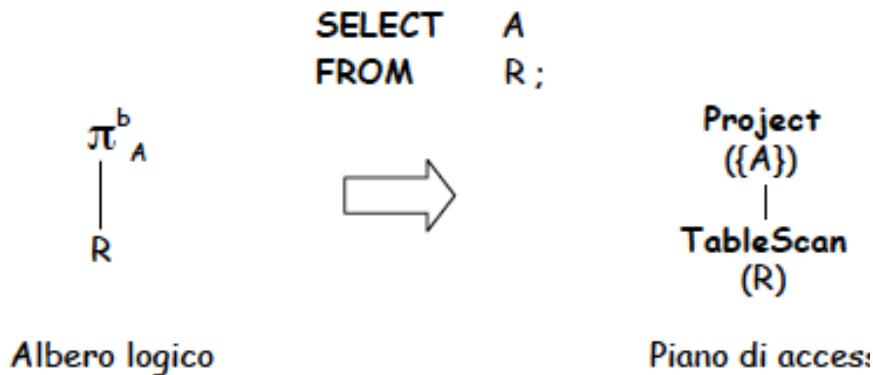
## Esempi: tableScan e SortScan



# OPERATORI FISICI PER LA PROIEZIONE

Operatori per  $\pi^b_{\{Ai\}}$  e  $\pi_{\{Ai\}}$

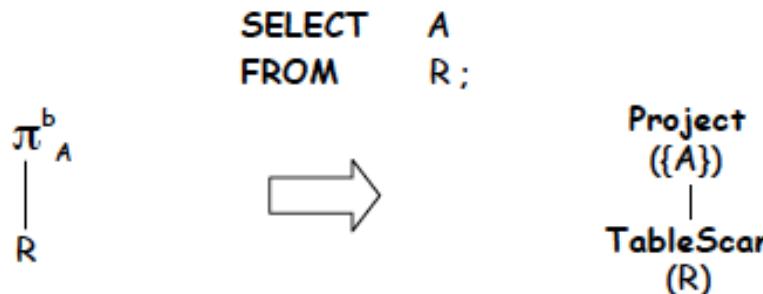
- Project ( $O, \{Ai\}$ ): per la proiezione dei record di  $O$  senza l'eliminazione dei duplicati;



# OPERATORI FISICI PER LA PROIEZIONE

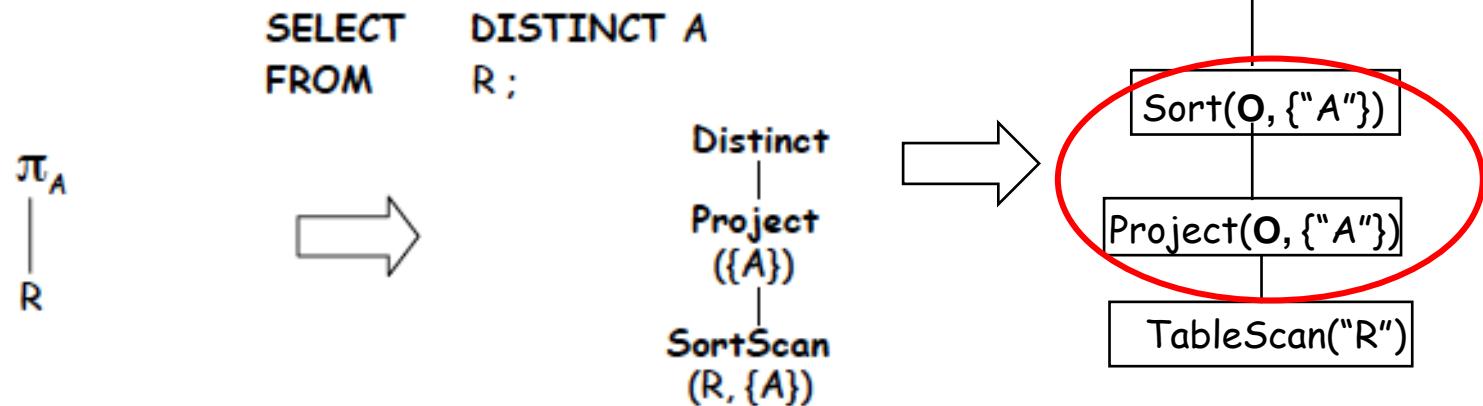
Operatori per  $\pi^b_{\{Ai\}}$  e  $\pi_{\{Ai\}}$

- Project ( $O, \{Ai\}$ ): per la proiezione dei record di  $O$  senza l'eliminazione dei duplicati;



Albero logico

- Distinct ( $O$ ): per eliminare i duplicati dei record ordinati di  $O$ ;



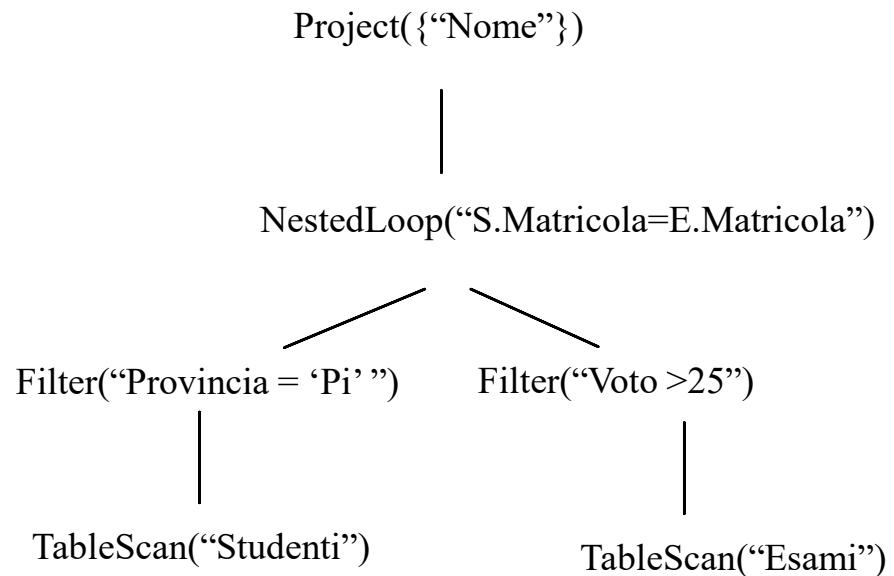
# PIANI DI ACCESSO

Un piano di accesso è un algoritmo per eseguire un'interrogazione usando gli operatori fisici disponibili.

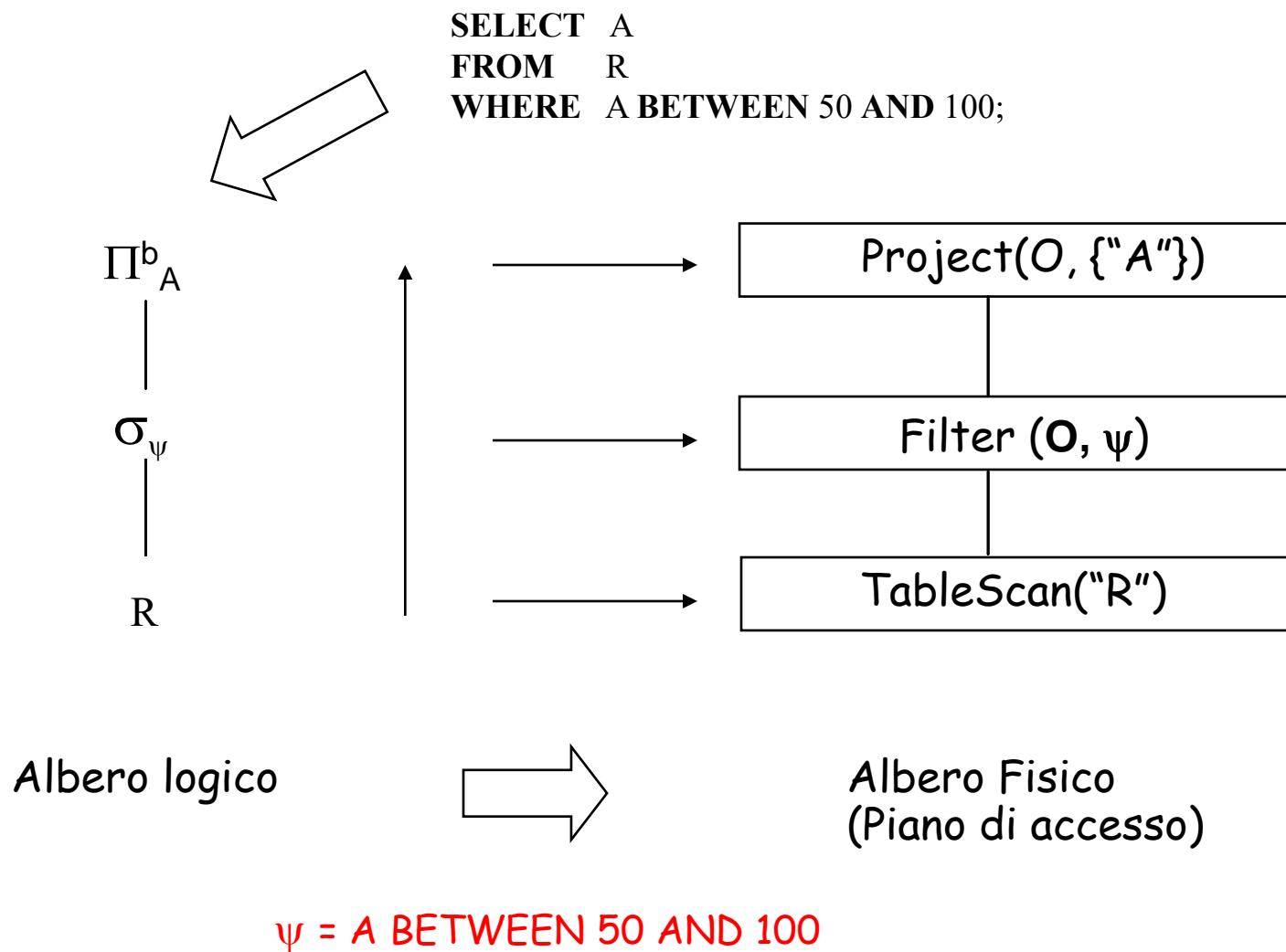
Interrogazione:

```
SELECT  Nome
FROM    Studenti S, Esami E
WHERE   S.Matricola=E.Matricola AND
        Provincia='PI' AND Voto>25
```

Piano di accesso:

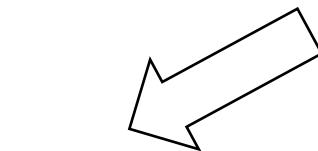


# 1) ESEMPIO DI PIANO DI ACCESSO: SFW

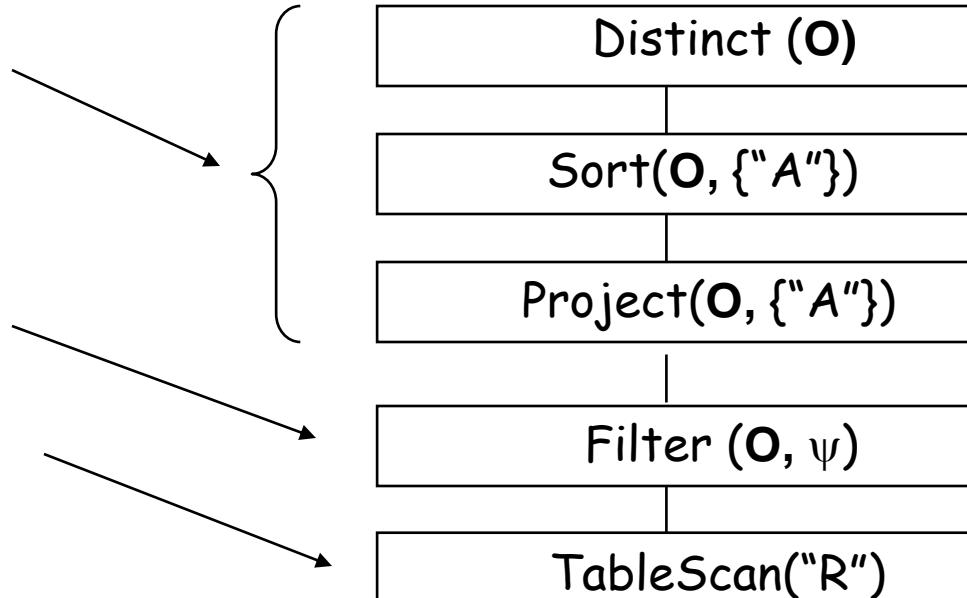


## 2) ESEMPIO DI PIANO DI ACCESSO: DISTINCT

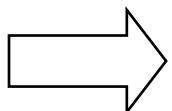
```
SELECT DISTINCT A  
FROM R  
WHERE A BETWEEN 50 AND 100;
```



$\Pi_A$   
 $\sigma_{\psi}$   
R

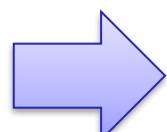


Albero logico

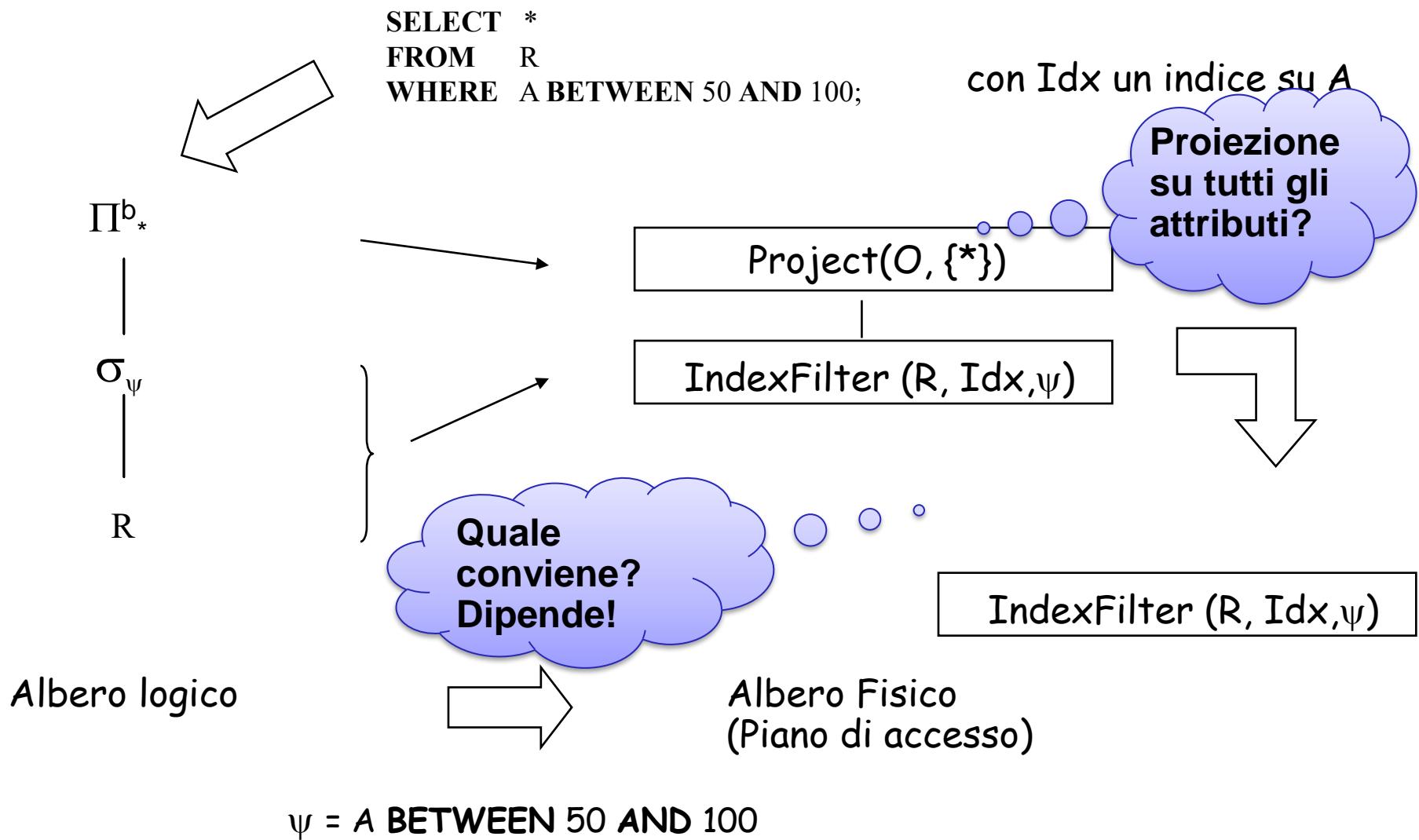


Albero Fisico  
(Piano di accesso)

$\psi = A \text{ BETWEEN } 50 \text{ AND } 100$



### 3) ESEMPIO DI PIANO DI ACCESSO CON INDICE

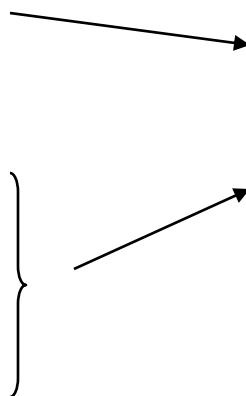


## 4) ESEMPIO DI PIANO DI ACCESSO CON INDICE

SELECT A  
FROM R  
WHERE A BETWEEN 50 AND 100;

con  $\text{Idx}$  un indice su A

$\Pi^b_A$   
|  
 $\sigma_{\psi}$   
|  
R

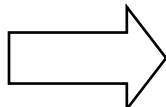


Project( $O, \{"A"\}$ )

IndexFilter ( $R, \text{Idx}, \psi$ )

IndexOnlyFilter ( $R, \text{Idx}, \psi$ )

