

# Basi di dati

28/09/2020

Progettazione

SQL + Progettazione

Laboratorio (Sviluppo applicativo)

} prova intercorso?

Libri di testo: Sistemi di basi di dati - Fluasti

Cos'è una base di dati? cosa è un insieme?

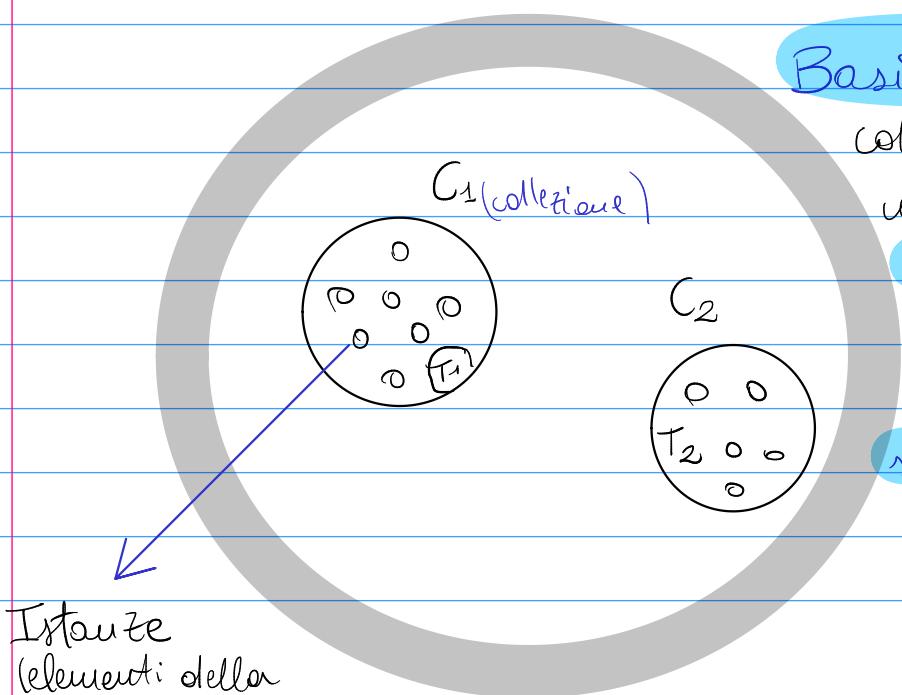
L'insieme è una collezione non ordinata e senza molteplicità degli elementi, una base di dati è una particolare collezione che ha una uniformità concettuale (caratteristiche in comune) dove si considera un dominio specifico, ristretto.

Quando parlo di collezione specifico che trattano lo stesso tipo, ad esempio collezione di record struct (nove A

nove B

// il tipo di collezione è espresso

)



Basi di dati (insieme di collezioni di oggetti uniformi) o collezioni OMOGENEE di dati STRUTTURATI che sono memorizzati su supporti permanenti (es. file)

Istante  
(elementi della collezione)

Si può considerare come deposito permanente gli istanze di struttura dati, il programma crea i dati, e prima che termini l'esecuzione, posso salvazueli all'interno; quando il programma è attivo posso andare a recuperare i dati dal deposito



struct (nuove stile)  
} (cognome stile)  
    |  
    |  
    |

(ordino, perci) è l'istanza della mia base di dati

Sistema per la gestione delle basi di dati (DBMS)  
(Data base management system) sono sistemi software  
che creano e gestiscono i dati base

Funzionalità di un DBMS (corri ci aspettiamo...)

- definizione delle collezioni e della loro struttura
- <sup>sono due cose diverse</sup> definizione di METADATI, manipolazione dei metadati (aggiunge, cancella e modifica)
- manipolazione dati (aggiungere, cancellare, modificare)
- recupero efficiente dei dati (lavoraggio di <sup>recupero</sup> interrogazione della BD e una minimizzazione delle interrogazioni)
- Sicurezza (protezione da intrusioni da parte di personale non autorizzato)

- Durability (garanzia di persistenza)  
la base di dati deve essere preservata anche in presenza di malfunzionamenti HW/SW o eventi drammatici
- Definizione delle viste esterne  
API per interfacciarsi ai linguaggi di programmazione (di alto livello)

Dove l'utente vede solo una sezione del database, la vista è una rielaborazione del database e una volta concluso il recupero avrà una interfaccia con il programmatore.

Immagino il DBMS come deposito, dove più utenti vi accedono. (es. prenotazione di un posto di un treno: l'utente "interroga" il database, quindi il deposito è condizionato e offre al luogo di scambio di informazioni)

## Definizione delle viste esterne

29/08/2020

API per interfacciarsi ai linguaggi di programmazione

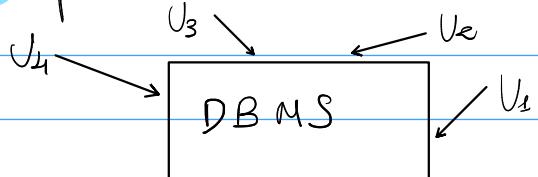
Vi possono accedere - UTENTI

- APPLICATIVI

per ogni tipologia d'utenza, devo avere delle viste esterne (i fini dell'uso della sottabase sono molti)

## Sicurezza e persistenza

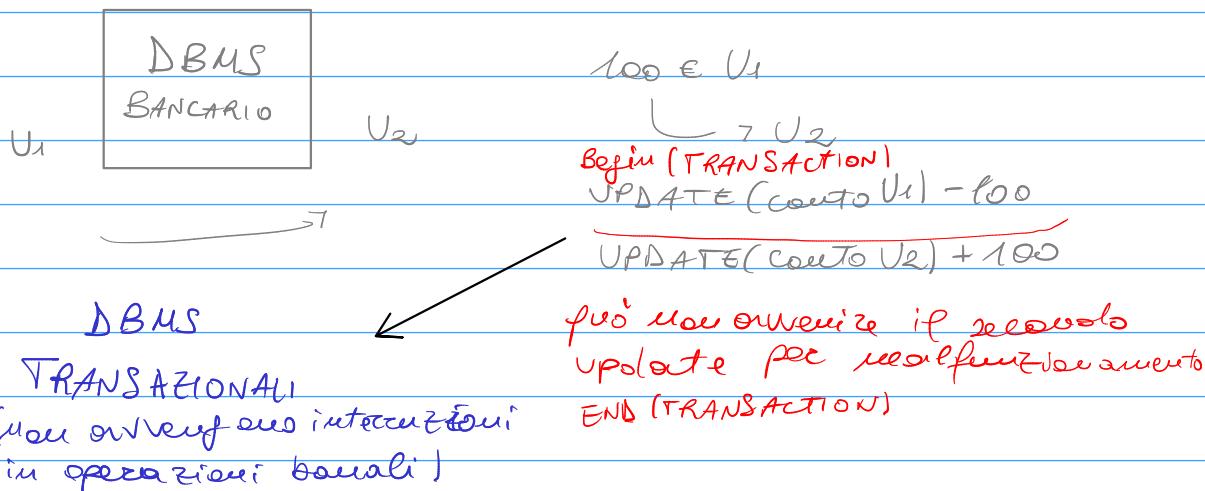
DBMS come una sorta di deposito di dati, dove più utenti possono accederi



## deposito condiviso di dati

punto critico del DBMS: incerenza dei dati (c'è l'accesso non sequenziale ai dati) -> INCONSISTENZA

- deve quindi gestire gli accessi concorrenti esempio



- ripristinare il contenuto della transazione (caso del crollo di transazione) -> GESTIONE DELLE TRANSAZIONI

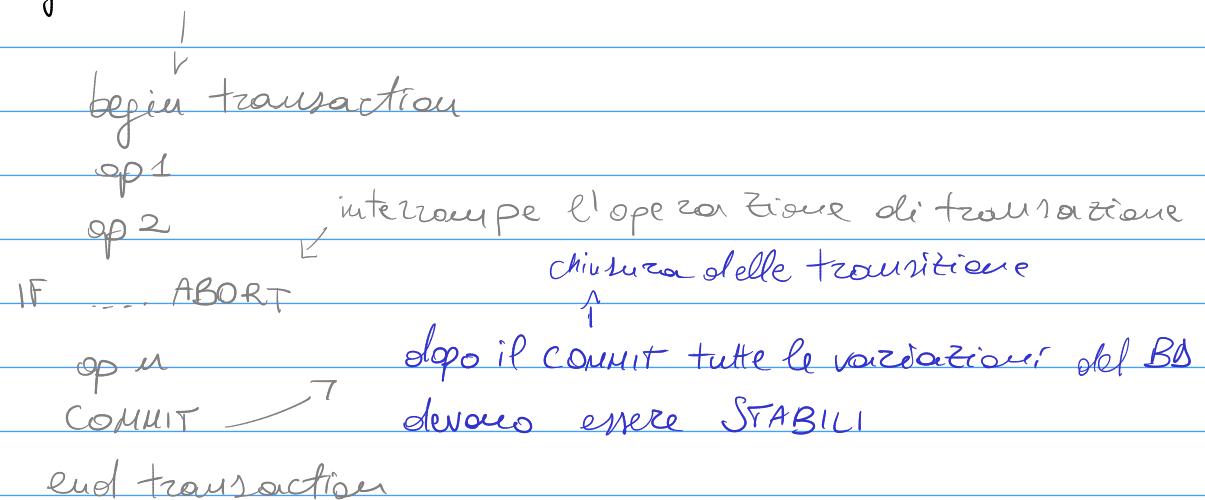
ACID (proprietà delle transazioni)

Atomicity (le operazioni o vengono completate o non vengono effettuate)

Consistency (richiede dati di buona qualità che rispettano le regole del DB)

Isolation (le operazioni devono avvenire in modo isolato, per gli accessi concorrenti)

Durability (persistenza: i dati sono stabili nel database)

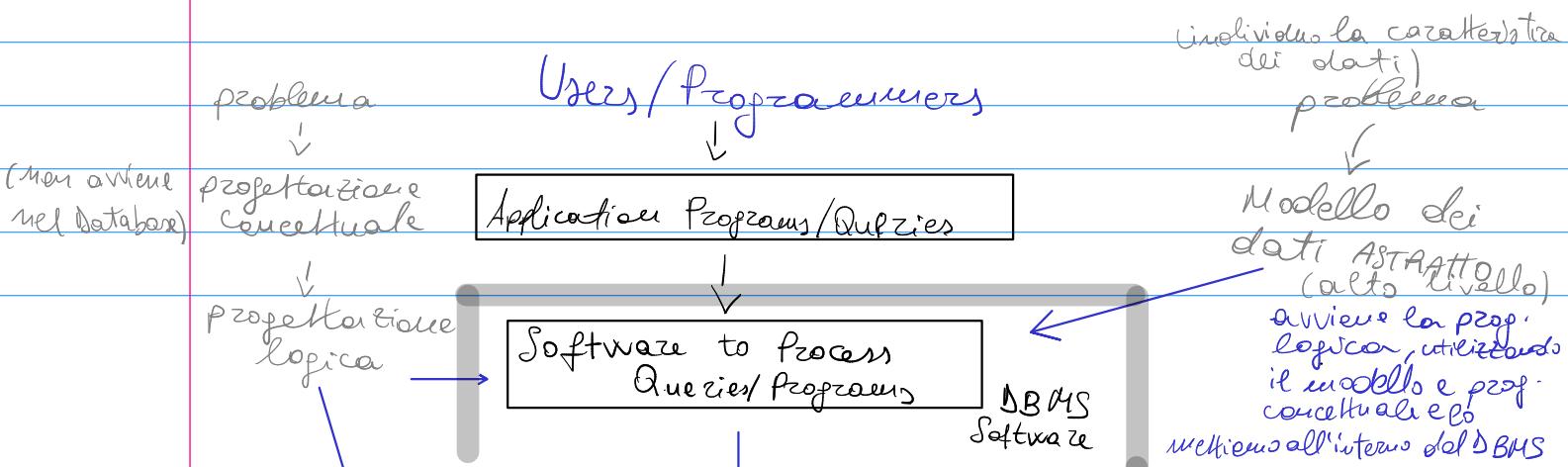


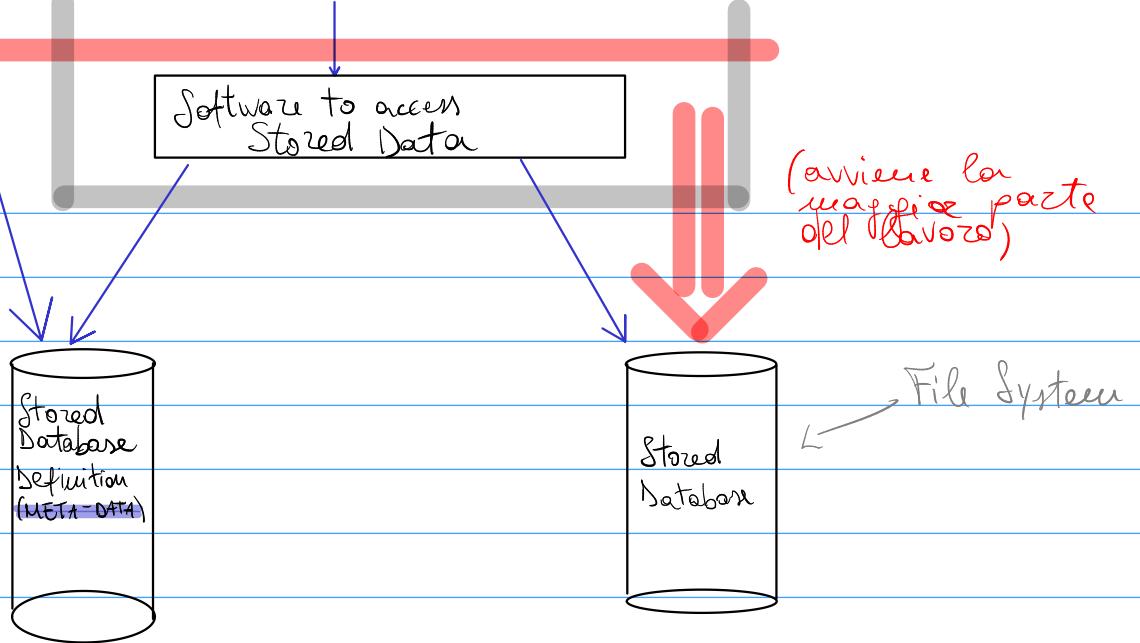
## DBMS transazionali

ha numerose operazioni di variazione del DB parallele e il contenuto della base di dati perfetta è aggiornato con grande frequenza

## DATA WAREHOUSE

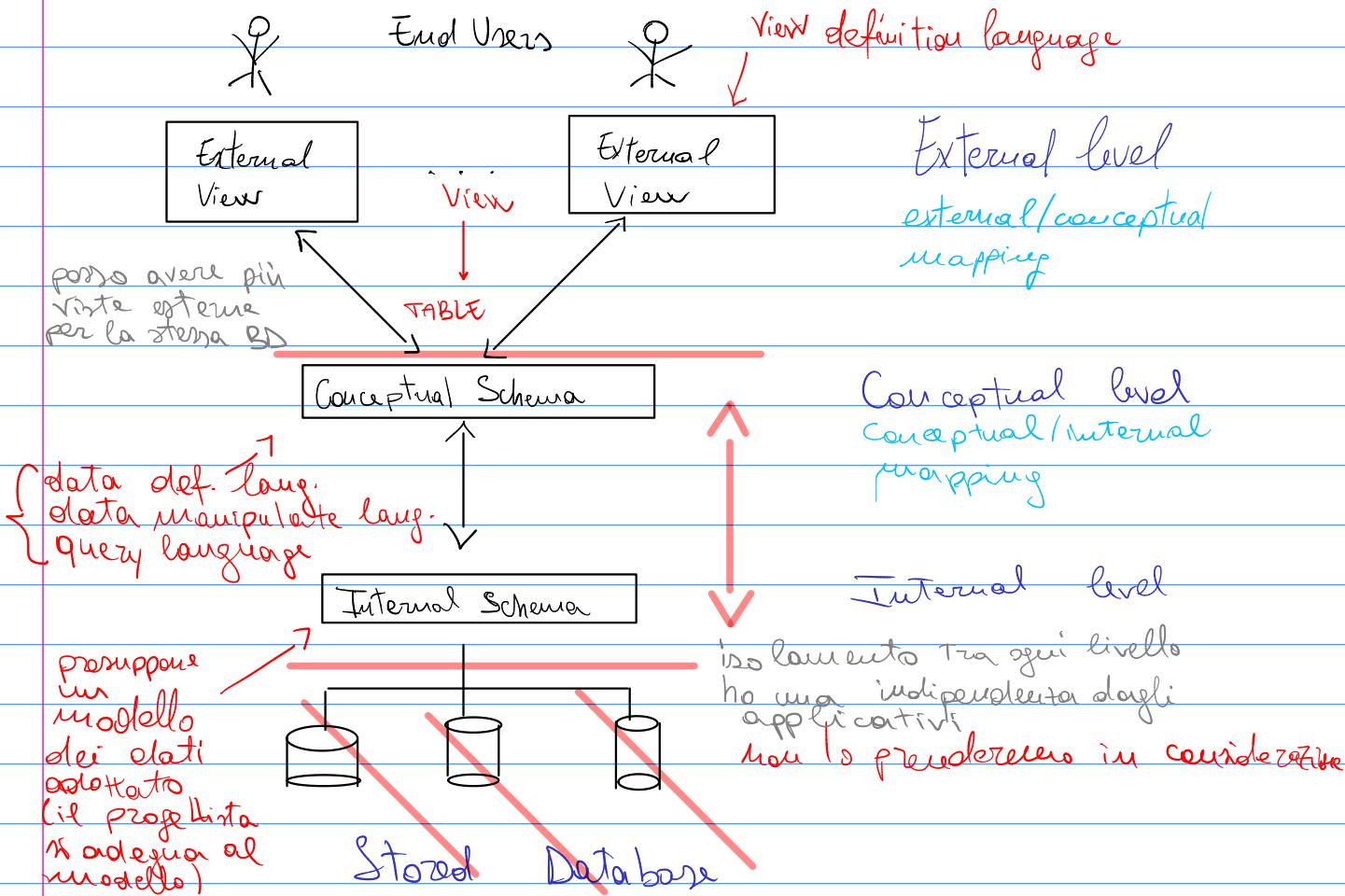
è un database dove i dati NON VENGONO AGGIORNATI, anzi  
non conserva gli storici e inserisce solo nuovi dati, l'unico problema  
è il peso stesso del database





- 1 Fase iniziale mi rendo conto dei dati importati
- 2 Descrivere la prop. concettuale in termini del linguaggio per la definizione dei dati (strettamente collegato al modello)

**Schema a tre livelli** l'architettura dei DBMS, a 3 livelli: esterno, concettuale, interno) isolata rende robusta l'architettura



Un DBMS relazionale (ha le tabelle) e attraverso le relazioni codifica le identità che voglio rappresentare.

Fase concettuale individuare informazioni rilevanti

Fase Prog. logica identificare la struttura tabella che devo codificare

Schema concettuale è l'insieme delle strutture che mi servono per codificare

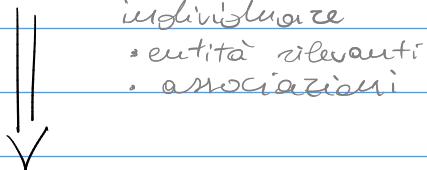
01/10/2020

## Progettazione di una base di dati:

- 1 Progettazione concettuale (strutture dati e le loro connessioni)
- 2 Progettazione logica

} entità e le loro  
relazioni

descrizione informale del problema (testuale)



descrizione strutturata del problema

→ CLASS DIAGRAM - (entità - relazioni)

standard  
descrittivo  
UML =>  
Unified  
Modelling  
Language

Le entità e le loro relazioni: è un concetto atomico/elementare che dovrà essere memorizzato

un concetto descrivibile in maniera univoca mediante:

- un insieme di attributi
- (eventualmente) le sue relazioni con altre entità

esempio gli attributi

Film

- Titolo
- Anno
- Registro
- Durata
- Genere
- Data uscita
- Nazione
- Locandina

Cliente

- Name
- Cognome
- Vita
- Città
- Email

- Cast
- Trama
- Lingua originale

## Proprietà

(1)

l'insieme dei attributi devono permetterci, in maniera univoca, di descrivere tutte le possibili istanze rappresentate dall'esistenza

(2)

gli attributi devono prevedere tutte le informazioni utili per il problema

Film

- Titolo (1,\*)
- Regista (1,\*) abbrev. di (1,1)
- il valore Durata 1 dell'attributo Cost (0,\*) abbreviato \*, si intende metti
- è obbligatorio Genere (1,\*)

### ATRIBUTI

- cardinalità dell'attributo  
il numero di valori che può assumere in una istanza (min-card, max-card)

Attributo (0, -) Non ci è informazione (non VASO)

↓  
Non importa  
NON RILEVANTE

Nazione (1,\*)

DataNascita (0,1)

Lingua 1

Locandina (0,1) BLOB (binary large object)

Trama (0,1)

Total cardinalità 1

Partiale = 0

↓ non vado a volerlo la struttura interna

Attributo singolo max val = 1

= multiplicità max val > 1 → \* saranno un problema

## Tipi di dato:

- Simple (il valore non ha struttura)
- Structured (il valore dell'attributo ha una struttura interna)

es. dato strutturato in attributo del cliente (l'indirizzo contiene via, n. civico, CAP), DATA è un caso particolare (è convenzionale)

es. dato semplice: integer, decimal (virgola fissa), float (virgola mobile), string (char/ var char), date, time, boolean

generi, valori possibili:

thriller

giatto

drammatico

romantico

avventuroso

} TIPO BASICO (il generi è una enumerazione)

- - -

Come è fatto un class diagram

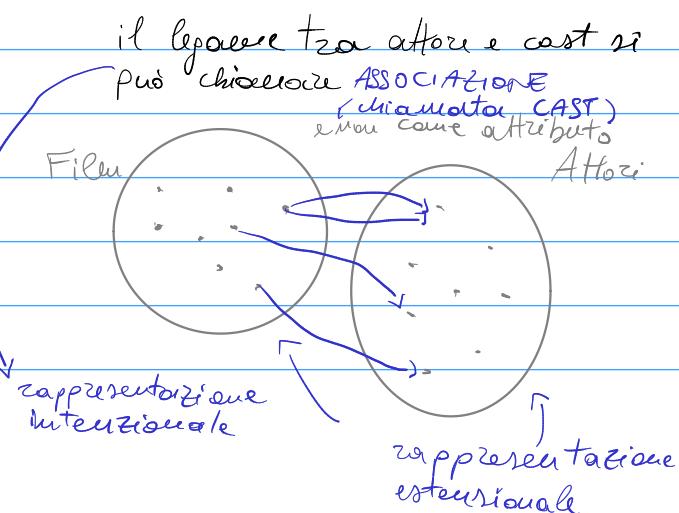
Una classe ha una definizione simile di "entità" ma ha delle caratteristiche aggiuntive. Graficamente è rappresentata da un rettangolo che è separato in 3 parti. La classe ha uscites per rappresentare l'entità (e ha un nome, non ci sono classi con lo stesso nome). Due spazi di definizione (parti strutturali), una parte rappresenta la collezione di attributi associate alle classi, che descrivono l'aspetto statico della classe, e l'altra parte descrive l'aspetto dinamico (metodi: operazioni che alterano gli stati) ci occupiamo dell'aspetto statico.

	FILM		«enumerazione» ENUM-GENERIC
Nella classe ci sono gli attributi	+Titolo(100): VARCHAR +Regista(1, *): VARCHAR +ANNO(1): CHAR +DataUscita(01): Date +Genre(1, *): ENUM-Genre	Nell'enumerazione ci sono i possibili valori	Triller Drammatico Giallo Poliziesco Animazione
→ tipo di associazione all'attributo			
partecipa			

05/10/2020

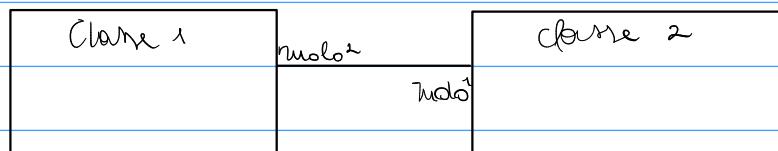
Una entità di cardinalità (1, 1) avrà un vincolo di consistenza

Attore
recita (0, *): Film
per partecipare è parziale *
è un'assoc. multe a molti
name
Cognome
dataN(1,1): DATE
dataH(0,1): DATE
Luglio N
Nazionalità



L'associazione che lega FILM e ATTORE, si chiama "ruolo".  
 "recita" è l'associazione che lega attore - film (ed è un ruolo) quindi  
 tra due entità ci sono due ruoli

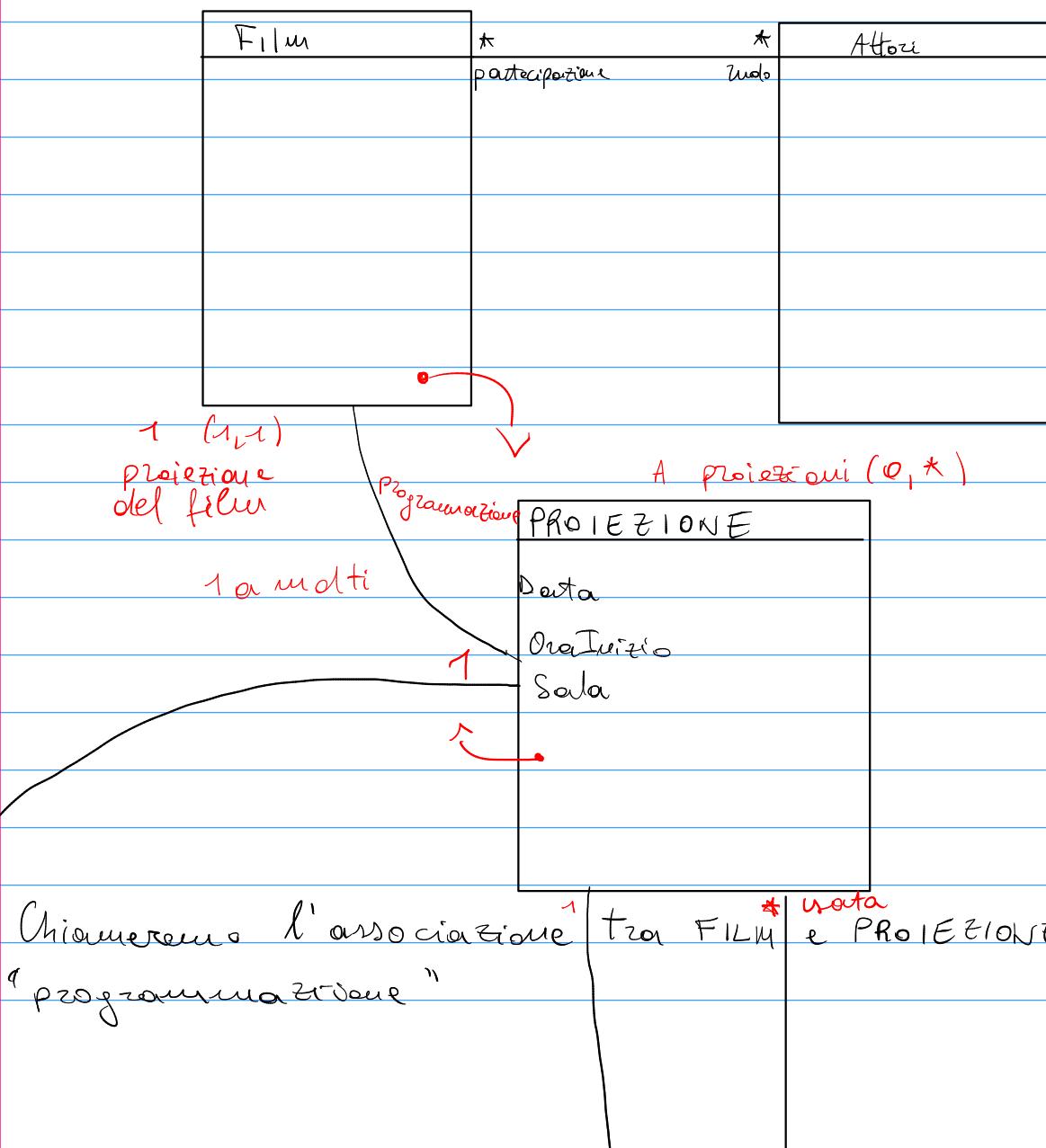
(Meme)

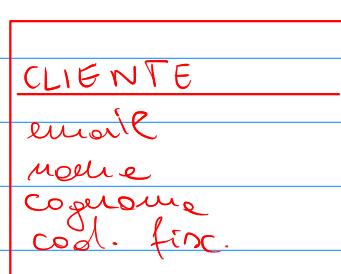


Nelle associazioni, la cardinalità caratterizza la struttura della base di dati. Mi dico indicazioni nelle entità.

L'associazione cast è di tipo MOLTI(\*) A MOLTI(\*) (perché: molti è su entrambe le direzioni)

Molti a molti

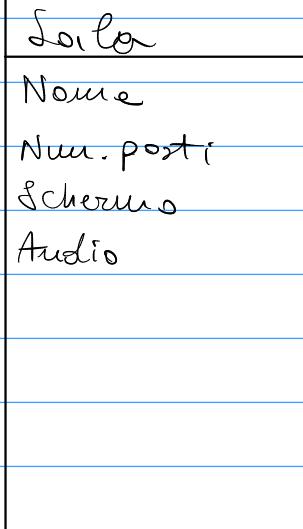




Prenotazione

N. posti

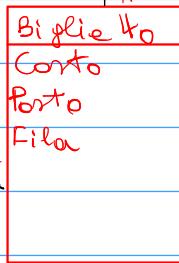
1 in



1 \*

(0,1)

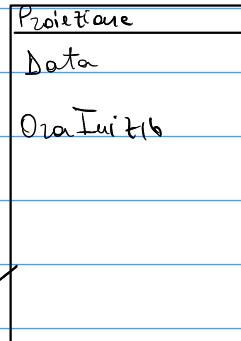
\*



più basso livello

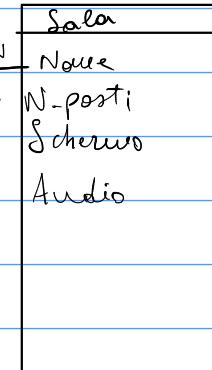
Invece di creare l'entità prenotazione, crea l'associazione tra cliente e proiezione

le proiezioni  
(0,\*)



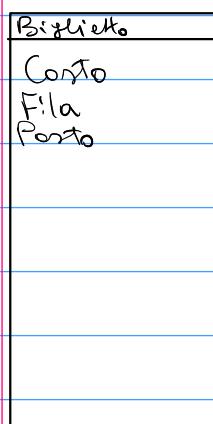
USA TA

\*



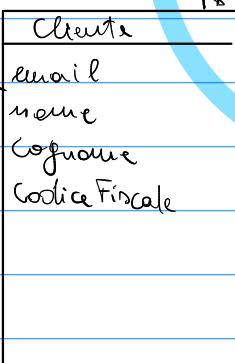
1 N

\*



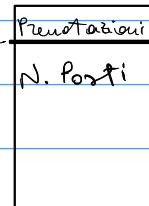
\*

(0,1)



Prenotazione

\*



CLASSE

DI

ASSOCIAZIONE

esprime attributi  
tramite associazioni  
e non tramite entità

Le classi di associazioni sono indispensabili per esprimere attributi delle associazioni MOLTI - A - MOLTI

Vincolo interno : legare tra biglietto e sali (regola di coesistenza)

il vincolo mi permette di scrivere un numero limitato (non posso riceverlo nel class diagram)

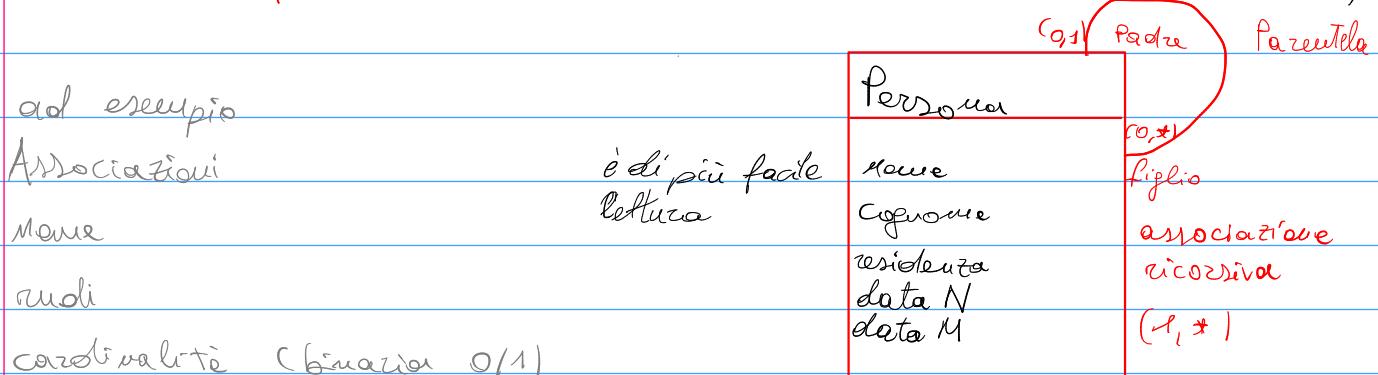
Progettazione -> Class Diagram

Ettità -> Classi

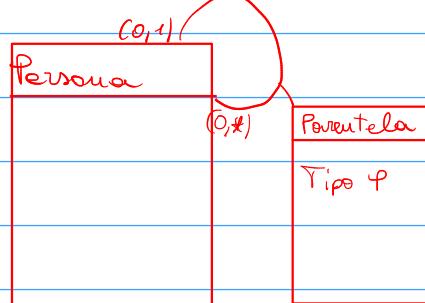
Associazioni -> Associazioni tra classi

Attributi delle associazioni -> Classi di associazione

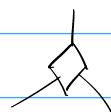
i vincoli che possono avere di cardinalità e di tipo (con le entità)



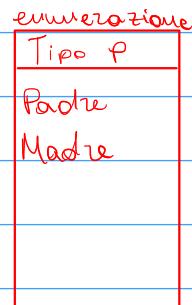
grado : è il numero di entità coinvolte



Per determinare il grado : è il numero di classi coinvolte simultaneamente

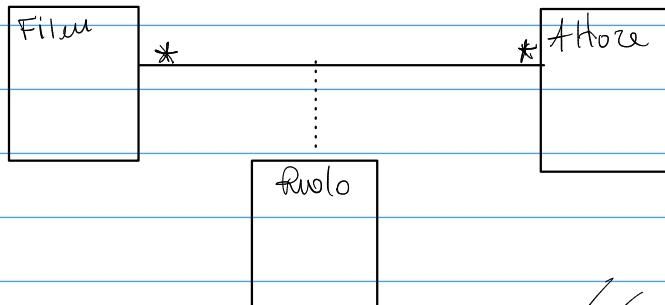


Grado 3, graficamente



06/10/2020

Grado associazioni : il numero delle classi che partecipano all'associazione.  
Le più ricorrenti sono quelle di grado 2. Ricorreva associa la medesima classe



+ alto livello  
Ci preferisce questa per la prog. concett.

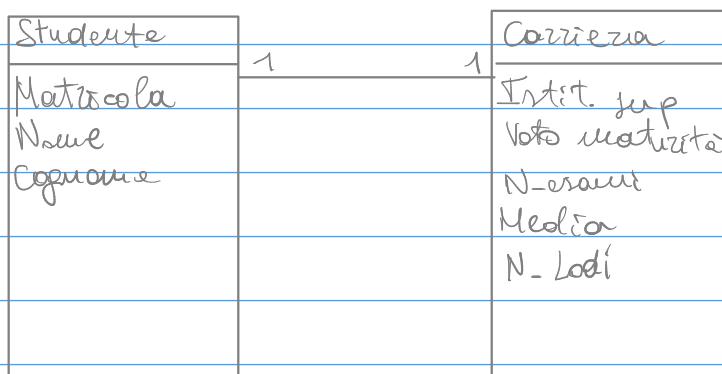


+ basso livello  
≤ IMPLEMENTAZIONE

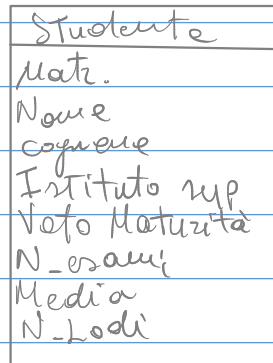
istanza di associazione

Cos'è l'associazione uno - a - uno

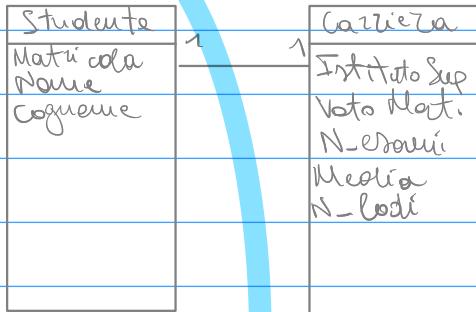
Esempio



Sceglio una delle due  
in base alle interrogazioni  
che devo fare  
(dipende dal contesto)



Vantaggi:  
Non devo  
calcolare i  
dati:  
(li calcolo  
una sola volta  
e li uso tante  
volte)



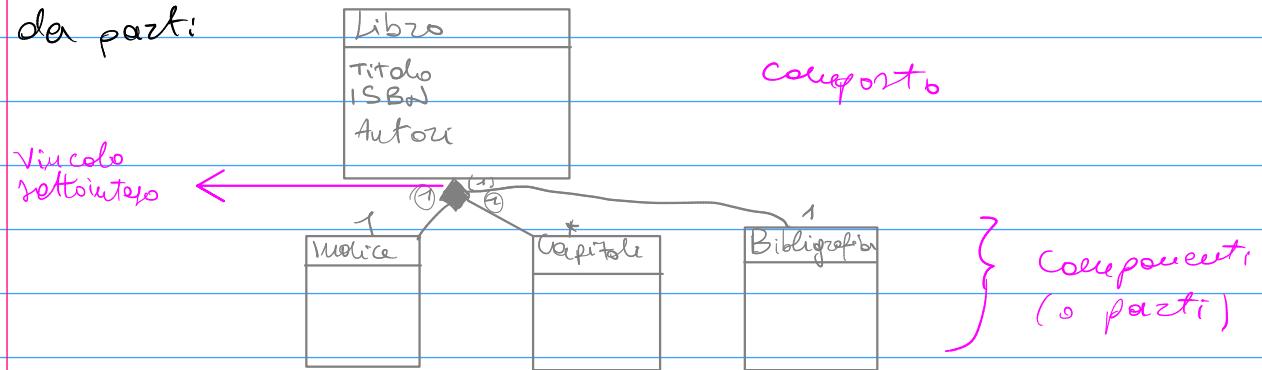
problema: Addebito

mesimi (è calcolato)  
media (è calcolata)

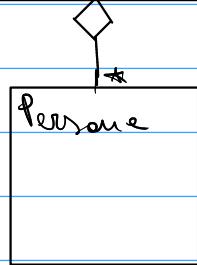
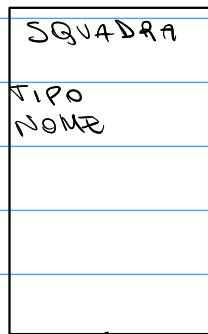
Svantaggio:  
- La gestione  
della consistenza

### Specie di associazioni delle associazioni:

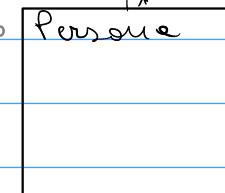
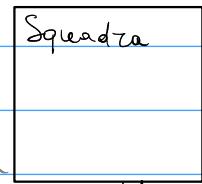
- **Composizione** permette di sottolineare che c'è un tutto composto da parti:



- A un componente è associato ad un unico composto
- B le componenti non possono esistere senza il composto
- **Aggregazione** Non sostengono i vincoli A e B, non ha regole di coerenza



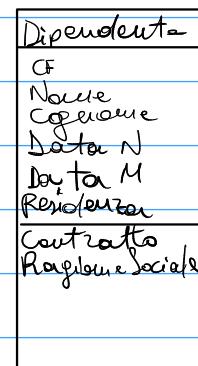
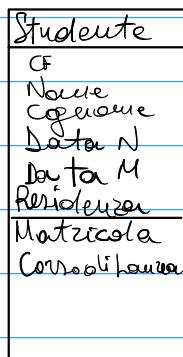
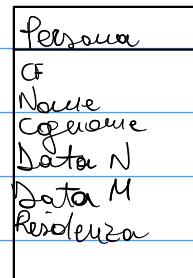
Una squadra è composta solo da persone  
 ←→  
 è la stessa cosa  
 →  
 non ho  
 l'obbligo che una persona sta in una  
 squadra soltanto



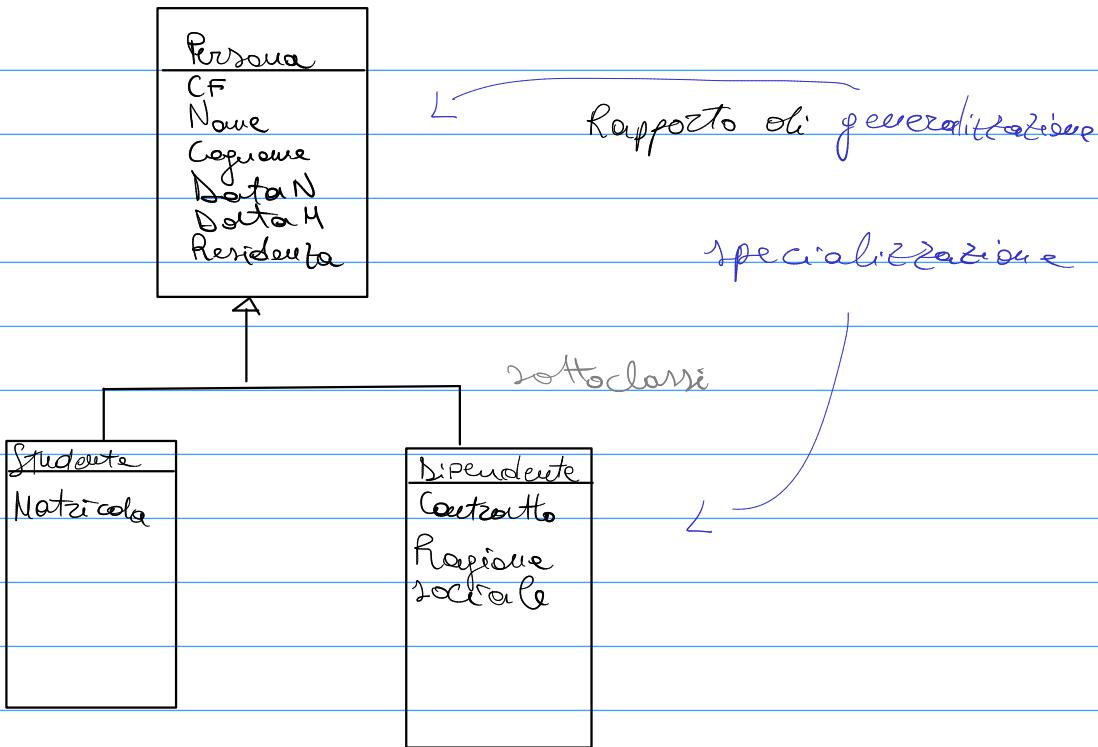
Le componenti hanno vita autonoma rispetto all'aggregato  
 (cosa che non succederà prima con le composizioni)

08/10/2020

## Specializzazione di entità

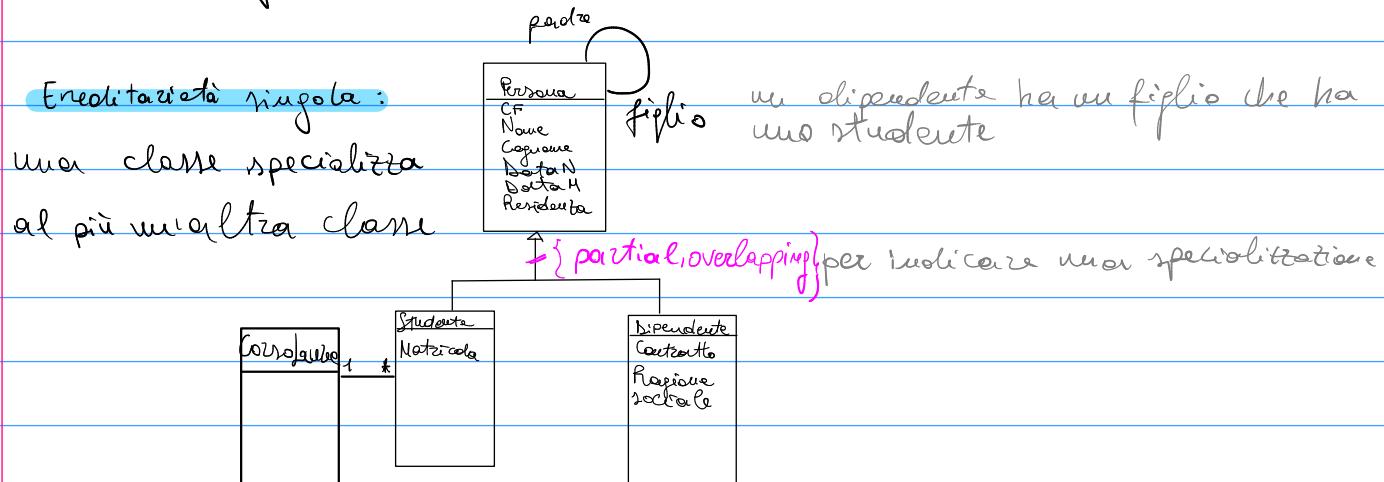


Nei class diagrammi posso avere una entità più generale



Gli attributi della specializzazione sono i mesi attributi propri e quelli della generalizzazione

**IMPORTANTE:** Una istanza della specializzazione è anche istanza della sua generalizzazione (non vale il viceversa)

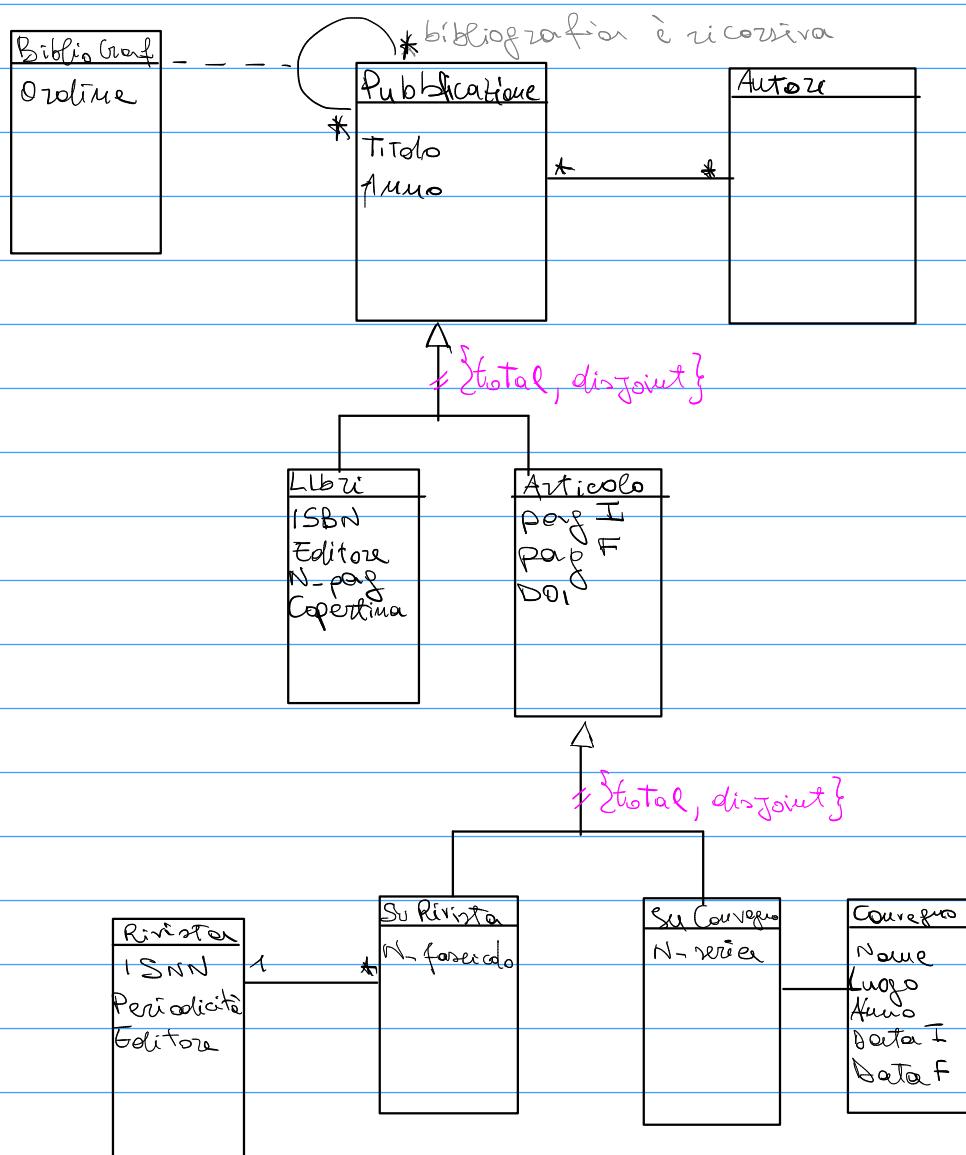


Una specializzazione può essere totale o parziale, disjoint o overlap.

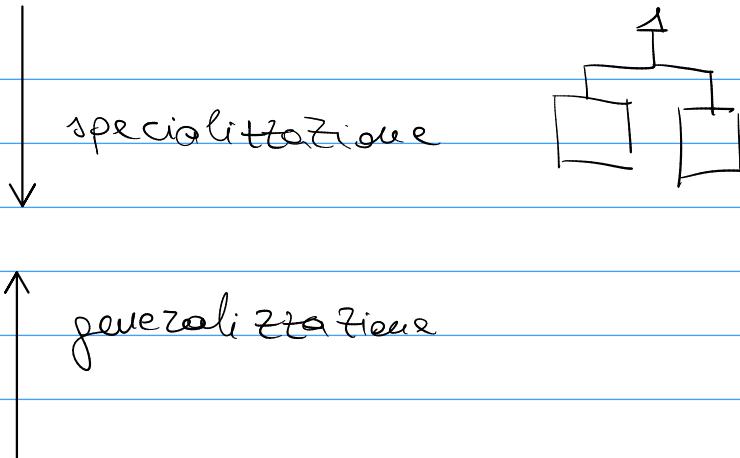
Totale se ogni persona è o un studente o un dipendente  
 Parziale se ogni persona può essere specializzata o in studente o in dipendente (non obbligatoriamente)

Dissintesa se quando specializza, non obbligato a specializzarlo  
(comune o nell'altra)

Overlap se quando fa la specializzazione eredita simultaneamente  
(in più d'una), è in sovrapposizione.