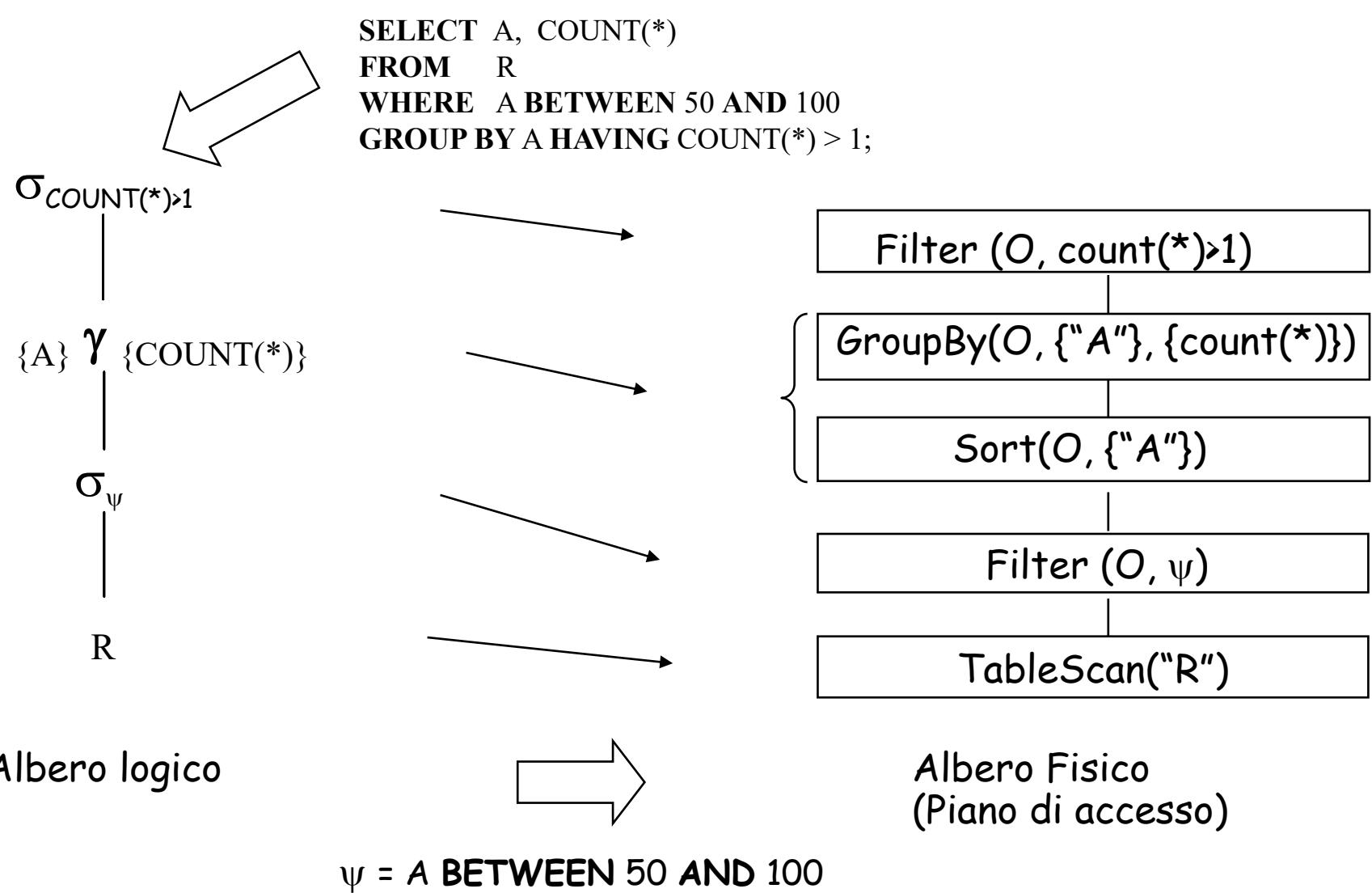
Albero logico



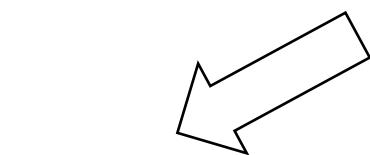
Albero Fisico  
(Piano di accesso)

$\psi = A \text{ BETWEEN } 50 \text{ AND } 100$

## 5) ESEMPIO DI PIANO DI ACCESSO: GROUP BY

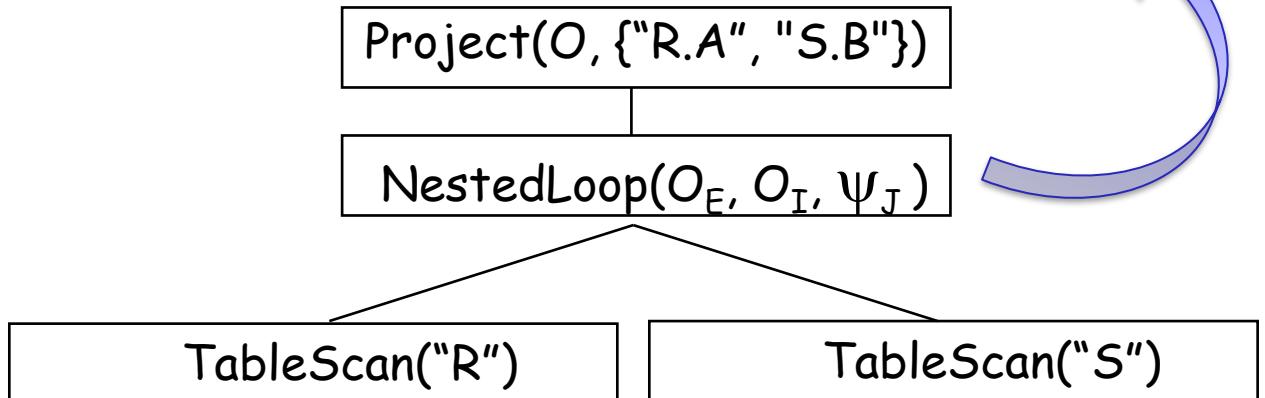
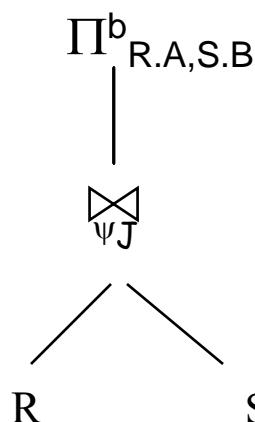


## 6) ESEMPIO DI PIANO DI ACCESSO: GIUNZIONE SENZA INDICE



```
SELECT R.A, S.B  
FROM   R , S  
WHERE  R.A = S.B;
```

```
foreach record r in R do  
foreach record s in S do  
  if ri = sj then  
    aggiungi <r, s> al risultato
```

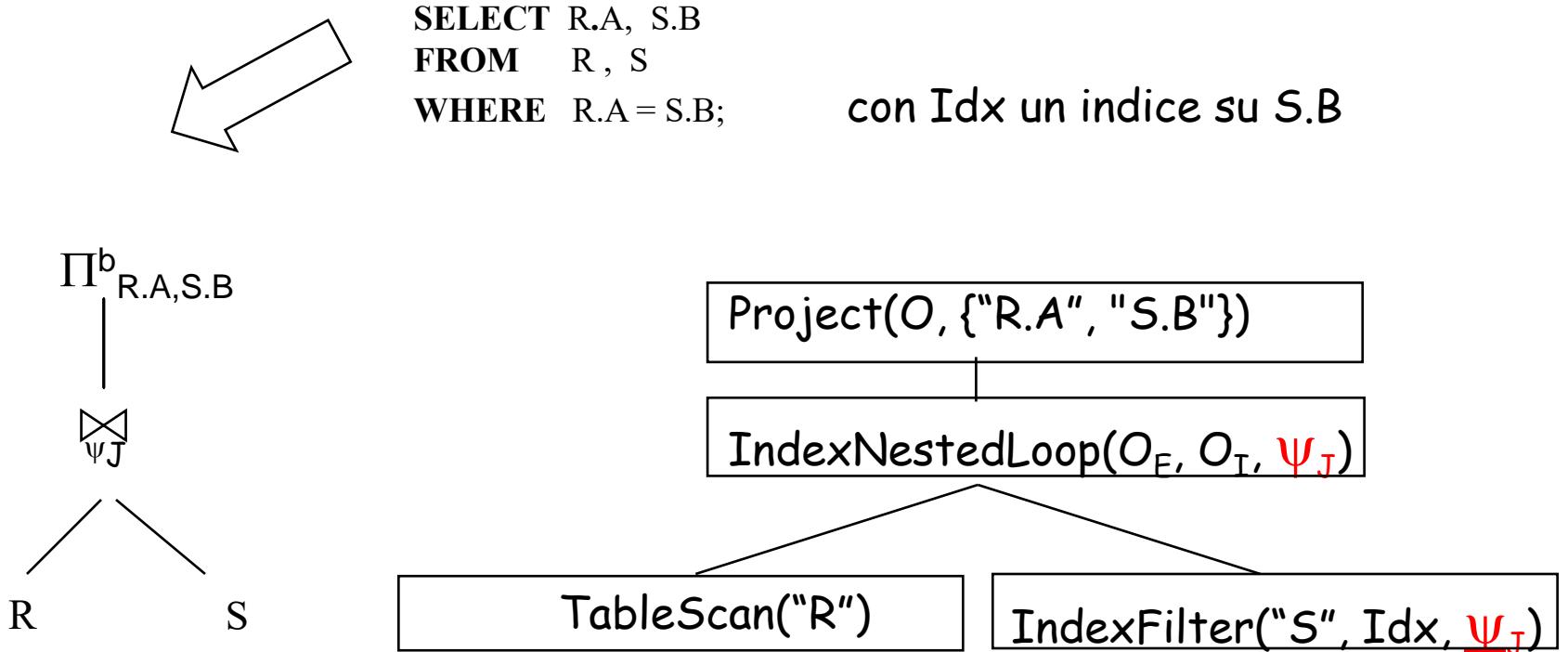


Albero logico

Albero Fisico  
(Piano di accesso)

$$\Psi_J = R.A = S.B$$

## 6) ESEMPIO DI PIANO DI ACCESSO: GIUNZIONE CON INDICE



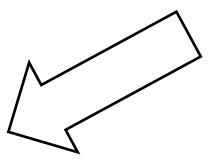
Albero logico

Albero Fisico  
(Piano di accesso)

$$\Psi_J = R.A = S.B$$

## 6) ESEMPIO DI PIANO DI ACCESSO: GIUNZIONE CON INDICE

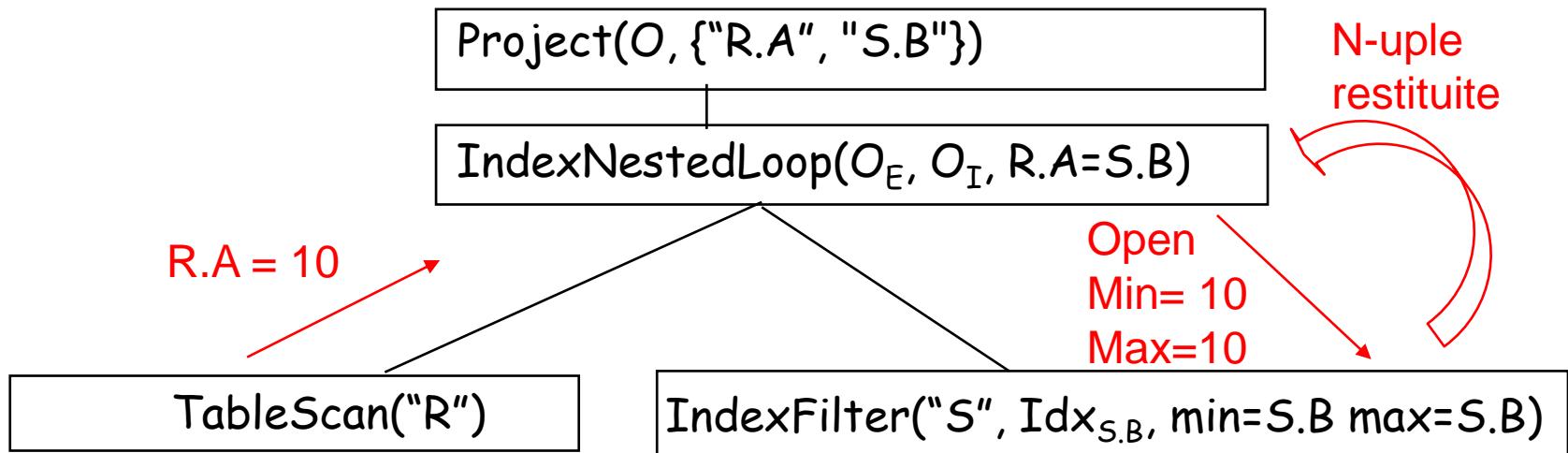
SELECT R.A, S.B  
FROM R, S  
WHERE R.A = S.B;      con Idx un indice su S.B



$\Pi^b_{R.A, S.B}$

$\bowtie_J$

R      S  
Albero logico



Albero Fisico  
(Piano di accesso)

$\Psi_J = R.A = S.B$

## 6) ESEMPIO DI PIANO DI ACCESSO: GIUNZIONE CON INDICE

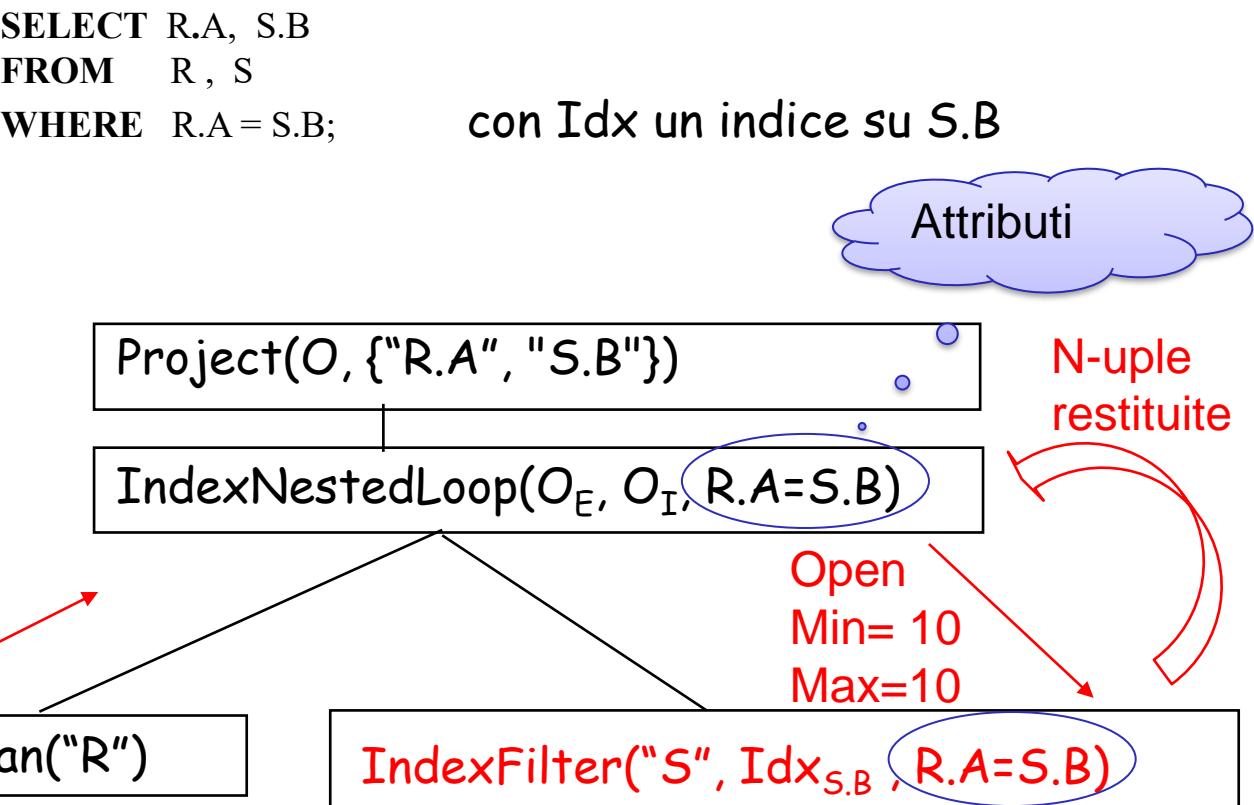
SELECT R.A, S.B  
FROM R, S  
WHERE R.A = S.B;

con Idx un indice su S.B

$\Pi^b_{R,A,S,B}$

$\bowtie_J$

R      S  
Albero logico

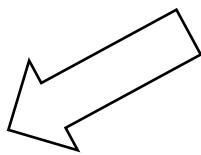


$$\Psi_J = R.A = S.B$$

Albero Fisico  
(Piano di accesso)

## 6) ESEMPIO DI PIANO DI ACCESSO: GIUNZIONE CON INDICE

SELECT R.A, S.B  
FROM R, S  
WHERE R.A = S.B;      con Idx un indice su S.B



$\Pi^b_{R.A,S.B}$

$\bowtie_J$

R      S

Albero logico

Project( $O, \{"R.A", "S.B"\}$ )

IndexNestedLoop( $O_E, O_I, R.A=S.B$ )

IndexFilter("R", Idx<sub>R.A</sub>, R.A=S.B)

ERRORE!!

Albero Fisico  
(Piano di accesso)

## ALTRI ESERCIZI

1) `SELECT A  
FROM R  
WHERE A BETWEEN 50 AND 100  
ORDER BY A;`

2) `SELECT DISTINCT A  
FROM R  
WHERE A BETWEEN 50 AND 100  
ORDER BY A;`

3) `SELECT DISTINCT A  
FROM R  
WHERE A BETWEEN 50 AND 100  
ORDER BY A;`

ed esiste un indice su A

4) `SELECT DISTINCT A  
FROM R  
WHERE A = 100  
ORDER BY A;`

- 1) A è una chiave
- 2) A non è una chiave

5) `SELECT A, COUNT(*)  
FROM R  
WHERE A > 100  
GROUP BY A;`

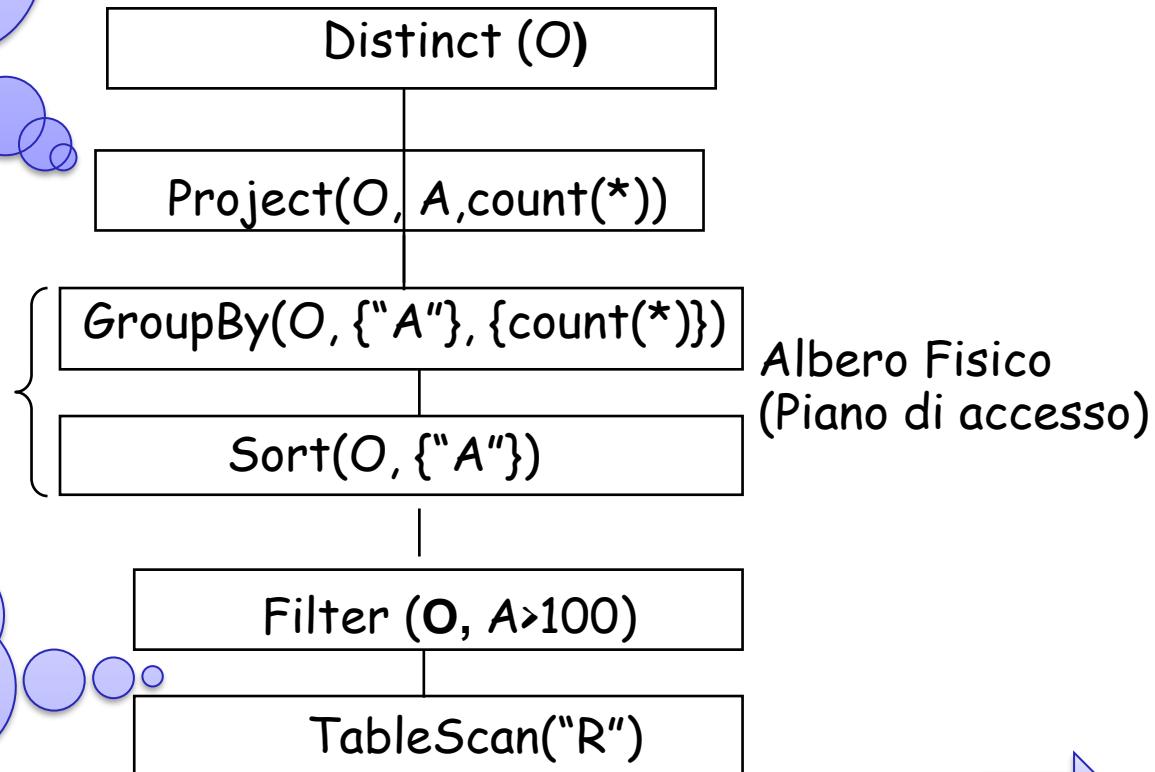
6) `SELECT DISTINCT A, COUNT(*)  
FROM R  
WHERE A > 100  
GROUP BY A;`

## Soluzione Esercizio 6 (senza indici)

$\Pi_{A, \text{count}(*)}$   
 $\{A\} \gamma \{\text{COUNT}(*)\}$   
 $\sigma_{A > 100}$   
R  
Albero logico

Non serve perché la group by lascia in output solo A e count(\*)

```
SELECT DISTINCT A, COUNT(*)  
FROM R  
WHERE A > 100  
GROUP BY A;
```



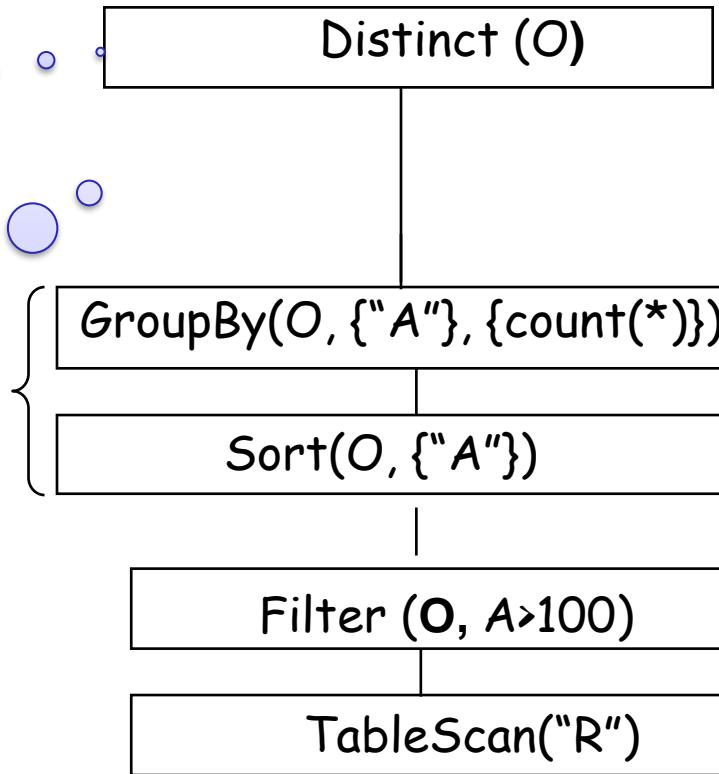
Avrei potuto fare il sort subito dopo tableScan?

## Soluzione Esercizio 6 (senza indici) Cont.

```
SELECT DISTINCT A, COUNT(*)
FROM R
WHERE A > 100
GROUP BY A;
```

$\Pi_{A, \text{count}(*)}$   
 $\{\text{A}\} \gamma \{\text{COUNT(*)}\}$   
 $\sigma_{A>100}$   
R  
Albero logico

Serve?  
 Bisogna fare  
 un sort su A  
e su Count  
per  
eliminare i  
duplicati?



Albero Fisico  
(Piano di accesso)

## Soluzione Esercizio 6 (con indici) Cont.

```
SELECT DISTINCT A, COUNT(*)  
FROM R  
WHERE A > 100  
GROUP BY A;
```

$\Pi_{A, \text{count}(*)}$

$\{\text{A}\} \gamma \{\text{COUNT}(*)\}$

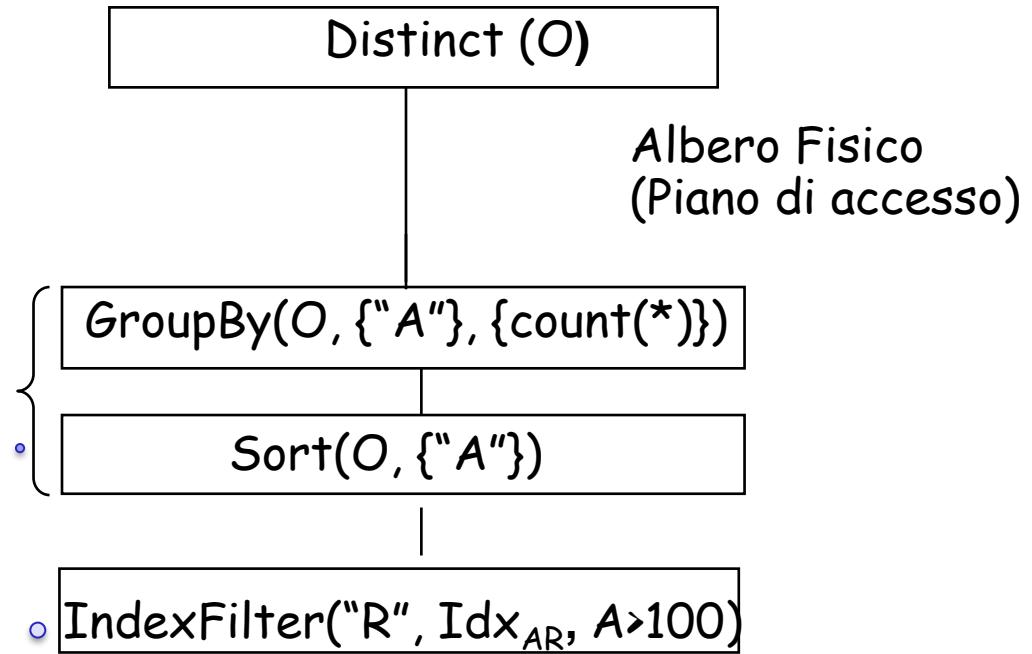
$\sigma_{A>100}$

R

Albero logico

Serve?

Bisogna specificare gli indici!!!



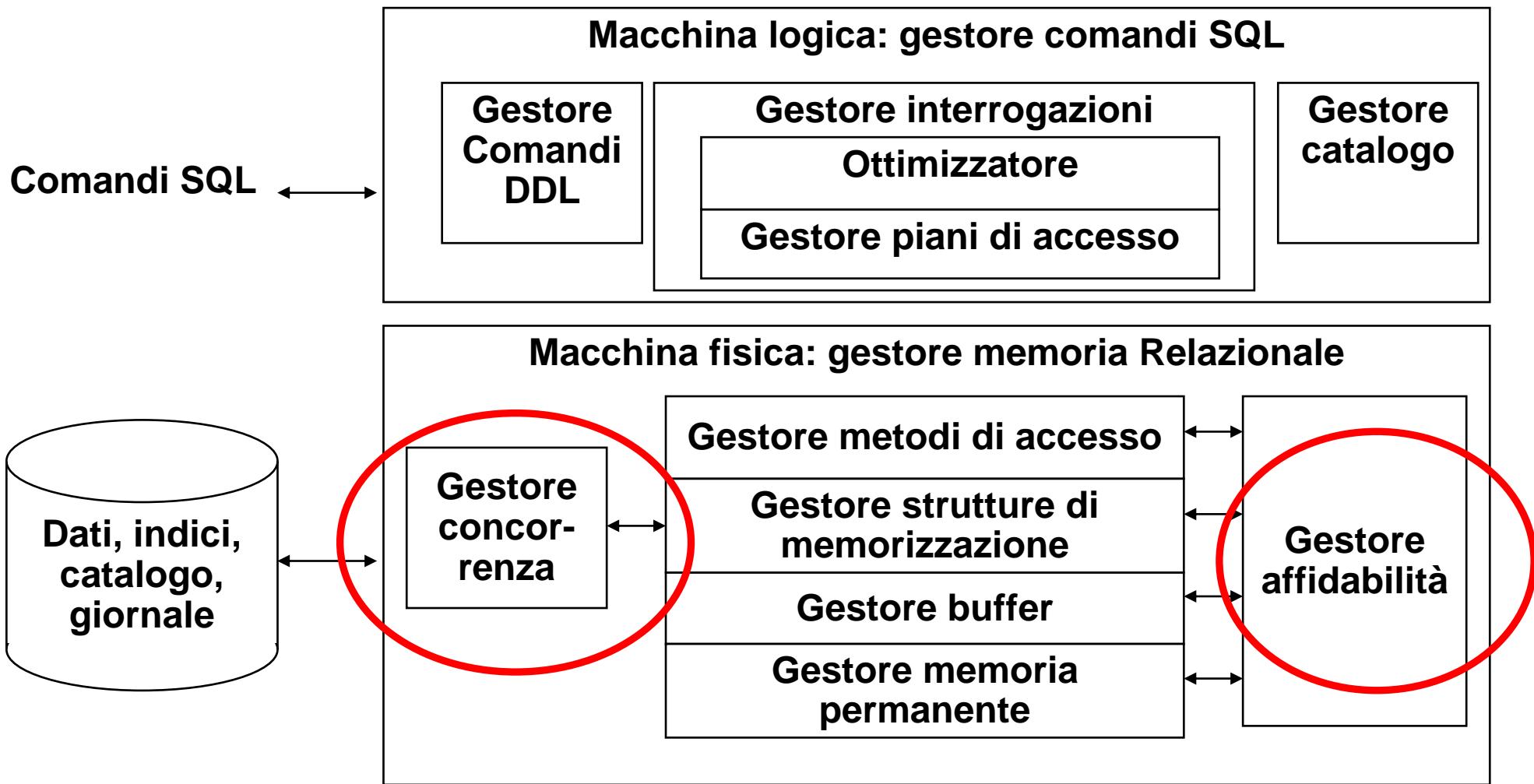
---

# **PARTE IV**

# **TRANSAZIONI**

# ARCHITETTURA DEI DBMS

---



Esempio. Gestione ordini su un sito di  
**ecommerce**  
(struttura del DB semplificata)

**ITEM(Codice, Descrizione, Prezzo, Quantita)**

**ORDINE(Id, Data, Ordinante, codOrdinato)**

```
SET NumItem=(SELECT COUNT(*) FROM ITEM WHERE
              (Codice=CodiceScelto));
IF (NumItem > 0) THEN
    UPDATE ITEM SET Quantita=Quantita-1 WHERE
              (Codice=CodiceScelto));
    INSERT INTO ORDINE(Data,Ordinante, codOrdinato) VALUES
              (NOW(), NomeOrdinante, CodiceScelto);
END IF;
```

**Il sistema va in crash in questo punto!**

## Esempio. Gestione ordini su un sito di ecommerce

(struttura del DB semplificata)

**ITEM(Codice, Descrizione, Prezzo, Quantita)**

**ORDINE(Id, Data, Ordinante, codOrdinato)**

```
SET NumItem=(SELECT COUNT(*) FROM ITEM WHERE
(Codice=CodiceScelto));
IF (NumItem > 0) THEN
    UPDATE ITEM SET Quantita=Quantita-1 WHERE
(Codice=CodiceScelto));
    INSERT INTO ORDINE(Data,Ordinante,ItemOrdinato) VALUES
(NOW(), NomeOrdinante, CodiceScelto);
END IF;
```

**Due ordini in contemporanea eseguono la query**

## Gestione delle Transazioni

---

- Le **transazioni** rappresentano l'unità di lavoro elementare (insiemi di istruzioni SQL) che **modificano** il contenuto di una base di dati.
- Sintatticamente un transazione è contornata dai comandi **begin transaction** (e **end transaction**); all'interno possono comparire i comandi di **commit work** e **rollback work**.

```
begin transaction  
update SalariImpiegati  
set conto=conto*1.2  
where (CodiceImpiegato = 123)  
commit work
```

```
begin transaction  
update SalariImpiegati  
set conto=conto-10  
where (CodiceImpiegato = 123)  
if conto >0 commit work;  
else rollback work
```

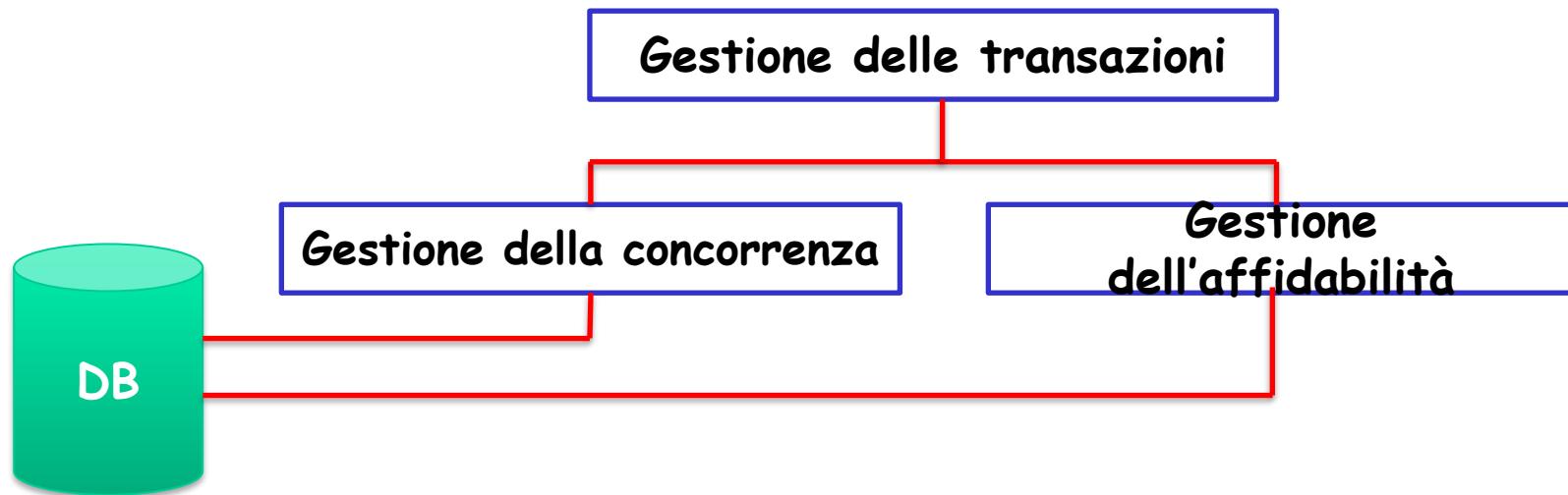
Le transazioni sono comprese tra una BEGIN transaction ed una commit/rollback

## PROPRIETA' ACID DELLE TRANSAZIONI

- **Atomicità** → La transazione deve essere eseguita con la regola del "tutto o niente".
- **Consistenza** → La transazione deve lasciare il DB in uno stato **consistente**, eventuali vincoli di integrità non devono essere violati.
- **Isolamento** → L'esecuzione di una transazione deve essere **indipendente** dalle altre.
- **Persistenza (Durability)** → L'effetto di una transazione che ha fatto commit work non deve essere perso.

# Funzioni del DBMS (in breve...)

- **Gestione dei dati**: cura la memorizzazione permanente dei dati ed il loro accesso
- **Gestione del buffer**: cura il trasferimento dei dati da memoria di massa a memoria centrale, e il caching dei dati in memoria centrale
- **Ottimizzazione delle interrogazioni**: seleziona il piano di accesso di costo ottimo con cui valutare ciascuna interrogazione



---

# GESTIONE DELL'AFFIDABILITÀ

## Cos'è una transazione?

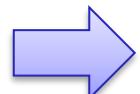
---

- Una **transazione** è un'unità logica di elaborazione che corrisponde a una serie di operazioni fisiche elementari (lettura/scritture) sul DB
- Esempi:
  - Trasferimento di una somma da un conto corrente ad un altro  
**UPDATE CC**  
**SET Saldo = Saldo - 50**  
**WHERE Conto = '123'**
  - Aggiornamento degli stipendi degli impiegati di una sede  
**UPDATE Imp**  
**SET Stipendio = 1.1\*Stipendio**  
**WHERE Sede = 'PISA'**

# GESTIONE DELLE TRANSAZIONI

---

- Una **funzionalità** essenziale di un DBMS è la **protezione dei dati** da malfunzionamenti e da interferenze dovute all'accesso contemporaneo ai dati da parte di più utenti.
- La transazione per il programmatore:
  - Una transazione è un programma sequenziale costituito da operazioni che il sistema deve eseguire garantendo:
  - **Atomicità, Consistenza, Serializzabilità , Persistenza**
  - **(Atomicity, Consistency, Isolation, Durability - ACID)**



## Le transazioni: atomicità e Persistenza

---

- **Definizione** Una **transazione** è una sequenza di azioni di lettura e scrittura in memoria permanente e di elaborazioni di dati in memoria temporanea.
- **Gestore dell'affidabilità:**
  - **Atomicità:** Le transazioni che terminano prematuramente (*aborted transactions*) sono trattate dal sistema come se non fossero mai iniziate; pertanto eventuali loro effetti sulla base di dati sono annullati.
  - **Persistenza:** Le modifiche sulla base di dati di una transazione terminata normalmente sono permanenti, cioè non sono alterabili da eventuali malfunzionamenti.

## Le transazioni: Serializzabilità

---

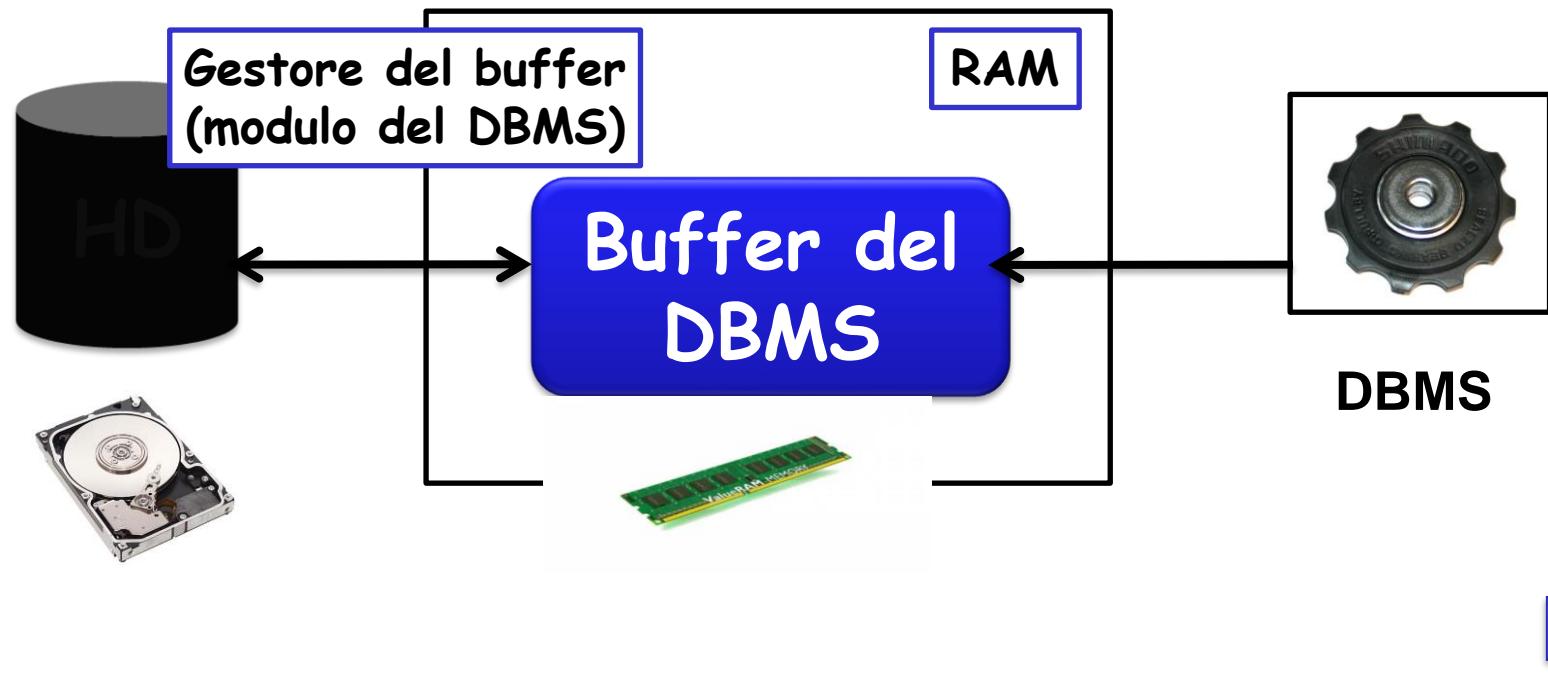
- **Definizione** Una **transazione** è una sequenza di azioni di lettura e scrittura in memoria permanente e di elaborazioni di dati in memoria temporanea:

Gestore della Concorrenza:

- **Serializzabilità:** Nel caso di esecuzioni concorrenti di più transazioni, l'effetto complessivo è quello di una esecuzione seriale.