12/10/2020



Ogni istanza di specializzazione lo è anche per la generalizzazione

Ereditarietà degli attributi (dalla classe padre)

Ereditarietà delle associazioni (dalla classe padre)

Ragione per l'utilizzo delle specializzazioni:

- i concetti hanno delle naturali strutturazioni per archiche (ad albero)
- i class diagrammi che in genere risultano, sono più compatte e quindi maggiore legibilità

Class Diagram: descrive dal punto di vista concettuale le strutture dati di un problema

- non siamo interessati ai dettagli implementativi
- (i dettagli saranno curati in fase di progettazione logica e fisica)

Nella fase della prog-logica:

- adottare un modello dei dati (modello dei dati relazionale) adotteremo il modello relazionale dei dati
  - ↳ NoSQL
  - (SQL relazionale)

- Nel passaggio dal modello concettuale al modello logico (relazionale) avremo delle difficoltà per caratteristiche usate nei class diagram:

Relazionale  
oggetti  
ORACLE  
POSGRESQL

- Attributi strutturati (il modello relazionale dei dati vuole valori non strutturati per gli attributi)
- Attributi con valore multiplo (il modello relazionale esige valore unico)
- Il modello relazionale non supporta direttamente le gerarchie di specializzazione e generalizzazione

Cosa ho prodotto nella progettazione concettuale:

- CLASS DIAGRAM (non sempre sufficiente, deve essere accompagnato da altri documenti) perché?

1) Una classe abbia 100 attributi (il class viene annotato dalla classe)

Soluzione: un documento di supporto chiamato dizionario delle classi

Classi	Attributi
Descrizione	Nome: tipo descrizione

Riprendiamo i class diagram del documento:

Dizionario delle classi

Classe	Descrizione	Attributi
Documento	descrivere un documento generico scritto da un autore	Nome: tipo stringa è il titolo Titolo: // è il titolo del documento

## Dizionario delle associazioni

Per ogni  
associazione

Associazione	Descrizione	Partecipanti
Bibliografia	descrive il legame tra documenti date dalle citazioni bibliografiche riportate in ogni documento	- documento; ruolo Cita(0,*); documenti; citati in bibliografia  - documento; ruolo citato da (0,*); documenti che citano il documento finito

## Dizionario dei vincoli

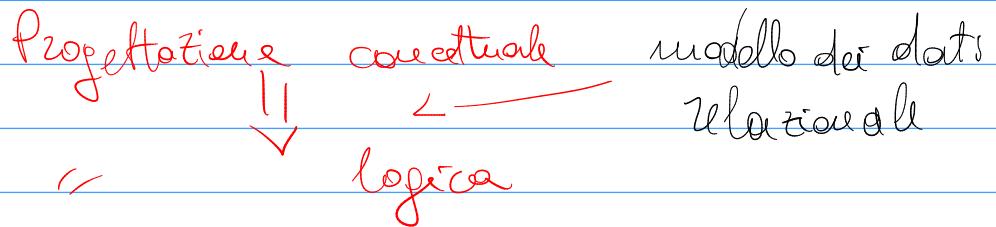
- Contiene i vincoli di coerenza che i dati del database devono rispettare
- Alcuni vincoli sono già espressi nel class diagram
- ad esempio: vincolo di cardinalità (di attributi e di associazioni)
- Attributi: total/disjoint delle specializzazioni

Nome
Cognome
data N
comune Nascita
CF ←
età

Vincolo  
vincolo (Attributo calcolato)

- a) il valore del codice fiscale deve essere coerente  
con i valori dei dati analogici

- Non vanno inseriti vincoli che sono già espresi nel Domain;
- i vincoli aggiuntivi dipendono dal problema in esame.
- i vincoli garantiscono la qualità interna dei dati della BD.



### Modello dei dati relazionale

- Schema relazionale

1) Un nome

2) Nomi di attributi

3) Il dominio dei valori di ciascun attributo

esempio: DATA(ANNO, MESE, GIORNO)

$$\text{dom(GIORNO)} := \{1, \dots, 31\}$$

$$\text{dom(MESE)} := \{1, \dots, 12\}$$

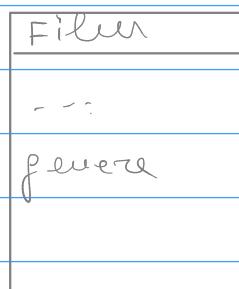
$$\text{dom(ANNO)} := \{1950, \dots, 2030\}$$

dipende dove  
voglio collocare il  
dominio

METADATI } Lo schema per una base di dati sarà un insieme di  
} schemi relazionali

ci ricorda una classe

è un  
dominio



Fissato uno schema relazionale  $R(A_1, \dots, A_n)$  una relazione  
o in  $R(A_1, \dots, A_n)$  è un sottoinsieme del prodotto cartesiano  
dei domini degli attributi

$$\mathcal{R} \subseteq \{1850 \dots 2030\} \times \{1 \dots 12\} \times \{1 \dots 31\}$$

zona delle terne

**insieme**  $\mathcal{R}_1 = \emptyset$  insieme vuoto = relazione vuota

(è la più piccola relazione nullo schema)

$\mathcal{R}_{MAX}$  = preso lo schema di prima - -

(il prodotto cartesiano è la relazione più grande sullo schema)

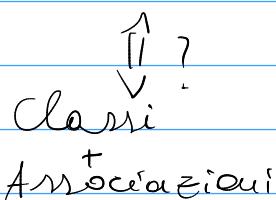
**Caratteristiche dell'insieme (in questo collezione)**

- Non ha molzione di molteplicità degli elementi (non ha duplicati)
- Le relazioni non sono ordinate

13/10/2020

Schemi relationali

(metadati)



Relazioni

(DATI)

Insieme di istanze di classi e associazioni

- Gli schemi relationali sono vicini strutturalmente alle classi

Person (cf, nome, cognome, datan)

Person
cf
nome
cognome
datan

Preso uno schema relazionale  $R(A_1, \dots, A_n)$

Superchiave è un sottoinsieme degli attributi; associando i valori agli attributi della superchiave si identifica in maniera univoca una istanza dello schema relazionale (ma ci sono due istanze distinte che assumono gli stessi valori negli attributi della superchiave)

Persona
CF
Matricola
Nome
Cognome
DataN

$\hookleftarrow \rightarrow$  Studente (CF, matr., nome, cognome, dataN)

Superchiavi:

- $\{CF, matr.\}$  non è una chiave candidata
- $\{CF\}$  "
- $\{matr.\}$  CHIAVE CANDIDATA
- $\{CF, matr., nome, cognome, dataN\}$  Non è chiave candidata

\* Dato uno schema esiste sempre una superchiave (l'insieme di tutti gli attributi è una superchiave)

Chiave Candidata è una superchiave minimale,

Eliminando un attributo dalla chiave candidata l'insieme risultante non è più una superchiave

Superchiave con un unico attributo è una chiave candidata  
Possono esistere più di una chiave candidata nello schema relazionale

ESAMI (Matricola, Corso, Voto, Data, Lode)

Primeria

$\{Matricola, Corso\}$  è una superchiave e candidata

$\{Matricola, Data\}$  Non è una superchiave

$\{Matricola\}$  Non è una superchiave

## Chiave Primaria

È una chiave candidata scelta come identificativo nello schema relazionale.

Se c'è un'unica chiave candidata, quella è anche chiave primaria, se ci sono più chiavi candidate devo scegliere la chiave primaria

Studente (CF, matr., nome, cognome, dataN)

Chiavi candidate {matricola} {CF}

Chiave Esterna è un sottosistema degli attributi (nello schema relazionale) che sono in corrispondenza binaria con la chiave primaria in un altro schema

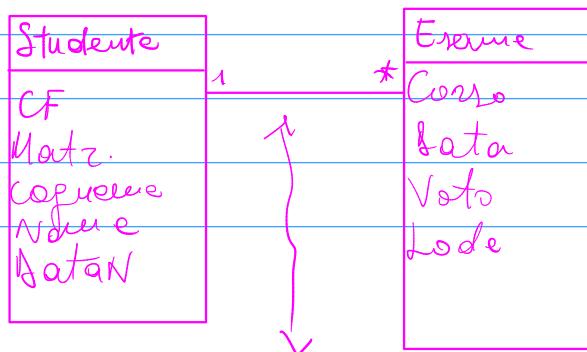
Studente (CF, matr., nome, cognome, dataN)

Esercizi (Matricola, Corso, Voto, Data, Lode)

abbiamo una corrispondenza binaria:

- stesso numero di attributi
- gli attributi corrispondenti devono avere lo stesso dominio (non serve che abbiano lo stesso nome)

{Matricola} in Esercizi è chiave esterna per Studente



dato un esame  
ho individuato lo  
studente che ha  
sostenuto l'esame

## Chiave primaria

Studente (cf, Matz, Nome, Cognome, DataN)

Esercizio (Matz, Corso, Data, Voto, Lode)

## Tutte generali

- 1) individuare la chiave primaria di A
- 2) aggiungere in B una chiave esterna in corrispondenza con la chiave primaria di A

A ( $A_1, \dots, A_n$ ) supponiamo che la chiave primaria sia  $A_i$   
cioè  $A\{A_1, \dots, A_i\}$   
e  $B(B_1, \dots, B_n, A_1, \dots, A_i)$  chiave esterna

Progettazione concettuale  
↓  
modello dei dati  
relazionale

15/10/2020

Progettazione Logica

## Assunzioni sul class diagram

- 1) non ci sono attributi strutturati nelle classi
- 2) non ci sono attributi con valori multipli
- 3) non ci sono gerarchie di specializzazione

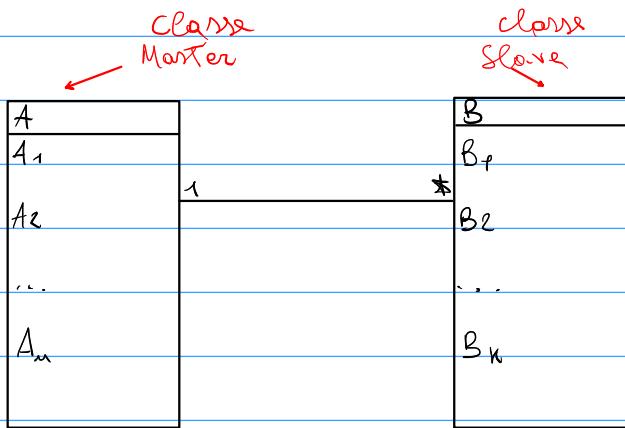
- Le entità (classi) sono descritte da uno schema relazionale

- Le associazioni sono codificate utilizzando le chiavi esterne

Il modo in cui una associazione viene codificata dipende dalla cardinalità dell'associazione

- uno -o - molti (binarie, grado due)
- associazioni binarie molti -o - molti
- associazioni molti a molti; con classe di associazione
- associazioni unarie
- associazioni uno a uno

Uno - a - molti

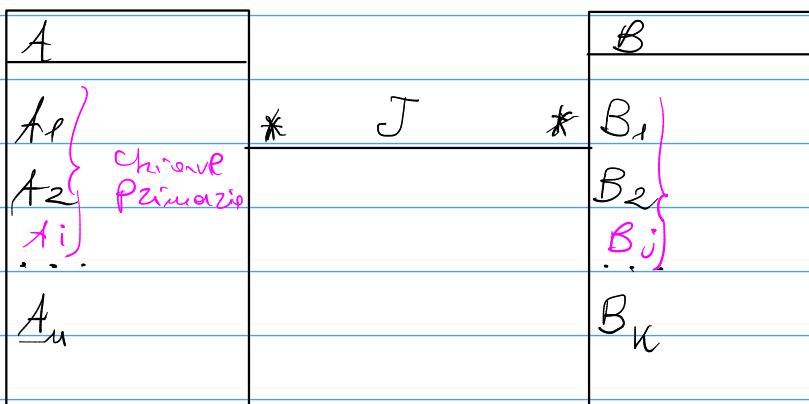


- 1) individuare la chiave primaria di A
- 2) aggiungere in B una chiave esterna in corrispondenza con la chiave primaria di A

$A(A_1 \dots A_n)$  supponiamo che la chiave primaria di A sia  $\{A_1, \dots, A_i\}$

$B(B_1, \dots, B_k, A_1, \dots, A_i)$  chiave esterna di A ( $A_1, \dots, A_i$ )

Molti - a - molti



- 1) Individuare le chiavi di A e di B
- 2) Creare uno schema relazionale per A e uno per B
- 3) Creare uno schema per l'associazione. Lo schema comprende le chiavi esterne per A e per B

$A (A_1 \dots A_n)$

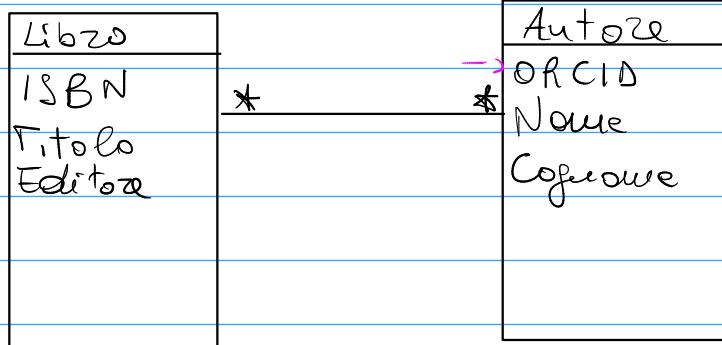
$B (B_1 \dots B_k)$

$J(A_1 \dots A_i, B_1 \dots B_j)$

chiave esterna (non riporta tutto  
la chiave chiave, basta che abbiano  
corrispondenza)

Esempio

chiavi ->



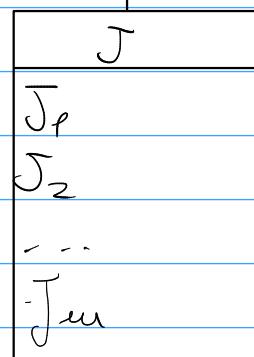
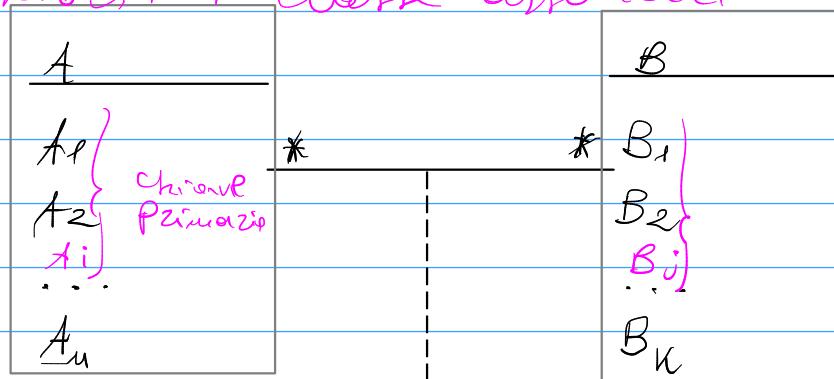
Libro (ISBN, Titolo, Editore)

Autore (ORCID, Nome, Cognome)

Scrire (ISBN, ORCID)

metto insieme e ne  
esce che:

Molti -o- Molti + classe associazione



classe di  
associazione

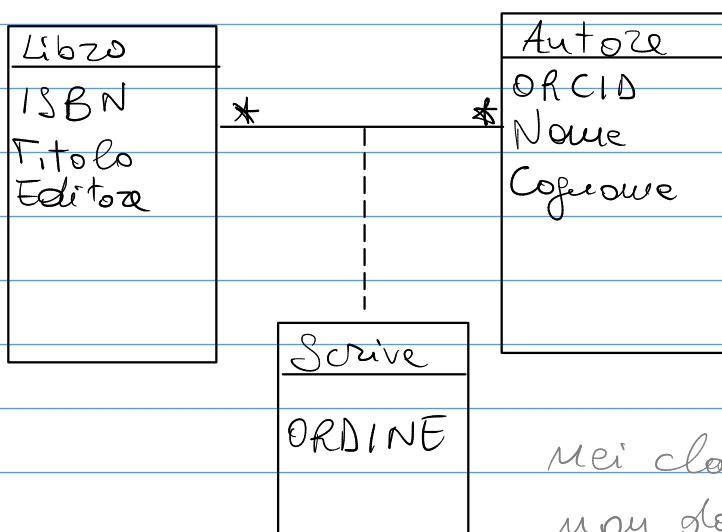
- 1) Individuare le chiavi di A e di B
- 2) Creare uno schema relazionale per A e uno per B
- 3) Creare uno schema per l'associazione. Lo schema comprende le chiavi esterne per A e per B
- 3.a) creare uno schema per la classe di associazione lo schema comprende le chiavi esterne per A e per B e per gli attributi della classe di associazione

A ( $A_1 \dots A_n$ )

B ( $B_1 \dots B_k$ )

J( $A_1 \dots A_i, B_1 \dots B_j, S_1 \dots S_m$ )

Esempio



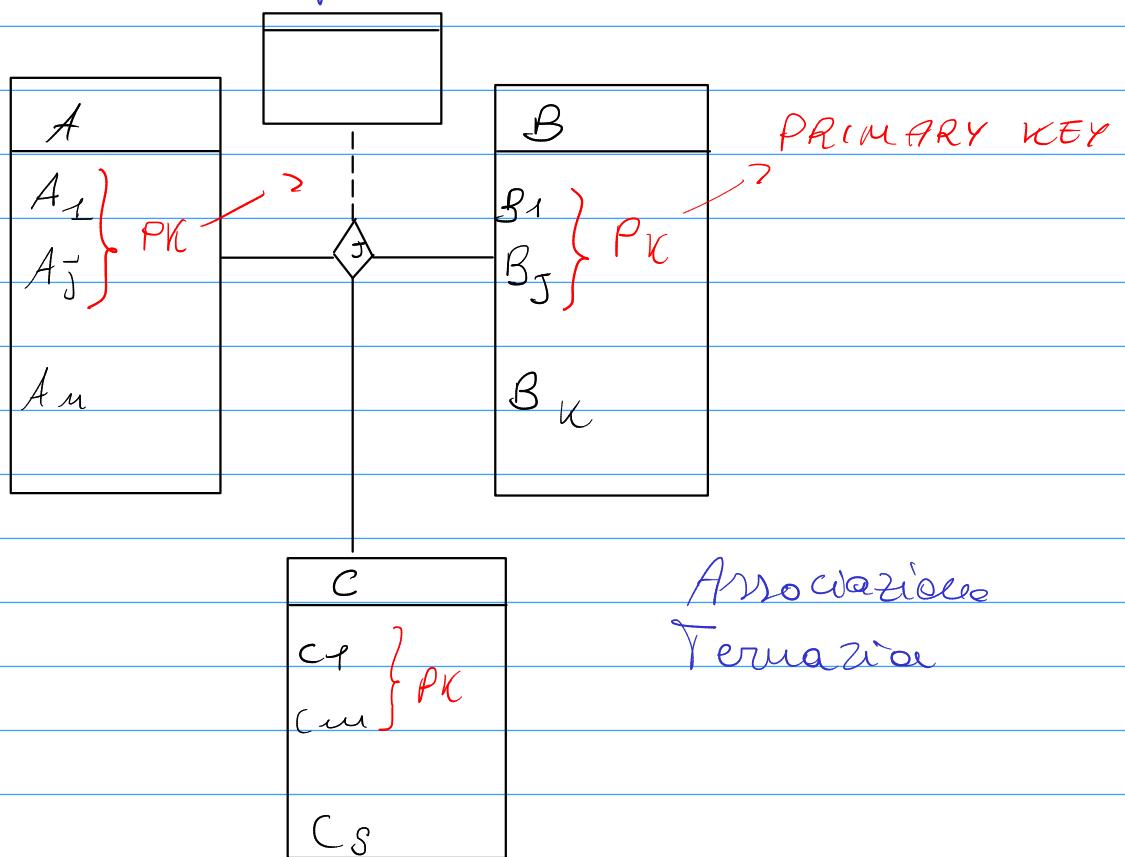
Libro (ISBN, Titolo, Editore)

Autore (ORCID, Name, Cognome)

Scrive (ISBN, ORCID, ORDINE)

Le class diagrammi  
non dovrebbero avere  
le chiavi esterne  
(vanno solo nello  
schema): sono un  
artificio tecnico

## Associazioni di profondità > 2

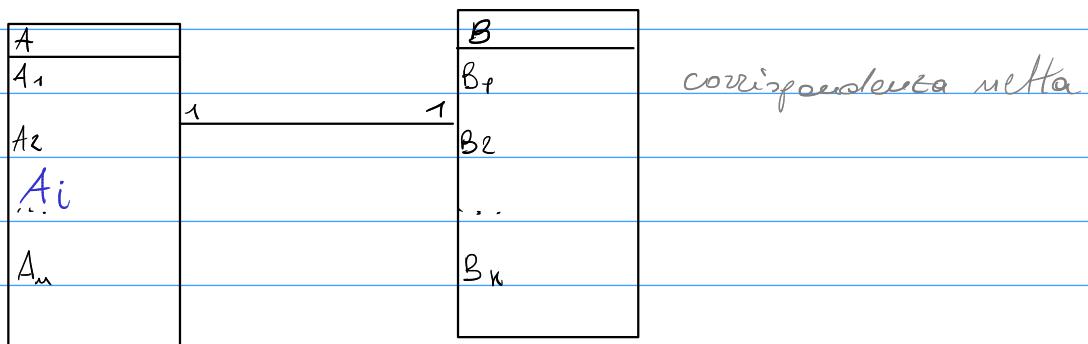


- 1) Creare uno schema per ogni classe
- 2) Individuare le chiavi primarie di ogni classe
- 3) Creare uno schema che contenga le chiavi esterne di ogni classe

$A(A_1 \dots A_n)$   
 $B(B_1 \dots B_n)$   
 $C(C_1 \dots C_s)$   
 $J(A_1 \dots A_i, B_1 \dots B_j, C_1 \dots C_m)$   
chiavi esterne

## Associazione uno-a-uno

- casi speciali di associazioni uno a molti  
(c'è uno sbilanciamento del ruolo)



corrispondenza netta

Per ogni istanza di A ne esiste una, e una sola  
associata di B e viceversa

NON SI DICE la chiave primaria di A in B  
SI DICE la chiave esterna di B in A

### Soluzioni:

1) Codificare A, Codificare B con la chiave esterna di A in B

$A(A_1 \dots A_n)$  chiave primaria

$B(B_1 \dots B_k, \underbrace{A_1 \dots A_i})$

chiave esterna

2) Codificare A con la chiave esterna di B in A, codificare B

$A(A_1 \dots A_n, \underbrace{B_1 \dots B_j})$  chiave esterna

$B(\underbrace{B_1 \dots B_j}, \dots B_k)$  chiave primaria

3) Codificare A con la chiave esterna di B in A, codificare B con la  
chiave esterna di A

$A(A_1 \dots A_n, B_1 \dots B_j)$  chiave primaria di A

$B(\underbrace{B_1 \dots B_j}, \dots B_k, \underbrace{A_1 \dots A_i})$

chiave esterna di A

(solo se si tratta di un'associazione uno-a-uno)

4) Un unico schema che comprende gli attributi di A e B

$U(A_1 \dots A_n, B_1 \dots B_k)$

solti<sup>re</sup> u<sup>nic</sup>a

e non faccio uso

di chiavi esterne e  
interne

Esempio

Studente		Carriera	
Matz		N-esami	
Noeve	1	N-Lode	
Cogn		Media	
		COD-LAUREA	

4) Studente (Matz, Noeve, Cognome, ... )

(Questa soluzione va bene solo se noi ogni interazione tocca tutti questi aspetti. Per le variazioni, fare una TABELLA UNICA, risulterà molto pesante dato che gli attributi di STUDENTE rimangono fissi, mentre la CARRIERA no.)

1) Studente (Matz, Noeve, Cognome )

Carriera ( MATR, N-esami, N-lode... )

chiave  
esterna

Sotto il punto di vista delle variazioni (update), consente di lavorare in maniera più mirata. Visto che gli update vengono fatti in memoria principale, ora avendo dei record più piccoli è vantaggioso perché il

costo sulla base di dati (le operazioni) sono quelle legate agli trasferimenti della memoria principale in quella secondaria.