# Gestione delle Transazioni

- Per aumentare l'efficienza prestazionale, tutti i DBMS utilizzano un buffer temporaneo di informazioni in memoria principale, il quale viene periodicamente scritto su memoria secondaria.



## LA TRANSAZIONE PER IL DBMS

---

- Una transazione può eseguire molte operazioni sui dati recuperati da una base di dati, ma al DBMS interessano solo quelle di lettura o scrittura della base di dati, indicate con  $r_i[x]$  e  $w_i[x]$ .
- Un dato letto o scritto può essere un record, un campo di un record o una pagina. Per semplicità supporremo che sia una pagina.
- Un'operazione di lettura  $r_i[x]$  comporta la lettura di una pagina nel buffer, se non già presente.
- Un'operazione di scrittura  $w_i[x]$  comporta l'eventuale lettura nel buffer di una pagina e la sua modifica nel buffer, ma non necessariamente la sua scrittura in memoria permanente. Per questa ragione, in caso di malfunzionamento, si potrebbe perdere l'effetto dell'operazione.

## TIPI DI MALFUNZIONAMENTO

---

- Fallimenti di **transazioni**: non comportano la perdita di dati in memoria temporanea né persistente (es.: violazione di vincoli, violazione di protezione, stallo)
- Fallimenti di **sistema**: comportano la perdita di dati in memoria **temporanea** ma non di dati in memoria persistente (es.: comportamento anomalo del sistema, caduta di corrente, guasti hardware sulla memoria centrale)
- **Disastri**: comportano la perdita di dati in memoria persistente (es.: danneggiamento di periferica)

**Gestore dell'affidabilità** → verifica che siano garantite le proprietà di **atomicità e persistenza** delle transazioni.

- Responsabile di implementare i comandi di:  
begin transaction, commit, rollback
- Responsabile di ripristinare il sistema dopo **malfunzionamenti software (ripresa a caldo)**
- Responsabile di ripristinare il sistema dopo **malfunzionamenti hardware (ripresa a freddo)**

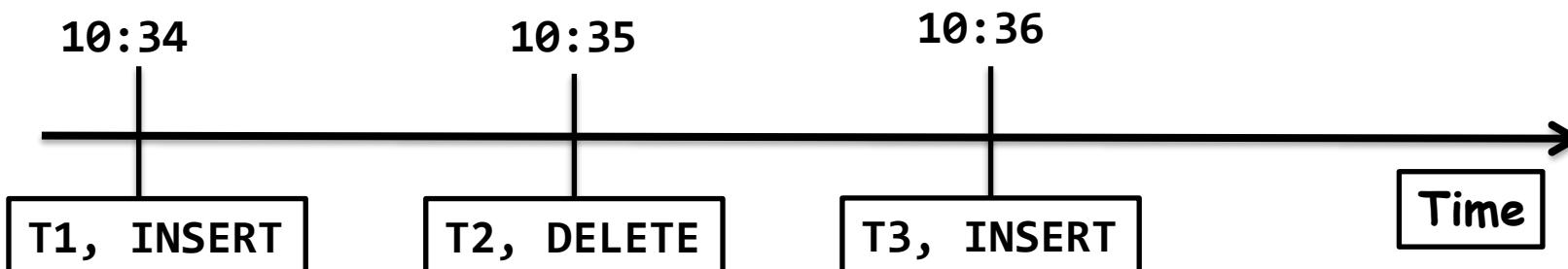
## Gestione delle Transazioni - File di LOG

- Il controllore di affidabilità utilizza un log, nel quale sono indicate tutte le operazioni svolte dal DBMS.

### LOG, struttura fisica

```
<10:34, 11/12/2019, Transaction: T1, INSERT(3,Mario, Rossi)  
INTO IMPIEGATI>  
<10:35, 11/12/2019, Transaction: T2, DELETE(3,Mario, Rossi)  
INTO IMPIEGATI>  
<10:36, 11/12/2019, Transaction: T3, INSERT(6,Maria, Bianchi)  
INTO IMPIEGATI>
```

### LOG, struttura logica



## Le primitive undo e redo

---

- Convenzioni notazionali:
  - data una transazione T, indicheremo con  $B(T)$ ,  $C(T)$  e  $A(T)$  i record di begin, commit e abort relativi a T, rispettivamente, e con  $U(T, O, BS, AS)$ ,  $I(T, O, AS)$  e  $D(T, O, BS)$  i record di update, insert e delete, rispettivamente su un oggetto O, dove  $BS$  è before state and  $AS$  è after state.
- I record del log associati ad una transazione, consentono di disfare e rifare le corrispondenti azioni sulla base di dati:
  - primitive di **undo**: per disfare un'azione su un oggetto O, è sufficiente ricopiare in O il valore BS (l'insert viene disfatto cancellando l'oggetto O)
  - primitiva di **redo**: per rifare un'azione su un oggetto O, è sufficiente ricopiare in O il valore AS (il delete viene rifatto cancellando l'oggetto O)

## Gestione delle Transazioni (file di log)

---

- Il log si presenta come un file sequenziale suddiviso in record:



- Due tipi di record:

- Record di transazione** → tengono traccia delle operazioni svolte da ciascuna transazione sul DBMS. Per ogni transazione, un record di begin (B), record di insert (I), delete (D) e update (U) e un record di commit (C) o di abort (A).
- Record di sistema** → tengono traccia delle operazioni di sistema (dump/checkpoint).

## L'operazione di dump (copia del DB)

---

- L'operazione di **dump** produce una copia completa della base di dati, effettuata in mutua esclusione con tutte le altre transazioni quando il sistema non è operativo.
- La copia viene memorizzata in memoria stabile (backup).
- Al termine del dump, viene scritto nel log un record di dump, che segnala l'avvenuta esecuzione dell'operazione in un dato istante. Il sistema riprende, quindi, il suo funzionamento normale.

# PROTEZIONE DEI DATI DA MALFUNZIONAMENTI

---

- Copia della BD (**DUMP**)
- Giornale/**log**: durante l'uso della BD, il sistema registra nel giornale/log la storia delle azioni effettuate sulla BD dal momento in cui ne è stata fatta l'ultima copia.
- Contenuto del giornale/log:
  - (T, begin);
  - Per ogni operazione di modifica:
    - la transazione responsabile;
    - il tipo di ogni operazione eseguita;
    - la nuova e vecchia versione del dato modificato:  
 $(T, write, address, oldV, newV);$
  - (T, commit) o (T, abort).

## Due Regole

---

- **REGOLE di SCRITTURA del LOG**
  - Regola Write Ahead Log (WAL) → la parte **BS** (before state) di ogni record di log deve essere scritta **prima che la corrispondente operazione venga effettuata** nella base di dati.

## Due Regole

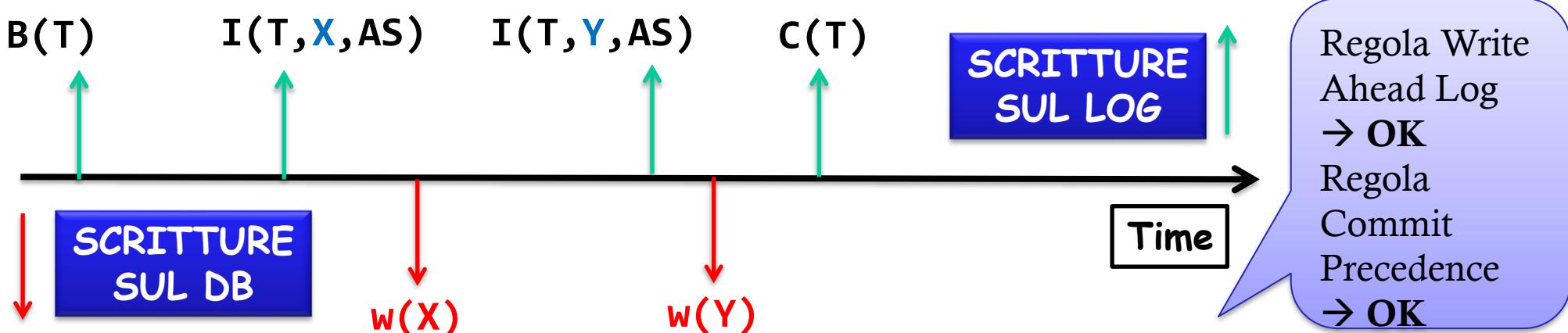
---

- **REGOLE di SCRITTURA del LOG**
  - Regola Write Ahead Log (WAL) → la parte **BS** (before state) di ogni record di log deve essere scritta **prima che la corrispondente operazione venga effettuata nella base di dati.**
  - Regola di Commit Precedence → la parte **AS** (after state) di ogni record di log deve essere scritta nel log **prima di effettuare il commit della transazione.**

# Regola Write Ahead Log

- **REGOLE di SCRITTURA del LOG**

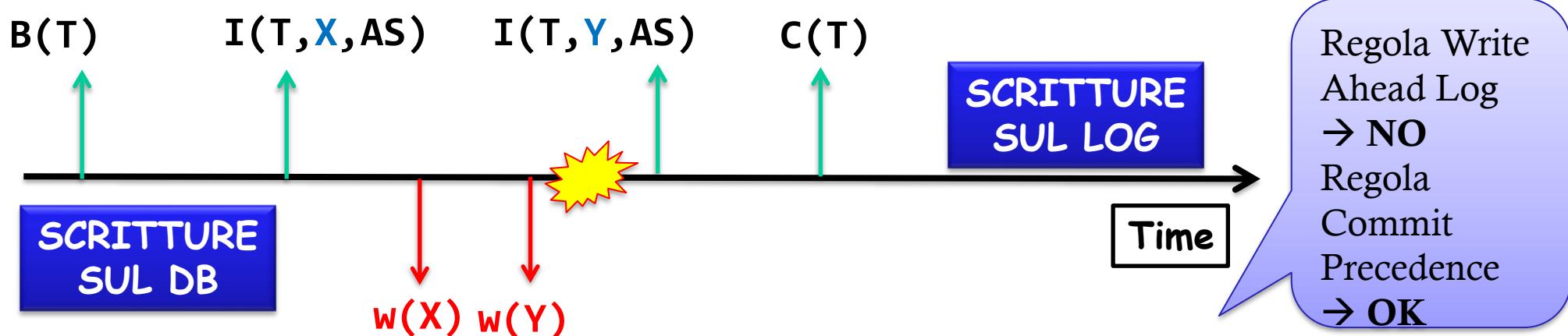
- Regola Write Ahead Log (WAL) → la parte **BS** (before state) di ogni record di log deve essere scritta **prima che la corrispondente operazione venga effettuata** nella base di dati.
- Regola di Commit Precedence → la parte **AS** (after state) di ogni record di log deve essere scritta nel log **prima di effettuare il commit della transazione**.



# Regola Write Ahead Log

- **REGOLE di SCRITTURA del LOG**

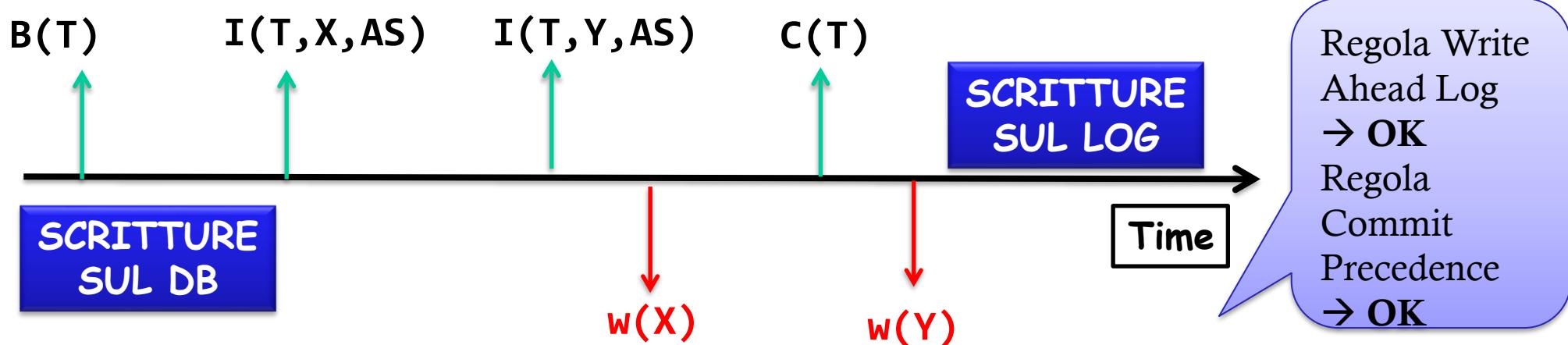
- Regola Write Ahead Log (WAL) → la parte **BS** (before state) di ogni record di log deve essere scritta **prima che la corrispondente operazione venga effettuata** nella base di dati.
- Regola di Commit Precedence → la parte **AS** (after state) di ogni record di log deve essere scritta nel log **prima di effettuare il commit della transazione**.



# Regola Write Ahead Log

- **REGOLE di SCRITTURA del LOG**

- Regola Write Ahead Log (WAL) → la parte **BS** (before state) di ogni record di log deve essere scritta **prima che la corrispondente operazione venga effettuata** nella base di dati.
- Regola di Commit Precedence → la parte **AS** (after state) di ogni record di log deve essere scritta nel log **prima di effettuare il commit della transazione**.



# GESTIONE DELL'AFFIDABILITA'

---

- Gli algoritmi si differenziano a seconda del modo in cui si trattano le scritture sulla BD e la terminazione delle transazioni
  - Disfare-Rifare ←
  - Disfare-NonRifare
  - NonDisfare-Rifare
  - NonDisfare-NonRifare
- Ipotesi: Le scritture nel giornale vengono portate subito nella memoria permanente!

## DISFARE

---

- Quando si portano le modifiche nella BD ?
  - Politica della **modifica libera** : le modifiche possono essere portate nella BD stabile prima che la T termini (disfare o steal).
- Regola per poter disfare: prescrittura nel giornale ("Log Ahead Rule" o "**Write Ahead Log**"):
  - se la nuova versione di una pagina rimpiazza la vecchia sulla BD stabile prima che la transazione T abbia raggiunto il punto di Commit, allora la vecchia versione della pagina deve essere portata prima sul giornale in modo permanente.

- Come si gestisce la terminazione ?
  - *Commit libero* : una T può essere considerata terminata normalmente prima che tutte le modifiche vengano riportate nella BD stabile (occorre rifare).
- Regola per poter rifare una T: ("Commit Rule")
  - *Le modifiche (nuove versioni delle pagine) di una T devono essere portate stabilmente nel giornale prima che la T raggiunga il Commit (condizione per rifare).*

## PUNTO DI ALLINEAMENTO ("CHECKPOINT")

---

- Al momento del ripristino, solo gli aggiornamenti più recenti tra quelli riportati sul giornale/log potrebbero non essere stati ancora riportati sulla base di dati. Come ottenere la certezza che non è necessario rieseguire le operazioni più vecchie?
- Periodicamente si fa un Checkpoint (CKP): si scrive la marca CKP sul giornale/log per indicare che tutte le operazioni che la precedono sono state effettivamente effettuate sulla BD.

## PUNTO DI ALLINEAMENTO ("CHECKPOINT")

---

- Un modo (troppo semplice) per fare il CKP:
  1. si sospende l'attivazione di nuove transazioni,
  2. si completano le precedenti, si allinea la base di dati (ovvero si riportano su disco tutte le pagine "sporche" dei buffer),
  3. si scrive nel giornale/log la marca CKP.
  4. Si riprende l'esecuzione delle operazioni.

## CHECKPOINT

---

- Si scrive sul giornale una marca di inizio checkpoint che riporta l'elenco delle transazioni attive (**BeginCkp**, {T<sub>1</sub>,...,T<sub>n</sub>})
- In parallelo alle normali operazioni delle transazioni, il gestore del buffer riporta sul disco tutte le pagine modificate
- Si scrive sul giornale una marca di **EndCkp**
- La marca di **EndCkp** certifica che tutte le scritture avvenute prima del **BeginCkp** ora sono sul disco.
- Le scritture avvenute tra **BeginCkp** e **EndCkp** forse sono sul disco e forse no.

## RIPRESA DAI MALFUNZIONAMENTI (disfare-rifare)

---

- Fallimenti di **transazioni**: si scrive nel giornale (T, abort) e si applica la procedura disfare.
- Fallimenti di **sistema**:
  - La BD viene ripristinata con il comando Restart (ripartenza di emergenza), a partire dallo stato al punto di allineamento, procedendo come segue:
    - **Le T non terminate vanno disfatte**
    - **Le T terminate devono essere rifatte.**
- **Disastri**: si riporta in linea la copia più recente della BD e la si aggiorna rifacendo le modifiche delle T terminate normalmente (ripartenza a freddo).