Prendiamo in considerazione il seguente esempio:

Studenti	Esami
MATR	DATA
NAME	VOTO
COGNOME	LODE
DATN	CORSO
RESIDEN	

È sbagliato scrivere:

STUDENTI - ESAMI (MATR, NAME, COGNOME, DATN,  
RESIDEN, DATA, VOTO, LODE,  
CORSO)

Perché i dati che rimangono fissi, verranno poi  
riplicati puntualmente (e il DB diventerà più pesante)

pubblicazione
DOI
TITOLO
ANNO
LUGO

0..1

LIBRO (ISBN, N\_PAG, DOI)  
ARTICOLO (COD\_A, TIPOART, PAG\_I,  
PAG\_F, DOI)

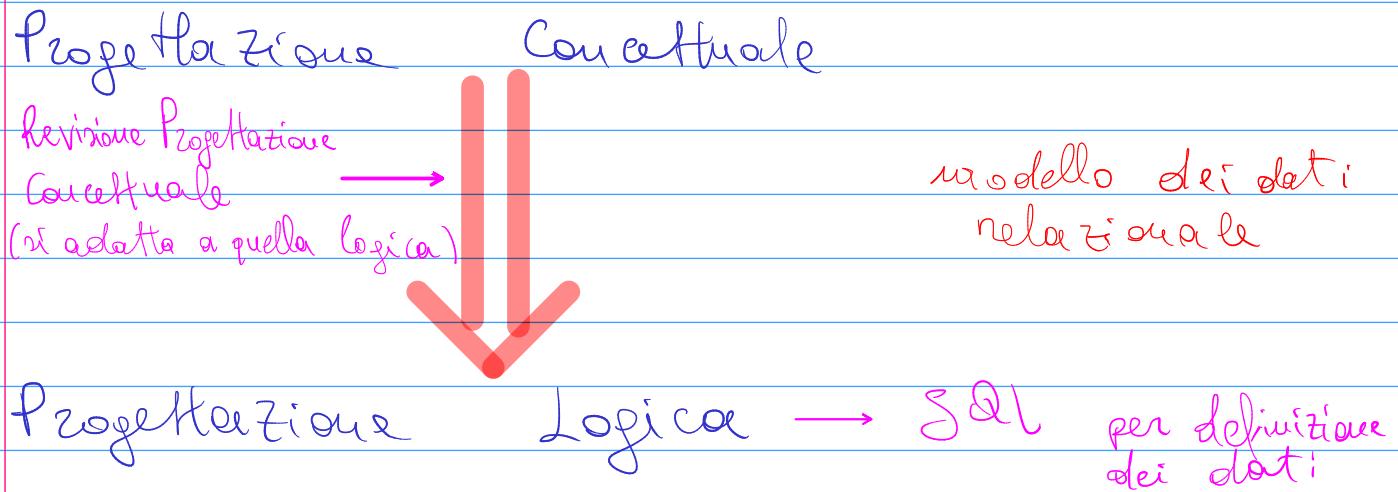
PUBBLICAZIONE (DOI, TITOLO, ANNO,  
LUGO)

ARTICOLO
TIPOART
PAG_I
PAG_F
COD_A

faccendo un solo  
schema, anche  
attributi non  
utilizzati (o libro  
o articolo)

Ri-capito Lando ...

19/10/2020



- Assunzioni nei class diagram (generali)
  - 1) Non ci sono attributi strutturati nelle classi
  - 2) Non ci sono attributi con valori multipli
  - 3) Non ci sono gerarchie di specializzazione  
non del tutto vere, in tutti i casi

Ri-strutturazione del class diagram di progettazione

Analisi delle classi

Analisi degli attributi derivati e delle ridondanze

Analisi delle associazioni uno-a-molti

Analisi degli attributi strutturati

Analisi degli attributi a valore multiplo

Codifica gerarchie

## Analisi delle chiavi

Quando abbiamo uno schema relazionale, è sempre possibile trovare una **superchiave**, di conseguenza è possibile trovare una **chiave primaria** (una fra le chiavi minimali, l'insieme degli attributi formano una superchiave). A cosa servono le chiavi primarie?

- Identificare univocamente in modo **ucciso**

CORSO AA (TITOLO, AA, TITOLARE, GRUPPO)

ESAMI (MATRICOLA, CORSO, VOTO, LODE, DATA)

CORSO
TITOLO
AA
TITOLARE
GRUPPO

ESAMI
MATRICOLA
CORSO
VOTO
LODE
DATA

PRIMARY KEY (TITOLO, AA, GRUPPO)

ESAMI (MATRICOLA, VOTO, LODE, DATA, AA, GRUPPO)

FOREIGN KEY (CORSO, AA, GRUPPO)

solo in corrispondenza

Si è soliti usare delle primary key con pochi attributi, per ovviare al problema di una primary key con vari attributi:

1) => **CHIAVI SINTETICHE / CHIAVI SURROGATE**

è un attributo chiave che non "appartiene" al concetto, ma che si utilizza per comodità tecniche. ad esempio sarebbe utile inserire un Codice Corso all'interno di CORSO AA, e fungerà come identificativo puro, privo di informazioni, che utilizzerà come chiave compatta. La chiave surrogata va inserita nel class diagramm revindicato, e **NON IN QUELLO CONCESSIONALE**

CORSO AA (Cod\_C, TITOLO AA, TITOLARE, GRUPPO)

ESAMI (MATR, VOTO, LODE, DATA, Cod\_C)

## Analisi attributi derivati

Personer
CF
Name
Cognome
DoctoR
--
/Età

è opportuno storediziale l'età ?

PERSONA (CF, Name, Cognome, DoctoR, Età)  
?

non è una informazione primitiva

- memorizzare o calcolare ?

1) Dico sapere come e quanti utilizzerò l'attributo calcolato

VANTAGGI: non lo devo calcolare quando faccio le interrogazioni delle istanze

Svantaggi: se non è di lungo uso, spreco memoria  
+ potrebbe richiedere frequenti variazioni  
+ richiede la gestione della consistenza  
tra dati memorizzati e dati calcolati

## Valutazione associazioni uno-a-uno

(se nona ridondante, l'entità è calcolata)

Studente
Matri

1  
\*

Carriera
N-esame
Medie
N-bandi

1

Esoni
Corsa
Nota
Lode
Data

\*

1

-> associazione ridondante

Se decido di mantenere in memoria la carriera, devo mettere una regola di coerenza (ogni volta che aggiungo un esame, aggiorno gli attributi di carriera)

## Attributi Strutturati

$R(A_1 \dots A_n) \rightarrow \text{dom}(A_1)$  i valori del dominio non hanno una struttura che posso sfruttare (è un problema che deriva dal modello relazionale)

- Ad un attributo si associa un unico valore! quindi  
**NIENTE VALORI MULTIPPI**

Personna
CF
Name
Cognome
<u>Residenza</u>
Telefono [0,*]

← attributo strutturato  
← attributo multiplo

### Soluzioni:

1)

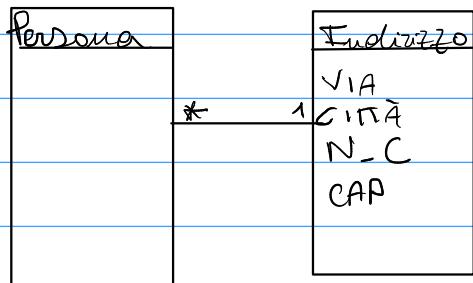
Personna
:
Via
N°C
CAP
CITTÀ
Nazione

quando gli elementi della residenza sono utilizzati nelle interrogazioni (singolarmente)

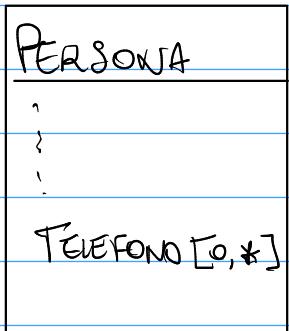
2)

Residenza
← è una stringa di caratteri Via Claudio, 21, 80125, Napoli, Italia

3)



Attributi a valori multipli



1) Un attributo per ogni valore

- NON SO QUANTI SONO I VALORI

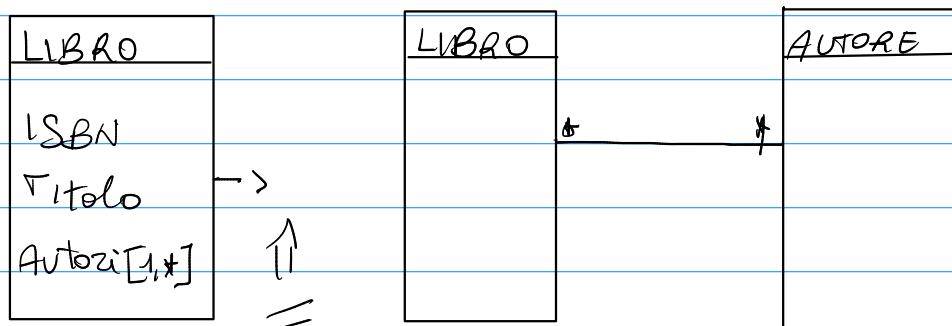
- ASSOCIARE UN

NUMERO DI ATTRIBUTI  
PORTA AL RISCHIO  
DELLO SCARSO UTILIZZO

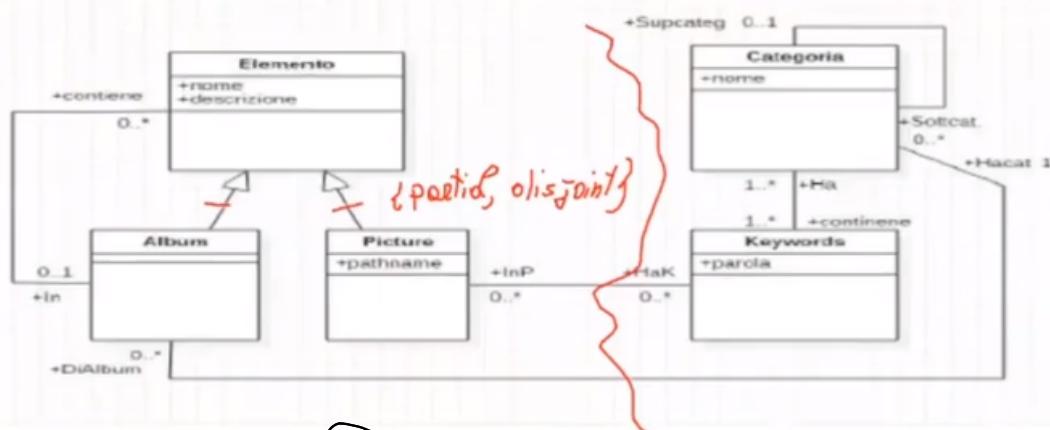
2) Codifica dei valori in un unico valore  
(ad es. stringa)

→  
l'estrazione dei valori da applicativo, e non da linguaggio di interrograzie

3) Entità esterna legata da associazione

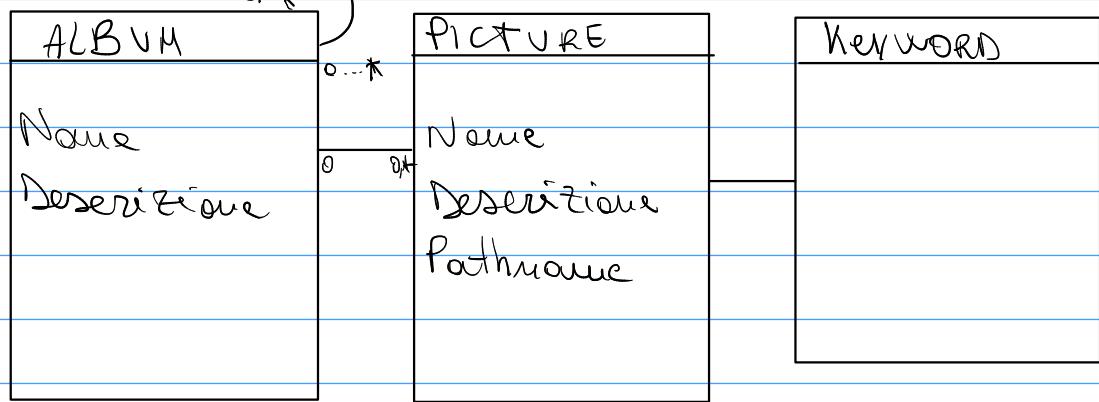


# Codifica Gerarchie



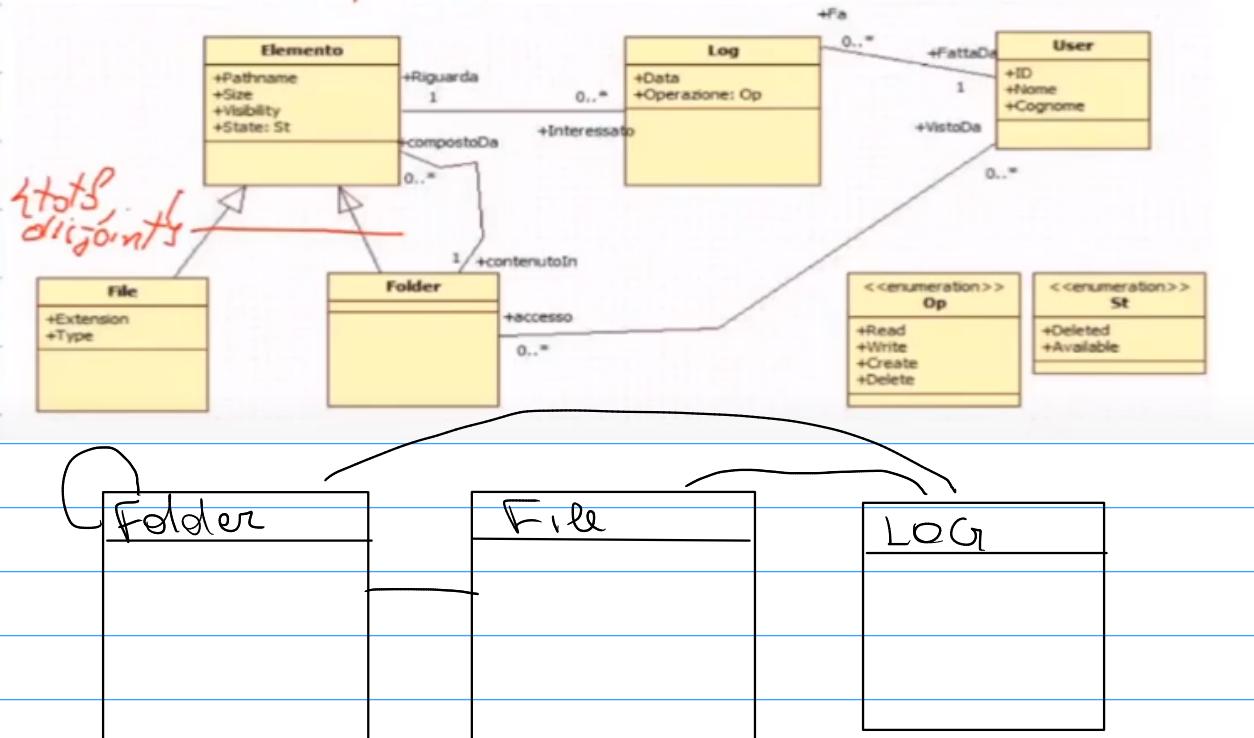
1)

G&W  
↓  
SPEC.



**NON** va bene, si può usare solo se la specializzazione è totale

## FILE SYSTEM



1) Schiacciato la generale su specializzazione

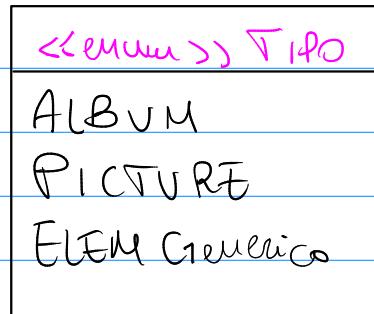
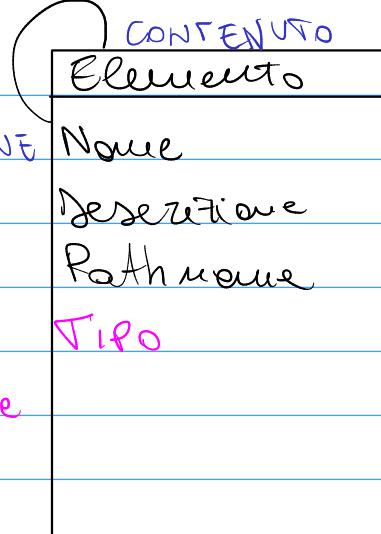
2) GEN

$\uparrow$  CONTIENE

SPEC

Attributo ->

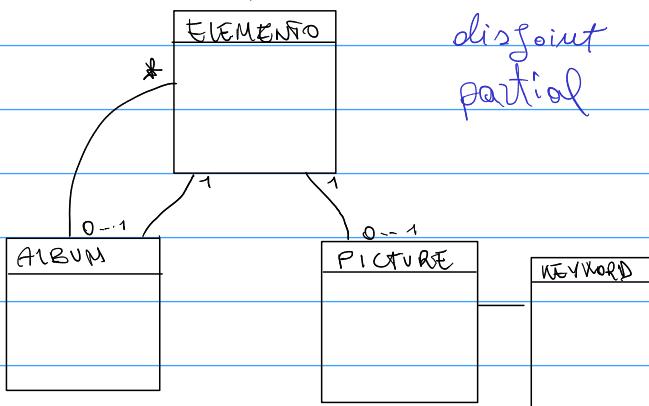
discriminante



Vincoli di consistenza aggiuntivi

- Se TIPO = PICTURE allora PathName ha valore associato
- " "  $\Leftrightarrow$  PICTURE " " " " NON " " "
- Se elemento partecipa alla ricorsiva come ruolo, CONTIENE allora TIPO = ALBUM

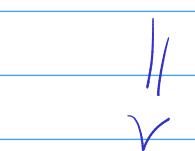
3) Trasformiamo le specializzazioni in binari associazioni



disjoint  
partial

Aaggiungere vincolo: **Disgiunzione**: un elemento non può essere associato simultaneamente ad ALBUM e PICTURE

Prog. concettuale → class diagram 1.0 20/10/2020



Revisione

Dizionario

- Classi
- Associazioni } 1.0
- Vincoli

Prog. concettuale → class diagram 2.0

- no gerarchie
- no attributi strutturati vincoli 1.0 + vincoli revisione
- no attributi di valore multiplo



Schema Logico

- Schema relazionali
- Chiavi primarie (primary key)
- Chiavi esterne (foreign key)

Prog. Logica

} Struttura portante  
dello schema logico



Metadati relazionali Standard SQL

- Definizione di strutture
- Definizione di Vincoli

Comandi Metadati:

inserimento  
create -,-.  
cancellazione  
drop  
modifica  
alter  
ricerca  
select

Comandi Dati:

insert  
delete  
update  
select

concentriamoci sui metadati

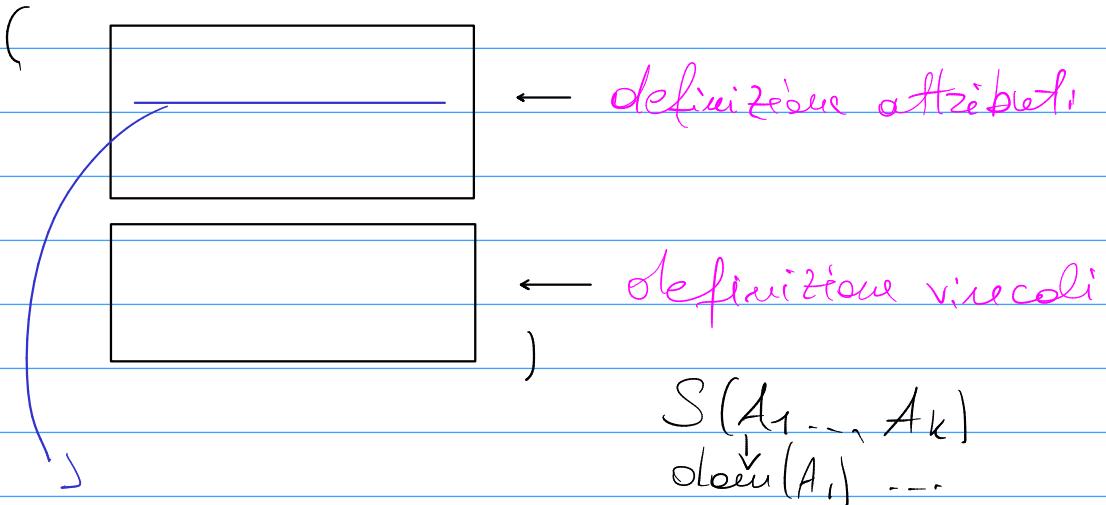
Schemi



Metadati

create schema <name>

$\rightarrow$  Schema relazionale  $\leftarrow$ , TABLE  
 create TABLE <name>  $\leftarrow$  unico nel DB



è unico  
 all'interno  
 della tabella

$\leftarrow$  <name attributo> <tipi attributo> [vincolo attributo]

Esempio

Studenti (Matr, CF, Nome, Cognome, DataN)

Esami (Matr, Corso, Voto, Lode, DATA)

CREATE TABLE studenti (

matricola	CHAR(9), PRIMARY KEY	char(u)
cf	CHAR(16), UNIQUE	{varyinf character(u)}
Nome	VARCHAR(20), NOT NULL	
Cognome	VARCHAR(20), NOT NULL	VARCHAR(u)
DataN	DATE NOT NULL	DATE

(YYYY-MM-DD)

YEAR(-) estrae anno

MONTH(-) = mese

DAY(-) // giorno

VINCOLI

TOTALITÀ - NOT NULL

TOTALITÀ + UNICITÀ  $\leftarrow$  PRIMARY KEY -  
 UNIQUE

CONSTRAINT < nome > & vincolo > [SET ENABLED / SET DISABLED]

in un modo alternativo:

CREATE TABLE studenti (

matricola CHAR(9),

cf CHAR(16),

Nome VARCHAR(20), NOT NULL

si può abilitare o

Cognome VARCHAR(20), NOT NULL

disabilitare il vincolo

dataN DATE NOT NULL

✓

CONSTRAINT pri1 PRIMARY KEY (matricola) SET ENABLED,  
(da solo è sufficiente)

CONSTRAINT uni1 UNIQUE (cf) SET ENABLED

)

Se voglio cancellare il vincolo:

-> ALTER TABLE studenti

dare if nome, matricola if vincolo

DROP CONSTRAINT uni1

è cancellato il vincolo Uni1

CONSTRAINT < nome > CHECK < espr. boole >

CONSTRAINT NN1 CHECK cognome IS NOT NULL

cognome IS NOT NULL OR nome IS NOT NULL

Eserci (matr, corso, data, lode, voto)

CREATE TABLE esami (

matr CHAR(9),

corso VARCHAR(20) NOT NULL,

data DATE NOT NULL,

Vincolo di  
dominio

voto INTEGER NOT NULL CHECK voto >= 0 AND voto <= 30

lode CHAR(1) DEFAULT = 'N' CHECK lode = 'N' OR lode = 'S'

CONSTRAINT v-lode CHECK lode = 'N' OR voto = 30 )

CONSTRAINT pri\_E PRIMARY KEY (matr, corso)

lode IN('N', 'S')

CONSTRAINT fk\_E FOREIGN KEY (matr)

REFERENCES studente(matricola)

FOREIGN KEY < lista attributi FK >

chiave primaria dell'altra tabella

REFERENCES < nome tabella riferita (lista attributi riferiti) >

## Vincoli

1) Vincolo Locale CHECK expr. bool.

2) Voto Compreso tra 0-30

3) Chiave Esterna vincolo di tipo interrelazionale  
(per elementi di tabelle diverse)

**Vincolo elementare:** esprime condizioni sui valori di un unico attributo

**Vincolo di esempla:** esprime condizioni su più attributi di una riga (riga)

**Vincolo intrarelazionale:** esprime condizioni su più record della stessa tabella

**Vincolo interrelazionale:** esprime condizioni su record appartenenti a tabelle diverse

CREATE TABLE < nome >

22/10/2020

att\_1 tipo\_1 [DEFAULT < valore def > [NOT NULL] [vincolo default]],

att\_n tipo\_n [DEFAULT < valore def > [NOT NULL] [vincolo default]],

CONSTRAINT < nome vincolo > < definizione vincolo >,

<def vincolo> :=

PRIMARY KEY (< lista attributi >)

UNIQUE (< lista attributi >)

CHECK < espressione booleana >

FOREIGN KEY (< lista attributi >) REFERENCES  
< nome tabella > (< lista attributi >)

FOREIGN KEY (att\_1 ... att\_n) REFERENCES

Al tipo di i ...

## PRIMARY KEY

gli attributi saranno referenziati dalle chiavi esterne, intendendo esprimere un vincolo di unicità, implica che tutti gli attributi della chiave primaria sono totali NOT NULL

- per garantire il controllo di unicità in maniera efficiente i DBMS usualmente costruiscono una struttura dati per agevolare l'operazione: INDICE (INDEX) (gli indici sono costate sotto il punto di gestione, anche le chiavi: la dichiarazione della chiave primaria si fa quando si vogliono avere dei benefici)

## FOREIGN KEY

- Referenzia una chiave primaria esterna
- Implica un vincolo di INTEGRITÀ REFERENZIALE
  - i valori che compaiono negli attributi di una chiave esterna DEVONO essere presenti nella chiave primaria referenziata
- inserimento nella tabella referenziata

inserisco in ESAMI '(N86/1000, ...)

esiste la matricola 'N86/1000' in STUDENTI

} Controlla il  
vincolo di  
integrità

## Violazione dell'integrità referenziale

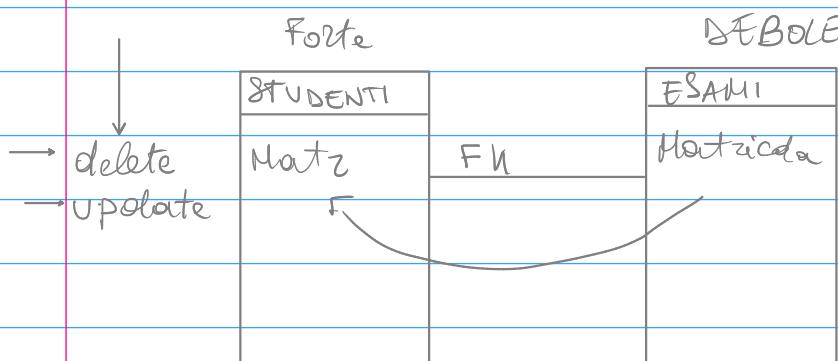
- DELETE nella tabella referenziata
- UPDATE nella tabella referenziata

## FOREIGN KEY FK1 (matr) REFERENCES

### STUDENTI (matrizza)

- N DELETE → No action
- CASCADE
- SET NULL
- SET DEFAULT

(se blocca l'operazione)



Riprendiamo la struttura della create table

26\10\2020

CREATE TABLE name tabella)

Vincolo  
/ dominio

(nuove attributi tipo [DEFAULT] [CHECK express\\_bool] )

[NOT NULL]

[UNIQUE]

[PRIMARY KEY]

per il vincolo: CONSTRAINT < nome constraint > PRIMARY KEY ( lista attributi )

UNIQUE ( lista attributi )

o vincolo di esempio

o -- INTRARELACIONALE

NO --

CHECK expression boolean

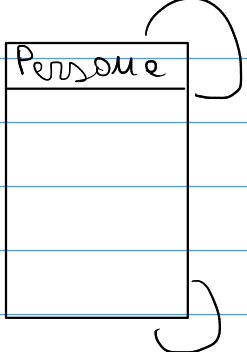
FOREIGN KEY ( lista attributi )

REFERENCES tabella ( lista attributi )

[gestione violazione integrità referenz.]

ON DELETE

ON UPDATE



PERSONE (CF, NOME, COGNOME)

PARENTEZA (CF, CF Madre, CF Padre)

1) Una persona ha al più un padre

CREATE TABLE parentela

CF CHAR (16),

CF Madre // ,

CF Padre // ,

100 200 400 CONSTRAINT v1 UNIQUE (CF, CF Padre)

100 200 300 100 300 UNIQUE (CF)

100 400

NON VA

BENE!

Quando viene cancellata una persona vengono cancellati  
l'informazione di parentela

CONSTRAINT FK1 FOREIGN KEY CCF

REFERENCES PERSONA(CCF)

ON DELETE CASCADE

ad esempio : ESAMI(MATR, CORSO, DATA, VOTO, LOSE)

CONSTRAINT UNICO\_ES UNIQUE(MATR, CORSO)

↳ UNIQUE(MATR) lo studente  
può fare  
un solo  
esame

UNIQUE(CORSO) solo uno  
studente  
può fare  
l'esame

OK -> CONSTRAINT PK PRIMARY KEY (Matr, Corso)

↳ CONSTRAINT PK1 PRIMARY KEY (Matr) // NON POSSO AVERE  
↔ PK2 ↔ (Corso) // DUE CHIAVI PRIMarie

CREATE TABLE Esami ( ...

CORSO --| PRIMARY KEY ↗ scritto così ci sono due  
MATR --| PRIMARY KEY ↘ chiavi primarie

## Linguaggio di interrogazione

SELECT

SQL

Algebra relazionale  
(di tipo astratto)

modello relazionale

- Select mette insieme una sequenza di operazioni elementari
- Nell' **algebra relazionale** ogni operazione corrisponde ad una operazione elementare

Select  $\rightarrow$  espressione  $\Rightarrow$  ottinuto  $\rightarrow$  eseguito  
algebra  
relazionale

## Modello relazionale dei dati

SCHEMA RELAZ STUDENTI (matricola, nome, cognome, dataN, citta)

non corrisponde alle regole (ovvero la rappresentazione di una relazione)

una relazione  $R$  su  $S$  è un sottoinsieme del prod. cartesiano  
 $\text{dom}(\text{matricola}) \times \text{dom}(\text{nome}) \times \dots$

la relazione  $R = \{r_1, r_2, r_3, r_4\}$  è un insieme

- Non ci sono duplicati
- $r_1, \dots, r_4$  non sono ordinati !!

## OPERAZIONI

- **Proiezione**  $\Rightarrow$  operazione unaria (lavoro su una relazione)

$\Pi_{\text{attr}_1, \text{attr}_2, \dots, \text{attr}_n} (r)$

• è scelta e significativa

- si restituivano i dati ad un sottoinsieme degli attributi:

ASSOCIO JN  
NOME ALLA  
RELATIONE

$r_{cm}$

$\Pi_{\text{name}, \text{cognome}} (r) \rightarrow$

lavoriamo per  
colonna

name	cognome
Ciro	Esposito
Vincenzo	Esposito
Ada	Rossi
Mario	Rossi

? (name, cognome)

$\Pi_{\text{cognome}} (r_{cm}) \rightarrow$

Cognome
Esposito
Rossi

con SQL

$\Pi_{\text{name}, \text{cognome}} (r) \text{ DISTINCT}$

$\left[ \begin{array}{l} \text{SELECT } \text{Name}, \text{cognome} \\ \text{FROM Studenti} \end{array} \right]$

name	cognome
Ciro	Esposito
Vincenzo	Esposito
Ada	Rossi
Mario	Rossi

$\Pi_{\text{cognome}} (r) \rightarrow$

Cognome
Esposito
Rossi

$\left[ \begin{array}{l} \text{SELECT DISTINCT cognome} \\ \text{FROM Studenti} \end{array} \right]$

$\text{SELECT Cognome}$   
 $\text{FROM Studenti}$

Cognome
Esposito
Esposito
Rossi
Rossi

per evitare  
duplicati

$\text{SELECT [ALL]DISTINCT lista attributi}$   
 $\text{FROM nome tabella}$

- Selezione lavoriamo per righe
- Uso:
- Non altera lo schema della relazione
- Restituisce un SOTTOINSIEME della relazione di partenza

rigua  $\sigma_{\text{condizione}} (\tau)$   
booleana

$\sigma_{\text{condizione}} (\tau)$   
< 'Espresso'

CONDIZIONE nome-attributo = 'valore'

nome-attributo relazione-confronto valore costante

Relazioni di confronto

=

<>

<
<=
>
>=

$\Rightarrow$  domini NUMERICI  
ordinati STRINGHE  
DATE

'a a' < 'aaa' TRUE

in ASCII vengono prima

'zzz' < 'aaa' TRUE

- nome-attributo IS NULL
- nome-attributo relazione-confronto 'costante'
- nome-attributo relazione-confronto nome-attributo

Studente (

, CittàN, CittàR )

$\sigma_{CittàN = CittàR} (\tau)$

dom(attributo)  $\cup \{ \text{NULL} \}$

Per gli attributi parziali assumo che il dominio dell'attributo include il valore speciale **NULL**

POSSONO ESSERE COMPOSTE CON AND, OR, NOT

partiale, i vivi hanno  
 ↓  
 valore NULL  
 (e non var?)

$\text{PER SO NT CCF, nome, cognome, dataN, dataM}$

$\Gamma \text{dataM IS NULL } (P) \rightarrow \text{vivi}$

$\Gamma \text{dataM IS NOT NULL } (P) \rightarrow \text{morti}$

Come gestire i valori indeterminati?

il valore è **UNDEFINED**

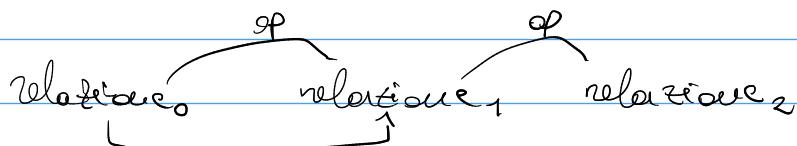
$\Gamma \text{dataM } (P) < '2020-10-26' \quad \text{NULL} < '2020-10-26' ?$

27/10/2020

Algebra relazionale

- dominio  $\rightarrow$  relazioni

$\rightarrow$  operazioni sulle relazioni



Con le operazioni possiamo scrivere **ESPRESSIONI** sulle relazioni

$\rightarrow$  quando valutiamo l'espressione

## OPERAZIONI

- Proiezione

$\Pi_{\text{name}, \text{cognome}}(r)$

- Selezione

$\{ \text{città} = 'Napoli' \text{ AND name} = 'Ciro' \}$

$\Pi_{\text{name}, \text{cognome}} ( \{ \text{città} = 'Napoli' \} )$

la relazione che ottengo è:

name	cognome
Ciro	Esposito
Riccardo	Esposito

$$\sum_{\text{cittàN} = \text{"Napoli}} (\Pi_{\text{nuore cognome}}(S)) = \emptyset$$

L'ordine delle operazioni è significativo

matr. cda	nome	cognome	dataN	cittàN	Lavoro	
N86001	ciro	Esperto	2000-10-01	Napoli	Programmatore	True
N86010	onida	Rossi	2000-11-02	Milano	FALSE	NULL INDEF.
N86020	mario	Rossi	2000-01-02	Milano	FALSE	CAMERIERE
N86030	Vincenzo	Esperto	2000-12-02	Napoli	TRUE	NULL INDEF.

- attributo IS NOT NULL
- attributo op. relaz. confronto veloce
- attributo op. relaz. " " attributo
- loro composizioni con connettivi logici AND, OR, NOT

$$\Pi_{\text{matr. cda}} (\text{Lavoro } \overset{(S)}{\text{IS NOT NULL}})$$

→ matr. cda studenti lavoratori

A	B	NOTA	A $\wedge$ B	A $\vee$ B
T	T	F	T	T
T	F	F	F	T
F	T	J	F	T
F	A	V	F	F
U	T	U	U	T
T	U	F	U	T
U	V	U	U	U
U	F	J	F	U
F	U	T	F	U

## Selezione SQL

$\Pi \rightarrow \text{SELECT MATRICOLA}$

$\text{FROM STUDENTI}$

$\sigma_c \rightarrow \text{WHERE Lavoro = 'prog' AND}$   
 $\text{CittàN = 'Napoli'}$

nome attributo

nometabella . nomeattr

Algebra  
relaz.

$\Pi_{\text{matricola}}$

RINOMINA

Cambia il nome della  
Colonna della tabella

$\text{SELECT S.Matricola AS RISULTATO}$   
 $\text{FROM Studenti AS S} \leftarrow \text{ALIAS}$  identifica in maniera  
 $\text{WHERE S.Lavoro = 'prog' AND}$   
 $S.CittàN = 'Napoli'$

Risultato

$\text{SELECT *}$  proietta tutti gli attributi  
 $\text{FROM ---}$   
 $\text{WHERE ---}$

S. matr
---------

$\sigma_c(S) \Leftrightarrow \text{SELECT *}$   
 $\text{FROM Studenti AS}$   
 $\text{WHERE C}$

## OPERAZIONI INSIEMISTICHE

- Relazione è un insieme
  - $\cap, \cup, \setminus$  intersezione, unione, differenza

$$\begin{array}{|c|c|c|} \hline r_1 & & \\ \hline A & B & C \\ \hline a & a & a \\ \hline a & b & c \\ \hline b & b & c \\ \hline c & c & c \\ \hline \end{array}
 \quad
 \begin{array}{|c|c|c|} \hline r_2 & & \\ \hline E & F & G \\ \hline a & a & a \\ \hline a & b & d \\ \hline b & b & c \\ \hline \textcolor{red}{0} & e & e \\ \hline \textcolor{red}{c} & c & c \\ \hline \end{array}
 = \quad
 \begin{array}{|c|c|c|} \hline ? & ? & ? \\ \hline \cancel{a} & a & a \\ \hline b & b & c \\ \hline \textcolor{pink}{c} & c & c \\ \hline \end{array}$$

L'operazione è definita se e solo se lo schema delle relazioni è compatibile.

- 1) stesso numero di attributi  $\pi_1$   $S(A_1 \cup A_k)$   
 $i$ -attributo di  $\pi_1$  ha lo stesso  $\pi_2$   $Z(B_1 \cup B_k)$   
 dominio dell' $i$ -attributo di  $\pi_2$

$$\dim(A_i) = \dim(B_i)$$

NON CI SONO RIPETIZIONI

$$\begin{array}{|c|c|c|} \hline
 & r_1 & \\ \hline
 A & B & C \\ \hline
 a & a & a \\ \hline
 a & b & c \\ \hline
 b & b & c \\ \hline
 c & c & c \\ \hline
 \end{array}
 \cup
 \begin{array}{|c|c|c|} \hline
 & r_2 & \\ \hline
 E & F & G \\ \hline
 a & a & a \\ \hline
 a & b & d \\ \hline
 b & b & c \\ \hline
 e & e & e \\ \hline
 c & c & c \\ \hline
 \end{array}
 =
 \begin{array}{|c|c|c|} \hline
 ? & ? & ? \\ \hline
 a & a & a \\ \hline
 a & b & c \\ \hline
 b & b & c \\ \hline
 c & c & c \\ \hline
 a & b & d \\ \hline
 e & c & e \\ \hline
 \end{array}$$

$$\begin{array}{c|c|c}
 A & B & C \\
 \hline
 a & a & a \\
 a & b & c \\
 b & b & c \\
 c & c & c
 \end{array}
 \xrightarrow{\quad r_1 \quad}
 \begin{array}{c|c|c}
 E & F & G \\
 \hline
 a & a & a \\
 a & b & d \\
 b & b & c \\
 c & c & c
 \end{array}
 = \quad
 \begin{array}{c|c|c}
 ? & ? & ? \\
 \hline
 a & b & c
 \end{array}$$

PARTITE(Data, Stadio, Sq1, Sq2, Goal1, Goal2)

- Squadre che hanno vinto allo stadio S. Paolo

Vince1  $\leftarrow \overline{\Pi}_{Sq1} \Gamma(P)$  Goal1 > Goal2 AND  
Stadio = 'S. Paolo'

Vince2  $\leftarrow \overline{\Pi}_{Sq2} \Gamma$  Goal2 > Goal1 AND  
Stadio = 'S. Paolo'

RISULTATO  $\leftarrow$  VINCE1  $\cup$  VINCE2

- Squadre che non hanno MAI vinto al S. Paolo

Tutte le squadre  $\setminus$  (Vince1  $\cup$  Vince2)

$(\overline{\Pi}_{Sq1}(P) \cup \overline{\Pi}_{Sq2}(P)) \setminus (Vince1 \cup Vince2)$

=> Operazione prodotto

=> Copie di squadre che hanno vinto nello stesso insieme  
di stadi

=> N-Goal segnati in una data

=> operazioni di conteggio

Le squadre che hanno vinto in casa in data  
(2020-10-25)

$\leftarrow \overline{\Pi}_{Sq1} \left( \Gamma_{data=2020-10-25}(P) \text{ AND } Goal1 > Goal2 \right)$

Le squadre che non hanno MAI vinto giocando in casa

$$\text{Tutte squadre} \leftarrow \overline{U}_{S_{G_1}}(P) \cup \overline{U}_{S_{G_2}}(P)$$

$$\text{Tutte Squadre} \setminus \overline{U}_{S_{G_1}}\left(\Gamma_{\text{Goal}_1 \rightarrow \text{Goal}_2}(P)\right)$$

$\Gamma_{\text{Goal}_1 \rightarrow \text{Goal}_2}(P)$

se lavora con la selezione lavoro su una singola riga  
alla volta

MA LAVORANO SU PIÙ  
ISTANZE

SQ1	SQ2	G <sub>1</sub>	G <sub>2</sub>	
A		0	1	← Perde
A		1	0	← Vince

Prodotto

29/10/20

$U_1 \times U_2 \rightarrow$  prodotto cartesiano di due relazioni

	A	B
$U_1$	a	b
	a	a

	C	D	E
$U_2$	c	c	c
	d	a	a

$U_1 = u_1$  attributi  
 $N_1$  elementi  
 $U_2 = u_2$  attributi  
 $N_2$  elementi

$U_1 \times U_2$

A	B	C	D	E
a	b	c	c	c
a	b	d	a	a
a	b	a	b	a
a	a	c	c	c
a	a	d	a	a
a	a	a	b	a

$U_1 \times U_2 = u_1 + u_2$   
attributi

$N_1 \times N_2$  elementi

MATR. NOME COGNOME

100	Ciro	Esposito
200	Mario	Rossi
300	Vito	Verdi

MATR. CORSO VOTO

100	BD I	25
200	BD I	24
100	BD II	30

MATR. NOME COGNOME MAT. CORSO VOTO

1 $\rightarrow$	100	Ciro Esposito	100	BD I	25
2 $\rightarrow$	100	" "	200	BD I	24
3 $\rightarrow$	100	" "	100	BD II	30
	200	Mario Rossi	100	BD I	25
	200	" "	200	BD I	24
	200	" "	100	BD II	30
	300	Vito Verdi	100	BD I	25
	300	" "	200	BD F	24
	300	" "	100	BD II	30

Condizione di giunzione

Di fatto c'è una uguaglianza tra la chiave degli studenti (mat<sub>1</sub>) e la chiave esterna di esami (mat<sub>2</sub>)

$$\boxed{\Gamma_{\text{mat}_1, \text{mat}_2}} = (n_1 \times n_2) \quad \begin{array}{l} \text{seleziona solo} \\ \text{determinate righe} \\ \text{della tabella} \end{array}$$

- Interrogazione : Gli studenti che non hanno fatto esami

$$\text{Studenti} f \leftarrow \overline{\prod}_{\text{mat}_2} (\Gamma_{\text{mat}_1, \text{mat}_2} = (n_1 \times n_2)) \rightarrow \begin{bmatrix} \text{mat}_2 \\ \hline 100 \\ 200 \end{bmatrix}$$

$$\text{Studenti TOT} \leftarrow \prod_{\text{mat}_2} (S)$$

$$\text{Studenti O} \leftarrow \text{Studenti TOT} \setminus \text{Studenti} f$$

o più semplicemente  $\prod_{\text{mat}_2} (\text{Studenti}) \setminus \prod_{\text{mat}_2} (\text{Esami})$

condizione di giunzione

Dubbio di  $\Delta_c \leftarrow \text{Join}_c \leftarrow$   
giunzioni

$$\Gamma_{\text{mat}_1=\text{mat}_2} (S \times E) \equiv S \bowtie_{\text{mat}_1=\text{mat}_2} E$$

$\hookrightarrow$  same attr. op confronto same attributo

Ricapporto (parte 1)

$r_1 \times r_2 \leftarrow$  prodotto cartesiano (prodotto libero delle righe)

$r_1 \bowtie_C r_2 \leftarrow$  prodotto con giunzione condizionata

una riga di  $r_1$  si congiunge con una riga di  $r_2$   
se le due righe soddisfano  $C$ .

(combinazione delle sole righe che soddisfano la condizione  $C$ )

$$\cap_C(r_1 \times r_2) = r_1 \bowtie_C r_2$$

### Giunzione Naturale

- Usa una condizione di uguaglianza implicito  
gli attributi di  $r_1$  in comune con  $r_2$  devono avere lo  
stesso valore  $r_1 \bowtie r_2$

Se non hanno attributi in comune (il nome e tipo dell'  
attributo) è un prodotto cartesiano

$$r_1 \bowtie r_2 \Rightarrow r_1 \times r_2$$

MATR NOME COGNOME		
100	Carlo	Esposto
200	Mario	Rossi
300	V.Gio	Veroli

MATR CORSO VOTO		
100	BD I	25
200	BD I	24
100	BD II	30

ma ci sono nomi  
di attributi in comune

MATR NOME COGNOME		
100	Carlo	Esposto
200	Mario	Rossi
300	V.Gio	Veroli

MATR CORSO VOTO		
100	BD I	25
200	BD I	24
100	BD II	30

MATR NOME COGNOME CORSO VOTO

100 Carlo Esposto BD I 25

100 " BD II 30

200 Mario Rossi BD I 24

Ha una  
colonna di  
meno!

$r_1 \bowtie r_2 \rightarrow$  prodotto con giunzione NATURALE, combinazione  
delle righe che soddisfano la condizione di  
uguaglianza sugli attributi di ugual nome

$S$  Studente(Matricola, Nome, Cognome)  
 $E$  ESAME(MAT, --)

$S \bowtie E$   
 $\text{matricola} =$   
 $\text{mat}$

Operazione Rimozione  
 $\rightarrow$  altera i nomi dello schema

che  $p(S) \bowtie E \rightarrow$  giunzione naturale  
 $\downarrow$   
 studente(MAT, Nome, Cognome)

se non avessi fatto l'operazione di rimozione, non  
 sarebbe stato possibile fare la giunzione naturale

$p(S) \bowtie E$   
 $\text{MATRICOLA} \leftarrow \text{MAT}$

un altro esempio

Residenza(CF, dataI, dataE, CittàR, Via)

Persona(CF, Nome, Cognome, dataN, dataM)

RazzaP(CF, CF Padre, CF Madre)

- CF, Nome e cognome delle persone che hanno fatto  
 un cambio di residenza cambiando città

Cambio  $\leftarrow \Pi_{CF} \left( \sigma_{CittàR \leftarrow} (R \bowtie p(R)) \right)$   
 $\text{Residenza}(CF_1, dataI_1, dataE_1, CittàR_1, Via_1)$

$\text{CittàR} \leftarrow$   
 CittàR<sub>1</sub>  
 AND  
 dataE<sub>1</sub> IS  
 NOT NULL

$\Pi_{CF, nome, cognome} (\text{Cambio} \bowtie P)$

$$R_{1S} \leftarrow \prod_{Cf} \left( \sum_{\text{città R1}} \left( R \bowtie p(R) \right) \right)_{\text{RESIDENTI}(Cf), \text{data}_1, \text{data}_2, \text{città}_1, \text{via}_1}$$

$$= \sum_{\text{città R1}} \prod_{Cf_1}$$

$$R_{1S} = \prod_{Cf} p(R)$$

R

100	30/10/2020	Napoli
100	1/11/2010	Roma

100	30/10/2020	- - -
-----	------------	-------

è una proiezione  
(venga determinata una riga per l'oggetto)

MATR	NOME	COGNOME	CORSO	VOTO
100	CIAO	Esperto	BD I	25
100	"	"	BD II	30
200	MARIO	Rossi	BD I	21
300	UGO	VERDI	NULL	NULL

02/11/2020

← giunzione esterna

IX

nell'operazione di giunzione la matr 300 si perde traccia

es. Matricole studenti senza esami  $\prod_{\text{matr}}(S \bowtie E)$

$$\prod_{\text{matr}} \left( \begin{array}{l} S \bowtie E \\ \text{corso matr = matr} \\ \text{NULL} \end{array} \right)$$

equivalente

$$S \bowtie E = S \setminus E$$

uguale perché non ci sono

OUTER JOIN:

Full JOIN:

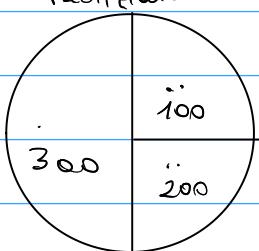
RIGHT JOIN

LEFT JOIN

$$S \bowtie E = S \bowtie E$$

righe da recuperare

Partizione



Raggruppamento

1) criterio di raggruppamento

⇒ Creare partizioni

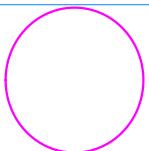
2) Operazioni conteggio sulle singole parti

Lista (possibilmente vuota) di ATTRIBUTI

(MATR) → tutte le righe che hanno lo stesso valore  
in matricola

CASO PARTICOLARE LISTA VUOTA

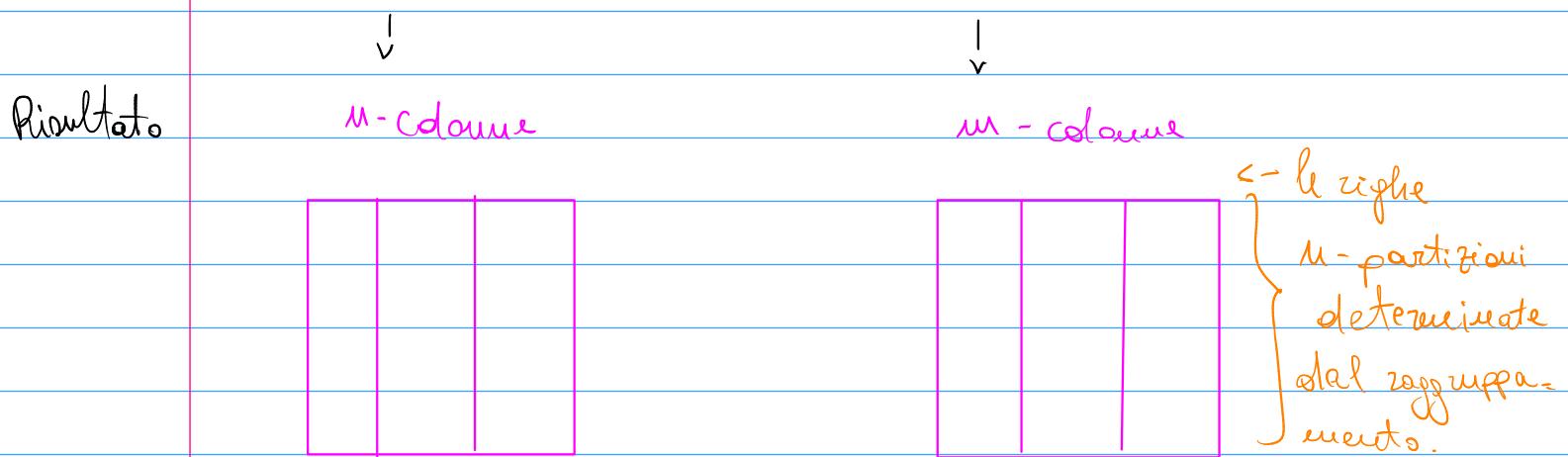
il GRUPPO coincide con la relazione



## Operazioni di conteggio (si fanno sui gruppi)

COUNT (attributo) conta gli elementi del gruppo con valore NOTNULL nell'attributo  
 SUM (attributo) somma i valori dell'attributo su tutti gli elementi del gruppo  
 MIN (attributo) ] Recupera i valori MAX MIN sull'attributo tra tutti gli  
 MAX (attributo) ] elementi del gruppo  
 AVG (attributo) Fa la media

Sintassi Operazione (Fornita in  
 lista raggruppamento) E' u  
 liste op. conteggio (relazione)



ad esempio : matr  $\vdash$  COUNT(corso) (S IXI  $\vdash$  matr = matr)

matr	COUNT(corso)
100	2
200	2
300	0

i NULL non viene conteggiato

Per ogni studente i m-esami, la media, il min. badi

matr  $\vdash$  COUNT(corso), AVG(voto), COUNT(Loste)

= il voto più alto che è stato preso

$\leftarrow \max(voto) \in$   
lista voto

Non è un valore, è  
un insieme che  
[30] contiene un valore

il max è calcolato su tutta la relazione

=> PER OGNI CORSO IL VALORE MAX

corso  $\leftarrow \max(\text{voto}) \in$   
Esami (mat, corso, voto, cfu, Data, lode)

Per ogni studente : il n. esami, n. crediti, per anno ?

matyear(Data)  $\leftarrow \text{COUNT}(\text{corso}), \text{SUM}(\text{cfu})$   
nominativi:

$\rho(\text{matr}, \text{year}(\text{Data})) \leftarrow (\text{COUNT}(\text{corso}), \text{SUM}(\text{cfu}))$   
 $\text{matr, anno, n-esami, n-cfu}$

=> Trovare nome, cognome e matricola degli studenti che nell'anno 2019 (esami sostenuti) hanno la media più alta

zappgrupp.  
in tutti gli  
altri

Matr. nome cogn. media


TAB  $\leftarrow \rho(\text{matr}, \text{nome}, \text{cognome}) \leftarrow \text{AVG}(\text{voto}) \left( \text{S IN } \text{YEAR}(\text{DATA}) = 2019 \right)$

100 righe

Soluzione 1 matr, nome, cognome  $\vdash_{\text{MAX}} \text{MAX}(\text{medio}) (\text{Tab})$

zighi?