## Ripresa a caldo

---

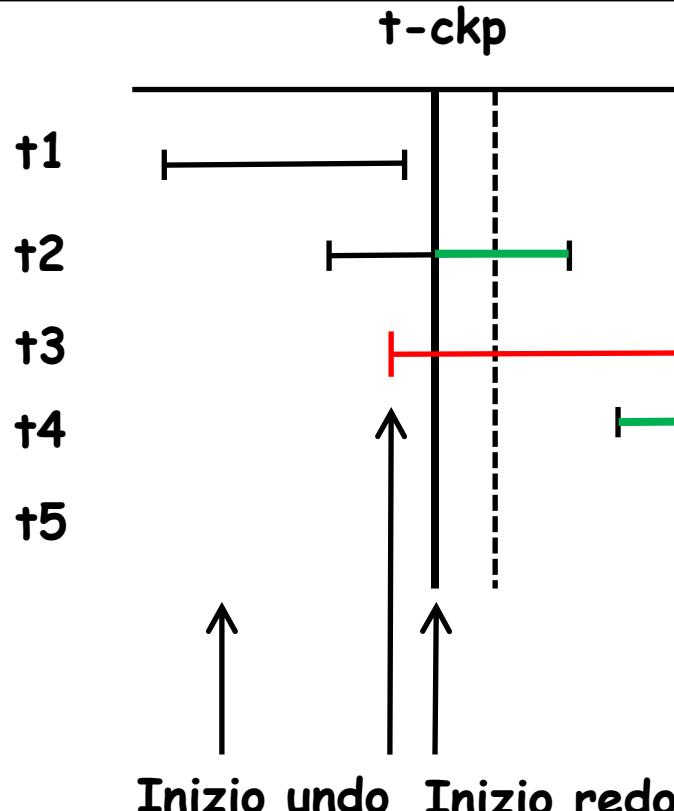
- Garantisce atomicità e persistenza delle transazioni
- Quattro fasi:
  - trovare l'ultimo checkpoint (ripercorrendo il log a ritroso)
  - costruire gli insiemi UNDO (transazioni da disfare) e REDO (transazioni da rifare)
  - ripercorrere il log all'indietro, **fino alla più vecchia azione** delle transazioni in UNDO, disfaccendo tutte le azioni delle transazioni in UNDO
  - ripercorrere il log in avanti, rifacendo tutte le azioni delle transazioni in REDO

## Ripresa a freddo

---

- Risponde ad un guasto che provoca il deterioramento della BD
  - Si ripristinano i dati a partire dal backup
  - Si eseguono le operazioni registrate sul giornale fino all'istante del guasto
  - Si esegue una ripresa a caldo

# Ripartenza dopo un fallimento



- $T_1$  va ignorata
- $T_2$  e  $T_4$  vanno rifatte
- $T_3$  e  $T_5$  vanno disfatte

$t-fail$

- trovare l'ultimo checkpoint (ripercorrendo il log a ritroso)
- costruire gli insiemi UNDO (transazioni da disfare) e REDO (transazioni da rifare)
- ripercorrere il log all'indietro, fino alla più vecchia azione delle transazioni in UNDO, disfacendo tutte le azioni delle transazioni in UNDO
- ripercorrere il log in avanti, rifacendo tutte le azioni delle transazioni in REDO

# Esercizio Applicare il protocollo rifare disfare

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), B(T6)

U(T3,O3,B5,A5), U(T5,O4,B6,A6),

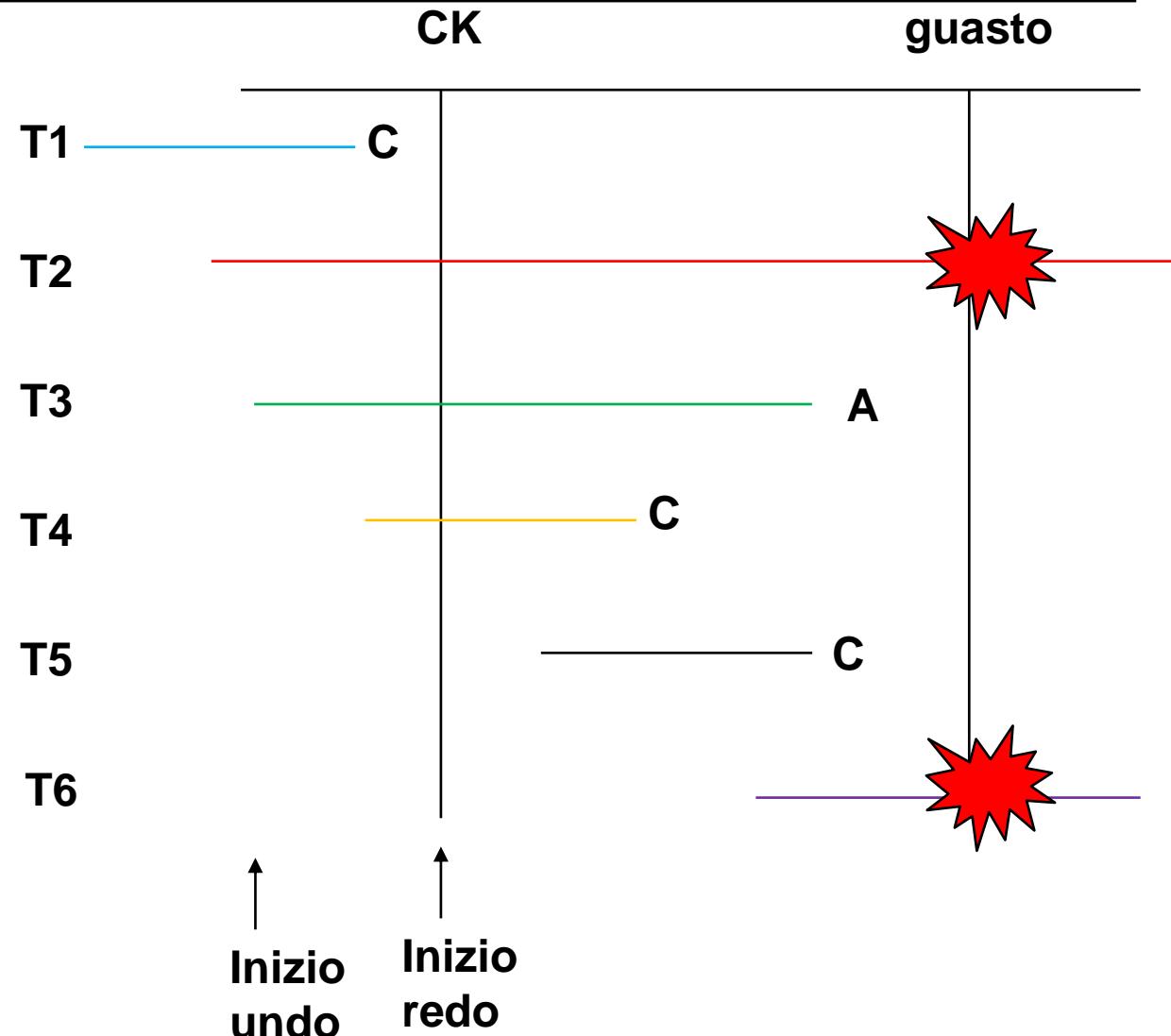
D(T3,O5,B7), U(T6, O6, B7, A7)

A(T3)

C(T5)

I(T2,O6,A8)

guasto



# Esempio di ripresa a caldo: fase 1

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), B(T6)

U(T3, O3, B5, A5), U(T5, O4, B6, A6),

D(T3, O5, B7), U(T6, O6, B7, A7)

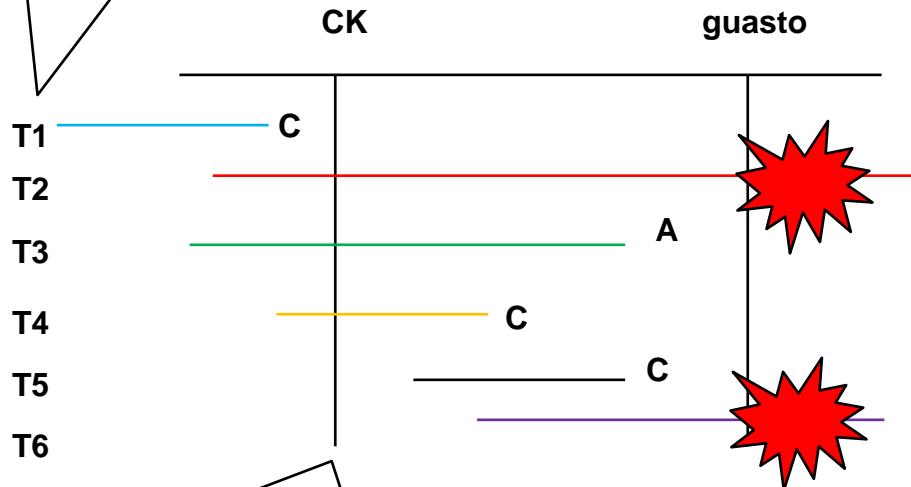
A(T3)

C(T5)

I(T2, O6, A8)

guasto

T1 fa il commit  
prima del  
checkpoint



Transazioni presenti nel record di Checkpoint:  
**{T2, T3, T4}**

Inizializziamo UNDO= {T2, T3, T4} e REDO={}

## Esempio di ripresa a caldo: fase 2

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), B(T6)

U(T3, O3, B5, A5), U(T5, O4, B6, A6),

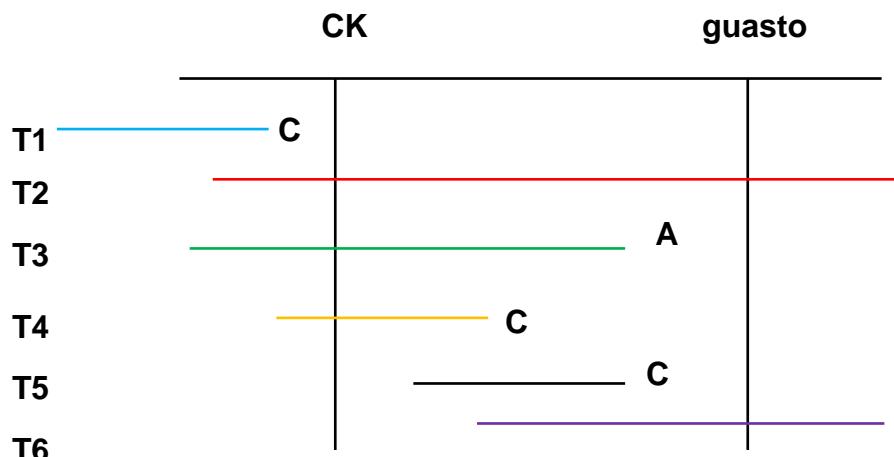
D(T3, O5, B7), U(T6, O6, B7, A7)

A(T3)

C(T5)

I(T2, O6, A8)

guasto



**UNDO = {T2, T3, T4} ?**

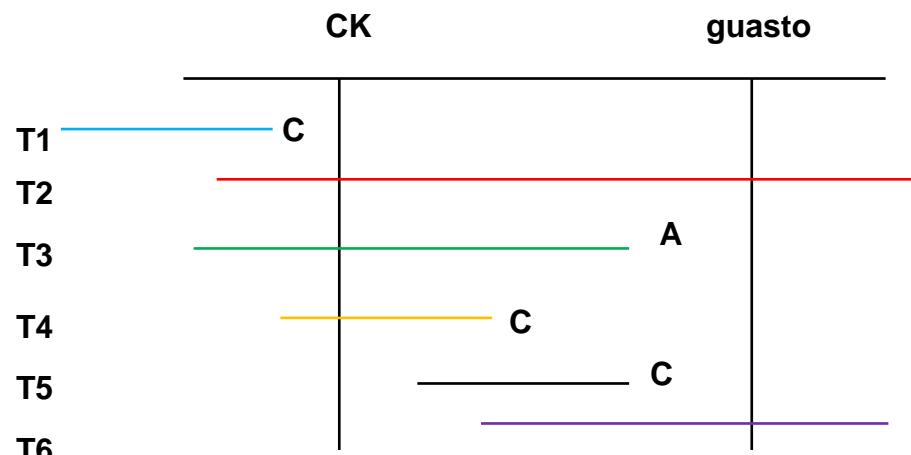
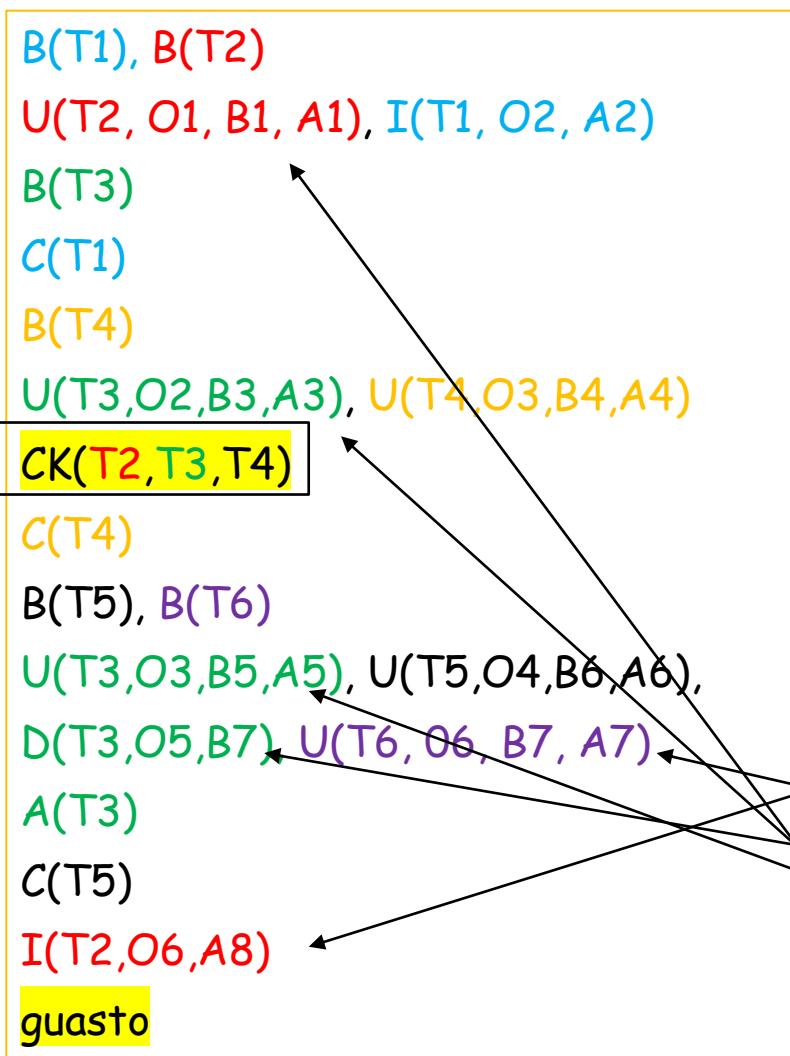
**C(T4) ⇒**  
UNDO = {T2, T3}  
REDO = {T4}

**B(T5) ⇒**  
UNDO = {T2, T3, T5}  
REDO = {T4}

**B(T6) ⇒**  
UNDO = {T2, T3, T5, T6}  
REDO = {T4}

**C(T5) ⇒**  
UNDO = {T2, T3, T6}  
REDO = {T4, T5}

# Esempio di ripresa a caldo: fase 3 - UNDO

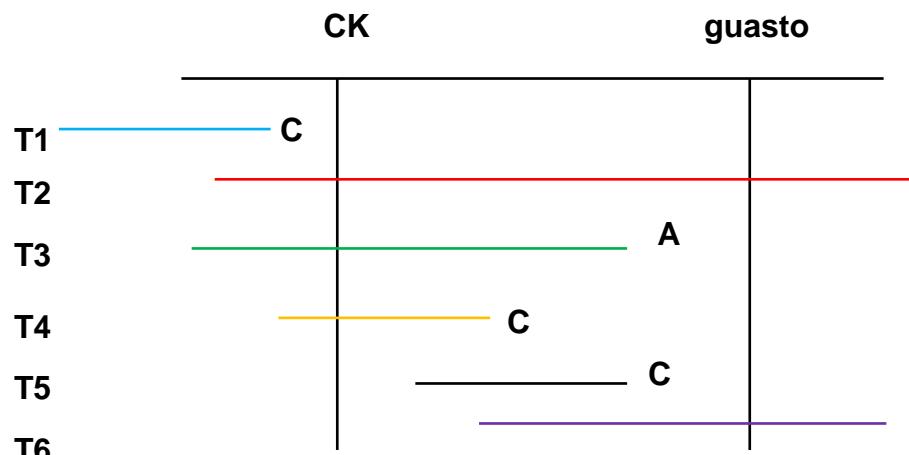


**UNDO = {T2, T3, T6}**  
**REDO = {T4, T5}**

|              |                  |
|--------------|------------------|
| $D(O6)$      | $\rightarrow T2$ |
| $U(O4 = B7)$ | $\rightarrow T6$ |
| $I(O5 = B7)$ | $\rightarrow T3$ |
| $U(O3 = B5)$ | $\rightarrow T3$ |
| $U(O2 = B3)$ | $\rightarrow T3$ |
| $U(O1 = B1)$ | $\rightarrow T2$ |

## Esempio di ripresa a caldo: fase 4 - REDO

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), B(T6)  
U(T3, O3, B5, A5), U(T5, O4, B6, A6),  
D(T3, O5, B7), U(T6, O6, B7, A7)  
A(T3)  
C(T5)  
I(T2, O6, A8)  
**guasto**



**UNDO = {T2, T3, T6}**  
**REDO = {T4, T5}**

**U(O3 = A4)** → T4  
**U(O4 = A6)** → T5

# GESTIONE DELL'AFFIDABILITA'

---

- Gli algoritmi si differenziano a seconda del modo in cui si trattano le scritture sulla BD e la terminazione delle transazioni
  - Disfare-Rifare
  - Disfare-NonRifare
  - NonDisfare-Rifare
  - NonDisfare-NonRifare
- Ipotesi: Le scritture nel giornale vengono portate subito nella memoria permanente!

---

# GESTIONE DELLA CONCORRENZA

# Serializzazione

---

- Uno schedule  $S$  si dice **seriale** se le azioni di ciascuna transazione appaiono **in sequenza**, senza essere inframezzate da azioni di altre transazioni.

$$S = \{T_1, T_2, \dots, T_n\}$$

- Schedule seriale ottenibile se:

- (i) Le transazioni sono **eseguite uno alla volta** (scenario non realistico)
- (ii) Le transazioni sono **completamente indipendenti** l'una dall'altra (improbabile)

## ACID: Proprietà di isolamento delle transazioni

---

- Il DBMS transazionale gestisce questi problemi garantendo la proprietà di **isolamento**
- La proprietà di isolamento di una transazione garantisce che essa sia eseguita come se non ci fosse concorrenza
- Questa proprietà è assicurata facendo in modo che ciascun insieme di transazioni concorrenti sottoposte al sistema sia “**serializzabile**”

## Gestione delle Transazioni - Problematica

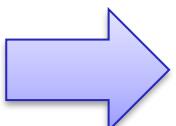
---

- In un sistema reale, le transazioni vengono eseguite in concorrenza per ragioni di efficienza / scalabilità.
- ... Tuttavia, l'esecuzione concorrente determina un insieme di problematiche che devono essere gestite ...

T1= Read(x);  $x=x+1$ ; Write(x); Commit Work

T2= Read(x);  $x=x+1$ ; Write(x); Commit Work

- Se  $x=3$ , al termine delle due transazioni  $x$  vale 5 (esecuzione sequenziale)  
... cosa accade in caso di esecuzione concorrente?