Soluzione 2  $\vdash_{\text{val MAX}} p (\text{MAX}(\text{medio}) (\text{Tab}))$

$T_{\text{MAX}}$

$\prod_{\text{matr}} \left( \begin{array}{l} \text{Tab} \propto T_{\text{MAX}} \\ \text{Nome} \\ \text{cognome} \end{array} \right)$

medio =  
 $\text{valmax}$

$\boxed{\text{VALMAX}}$

un altro esempio  $\Rightarrow$  PERSONA (CF, NOME, COGNOME, DATA\_N, DATA\_M, CITTÀ)

PRESIDENTE (CF, VIA, CITTÀ, DATA\_I, DATA\_F)

GENITORI (CF, CF\_PADRE, CF\_MADRE)

Trovare le persone che hanno avuto residenza sempre nella città di  
nascita

Sol ① Trovo chi non va bene: Quelli che ha residenza diversa dalla città  
di nascita

Risultato: Tutti \ Quelli che non vanno bene

NO  $\leftarrow \prod_{\text{CF}} \left( \Gamma_{\text{CITTÀ}_N \leftrightarrow \text{CITTÀ}} (P \propto \text{Residenza}) \right)$

sì  $\prod_{\text{CF}} (P) \setminus \text{NO}$

2) Conteggio  $\leftarrow$  Conto (e rezidenze  $\leftrightarrow$  Città)  
Il conteggio = 0 OK

$$\text{TAB} \leftarrow \rho \left( \text{CF} \in \text{COUNT}(\text{città}) \left( \pi_{\text{Città} \leftrightarrow \text{città}} (\text{P} \bowtie \text{R}) \right) \right)_{\text{CF}, \text{Ncittà}}$$

$\Rightarrow$  Trovare le persone che hanno avuto residenza sempre nella stessa città

Con conteggio: Conto il numero di città diverse in cui ho avuto la residenza

$$\text{TAB} \leftarrow \rho \left( \text{CF} \in \text{COUNT}(\text{città}) \left( \pi_{\text{CF}, \text{città}} (\text{R}) \right) \right)_{\text{CF}, \text{Ncittà}}$$

1  
città diverse

$\pi$   
 R<sub>1</sub> → Napoli  
 R<sub>2</sub> → Napoli  
 R<sub>3</sub> → Roma  
  
 Roma → Napoli  
 → Roma

RISULTATO  $\pi_{\text{Ncittà} = 1} (\text{Tab})$   
 perché è il caso delle città (almeno in una città gli deve trovare) : ha vissuto in una sola città

Senza operazioni di conteggio

Le persone no : ha due residenze in città diverse

$$\text{NO} \leftarrow \overline{\pi}_{\text{CF}} \left( \pi_{\text{Città} \leftrightarrow \text{città}} (\text{R} \bowtie \rho (\text{A}))_{\text{CF}, \text{data}, \text{I}, \text{Città-L}, \text{via}, \text{I}, \text{data}, \text{F}} \right)$$

$$S_1 \leftarrow \pi_{\text{CF}} (\text{P}) \setminus \text{NO}$$

$\Rightarrow$  le coppie di persone che hanno avuto residenza esattamente nelle stesse città

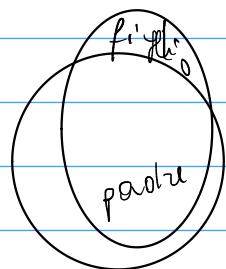
$\Rightarrow$  le persone che hanno avuto residenza nelle stesse città del Paese

P PERSONA (CF nome, cognome)  
 R RESIDENZA (CF dataI, dataT, cittaR -- )  
 Z PARENTELA (CF, CF Padre, CF Madre)

03/11/2020

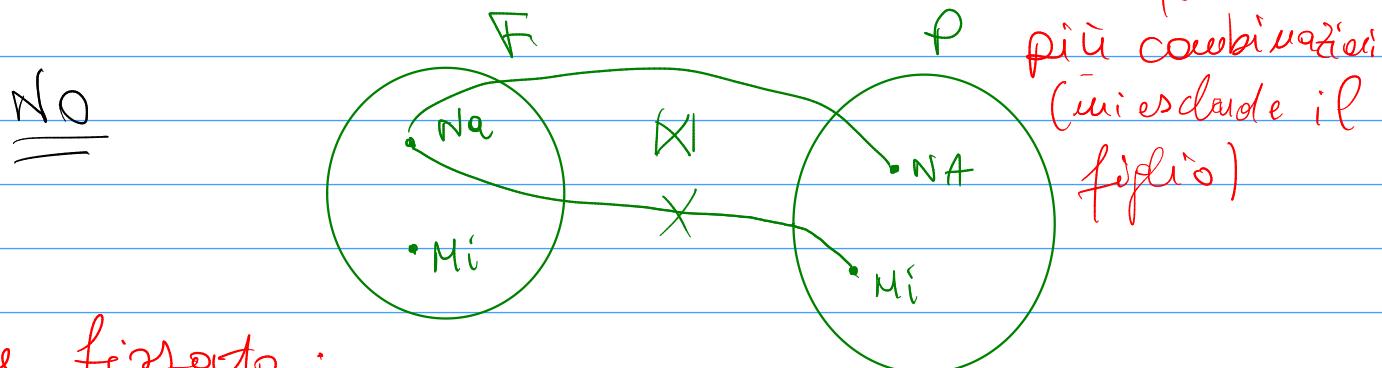
$R \bowtie Z \bowtie P(R)$   
 =  $\downarrow$   $\downarrow$   $\downarrow$   
 Residenza Padre Residenza  
 figlio Padre

$w_0 \leftarrow \prod_{CF} (\sigma_{CH})$   
 $citt_a <_s citt_b \wedge$



R1 Figlio  
 100 Napoli  
 100 Milano  
 R2 Padre  
 200 Milano  
 200 Napoli

la città  
non è finita =>



valore fissato:

Trovare le persone che hanno avuto residenza solo nella città di nascita del padre.

$No \leftarrow \prod_{CF} \sigma (P(P) \bowtie Z \bowtie R)$   
 $\underbrace{G_{\text{Non } P}}_{C_{FP} - RP} \quad \underbrace{C_{FP} =}_{C_{F \text{ Padre}}} \quad C_{\text{Ita } R}$

$\prod_{CF} (P)$

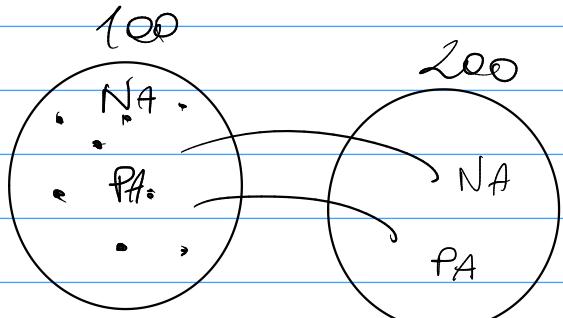
Fino la persona di  $CF = 100$

Trovare le persone che hanno avuto la residenza  
nelle stesse città di 100

$$\text{Città } 100 = \prod_{\text{città}} \Gamma_{CF=100}(R)$$

Napoli ✓  
Palermo

$$\prod_{CF} (P) \times \text{Città}_{100}$$

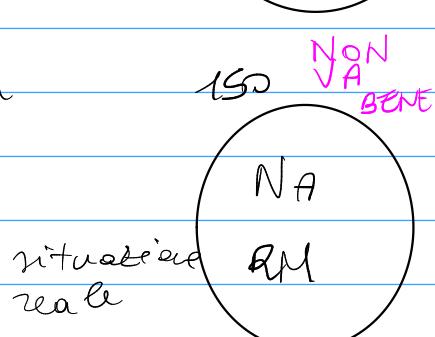


$$\prod_{CF, \text{città}} (R)$$

200	NA
200	PA
150	NA
150	PA
100	NA
100	PA

) situazione ideale

200	NA
200	PA
150	NA
150	RM
100	NA
100	PA



Hanno le stesse città va bene

$$NO_1 \leftarrow \prod_{CF} (\prod_{CF} (P) \times \text{Città}_{100} \setminus \prod_{CF, \text{città}} (R))$$

$$\text{città}_{100} \leftarrow \text{Tartà} (\Gamma_{CF=100} (R))$$

$$\text{ideale} \leftarrow \prod_{CF} (P) \times \text{Città}_{100}$$

$$\text{reale} \leftarrow \prod_{CF, \text{città}} (R)$$

$$NO_1 \leftarrow \prod_{CF} (\text{IDEALE} \setminus \text{REALE})$$

(controllo se mancano città ai reali)

$$NO_2 \leftarrow \prod_{CF} (\text{REALE} \setminus \text{IDEALE})$$

(controllo se hanno più città del necessario)

$$\text{Soluzione } \prod_{CF} (P) \setminus (NO_1 \cup NO_2)$$

# SQL

①

SELECT DISTINCT S.nome, S.cognome, S.CF  
FROM Studenti AS S JOIN Esami AS E  
WHERE S.matr = E.matr

SXE  
Prodotti

Π<sub>matr = matr</sub> (SXE)  
matr  
nome  
cognome  
CF

differenza è che ora utilizzo la joinzione  
con condizione, prima ho usato la clausola  
WHERE

②

SELECT DISTINCT S.nome, S.cognome, S.CF  
FROM Studenti AS S JOIN Esami AS E ON S.matr = E.matr

Π<sub>matr</sub> (S  $\bowtie$  E)  
matr = matr  
nome, cognome, CF

Natural JOIN



SELECT P.CF AS FIGLIO, PA.CF\_padre, AS Padre  
FROM Persone AS P Natural JOIN  
Parentela AS PA  
WHERE P.DatM IS NOT NULL

• Persone che hanno avuto residenza in due città diverse

SELECT

FROM Residenza AS R1 JOIN  
Residenza AS R2 ON R1.CF = R2.CF  
WHERE R1.Citta <> R2.Citta

# Regole per la logica relazionale

- Il nome di un attributo

TABELLA. attributo <  
AS. Studenti. CF oppure  
Abbreviato CF ALIAS. nome attributo

- Consigliabile def. e uso degli ALIAS

- $\cap \times_S$  FROM  $\cap, S$

$\cap \bowtie_S$   $\cap$  NATURAL JOIN S

$\cap \bowtie_C S$   $\cap$  JOIN S ON C

$\Rightarrow$

SELECT S.CF, S.nome, S.cognome  
FROM Studenti AS S  
 $\cap \bowtie_C S$  LEFT OUTER JOIN Esami AS E  
 $\cap$  LEFT OUTER JOIN S ON S.matr = E.matr  
ON C WHERE E.corso IS NULL

$\cap \bowtie_R S$

$\cap$  RIGHT OUTER JOIN S

ON C

$\cap \bowtie_L S$

$\cap$  (FULL) OUTER JOIN S ON C

può essere scritto così, abbreviato

• ORACLE

• POSTGRESQL OPEN SOURCE

→ le righe di R odi S  
che non troviamo  
possibilità di  
accoppiarsi, scompongo  
stallo o giurzire  
OUTER JOIN : completa la funzione trattandolo  
le righe

R GIUNZIONE S Natural Join  
Join ON  
a coppia righe di R e  
di S perché la condizione  
di giunzione è soddisfatta

05\11\20

$\text{cods } E \text{ count}(\text{codFilm}) (\text{Proiez } S \times \text{Proiez } F)$

↑  
non otengo il n-film  
ottengo il numero di proiezioni

$\text{cods } E \text{ count}(\text{codF}) (\prod_{\text{cods}, \text{codF}} (\text{Proiez } S \times \text{Proiez } F))$

Proiez F

P <sub>1</sub>	F <sub>1</sub>	...
P <sub>2</sub>	F <sub>2</sub>	...
P <sub>3</sub>	F <sub>3</sub>	...

Cod S

Count (CodF) (Proiez F, Proiez S)

S <sub>1</sub>	3
S <sub>2</sub>	3

3 sono le proiezioni

$\text{cods } E \text{ Count}(\text{codF}) (\prod_{\text{codF codS}} ( ))$

$\prod_{\text{cods}, \text{codF}}$

S <sub>1</sub>	1
S <sub>2</sub>	1

3)  $\text{Count} \leftarrow \rho(N\_spot)$   
 $\text{codFilm, N\_spot}$

$\text{MM} \leftarrow \rho(E^{\max(N\_spot)} (\text{Count}))$   
 $\max(N\_spot)$

SELECT

proiezione

FROM

giuntioni/prodotto

WHERE

selezione

SELECT DISTINCT  $A_1 \dots A_K$

FROM  $(T_1 \bowtie_c T_2)$

WHERE C

$$\Pi_{A_1, \dots, A_K} (\Sigma_C (T_1 \bowtie_c T_2))$$

S-STUDENTI(Matricola, CF, Nome, Cognome), E-ESAMI(Matricola, Corso, Voto, Lode, Dato)

$$\Pi_{CF, N, C} (\Sigma_{\substack{Corso \\ \text{null}}} (S \bowtie E))$$

SELECT S ...

FROM Studenti AS S LEFT OUTER JOIN ESAMI AS E ON S.matr = E.matr

WHERE E.Corso IS NULL

$$\leftarrow \overline{\Pi}_{matr(S)} \setminus \overline{\Pi}_{matr(E)}$$

(SELECT Matr  
FROM Studenti) ] Query 1

EXCEPT

] Indipendentemente

(SELECT Matr  
FROM ESAMI) ] Query 2

recepita  
dal Goering min

SELECT P. SQ1

FROM PARTIE AS P

WHERE P. Studio = 'S. Paolo' AND  
P. G1 > PG2 )

UNION [ALL / DISTINCT]

Ci riporta tutto anche i duplicati, se non dup.

SELECT P. SQ1

FROM PARTIE AS P

WHERE P. Studio = 'S. Paolo' AND  
P. G2 > PG1 )

(SELECT

{ Union  
Except  
Intersect }

(SELECT

→ Where GLI studenti che non hanno esami

→

Trova gli studenti che hanno un minimo ruolo d'esami

la sotto relazione è  
correlata

SELECT S. matr

From Studenti AS S

WHERE NOT EXISTS

eseguita { (SELECT \*

più volte } FROM Esami AS E

WHERE E.matr = S.matr

= RIVOLGIMENTO

se VERA

NOT EXISTS è una condizione soddisfatta  
quando:

EXIST (SELECT)

tabella vuota  $\emptyset$   
FALSE

Almeno una riga  
TRUE

Non mi importa il contenuto, uso \* per proiettare tutto

Trova gli studenti che non stanno nell'insieme degli studenti con esami

SELECT

FROM STUDENTI AS S

WHERE S.matr NOT IN

eseguita { SELECT DISTINCT E.matr  
1 volta { FROM ESAMI E }

non appartiene  
all'insieme

è scorrelata perché non compare nella sottointerruzione, e quindi può essere eseguita indipendentemente (la sottointerruzione)

SELECT

→ TABELLA  
non ha valori  
→ non viene memorizzata

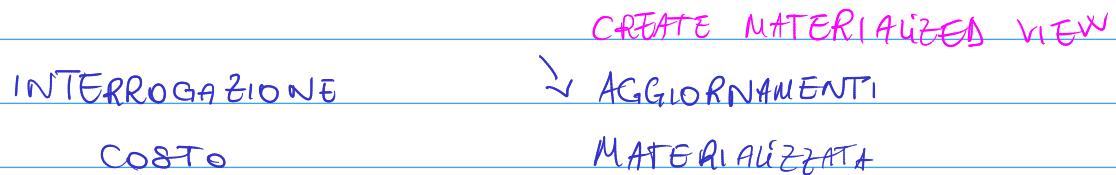
Dare un nome:

CREA UNA VIEW (senza memorizzare)

→ CREATE MATERIALIZED VIEW

il problema delle tabelle memorizzate: se i dati usati per la vista materializzata cambiano, cambia anche la vista materializzata (fa uscire per evitare calcoli!)

Se ho dati corretti, il costo è il più basso possibile della vista materializzata (che viene però aggiornata)



diventa un peso se lo dovrò aggiornare molte volte, le liste sono delle query che hanno un nome (è comunque una tabella)

SELECT → REWRITING → ALGEBRA REL.  
SELECT OTTIMIZZATA

Ritorniamo alle interrogazioni con le clausole:

WHERE ↗ [NOT] EXIST (SELECT <sub>unico attributo</sub>)

(attr<sub>1</sub>, ..., attr<sub>n</sub>) [NOT] IN (SELECT A<sub>1</sub> ... A<sub>m</sub>  
FROM )

il numero di attributi che stanno nella tupla dove corrispondere al numero di attributi della proiezione della clausola SELECT

(S.matr) NOT IN (SELECT E.matr  
FROM )  
posso avettere,  
è un solo attributo ( ) E?

l'insieme, in questo caso, dovrà essere costituito da un solo attributo; la struttura dell'elemento deve essere con medesima struttura della collezione che sto a recuperando  
→ devono corrispondere per tipo e numero di attributi

# SQL

09/11/2020

**SELECT  
FROM**

WHERE ~~(@)~~ [NOT] EXIST (SELECT

[NOT] è recuperata almeno una riga  
(CHECK rispetto a D)

$(A_1 \dots A_k) [\text{NOT}]$  IN (SELECT B<sub>1</sub> ... k)

CHECK APPARTENENZA ALL'INSIEME  
semplice, senza struttura diversa:  
 $A \text{ NOT } IN (SELECT B)$

Se vogliano fare un confronto scalare, per un'interrogazione invertita

A = (SELECT  
      ^>  
      ^< FROM  
      ^<= ^>  
      ^--)

Exp.

# SCALARS

## VOCABULARY

) } restituibile una volta  
Ripa

confronto un attributo  
con un valore calcolato

$A \Leftarrow \text{ALL} (\text{SELECT } B)$

$A \leq$  di tutti i valori selezionati  
 $A \leq A \text{ ANY } ( \text{SELECT } B )$

A ≤ di qualche elemento selezionato

- Tutto quello compare in una clausola WHILE ha un valore booleano (TRUE) FALSE)

## Algebra relazionale

< lista attributi > F < lista operazioni conteggio >

matr F count(corso), min(voto), max(voto) (Esami)

P (matr F count(corso), min(voto), max(voto) (Esami))  
matr, N\_Esami, voto\_min, voto\_max

SELECT E.matr, COUNT(F.corso) AS N\_Esami, MIN(E.voto) AS voto\_min, MAX(E.voto) AS voto\_max  
FROM ESAMI E  
GROUP BY E.matr

Regola!

Tutti gli attributi che comparevano nella clausola SELECT in forma non aggregata (ma sono argomento di COUNT, MIN, MAX, AVG, SUM ) DEVONO comparire nella clausola GROUP BY

SELECT E.matr, S.cognome, S.nome, COUNT(E.corso) AS N\_Esami, MIN(E.voto) AS voto\_min,  
MAX(E.voto) AS voto\_max

FROM STUDENTE AS S LEFT OUTER JOIN ESAMI AS E ON S.matr = E.matr  
GROUP BY S.matr

dunque è sintatticamente sbagliata perché S.cognome, S.nome comparevano in forma non aggregata ma non comparevano nella clausola GROUP BY . Per renderla corretta aggiungo S.cognome e S.nome nella clausola GROUP BY

SELECT E.matr, COUNT(E.corso) AS N\_Esami, MIN(E.voto) AS voto\_min  
MAX(E.voto) AS voto\_max  
FROM ESAMI E

- 1) Clauses **FROM** (lavoro su Esami)
- 2) WHERE non c'è, quindi lavoro su tutte le righe
- 3) Group BY non c'è raggruppamento, unico gruppo in cui confluiscono tutte le righe  
=> viene restituita un'unica riga

4) **SELECT** quale matricola proietto? stanno tutte nello stesso gruppo!!

**SELECT**

**FROM**

**WHERE** condizione su righe

**GROUP BY**

Oltre al group by, c'è HAVING (clausola simile a WHERE)

**HAVING** condizione su gruppi e conteggi

Ordine di esecuzione delle operazioni elementari:

- 1) Clauses **FROM** : calcolo prodotti e somme
- 2) Clauses **WHERE**: filtro le righe del passo 1)
- 3) Creo i gruppi sulle righe filtrate al passo 2)
- 4) filtro i gruppi - per valori del raggruppamento o per valori di conteggi
- 5) **SELECT** operazione di proiezione e conteggio sui gruppi  
(VIENE PROIETTATA UNA RIGA PER OGNI GRUPPO !!)

Dopo un group by non c'è disponibilità degli attributi che non sono oggetto di raggruppamento (fatto le colonne)

esempio:

Le medie degli esami sostenuti dagli studenti nel 2020

**SELECT** E.matr, COUNT(E.corso) AS N\_esami,  
AVG(E.voto) AS Medior

**FROM** Esami AS E **tutti gli esami**

**WHERE** YEAR(E.data) = '2020' **solo esami 2020**

**GROUP BY** E.matr

**gruppi di esami per studente**

**HAVING** COUNT(E.corso) > 2

**solo gruppi di almeno 3 esami**

Meno efficiente, stesso risultato (lo useremo se vogliamo selezionare la data)

SELECT E.matr, COUNT(E.corso) AS N\_esami

→  $\text{MAX}(\text{AVG}(E.voto))$  AS Media NO

From Esami AS E

GROUP BY E.matr, YEAR(E.data)

HAVING COUNT(E.corso) >= 2 AND YEAR(E.data) = '2020'

voglio selezionare lo studente con la media più alta

SELECT E.matr, COUNT(E.corso) AS N\_Esami

CORRELATE

FROM Esami AS E

GROUP BY E.matr

HAVING AVG(E.voto) >= ALL (

SELECT AVG(E.voto)  
From Esami AS E  
GROUP BY E.matr

Ricorda che due livelli di raggruppamento  
non possono stare "annidati"

## Viste

Sono interrogazioni a cui si associa un nome: il risultato della interrogazione non è memorizzato, vista è una tabella virtuale e può essere utilizzata IN LETTORE come fosse una tabella, non può essere utilizzata con operazioni di side effect

CREATE VIEW namevista (lista parametri)

AS SELECT --

⇒ CREATE VIEW conteggio(Matricola, N\_esami, Media)

AS

SELECT E.matr, COUNT(E.corso), AVG(E.voto)

From Esami AS E

GROUP BY E.matr

```

SELECT C.matricola
FROM conteggio AS C
WHERE C.Media = (SELECT MAX(Media)
                  ↑
                  FROM Conteggio AS H)

```

interrogazione  
 L' = interroga<sup>zione</sup>  
 se la re<sup>turna</sup>  
 (restituisce un  
 valore)

Lo posso fare?

SELECT C.matricola

From

(SELECT E.matr, COUNT(E.corso), AVG(E.voto)

From Esami AS E

GROUP BY E.matr) AS C

WHERE C.Media = 25

Già ho esteso la definizione di  
 conteggio (che è una interrogazione  
 select). Nella clausola From ci sono  
 delle sottointerrogazioni. SVANTAGGIO  
 PERDITA IN VISIBILITÀ

- Se mai c'è HAVING, non ha senso parlare di raggruppamento
- CREATE è un metodo esterno, compone solo con il nome della tabella
- View è un nome logico globale (con valori associati)
- Per conteggi con criteri diversi, si può ripetere la interrogazione in tanti sottocounteggi per poi unirli con una  
giunzione

cosa abbiamo visto nella  
clausola SELECT

SELECT nomi attributi, COUNT()

ndo messo  
10/11/2020

SELECT <espressione 1> AS <nome 1> ... <espressione n> AS <nome n>

con <espressione> si intendono le composizioni di espressioni, attributi, valori costanti o operazioni caratterizzate da tipi diversi gli altri

ad esempio:  
→  
1 →  
SELECT E.matr, YEAR(E.Data), AVG(voto), AVG(voto)/30 \* 110

1 FROM ESAME si può mettere l'AS

1 se voglio la media  
non

2 GROUP BY E.matr, YEAR(E.Data)

3 ORDER BY E.matr ASC, YEAR(E.Data) DESC

ASC indicata per  
ultimo, non  
obbligatoria

ASC Ascendente - default

DESC Descendente

Studenti (Matricola, Nome, Cognome)

( SELECT 'Triennale', S.nome || S.cognome  
FROM Studenti AS S  
WHERE S.matricola LIKE "N8%" )  
UNION

( SELECT 'Magistrale', S.nome || S.cognome  
FROM Studenti AS S  
WHERE S.matricola LIKE "N9%" )

Il like può essere usato solo con la clausola WHERE

Matricola LIKE "N8%"

inizia con 'N8'

espressione con metacaratteri

una stringa arbitraria di caratteri anche  
di lunghezza 0

SELECT (expression) ---

12\11\2020

### - funzioni predefinite

- legate ai tipi di dato
- possono essere estese con funzioni def. da utente  
*(funzioni collegate alle stringhe)*

ad esempio

LENGTH (String) → lunghezza stringa

restituisce un intero

$$\text{LENGTH}('pippo') = 5$$

INSTR (String1, String2, pos, n-00) → *0 se la sottostringa*

$$\text{INSTR}('N860001000', '00', 1) = 4$$

*String2 non occorre in string1*

$$\text{INSTR}('N860001000', '00', 5) = 5$$

→ *i dove è la prima occorrenza della stringa*

$$\text{-- ( -- / 6 ) = 8}$$

$$\text{-- ( -- / 1, 3 ) = 8}$$

*Nell'unico posto dalla posizione 1 calcolo terza occorrenza*

SUBSTR (String1, pos, lunghezza)

A = "ciao & esperto & xxx..."  
↓      ↓      ↓  
6      14      14  
↑  
INSTR(A, '&')

nuove SUBSTR(A, 1, INSTR(A, '&') - 1) = 'ciao'

è - 5

cognitive SUBSTR(A, INSTR(A, '&') + 1, INSTR(A, '&', 1, 2) - INSTR(A, '&', 1, 1))  
6                  14          8          6

cf SUBSTR(A, INSTR(A, '&', 1, 2) + 1)

SUBSTR(A, INSTR(A, '&', 1, 2), 16)

LENGTH()

INSTR()

SUBSTR()

## vincoli

- dominio
- tupla (esempla)
- Interrelazionali

} Definizione Tabella

Nella definizione di una tabella SOLO VINCOLI INTRARELAZIONALI

con i vincoli di tipo interrelazionali → ASSERTION

CREATE ASSERTION <nome assertione>

CHECK proprietà

→ espressione booleana

• EXIST <SELECT>

|| • NOT EXIST <SELECT>

CHECK NOT EXIST <qualcosa che non voglio>

esempio

FILM (CodF, titolo, --)

PROIEZIONE (CodP, CodSala, Data, Orario, Durata)

BIGLIETTI (CodP, CodPosto)

SALA (CodSala, N-posti)

non posso avere più biglietti per una proiezione dei posti della sala

CREATE ASSERTION capienza

CHECK NOT EXIST

(SELECT

FROM PROIEZIONE AS P JOIN

BIGLIETTO AS B ON P.CodP = B.CodP

GROUP BY P.CodP

HAVING COUNT(\*) >

(SELECT S.N-posti

FROM SALA S

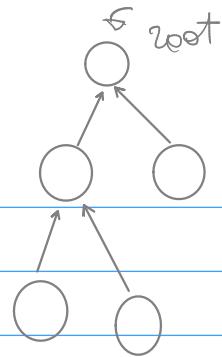
WHERE S.CodS = P.CodS )

Costruiamo un albero

TREE (CodT, radice)

NODE (CodN, Label, CodT)  
etichetta ]

ARC (CodA, label, NodoP, NodoF)



La radice di codT deve essere un nodo dell'albero

CREATE ASSERTION <radice> CHECK NOT EXIST

(SELECT \*

FROM TREE AS T JOIN NODE AS N ON T.radice = N.CodN  
WHERE N.CodT <> T.CodT )

I nodi di un arco stanno nello stesso albero

CREATE ASSERTION stemT

CHECK NOT EXIST

(SELECT

FROM (ARCS AS A JOIN

NODE AS N1 ON A.NodeF = N1.CodN)

JOIN NODE AS N2 ON A.NodeP = N2.CodN

WHERE N1.CodT <> N2.CodT

Con il check exist può esistere anche un solo elemento nella assertione

## MANIPOLAZIONE DATI

- **INSERT** inserisce righe in una tabella

- **DELETE** cancella righe da una tabella

- **UPDATE** cambia i valori di righe in una tabella

DELETE FROM <nuova tabella>

[WHERE <Condizione>] opzionale

WHERE TRUE

L'senza clausola WHERE cancello TUTTE le righe !!!

per cancellare l'intera tabella -> DROP TABLE <nuova-tabella>

INSERT INTO <nuova tabella> (lista attributi)

VALUES ( ), ( ), ( )

ESAMI (Matr, Voto, Data, Lode, CodE)

INSERT INTO Esami (Matr, CodE, Voto, Data, Lode)

VALUES (186000000, 80, 25, DATE "2020-11-11", NULL),

## Ficcapitolazione della manipolazione dati

16/11/2020

# INSERT      DELETE      UPDATE

STIRTE FROM < nome - tabella > [ WHERE (Condizione) ] ^ unica } cancellazione righe

Drop < nome - tabella> cancellazione tabella

**ALTER TABLE** <nome-tabella> } per la cancellazione  
**DROP COLUMN** <nome-colonna> } di una colonna  
(vengono cancellati  
anche i vincoli)

**INSERT** ← Inserire righe in una tabella  
ho due possibilità di inserimento: Dati e Dati calcolati

## DATI :

`INSERT INTO <nuova tabella>  
[ (lista attributi) ]`

ESAMI (Matr, Voto, Lode, data, CodE)

INSERT INTO ESAMI

(Matr, data, CodE, Voto, lode)

VALUES ('N860001000', SYSDATE, 'BDE', 28)

INSERT INTO ESAMI

VALUES ('N860001000', 28, NULL, SYSDATE, 'BDE')

Affezione! L'inserimento sarà accettato SOLO SE tutti i vincoli ENABLED (domino, intrerelazioni, unique) sono soddisfatti !!

Se il vincolo viene vietato. NO!

DATI CALCOLATI:

INSERT INTO <nuova tabella>

[lista-attrIBUTI]

(SELECT ...)

L'elista proiezione della SELECT è compatibile con la lista attributi se dichiarata, altrimenti fa dichiarazione della TABLE

CARIERA (Matr, N\_Esami, MEDIA\_30, MEDIA\_10)

DELETE FROM CARIERA C

WHERE Matr = 'N860001000'

INSERT INTO CARIERA

(SELECT (Matr, COUNT(CodE), AVG(Voto), AVG(Voto)\*10/30

FROM ESAMI E

WHERE E.Matr = 'N860001000'

GROUP BY E.Matr)

una condizione  
assente equivale  
a TRUE nella  
clausola WHERE

Cambiamento di record esistenti:  
UPDATE <tabella> [AS<alias>]

SET (Nome\_attributo) = (espressione)  
(      ) = ( )

[WHERE <condizione>]  
il cambiamento riguarda  
TUTTI i record che soddisfano  
la condizione dello WHERE

UPDATE Carrera

SET n\_Exami = (SELECT COUNT(\*)  
FROM Esami

WHERE Matr = 'NB6001000',

Media\_30 = (SELECT AVG(Voto)  
FROM Esami

WHERE Matr =

WHERE Matr = 'NB6001000'

Esempio MAGAZZINO(CodA, Descrizione, Scorta)

P. key → ORDINI (CodO, Data, PI, Eseguito)

COMP\_ORDINI (CodO, CodA, quantità)

P. key CARRELLO (CodC, PI, Data, Completato)

COLL\_CARRELLO (CodC, CodA, quantità)

Lo che il carrello di codice 'c1000' è stato completato  
e deve essere trasformato in ordine.

## Operazioni elementari

ho inserito dati  
costanti e dati  
calcolati !!

### ① Creo Ordine

INSERT INTO ORDINI

```
(SELECT CodC, SYSDATE, PI, 'NO'  
FROM carrello  
WHERE CodC = 'C1000')
```

### ② Creare Composizione Ordine

Insert into COMPODINE

```
(SELECT *  
FROM COMPACARRELLO  
WHERE CodC = 'C1000')
```

### ③ Cancellazione composizione

```
DELETE FROM  
COMPACARRELLO  
WHERE CodC = 'C1000'
```

Se cancello prima  
il carrello avrò  
una violazione di  
**CONSISTENZA**

### ④ Cancellazione carrello

```
DELETE FROM  
Carrello  
WHERE CodC = 'C1000'
```

• Se nella chiave esterna di COMPACARRELLO ho aggiunto la  
clausola : ON DELETE  
CASCADE

Allora la 3) e la 4) possono essere fatte con  
un'unica operazione : DELETE FROM Carrello  
WHERE CodC = 'C1000'

## ⑤ Aggiornamento seosta magazzino

UPDATE Magazzino M — sottraggo

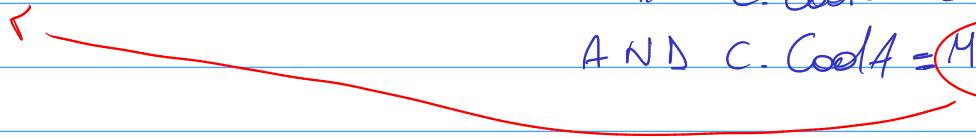
SET M.Scorta = M.Scorta - (SELECT C.Quantità

FROM Comprobante C

WHERE C.CodO = 'C1000'

AND C.CodA = M.CodA)

M.CodA



Quali  
righe  
modificare?

WHERE M.CodA IN

SELECT C.CodA

FROM Comp.Ordine, C

WHERE C.CodC = 'C1000')

da update farla alla clausa

WHERE!!

non posso  
fare FROM  
Comp.Ordine  
perché è cancellato

Inserire una registrazione

17/11/2020

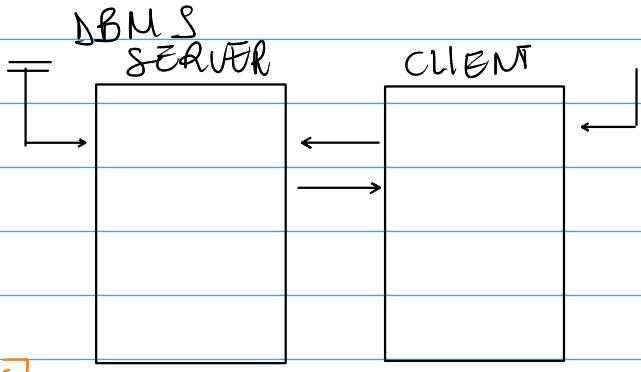
- Insert
- Delete
- Update

↳ Aggiorna i singoli attributi di una riga

PLSQL

PSQL

Eseguito  
interamente  
al DBMS



Linguaggi strutturati  
di programmazione  
attraverso API

JAVA

JDBC

Bell'applicativo  
è nel client

protocollo

JDBC

Java database  
connectivity

Un linguaggio di programmazione che permette di  
strutturare i comandi del DBMS

## Trigger

- Strumento per la gestione della consistenza di una base di dati;
- Quando capta un evento fa delle operazioni per gestire la consistenza dei dati;

Evento → Azione

⇒ vincoli di consistenza se rispetto i vincoli, ho successo  
(Azioni compiuta con successo)

FACCIO L'AZIONE

Se non rispetto i vincoli:  
(azione bloccata)

Se scateno una azione da un evento, possono esserci delle azioni a catena (come la gestione dell'integrità referenziale)

Evento → Azione

↓  
↙

(Altri eventi sulla base di dati)

Esempio : gestire integ. ref. ref.

→ ON DELETE  
→ CASCADING  
Evento → Azione  
(Azione a catena)

→ ON UPDATE →

Sintassi del trigger

CREATE TRIGGER < nome >

< quando > < evento >

gli eventi che andiamo a considerare :

ziga [ ] INSERT ON < tabella >

DELETE ON < tabella >

livelli di granularità

attributo [ ] UPDATE OF < attributo > diversi

ON < tabella >

quando voglio fare l'azione :

BEFORE (A moite) preventiva

AFTER (A valle) consultiva

INSTEAD OF

può essere usata su viste e non tabelle !!

le viste sono comunque entità virtuali (non memorizzate)

CREATE TRIGGER <nome>

<quando> <evento>

<granularità> → FOR EACH ROW (se non è unica)

Si verificano due casi:

L'azione deve essere compiuta per ogni riga?  
oppure per l'intero blocco dell'operazione?

INSERT S righe → azione per ogni riga? → 5azioni

↓  
azione unica per l'intera operazione  
→ 1 azione

CREATE TRIGGER <nome>

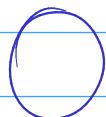
<quando> <evento>

<granularità> → FOR EACH ROW

<Condizione> per attivare il trigger

BEGIN bisogna soddisfare la condizione

AZIONE



Corpo procedurale trigger

END

(è una PROCEDURA)

CALL

(nuove (parametri))  
RETURN ---

, SEQUENCE

è una struttura che genera chiavi surrogate

ESAMI (Matr, Corso, ...)

ESAMI (CodE, Matricola, Corso, ...)

Chiave surrogata  
è un

Contatore (contine una variabile) → Dati

max valore usato

101

numero

Sintassi (Oracle):

si affiora

102

CREATE SEQUENCE *esame*

START WITH <numero>

INCREMENT BY <numero>

MIN VALUE <valminimo> <= numero di partenza

MAX VALUE <max numero>

CREATE SEQUENCE K-esame

↓  
10.500

START WITH 10500

INCREMENT BY 1

MIN VALUE 0

MAX VALUE 1.000.000,

può non essere meno  
il max value

① quando un esame, prima di inserirlo modifico la CodE  
sostituendolo con la chiave sintetica

CREATE TRIGGER B-IN-ES

il trigger scatta solo quando

BEFORE INSERT ON Esame

/ è NULL

FOR EACH ROW WHEN New.CodE IS NULL

BEGIN

NEW.CodE := K-esame.NEXTVAL

assegnamento

END

L'azione e si  
trovano tra  
BEGIN e END

l'inserimento con  
new, nell'update  
new e old, nel delete  
old

Prendo la ziga corrente

- Come si chiama

NEW

se non viene considerata  
è perché non è dichiarata  
FOR EACH ROW !!

Insert

NEW  
↓  
UPDATE  
OLD / DELETE  
↓ UPDATE

fondamentale  
(non potremo usazla)

un altro esempio

CREATE TRIGGER Normalizzazione

BEFORE INSERT ON STUDENTE

FOR EACH ROW

no niente

BEGIN

tutto in minuscule  
New.Name := UPPER (NEW.Name)

New.Cognome := UPPER (NEW.Cognome)

END

Quando il carrello viene completato, fa la trasformazione  
in ordine (Completo 'N' → 'S')

CREATE TRIGGER crea\_ordine

AFTER UPDATE OF Completo ON Carrello

FOR EACH ROW

WHEN OLD.Completo = 'N' AND  
NEW.Completo = 'S'

BEGIN

INSERT INTO ORDINE

VALUES (New.CodE, SYSDATE, NEW.PI, 'No') ;

INSERT INTO Comporzione anche  
 (SELECT \* old. Cod C  
 FROM Comprarelio C  
 WHERE C.Cod C = New.Cod C)  
 DELETE FROM Comprarelio C  
 WHERE C.Cod C = NEW.Cod C;  
 UPDATE C 'C 100'  
 ;  
 END NEW Cod C to  
inserendo ora

| UPDATE Magazzino M  
 | SET M.sconta = M.sconta - (SELECT C.quantità  
 | FROM Comporzione  
 | WHERE C.Cod O = NEW  
 | Cod. C  
 V WHERE M.Cod A IN AND C.Cod A = M.Cod A  
 | )

(SELECT  
 FROM Comporzione C  
 WHERE C.Cod O = NEW.Cod C)

Trigger Events  $\rightarrow$  Action

19/11/20

BEFORE - INSERT ON

AFTER - DELETE ON

INSTEAD OF - UPDATE OF ... ON

BEGIN

} Procedure

END

oracle

PL/SQL

JAVA

JDBC

TREE (CodT, root)  $\downarrow$  integer  
NODE (CodT, CodN, Label)  
ARC (CodT, Srg, Trg, Label)  
source target  
(padre) (figlio)

Quando cambia etichetta del nodo su un albero, dovo cambiare l'etichetta dei nodi del sottoalbero radicato nel nodo

CREATE TRIGGER Cambia-E

AT EACH UPDATE OF Label ON NODE

FOR EACH ROW

BEGIN

UPDATE NODE N

SET N.Label = N.Label + (NEW.Label - OLD.Label)

WHERE N.CodN IN N.CodN è nell'insieme dei figli  
di NEW.CodN

NEW.CodN è

il nodo cambiato

insieme figli di ✓  
NEW.CodN

SELECT

FROM ARC A

WHERE NEW.CodT = A.CodT AND

A.Srg = NEW.CodN

stesso albero

è il padre

<- Sfatto a mio vantaggio la natura reattiva del trigger

<= Devi prestare attenzione !

TAB(Cod, Num)

CREATE TRIGGER dec

AFTER UPDATE OF Num ON TAB

FOR EACH ROW WHEN NEW.Num > 0

BEGIN

UPDATE TAB C

SET Num = Num - 1

WHERE NEW.Cod = C.Cod

END

Traesco un processo

di decremento illuminato

UPDATE TAB

SET NUM = 3

WHERE Cod = A4

TAB

A4	5
BB	7

TRIGGER INSTEAD OF

- AFTER { } => Lavoriamo su tabella  
BEFORE }

- INSTEAD OF => Viste

Studenti (Matr, Nome, Cognome, CorsoL)

Eserci (MATR, Corso, Voto, LODE)

```
VISTA_INF <- SELECT  
FROM Studenti S JOIN  
ESAMI E  
ON S.matr = E.matr  
WHERE Corso = 'N86'
```

Le viste possono essere usate come query come fossero tabelle reali

```
SELECT *  
FROM Vista_Inf V  
WHERE V.Cognome LIKE "Esposito"
```

- Possiamo utilizzarle per
- INSERT      UPDATE      DELETE      ?      NO
- DELETE FROM Vista\_Inf  
WHERE V.cognome LIKE "Esposito"

Quando ho una vista :

- 1) la vista è un filtro su una tabella

```
CREATE VIEW S_Inf  
AS
```

```
(SELECT *  
FROM Studenti  
WHERE Corso = 'N86')
```

questa vista contiene:  
- Chiave primaria  
- tutti gli attributi totali

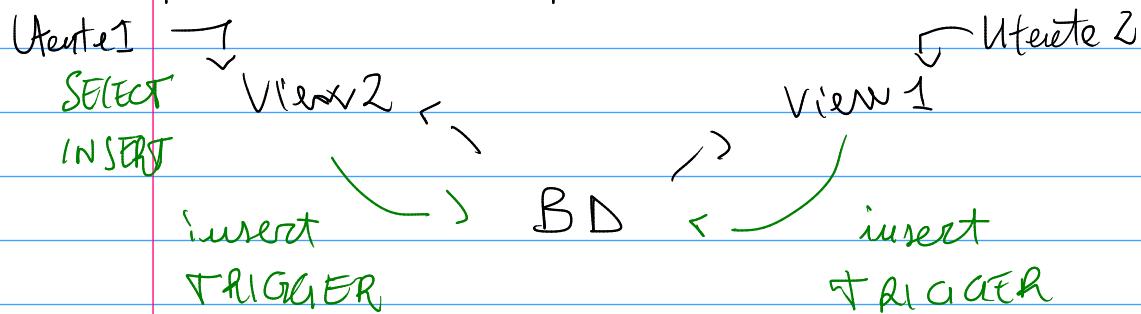
2) Posso usare la vista come forse la tabella su cui si basa

INSERT INTO S-Suf  
( )

=> AD OGNI OPERAZIONE SULLA VISTA, CORRISPONDE ANALOGA OP OPERAZIONE SULLA TABELLA MEMORIZZATA

Nel caso più generale, uso un TRIGGER (con clausola INSTEAD OF)

può essere usata per limitare la vista



Esempio CREATE VIEW Esoceli\_inf (Matr, Nome, Cognome, corso, voto, lode)

SELECT

FROM Studenti S JOIN

ESAMET E

ON S.Matr = E.Matr

WHERE CorsoL = 'N86'

la vista è basata su una

INSERT INTO Esoceli\_inf

già avere una struttura

VALUES ('N86001000', 'Ciro', 'Esposito', 'BDI', 25, null)

non è gestita automaticamente,

quindi crea un TRIGGER (gli dice come gestire degli inserimenti)

l'evento qui non  
occorre perché viene  
riempito

CREATE TRIGGER Ins\_View  
INSTEAD OF INSERT ON Esame\_Inf  
FOR EACH ROW  
BEGIN  
IF NOT EXISTS (SELECT \*  
FROM STUDENTI S  
WHERE S.matr = NEW.matr)  
THEN  
INSERT INTO Studenti  
VALUES (NEW.matr, NEW.nome,  
NEW.cognome, 'N86')  
END IF  
END IF  
j) sequenzializzazione di istruzioni  
INSERT INTO Esame  
VALUES (NEW.matr, NEW.corso, NEW.voto,  
NEW.punteggio)  
END

in genere:

BEGIN  
DECLARE  
dichiarare  
[sezione di dichiarazione di variabili]  
END DECLARE  
BEGIN

body

gestione  
e eccezioni  
EXCEPTION  
END

BEGIN

IF NOT EXISTS (SELECT \*

FROM STUDENTI S

WHERE S.matr = NEW.matr)

THEN

INSERT INTO Studenti

VALUES (NEW.matr, New.nome,  
NEW.cognome, 'N86')

END IF

(j) sequenzializzazione di istruzioni

INSERT INTO Esami

VALUES (NEW.Matricola, NEW.Corso, NEW.voto,  
NEW.Ragione)

END

Supponiamo di definire una revisione  
di definizione

BEGIN

DECLARE

CNT INTEGER := 0

il doppio begin è necessario

perché c'è la dichiarazione

END DECLARE

BEGIN

per vedere se ho uno studente : risultato di

SELECT COUNT(\*) INTO CNT

FROM Studenti S

WHERE S.matr = NEW.matr

una SELECT

memorizzata in una

variabile

IF CNT = 0

THEN

END IF

il costrutto generale

SELECT expr<sub>1</sub>, expr<sub>N</sub>

INTO

Ci deve essere compatibilità

di tipo e di numero

var<sub>1</sub>, , var<sub>N</sub>

ATTENZIONE: il costrutto è utilizzabile solo  
se la select restituisce al più una riga!

(altrimenti andiamo incontro a problemi di  
runtime)

# Basi di Dati

## Lezione del 19-11

**Trigger** -> meccanismo di evento-azione

```
INSERT ON  
DELETE ON  
UPDATE OF ... ON
```

Possiamo agire prima o dopo (**BEFORE, AFTER**) e **INSTEAD OF** (che non lavora sulle tabelle ma sulle viste).