## Gestione delle Transazioni - Perdita di Aggiornamento

---

T1= Read(x);  $x=x+1$ ; Write(x); Commit Work

T2= Read(x);  $x=x+1$ ; Write(x); Commit Work

| X=3         | Transazione1 (T1)       | Transazione2 (T2) | T2 scrive 4 |
|-------------|-------------------------|-------------------|-------------|
|             | Read(x)                 |                   |             |
|             | $x=x+1$                 |                   |             |
|             |                         | Read(x)           |             |
|             |                         | $x=x+1$           |             |
|             |                         | Write(x)          |             |
|             |                         | Commit work       |             |
| T1 scrive 4 | Write(x)<br>Commit work |                   |             |

## Gestione delle Transazioni - Lettura sporca/impropria

---

T1= Read(x);  $x=x+1$ ; Write(x); Rollback Work  
T2= Read(x); Commit Work

| X=3 | Transazione1 (T1) | Transazione2 (T2) |
|-----|-------------------|-------------------|
|     | Read(x)           |                   |
|     | $x=x+1$           |                   |
|     | Write(x)          |                   |
|     |                   | Read(x)           |
|     |                   | Commit work       |
|     | Rollback work     |                   |

T2  
legge 4!

## Gestione delle Transazioni - Letture inconsistenti/non riproducibili

**T1= Read(x);                    Read(x); Commit Work**

**T2= Read(x); x=x+1; Write(x); Commit Work**

X=3

# T1

## legge 3!

# T1

## legge 4!

| Transazione1 (T1) | Transazione2 (T2) |
|-------------------|-------------------|
| Read(x)           |                   |
|                   | Read(x)           |
|                   | x=x+1             |
|                   | Write(x)          |
|                   | Commit work       |
| Read(x)           |                   |
| Commit work       |                   |

# GESTIONE DELLA CONCORRENZA - ANOMALIE

- L'esecuzione concorrente di transazioni è essenziale per un buon funzionamento del DBMS.
- Il DBMS deve però garantire che l'esecuzione concorrente di transazioni avvenga senza interferenze in caso di accessi agli stessi dati.

Gestione CC: X=1000 euro

| T1           | tempo | T2           |
|--------------|-------|--------------|
| begin        | t1    | -            |
| r[x]         | ↓     | begin        |
| -            | t2    | r[x]         |
| -            | ↓     | x := x - 800 |
| x := x + 500 | t3    | -            |
| -            | t4    | w[x]         |
| w[x]         | t5    | *Commit*     |
| *Commit*     | t6    | ↓            |

Se eseguite in serie avrei:  
 $1000+500-800=700$  euro

Se in concorrenza potrei avere un'anomalia

# SERIALITÀ E SERIALIZZABILITÀ

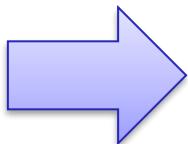
---

- **Definizione** Un'esecuzione di un insieme di transazioni  $\{T_1, \dots, T_n\}$  si dice **seriale** se, per ogni coppia di transazioni  $T_i$  e  $T_j$ , tutte le operazioni di  $T_i$  vengono eseguite prima di qualsiasi operazione  $T_j$  o viceversa.
- **Definizione** Un'esecuzione di un insieme di transazioni si dice **serializzabile** se produce lo stesso effetto sulla base di dati di quello ottenibile eseguendo serialmente, in un qualche ordine, le sole transazioni terminate normalmente.

## Controllo della concorrenza

---

- Nella pratica i DBMS implementano tecniche di controllo di concorrenza che garantiscono direttamente la serializzabilità delle transazioni concorrenti.
- Tali tecniche si dividono in due classi principali:
  - Protocolli **pessimistici**/conservativi: tendono a «**ritardare**» l'esecuzione di transazioni che potrebbero generare conflitti, e quindi anomalie, rispetto alla transazione corrente. Cercano quindi di prevenire.
  - Protocolli **ottimistici**, che permettono l'esecuzione sovrapposta e non sincronizzata di transazioni ed effettuano un controllo sui possibili conflitti generati **solo a valle del commit**.



## Controllo della concorrenza ottimistico

---

- Ogni transazione effettua «liberamente» le proprie operazioni sugli oggetti della base di dati secondo l'ordine temporale con cui le operazioni stesse sono generate.
- Al commit, viene effettuato un controllo per stabilire se sono stati riscontrati eventuali conflitti, e nel caso, viene effettuato il rollback delle azioni delle transazioni e la relativa riesecuzione.
- In generale, un protocollo di controllo di concorrenza **ottimistico** è basato su 3 fasi:
  - Fase di **lettura**: ogni transazione legge i valori degli oggetti della BD su cui deve operare e li memorizza in variabili (copie) locali dove sono effettuati eventuali aggiornamenti
  - Fase di **validazione**: vengono effettuati dei controlli sulla serializzabilità nel caso che gli aggiornamenti locali delle transazioni dovessero essere propagati sulla base di dati
  - Fase di **scrittura**: gli aggiornamenti delle transazioni che hanno superato la fase di validazione sono propagati definitivamente sugli oggetti della BD.

## Controllo della concorrenza - pessimistici/conservativi

---

- Nella pratica i DBMS implementano tecniche di controllo di concorrenza che garantiscono direttamente la serializzabilità delle transazioni concorrenti.
- Tali tecniche si dividono in due classi principali:
  - Metodi basati su lock
  - Metodi basati su timestamp

## Gestione delle Transazioni - Implementazione

---

- Come implementare il controllo della concorrenza?
  - I DMBS commerciali usano il meccanismo dei lock → per poter effettuare una qualsiasi operazioni di lettura/scrittura su una risorsa (tabella o valore di una cella), **è necessario aver precedentemente acquisito il controllo (lock) sulla risorsa stessa.**
- 
- Lock in lettura (accesso condiviso)
  - Lock in scrittura (mutua esclusione)

## Gestione di transazioni mediante lock

---

- I DBMS per evitare anomalie nelle transazione concorrenti usano diverse tecniche
- Una delle più comuni è basata su lock
- Il **lock** è un meccanismo che blocca l'accesso ai dati ai quali una transazione accede ad altre transazioni
  - lock a **livello di riga, tabella, pagina** (multi **granularità**)
  - lock in operazioni di scrittura (**mutua esclusione**) /lettura (**accesso condiviso**) (**multimodale**)
- In generale, quando una risorsa è bloccata, le transazioni che ne richiedono l'accesso vengono in genere messe in coda
  - quindi devono aspettare (che il lock sia rimosso)
- In sostanza, questo è un meccanismo efficace, ma influisce sulle prestazioni

# Gestione delle Transazioni

---

- Su ogni lock possono essere definite due operazioni:
  - Richiesta del lock in lettura/scrittura.
  - Rilascio del lock (*unlock*) acquisito in precedenza.

Transazione #0 (T\_0):

$r(x)$   
 $w(y)$

CODICE UTENTE



Transazione #0 (T\_0):

$lock_r(x)$

$r(x)$

$unlock(x)$

$lock_w(y)$

$w(y)$

$unlock(y)$

CODICE con LOCK

## SERIALIZZATORE 2PL STRETTO

---

- Il gestore della concorrenza (serializzatore) dei DBMS ha il compito di stabilire l'ordine secondo il quale vanno eseguite le singole operazioni per rendere serializzabile l'esecuzione di un insieme di transazioni.
- **Definizione** Il protocollo del blocco a due fasi stretto (**Strict Two Phase Locking**) è definito dalle seguenti regole:
  1. Ogni transazione, prima di effettuare un'operazione acquisisce il blocco corrispondente (chiede il lock)
  2. Transazioni diverse non ottengono blocchi in conflitto.
  3. I blocchi/lock si rilasciano alla terminazione della transazione (cioè al commit)

# Gestione delle Transazioni

---

- **Strict Two Phase Lock (2PL)** → I lock di una transazione sono rilasciati solo dopo aver effettuato le operazioni di commit/abort.
- **PROBLEMA:** I protocolli 2PL possono generare schedule con situazioni di deadlock (stallo).

TRANSAZIONI

$T_1 = r(x), w(y), \text{Commit}$   
 $T_2 = r(y), w(x), \text{Commit}$

SCHEDULE



| $T_1$     | $T_2$     |
|-----------|-----------|
| r_lock(x) |           |
|           | r_lock(y) |
| r(x)      |           |
|           | r(y)      |
| w_lock(y) |           |
|           | w_lock(x) |

## Gestione delle Transazioni - deadlock - timeout

---

- Per gestire le situazioni di **deadlock** causate dal gestore della concorrenza, si possono usare **tre tecniche**:
  1. Uso dei **timeout** → ogni operazione di una transazione ha un **timeout** entro il quale deve essere completata, pena annullamento (abort) della transazione stessa.
- $T_1$ : r\_lock(x, **4000**), r(x), w\_lock(y, **2000**), w(y), commit, unlock(x), unlock(y)

- Per gestire le situazioni di **deadlock** causate dal gestore della concorrenza, si possono usare **tre tecniche**:
2. **Deadlock avoidance** → prevenire le configurazioni che potrebbero portare ad un deadlock ... COME?
- ✧ Lock/Unlock di tutte le risorse allo stesso tempo.
  - ✧ Utilizzo di **time-stamp** o di **classi di priorità** tra transazioni (problema: può determinare **starvation!**)
    - ✧ **starvation**: quando una transazione è impossibilitata a proseguire la sua esecuzione per un periodo di tempo indefinito, mentre le altre transazioni del sistema proseguono tranquillamente.

- Per gestire le situazioni di **deadlock** causate dal gestore della concorrenza, si possono usare **tre tecniche**:

3. **Deadlock detection** → utilizzare **algoritmi per identificare eventuali situazioni di deadlock**, e prevedere meccanismi di recovery dal deadlock

- **Grafo delle richieste/risorse** utilizzato per identificare la presenza di cicli (corrispondenti a deadlock)
- In caso di ciclo, si fa abort delle transazioni coinvolte nel ciclo in modo da eliminare la mutua dipendenza ...

## CONDIZIONI DI STALLO

---

- Il problema si può risolvere con tecniche che prevengono queste situazioni (deadlock prevention), oppure con tecniche che rivelano una situazione di stallo e la sbloccano facendo abortire una o più transazioni in attesa (deadlock detection and recovery).

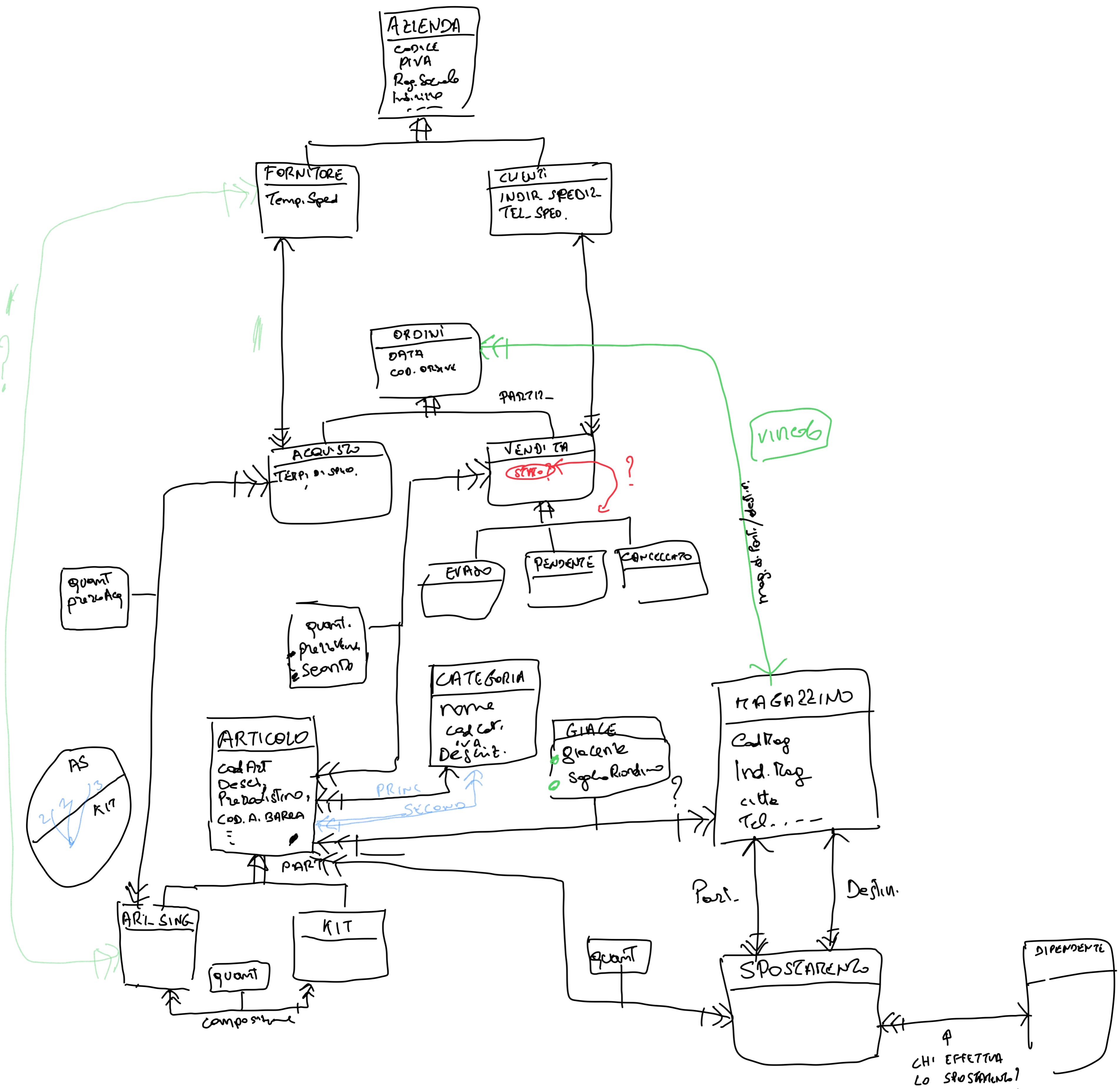
- Un metodo alternativo al 2PL per la gestione della concorrenza in un DBMS prevede l'utilizzo dei **time-stamp delle transazioni** (metodo **TS**).
  - Ad ogni transazione si associa un **timestamp** che rappresenta il momento di inizio della transazione.
  - Ogni transazione **non può leggere o scrivere un dato scritto da una transazione con timestamp maggiore**.
  - Ogni transazione **non può scrivere su un dato già letto da una transazione con timestamp maggiore**.

# Livelli di isolamento/consistenza per ogni transazione

---

- SERIALIZABLE assicura che
  - la transazione T legge solo cambiamenti fatti da transazioni concluse (che hanno fatto il commit)
  - nessun valore letto o scritto da T verrà cambiato da altre transazione finché T non è conclusa
  - se T legge un insieme di valori acceduti secondo qualche condizione di ricerca, l'insieme non viene modificato da altre transazione finché T non è conclusa
- REPEATABLE READ assicura che
  - la transazione T legge solo cambiamenti fatti da transazioni concluse (che hanno fatto il commit)
  - nessun valore letto o scritto da T verrà cambiato da altre transazione finché T non è conclusa
- READ COMMITTED assicura che
  - la transazione T legge solo cambiamenti fatti da transazioni concluse (che hanno fatto il commit)
  - T non vede nessun cambiamento eventualmente effettuato da transazioni concorrenti non concluse tra i valori letti all'inizio di T
- READ UNCOMMITTED
  - a questo livello di isolamento una transazione T può leggere modifiche fatte ad un oggetto da un transazione in esecuzione; ovviamente l'oggetto può essere cambiato mentre T è in esecuzione. Quindi T è soggetta a effetti fantasma

30 OTTOBRE 2023



|       |    |       |           |
|-------|----|-------|-----------|
| Mag A | 10 | penne | soglia 15 |
| Mag B | 5  | penne | soglia 10 |
| ;     | ;  | ;     | ;         |

|       |    |       |                 |
|-------|----|-------|-----------------|
| Mag A | 10 | penne | soglie Penne 30 |
| Mag B | 5  | penne |                 |
| ;     | ;  | ;     |                 |

# BASE DI DATI

Esercizio: Agenzia pubblicitaria

- Progettazione concettuale
- Progettazione logica

# Esercizio: agenzia pubblicitaria

- L'agenzia è composta da uno staff creativo e da uno staff amministrativo per la realizzazione di campagne pubblicitarie. Una campagna pubblicitaria ha un responsabile dello staff amministrativo ed è realizzata da uno staff creativo.
- I membri sia dello staff amministrativo che di quello creativo vengono pagati secondo il loro livello di impiego.
- La retribuzione dei livelli oscilla nel tempo. Ogni livello può avere una retribuzione diversa nel tempo, ma una particolare retribuzione viene usata univocamente per un livello alla volta.
- I membri degli staff (amministrativo o creativo) possono ricevere più livelli durante il loro impiego: è necessario sapere quando un membro dello staff comincia ad essere pagato per un certo livello e quando finisce di essere pagato per quel livello.
- Bisogna specificare la qualifica (disegnatore, progettista, eccetera) dello staff creativo.
- Solo gli amministratori possono ricevere un bonus di produzione in base al numero di campagne pubblicitarie che hanno gestito.
- I Clienti contattano una persona dello staff amministrativo per specificare i requisiti della loro campagna pubblicitaria e richiederne il preventivo.
- I clienti possono richiedere più campagne pubblicitarie.
- Le campagne sono tipicamente prodotte da squadre di impiegati provenienti dallo staff creativo.
- I membri dello staff creativo possono lavorare su più campagne pubblicitarie contemporaneamente.
- I clienti registrati devono poter chiedere un preventivo della campagna pubblicitaria che preveda una descrizione, una data di inizio, una data di fine, i costi stimati e lo stato (che può essere in fase di valutazione, non confermato, confermato, eccetera).
- Se il cliente accetta il preventivo, la campagna pubblicitaria dovrà essere realizzata dallo staff creativo.
- I componenti dello staff amministrativo e creativo non possono richiedere campagne pubblicitarie.

| Livello     |
|-------------|
| Descrizione |
| NomeLivello |

| RetribuzioneLivello |
|---------------------|
| CodRetribuzione     |
| Retribuzione        |
| DataInizioRetrib    |
| DataFineRetrib      |

| MembriStaff    |
|----------------|
| Matricola      |
| Nome           |
| Cognome        |
| DataAssunzione |

| Cliente       |
|---------------|
| CodiceCliente |
| Nome          |
| Cognome       |
| Email         |

Per semplicità non  
distinguiamo i clienti  
registrati da quelli che non lo  
sono

si è interessati a  
mantenere lo storico o la  
retribuzione in corso?

| CampagnaPubblicitaria |
|-----------------------|
| CodiceCampagna        |
| Titolo                |
| CostiStimati          |
| DataInizio            |
| DataFine              |
| Stato                 |

| Livello     |
|-------------|
| Descrizione |
| NomeLivello |

| RetribuzioneLivello |
|---------------------|
| CodRetribuzione     |
| Retribuzione        |
| DataInizioRetrib    |
| DataFineRetrib      |

| MembriStaff    |
|----------------|
| Matricola      |
| Nome           |
| Cognome        |
| DataAssunzione |



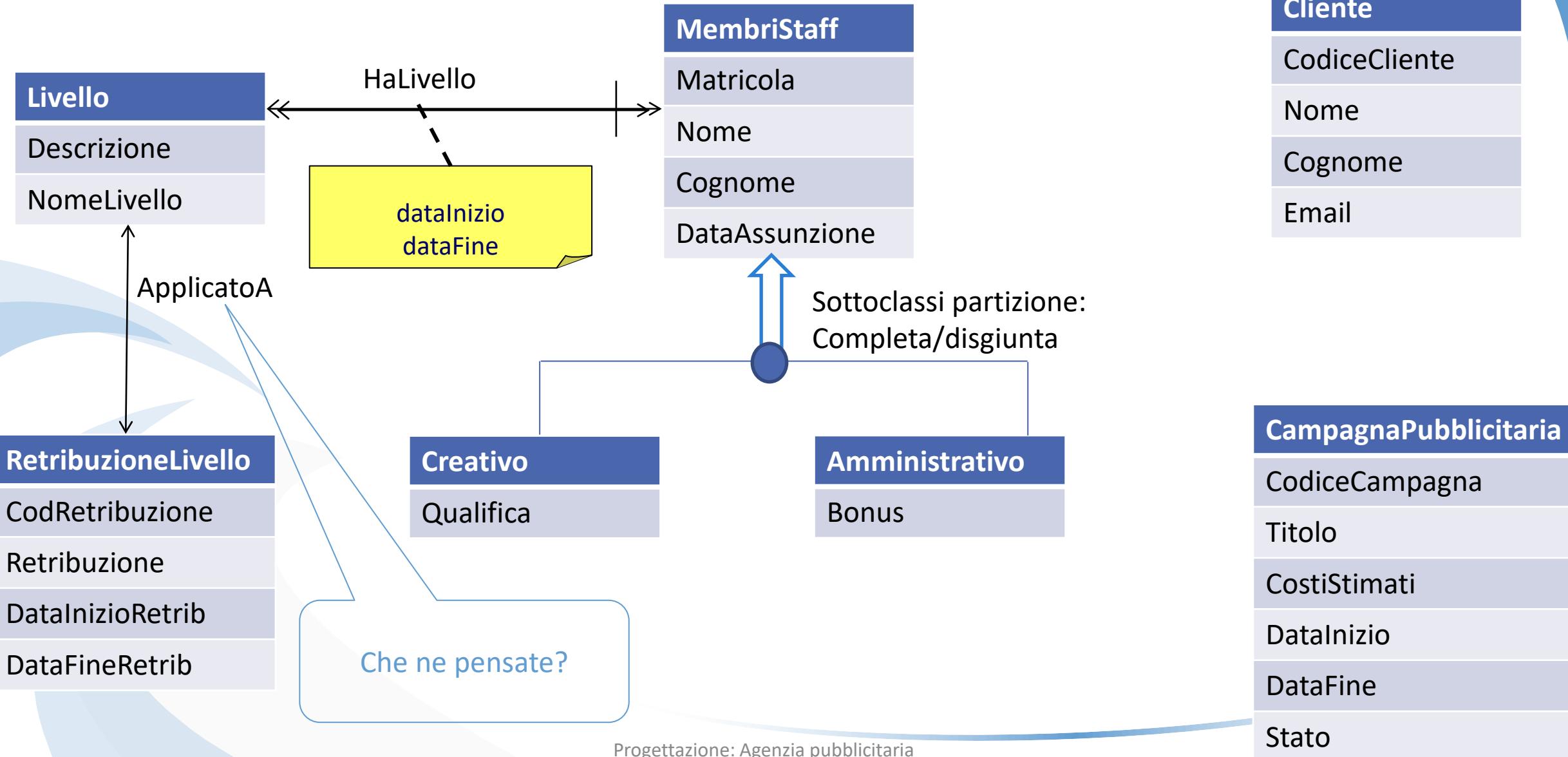
Sottoclassi partizione:  
Copertura/disgiunta  
Partizione

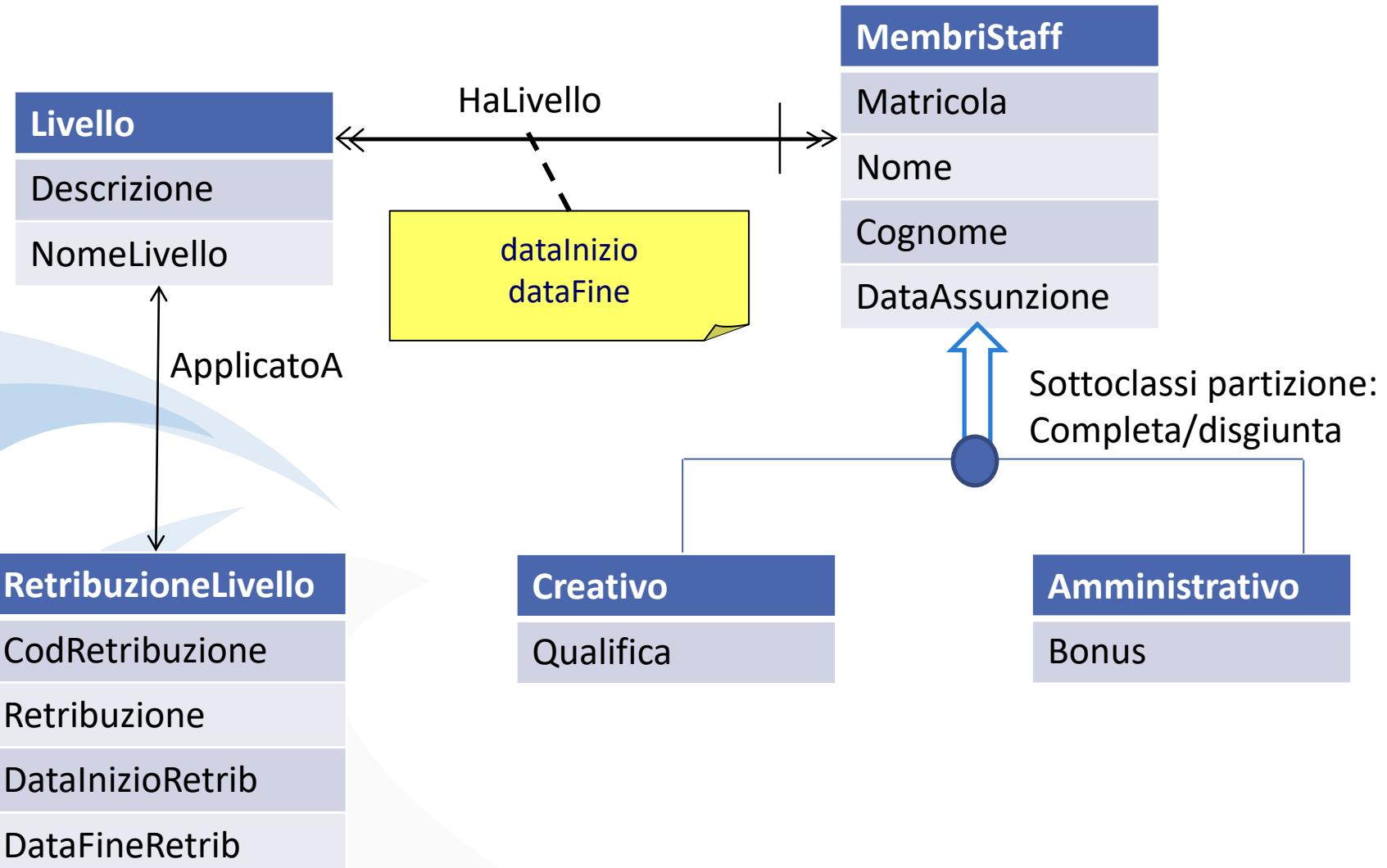
| Creativo  |
|-----------|
| Qualifica |

| Amministrativo |
|----------------|
| Bonus          |

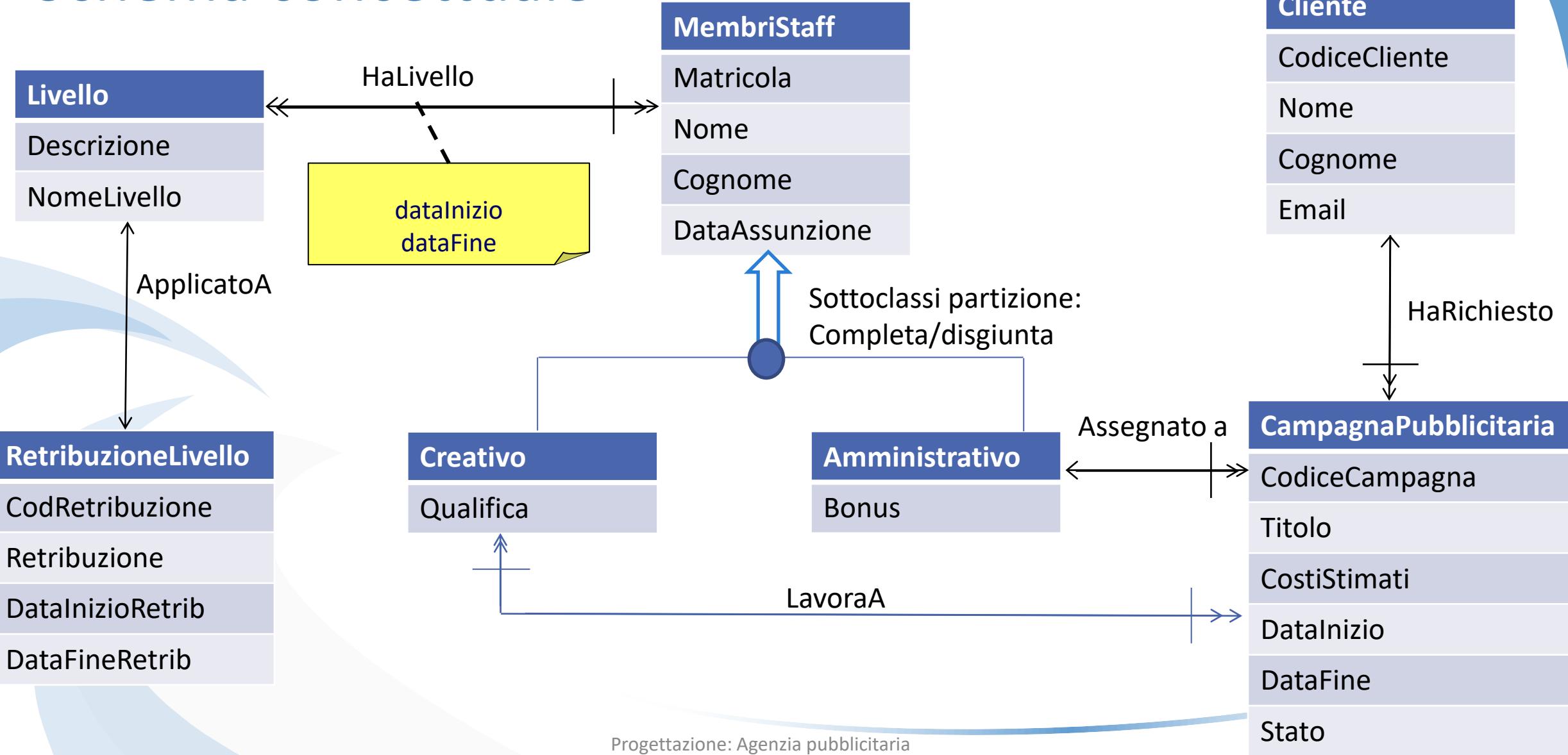
| Cliente       |
|---------------|
| CodiceCliente |
| Nome          |
| Cognome       |
| Email         |

| CampagnaPubblicitaria |
|-----------------------|
| CodiceCampagna        |
| Titolo                |
| CostiStimati          |
| DataInizio            |
| DataFine              |
| Stato                 |





# Schema concettuale



# Progettazione logica

- I passo: traduzione iniziale delle classi non coinvolte in gerarchie
- II passo: traduzione iniziale delle gerarchie
- III passo: traduzione degli attributi multivaleore
- IV passo: traduzione delle associazioni molti a molti
- V passo: traduzione delle associazioni uno a molti
- VI passo: traduzione delle associazioni uno a uno
- VII passo: introduzione di eventuali ulteriori vincoli
- VIII passo: progettazione degli schemi esterni

# I passo: traduzione iniziale delle classi non coinvolte in gerarchie e chiavi primarie

| Livello     |
|-------------|
| Descrizione |
| NomeLivello |

| RetribuzioneLivello |
|---------------------|
| CodRetribuzione     |
| Retribuzione        |
| DataInizioRetrib    |
| DataFineRetrib      |

| Cliente       |
|---------------|
| CodiceCliente |
| Nome          |
| Cognome       |
| Email         |

| CampagnaPubblicitaria |
|-----------------------|
| CodiceCampagna        |
| Titolo                |
| CostiStimati          |
| DataInizio            |
| DataFine              |

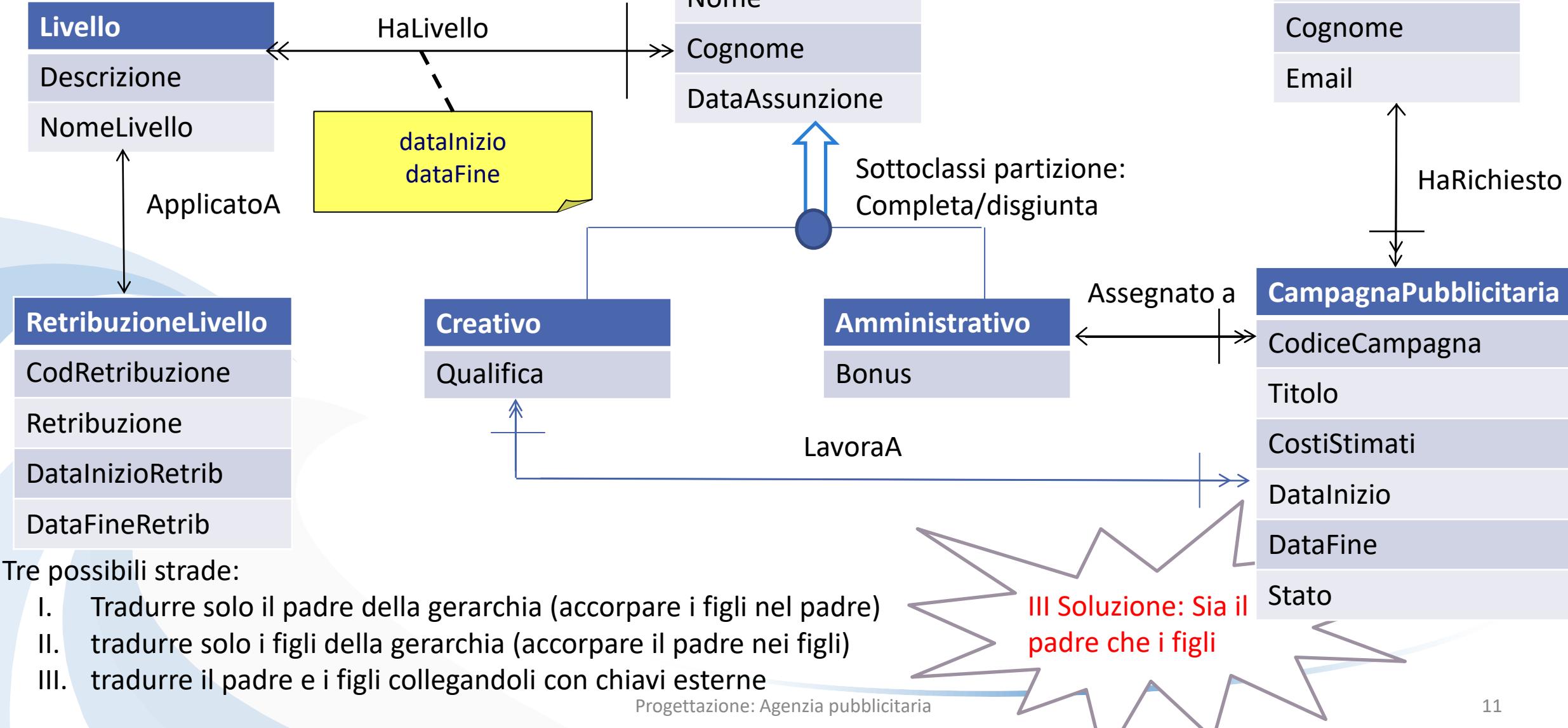
| Livello                 | T  | RetribuzioneLivello  | T  |
|-------------------------|----|----------------------|----|
| codice CHAR(16)         | PK | CodRetribuzione      | PK |
| Descrizione VARCHAR(15) |    | CHAR(3)              |    |
| NomeLivello DATE        |    | Retribuzione INTEGER |    |

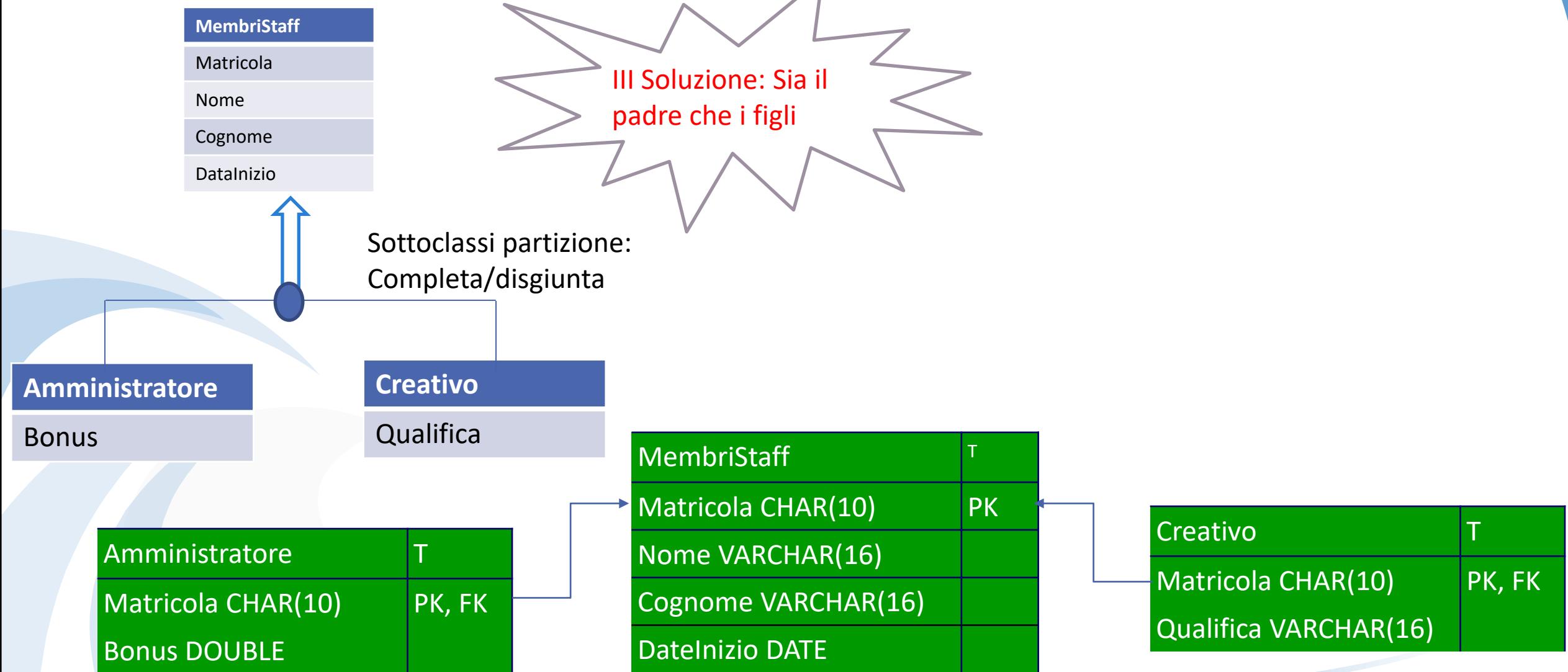
|                       |
|-----------------------|
| DateInizioRetrib DATE |
| DateFineRetrib DATE   |

| Cliente                   | T  | CampagnaPubblicitaria   | T  |
|---------------------------|----|-------------------------|----|
| CodiceCliente VARCHAR(16) | PK | CodiceCampagna          | PK |
| Nome VARCHAR(16)          |    | VARCHAR(16)             |    |
| Cognome VARCHAR(16)       |    | Titolo VARCHAR(16)      |    |
| Email VARCHAR(16)         |    | CostiStimati INTEGER    |    |
|                           |    | DataInizioCampagna DATE |    |
|                           |    | DataFineCampagna DATE   |    |
|                           |    | Stato char(2)           |    |

# Il passo: traduzione iniziale delle gerarchie



## Il passo: traduzione iniziale delle gerarchie

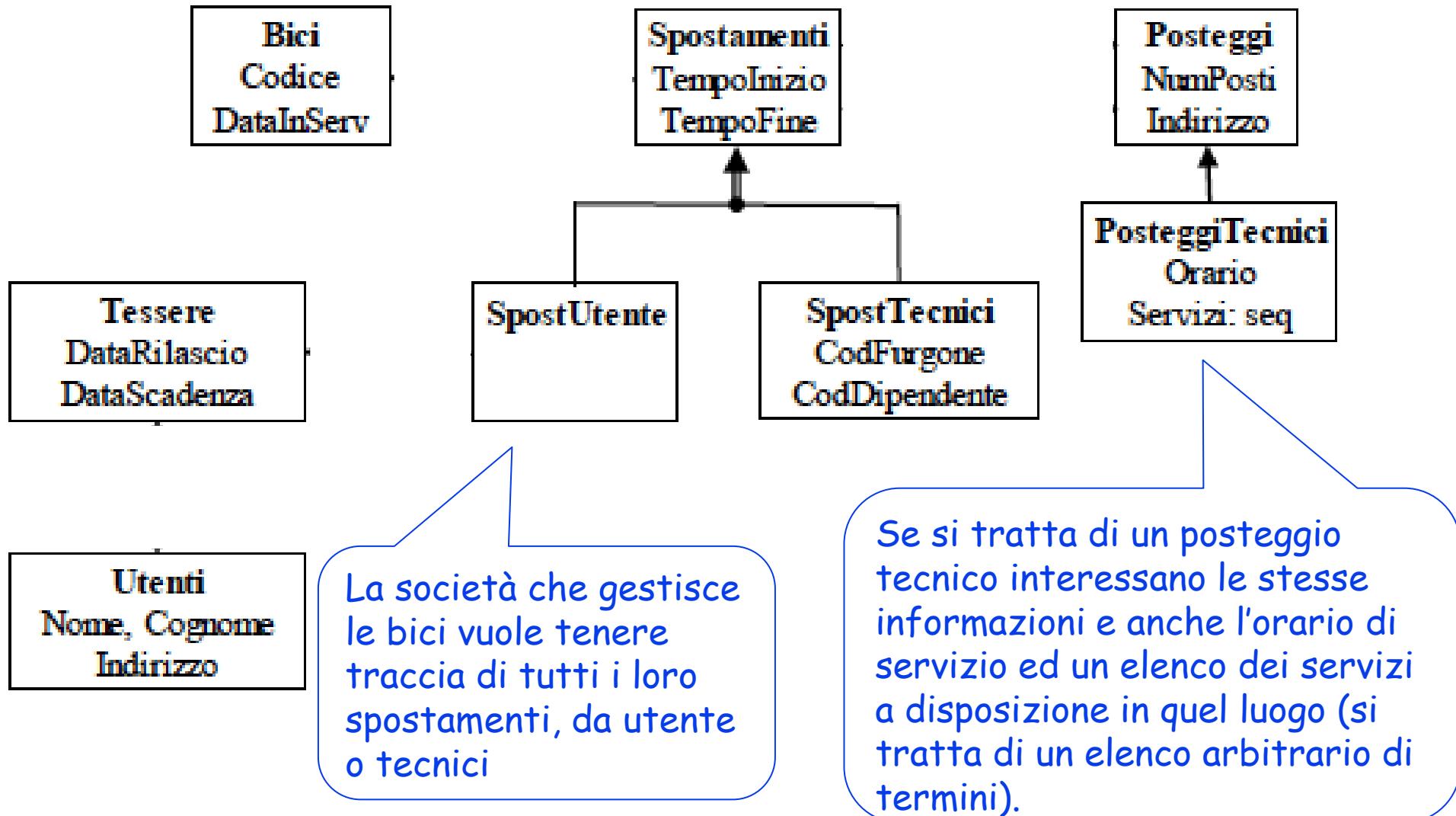


# Esercizio: Primo compitino di Basi di Dati - 9/4/2018

---

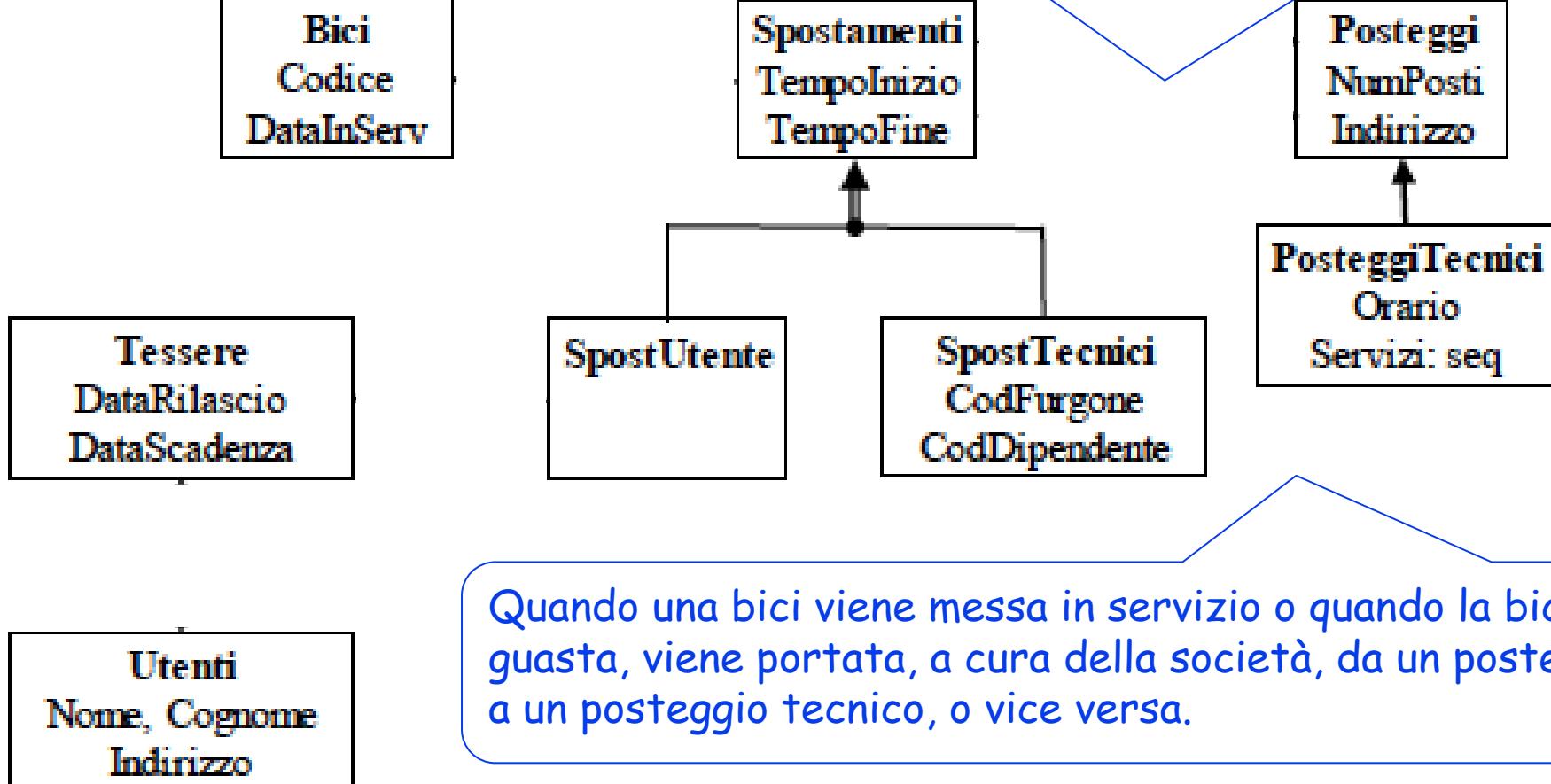
Una città ha un servizio di bici pubbliche basato su posteggi: un utente arriva a un posteggio, si identifica tramite tessera, ritira una bici, e la lascia ad un posteggio diverso. Quando una bici viene messa in servizio o quando la bici si guasta, viene portata, a cura della società, da un posteggio a un posteggio tecnico, o vice versa. La società che gestisce le bici vuole tenere traccia di tutti i loro spostamenti, da utente o tecnici. Di ogni bici interessano un codice e la data di inizio servizio. Di ogni posteggio interessa conoscere il numero di posti, l'indirizzo, gli spostamenti che sono iniziati dal posteggio, e gli spostamenti terminati sul posteggio. Se si tratta di un posteggio tecnico interessano le stesse informazioni e anche l'orario di servizio ed un elenco dei servizi a disposizione in quel luogo (si tratta di un elenco arbitrario di termini). Per ogni spostamento interessa conoscere la bici coinvolta, a che ora e da quale posteggio è iniziato, e a che ora e in quale posteggio si è concluso. Se era uno spostamento da utente, interessano la tessera utente e la bicicletta spostata. Se era uno spostamento tecnico, interessa il codice che identifica il furgoncino che lo ha effettuato e il codice del dipendente che lo guidava. Di ogni tessera utente interessano la data di rilascio, la data di scadenza, e gli utenti che sono associati alla tessera - che possono essere più d'uno. Per ogni utente interessano nome, cognome, indirizzo, e l'insieme delle tessere associate.

# Soluzione: Primo compitino di Basi di Dati - 9/4/2018



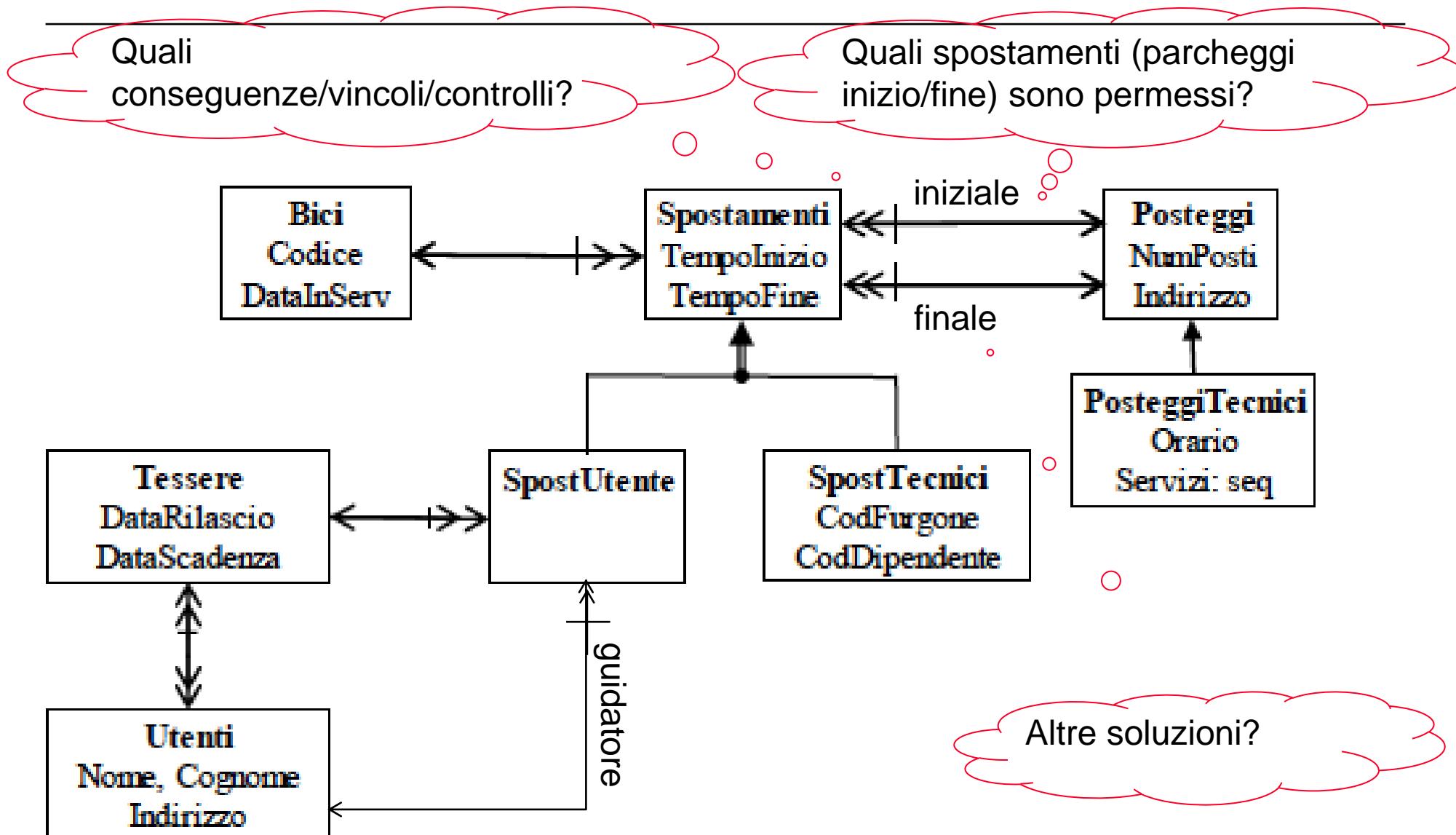
# Soluzione: Primo compitino di Basi di Dati - 9/4/2018

Se si tratta di un posteggio tecnico interessano le stesse informazioni e anche l'orario di servizio ed un elenco dei servizi a disposizione in quel luogo (si tratta di un elenco arbitrario di termini).



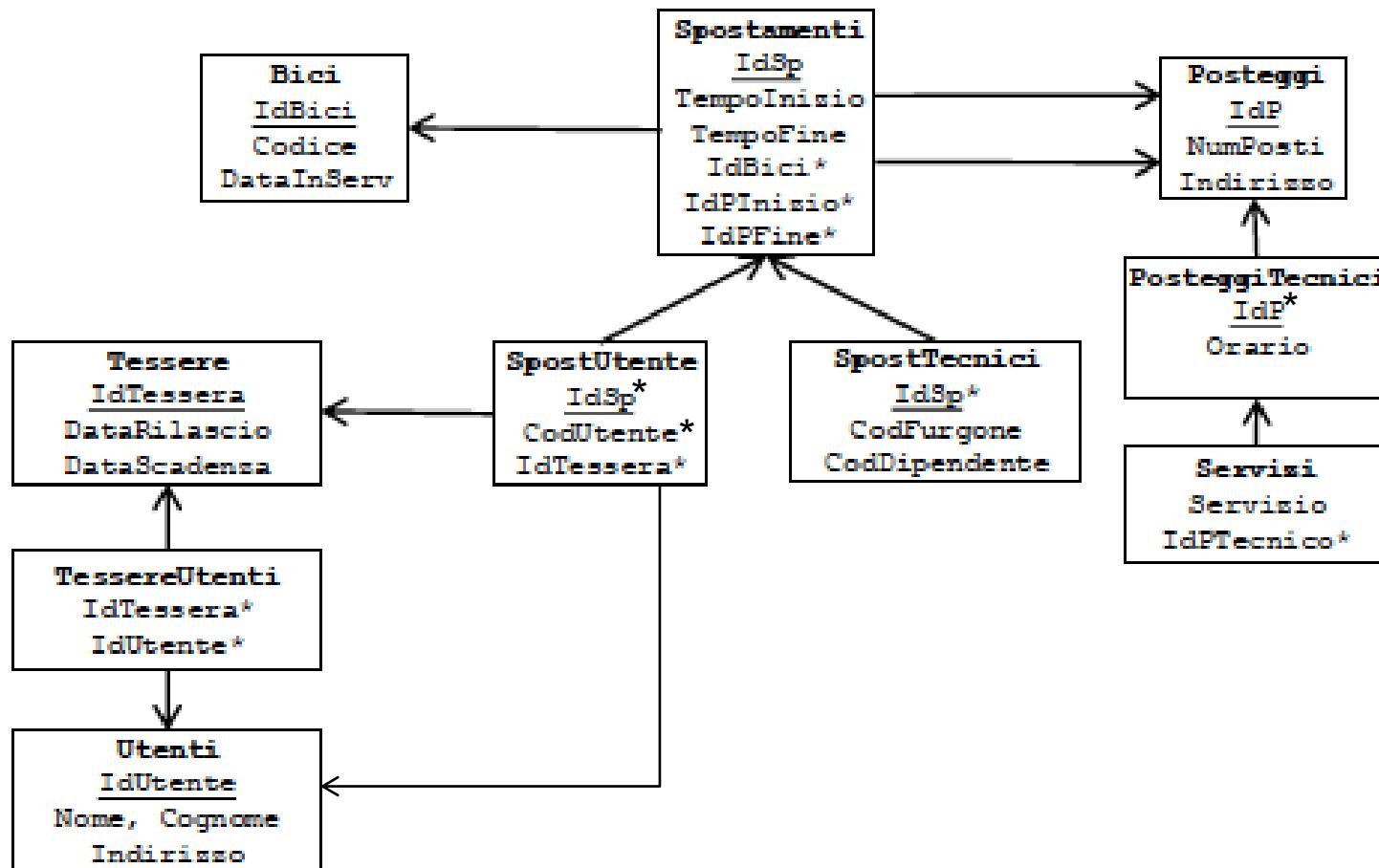
Quando una bici viene messa in servizio o quando la bici si guasta, viene portata, a cura della società, da un posteggio a un posteggio tecnico, o vice versa.

# Soluzione I: Primo compitino di Basi di Dati - 9/4/2018 - Concettuale



# Soluzione I: Primo compitino di Basi di Dati - 9/4/2018 - Logico

---



## Soluzione II: Primo compitino di Basi di Dati - 9/4/2018 - Concettuale

Altre soluzioni?

Quali conseguenze/vincoli/controlli?

E il logico?