**Azione** → BEGIN ... END definito attraverso una procedura

```
TREE (codT, root)  
NODE (CodT, CodN (chiave primaria), label (INTEGR))  
ARC (CodT, srg (chiave esterna), Trg (target), label (INTERGER))
```

Immaginiamo che l'albero presenti una sorgente (padre) e abbia come archi (i target) i suoi due figli. Quando cambio l'etichetta del nodo in un albero, devo cambiare allo stesso modo l'etichetta dei nodi del sottoalbero radicato nel nodo

```
CREATE TRIGGER cambia_E  
AFTER UPDATE OF label ON NODE  
FOR EACH ROW  
BEGIN  
UPDATE NODE N  
SET N.label = N.label +  
      (OLD.label - NEW.label)  
WHERE N.codn IN  
      (SELECT A.Trg  
       FROM ARC A  
       WHERE NEN.codT = A.codT AND A.Srg = NEW.CodN)
```

Sfrutto a mio vantaggio la natura reattiva del trigger. Bisogna prestare attenzione

```
TAB (Cod, Num)  
CREATE TRIGGER dec  
AFTER UPDATE OF Num ON TAB  
FOR EACH ROW  
BEGIN
```

```
UPDATE TAB C
SET Num = Num - 1
WHERE NEW.Cod = C.Cod
END
```

## TRIGGER INSTEAD OF

AFTER e BEFORE lavorano su tabelle; INSTEAD OF lavora invece su viste

```
STUDENTI (MATR, Nome, Cognome)
ESAMI (MATR, CORSO, VOTO, LODE)
VISTA_INF <- SELECT
    FROM STUDENTI S JOIN ESAMI E ON S.MATR= E.matr
    WHERE CORSOL = 'N86'
```

Le viste possono essere usate per interrogare (query) come fossero tabelle reali

```
SELECT
FROM VISTA_INF V
WHERE V.cognome LIKE 'Esposito'
```

In linea generale non possiamo usarle per INSERT, UPDATE e DELETE

```
DELETE FROM VISTA_INF
WHERE V.cognome LIKE 'Esposito'
```

Quando ho una vista:

- la vista è un filtro su una unica tabella

```
CREATE VIEW S_INF
AS
(SELECT *
FROM Studenti
WHERE CORSOL = 'N86' )
```

Questa vista contiene la chiave primaria della tabella e tutti gli attributi totali. In questo caso posso usare la vista come fosse la tabella su cui si basa (in oracle)

Nel caso più generale uso un TRIGGER con clausola INSTEAD OF

```
CREATE VIEW Esami_Inf (MATR, Nome, Cognome, corso, voto, lode)
SELECT Matr, Nome, Cognome,
FROM Studenti S JOIN
    ESAMI E
    ON S.matr = E.matr
```

```

WHERE CorsoL = 'N86'

INSERT INTO ESAME_INF
VALUES ('N86001000', 'Ciro', 'Esposito', 'BDI', 25, NULL)

CREATE TRIGGER Ins_View
INSTEAD OF INSERT ON Esame INF
FOREACH ROW
BEGIN
    IF NOT EXISTS (SELECT
                    FROM Studenti S
                    WHERE S.matr= NEW.matr)
    THEN
        INSERT INTO Studenti
        VALUES (NEW.matr, NEW.nome, NEW.cognome, 'N86')
    ENDIF;
    INSERT INTO Esami
    VALUES (NEW.matr, NEW.corso, NEW.voto, NEW.lode)
END

BEGIN
    DECLARE
        Sezione di dichiarazione di variabili
    END DECLARE
    BEGIN
        BODY
        EXCEPTION
    END

```

## Lezione del 23-11

Dichiarazione del TRIGGER:

- Azione (scritta in un linguaggio detto PLSQL (linguaggio proprietario di Oracle, procedurale che estende l'insieme di istruzioni DBMS con costrutti di controllo)) contenuta tra BEGIN e END

### PLSQL

- Procedurale
- Proprietario di Oracle
- È eseguito dal DBMS (lato server)
- Estende l'insieme di istruzioni DBMS con costrutti di controllo

Dati del DB possono essere trasferiti all'interno delle variabili (e viceversa)

Dati DB (deposito di dati) \iff dati nella variabile (elaborazione dati)

## 1. Variante SELECT -> il risultato è inviato allo STANDARD OUTPUT

SELECT *expr<sub>1</sub>* , ..., *exp<sub>n</sub>*

*INTO var<sub>1</sub>, ..., var<sub>n</sub>*

FROM

WHERE

Condizione necessaria → la select restituisce al più una riga.

Nel caso dovessi recuperare più di una riga non posso utilizzare SELECT ... INTO.

Devo utilizzare una struttura che si chiama **CURSOR**.

Attraverso il **CURSOR**: o

Ottengo una tabella → l'output viene riportato in un buffer e attraverso il **CURSOR** posso trasferire il contenuto riga per riga all'interno delle variabili

MEMO:

Tutto quello che è in parentesi quadre è opzionale

### SQL STANDARD

```
DECLARE NomeCursore [scroll]
      CURSOR FOR SELECT SQL
      [FOR < READ ONLY]
      UPDATE [OF Attributo, ..., Attributo]
```

- **FOR READ ONLY** (default) -> i dati recuperati dal cursore non saranno modificati nel DB (nella procedura posso fare quello che voglio)
- **DEFAULT NO SCROLL** -> il risultato può essere scandito solo sequenzialmente-> l'ordine di scansione sequenziale non è alterabile

Con lo scroll l'ordine di scansione è alterabile da utente

- **DECLARE** <- Associo un cursore (iteratore) alla tabella risultato della SELECT
- **OPEN** (<nome\_cursore>) <- manda in esecuzione la SELECT inizializzando il buffer
- **CLOSE** (<nome\_cursore>) <- (operazione duale dell'OPEN) disalloca il buffer
- **FETCH**[Posizione FROM] NomeCursore INTO var\_{1}, ..., var\_{n} <- trasferisce i dati della riga corrente nelle variabili (var\_{1}, ..., var\_{n})

Associato ad una FECTH c'è il concetto di riga corrente -> la FECTH aggiorna la posizione corrente

```
FETCH [Posizione FROM] < cursore:
```

NEXT

PRIOR

FIRST

```
LAST  
ABSOLUTE  
RELATIVE
```

Nel caso NO SCROLL -> FETCH <nome\_cursore> INTO < lista variabili > (il NEXT è implicito)

**PROCEDURE** <- dette anche STORED PROCEDURE (lo stored ci dice che è un aspetto procedurale, eseguito sul lato del DB)

```
CREATE PROCEDURE < nome prod >  
(par_{1} Tipo_{1} [IN/OUT/INOUT]  
par_{2} Tipo_{2} ...)
```

```
CREATE FUNCTION <nome funzione>  
(par_{1} Tipo_{1} [IN/OUT/INOUT]  
par_{2} Tipo_{2} ...)  
RETURN tipoR
```

Funzione che data la matricola dello studente restituisce il nome e cognome in maiuscolo concatenato (NOME-COGNOME)

Tabella Studente con (Nome, Cognome, Matricola)

```
CREATE FUNCTION nc  
(matricola Studente.matricola%TYPE)  
RETURN VARCHAR2(40)  
temp VARCHAR2(40) := '' (nome      tipo      init (opzionale))  
BEGIN  
    SELECT UPPER(S.nome) || '-' || UPPER(S.cognome)  
        into temp  
    FROM Studenti S  
    WHERE S.matricola = matr
```

[DICHIARAZIONE VAR]

BEGIN

...

EXCEPTION

...

END

## Lezione del 24-11

In una procedura il cursore può essere aperto/chiuso più volte → lanciare più volte interrogazioni

`FETCH < nome cursore > INTO variabile`  
→ dalla CURRENT del cursore alle variabili

`CURSOR < nome cursore > AS SELECT`  
→ Il Tipo < nome\_cursore > % ROWTYPE → il tipo di dato della riga restituito dalla SELECT →  
RECORD con campi: un campo per ogni attributo

Per accedere ai campi:

`nomerecord.nomecampo`

`miavariabile.< campo >`

Informazioni while per la scansione del cursore

```
< nome cursore > % NOT FOUND
```

exp booleana vera se non c'è più nessuna riga da esaminare nel cursore

```
< nome cursore > % FOUND
```

c'è ancora una riga ancora da scandire

```
< nome cursore > % ROWCOUNT
```

Intero. Indica il numero di righe del cursore scandite

Iterazioni:

```
WHILE < CONDIZIONE >
  LOOP
  ...
ENDLOOP

FOR I IN Cursore / SELECT
  LOOP
  ...
END
```

Intero per ogni riga recuperata dalla esecuzione della SELECT

- Non serve aprire/chiudere il cursore (è aperto automaticamente dall'inizio del ciclo for)
- Non serve dichiarare la variabile di riga
- Non serve fare FETCH (ad ogni ciclo la FETCH è implicita -> questa viene associata al record)

```
FOR I IN expr_{1} ... expr_{2}
  LOOP
```

```
...  
END LOOP
```

## COSTRUTTI DI SCELTA

```
IF (cond) THEN istruzioni  
[ELSE]  
    istruzioni  
END IF  
IF (cond) THEN  
    IF (cond) THEN  
    ENDIF  
ELSE  
ENDIF
```

```
CASE Expr  
WHEN (condizione) THEN istr  
  
WHEN (condizione) THEN istr  
  
ELSE istruzione  
  
END CASE
```

```
ISTRUZIONE  
RAISE (nome eccezione)
```

```
DECLARE
```

mia ecc. EXCEPTION (questo tipo di eccezione è una mia eccezione che mi comporta qualcosa)

```
IF < ... > THEN RAISE mia ecc.
```

Una volta sollevata un'eccezione non è possibile riprendere il corpo di esecuzione procedurale

```
EXCEPTION
```

```
WHEN mia ecc THEN istruzioni
```

Esempi di procedure:

DBMS:

- METADATI (CREATE per i METADATI)
- DATI

I metadati vengono memorizzati e strutturati secondo il modello relazionale. Per i metadati si utilizza generalmente il termine DIZIONARIO

Procedura che prende in ingresso il nome di una tabella e restituisce la definizione della tabella

```

CREATE FUNCTION creatab
  (nome USER_TABLES% TableNome)
RETURN (VARCHAR2(2000))
risult VARCHAR2(2000):
  "CREATE TABLE " || nome || '(';
CURSOR CC AS
  SELECT Column_Nome, DataType
  FROM USER_TAB_COLUMNS
  WHERE Table_Name = nome
  ORDER BY Position
BEGIN
FOR I IN CC
  LOOP
    risult := risult || I.Column_Name || ' ' || I.Data_Type || ',';
  ENDLOOP
  risult := SUBSTR(risult, 1, LENGTH(risult - 1))
  RETURN risult
END

```

# Lezione del 30-11

---

TREE(CodT, Radice)

NODE(CodN, Label, CodT)

ARCO(CodA, label, CodT, Padre, Figlio)

Funzione che dato un nodo restituisca la somma delle etichette dei nodi dal nodo fino alla radice

```
CREATE FUNCTION sommanodi(nodo NODE%TYPE)
  RETURN INTEGER
  CURRENT NODE%TYPE := nodo
  SOMMA INTEGER := 0

  BEGIN
    SELECT label INTO somma
    FROM NODO
    WHERE CodN = CURRENT;

    WHILE EXIST (SELECT
                  FROM ARCHI
                  WHERE A.Figlio = CURRENT)
      LOOP
        SELECT A.Padre INTO CURRENT
        FROM ARCHI A
        WHERE A.Figlio = CURRENT;

        SELECT label+somma INTO somma
        FROM NODO
        WHERE CodN = CURRENT;

    RETURN SOMMA
```

```
CREATE FUNCTION sommanodi(nodo NODE%TYPE)
  RETURN INTEGER AS //una volta messa l'intestazione per iniziare il
corpo della procedura è richiesto l'AS
PROF INTEGER := 1; //inizializzo la profondità
BEGIN
DELETE FROM TMP; //cancello il contenuto -> inizio da zero
INSERT INTO TMP VALUES (nodo, 0) // lo metto a profondità zero
LOOP
  INSERT INTO TMP
```

```

(SELECT A.figlio, PROF
  FROM TMP AS T JOIN ARCHI AS A ON T.CodN = A.padre
 WHERE T.profondità = PROF - 1);
EXIT WHEN NOTEXIST
(SELECT *
  FROM TMP T
 WHERE T.profondità = PROF)
PROF := PROF +1
ENDLOOP

SELECT (label) INTO risultato
FROM TMP JOIN NODON ON T.CodN = N.Cod

RETURN risultato

END

```

In questo modo compio un analisi di tutta la struttura dell'albero, riempiendo una tabella in cui per ogni nodo la sua profondità a partire dalla radice.

L'uso della profondità mi permette di evitare di considerare dei nodi con profondità più alta (e che ho già esaminato). Non faccio le cose due volte.

## Lezione del 01-12

```

TMP(CodN)
(root NODO%CodN)

BEGIN
DELETE FROM TMP;
INSERT INTO TMP VALUES (root);
CONT := 1 // assegno un contatore
LOOP

INSERT INTO TMP
(SELECT A.figlio
  FROM TMP JOIN ARCHI AS A ON T.CodN = A.Padre
 WHERE A.figlio NOT IN
 (SELECT *
   FROM TMP))
SELECT COUNT(*) INTO CONT2 FROM TMP;
EXIT WHEN CONT = CONT2;

```

```
CONT := CONT2  
ENDLOOP
```

Ho due contatori (CONT tiene il conteggio delle righe che stanno prima dell'inserimento).

#### Tabella espansa

Dato un albero cerchiamo le sue informazioni  
ESPANSA(NODO, DISCENDENTE, PROFONDITÀ)

#### SQL DINAMICO

Si parla di SQL dinamico quando i comandi SQL non sono scritti direttamente dal programmatore ma sono programmati. Parto da una variabile e costruisco il comando (lo preparo e lo mando in esecuzione)

```
VARIABILE STRINGA comando
```

```
comando := ...
```

```
PREPARE comando // esamina il comando e ne fa una valida sintattica  
EXECUTE comando
```

Alternativamente posso fare anche un **EXECUTE IMMEDIATE** (mando direttamente il comando). Il rischio è che il comando possa essere mandato in esecuzione tante volte ma con una sola PREPARE.  
Esempio

Supponiamo di avere una procedura

```
CREATE PROCEDURE ADDPK  
(tabella UNSER_TABLE%TABLE_NAME,  
 elenco VARCHAR2(1000))
```

```
COMANDO VARCHAR2(1000) := "ALTER TABLE" || tabella || 'ADD CONSTRAINT pk'  
PRIMARY KEY ('||elenco||' ', '');
```

```
BEGIN
```

```
EXECUTE IMMEDIATE comando  
END
```

, separatore nome attributo

Avrei potuto:

- verificare se tabella è un nome corretto;
- se gli elementi di elenco sono corretti

```

SELECT COUNT(*) INTO CK
FROM USER_TABLE
WHERE TABLE_NAME = tabella
LOOP

```

I campi su cui si fa l'interrogazione sono variabili → vengono decise dinamicamente

```

SELECT COUNT(*)
FROM TAB
WHERE ? //viene costruita in maniera dinamica in base all'analisi

```

es.

```

CREATE FUNCTION Contarighe (condizione VARCHAR2(2000))
ATTRIBUTO VALORE VARCHAR(100)
VAR INTEGER;
CORRENTE VARCHAR(1000) := condizione
...

```

comando VARCHAR2(2000):=

```

'SELECT COUNT(*) INTO VAR
FROM TAB
WHERE TRUE'
BEGIN
WHILE INSTR(corrente,',',') >= 1
LOOP
    pos1 := INSTR(corrente, 1, ',', ')
    POS2 := INSTR(corrente, 1, ',', ', 2)
    attributo := SUBSTR(corrente, 1, instr( pos1 - 1)
    IF pos2 >= 1
        THEN
            corrente := SUBSTR(corrente, pos2 + 1)
            valore := SUBSTR(corrente, pos1 + 1, pos2 - pos1)
        ELSE
            corrente := '';
            valore := SUBSTR(corrente, pos1 + 1)
        ENDIF
        comando := comando || 'AND' || attributo || '=' || valore;
    ENDLOOP
EXECUTE IMMEDIATE comando
RETURN VAR

```

---

Nel caso in cui si utilizzi un cursore per recuperare le righe

```

comando VARCHAR2 ....
miocursore REF SYSCURSOR //nome che riferirò al cursore che sto costruendo

BEGIN
    costruisco il cursore
    OPEN miocursore USING comando
    WHILE miocursore%FOUND
        LOOP
            FETCH miocursore INTO riga
            risultato:= risultato || riga
        ENDLOOP

```

Ho costruito un cursore, ho dichiarato un riferimento ad un cursore e al momento dell'apertura del cursore associo a questo riferimento (miocursore) il comando che ho costruito.

Creo l'associazione tra il comando preparato e il riferimento al cursore.

---

## Lezione del 03-12 (barra)

### JDBC

API JAVA per accedere alle basi di dati relazionale.

Insieme di classi e interfacce che permettono a Java di operare con le Basi di Dati

L'accesso diretto dei dati ha vantaggi e svantaggi.

Vantaggi:

- Efficente per soluzioni piccole
- Svantaggi
- Sviluppo lento
- Non flessibile
- Non condivisibile
- Distoglie dalla 'business logic'

Utilizzo di DBMS

- Applicazione general-purpose per manipolare le basi di dati
- Fornisce un modo per memorizzare informazioni in strutture dati efficienti
- Permette ricerche da più direzioni
- Architettura a due livelli
  - client (che conosce come gestire i dati)
  - DBMS (che gestisce lo storage ed il retrieval dei dati)

Si usiranno delle classi proprie di SQL non interne alle java.utils

Per gestire l'interazione con il DBMS

- **EMBEDDED SQL**

- il programma colloquia direttamente con il DBMS
- Gli statement SQL sono compilati utilizzando un precompilatore specifico del DBMS
- SQL diventa parte del codice

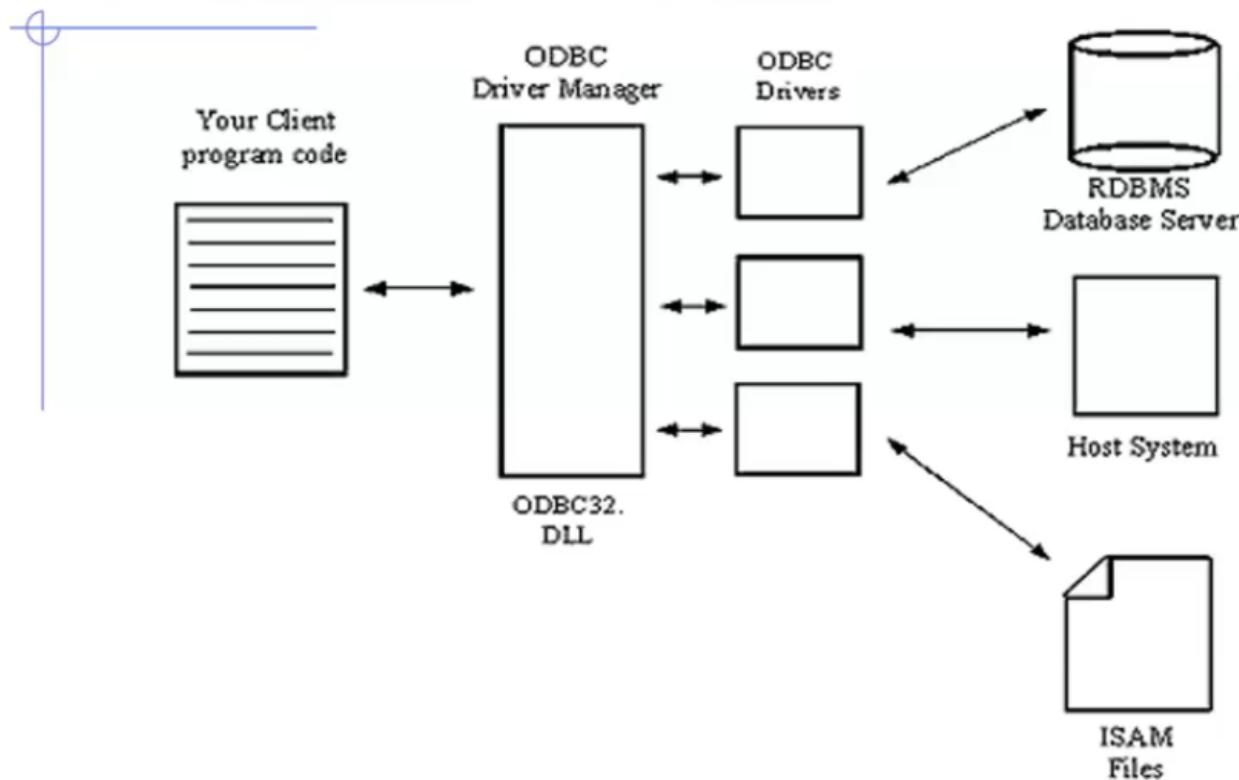
- **Call Level Interface**

- Ci si collega al DBMS attraverso a

Esistono vari tipi di CLI → JDBC, ODBC

ODBC → API della microsoft che permette l'accesso a dati residenti in DBMS diversi. È supportato da tutti i DBMS in commercio e gestisce le richieste SQL convertendole in un portmato comprensibile ad ogn DBMS.

# Architettura ODBC



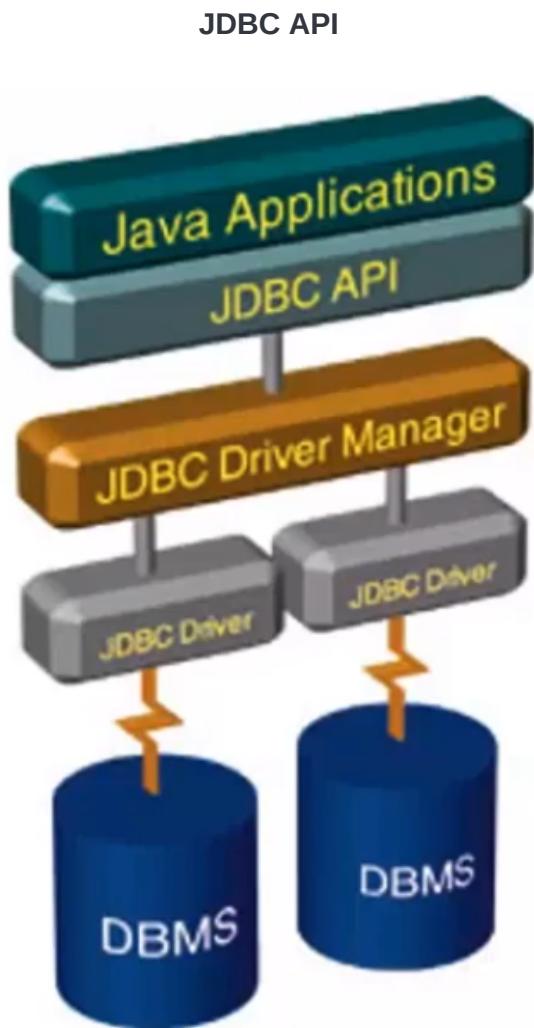
JDBC → è in grado di interagire con qualsiasi DBMS

Supporta 4 tipologia di Driver:

- Tipo 1 → ODBS

- wrapper da JDBC ad ODBC
- traduzione diretta
- è incluso nel jdk, ma è molto lento

- Tipo 2
  - wrapper da JDBC a driver nativo DBMS
    - traduce da JDBC in chiamate JNI al codice C/C++ del DBMS driver
    - facile da realizzare, ma un bug del driver causa crash del sistema (non è adatto al web e non era facile da configurare)
- Tipo 3
  - Driver Pure Java per DB Middle Ware
  - le chiamate JDBC venivano tradotte in un net-protocol indipendente dal DBMS
    - utopico, ma non realizzabile
- Tipo 4
  - driver Pure Java per DBMS
    - traduce le chiamate JDBC in un net-protocol comprensibile dal DBMS
    - molto veloce e facile da configurare, ma non è supportato da tutti i DBMS (richiede 1 Driver per ogni DBMS, lato client)



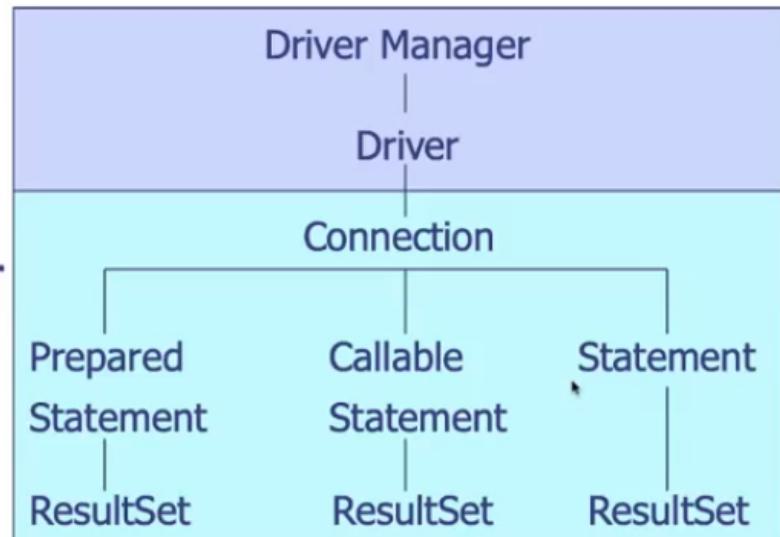
Presenta 2 livelli:

- JDBC driver (verso il DBMS)

- JDBC API (verso l'applicativo)

## Driver Layer

## Application Layer



### Puntualizzazione

Tutte le classi fondamentali di JDBC sono interfacce.

- Non sono implementate in JDBC
- Sono implementate dallo specifico driver

JDBC definisce un insieme di tipo SQL nella classe **java.sql.Types**

Ogni applicazione JDBC deve:

- Caricare un Driver (Driver Manager)
- Ottenere una connessione (Classe Connection)
- Eseguire uno Statement (Classe Statement)
- Gestire i risultati
- Rilasciare le risorse

Caricare il Driver

```

import java.sql.*;

try {
    Class.forName(NOME_DRIVER);
} catch(ClassNotFoundException) {
}
  
```

```
//Driver non Trovato
}
```

L'inizializzatore statico del driver, chiamato dalla JVM al caricamento della classe, si registra nel Driver Manager (automatico)

```
public class MyDriver
{
    static
    {
        new Mydriver();
    }
    public MyDriver()
    {
        java.sql.DriverManager.register(this);
    }
}
```

Ottenerne una connessione

Connection con =

```
DriverManager.getConnection(
    URL_MY_DATABASE);
```

Gli URL in JDBC sono una stringa formata da nodi, separati da ':' o '/'

È formato da

protocollo:sottoprotocollo:databasename

Il Driver Manager trova il driver appropriato chiamando il metodo **acceptURL(URL)** di ogni driver caricato. Il primo driver che risponde true, stabilisce una connessione. È possibile in questo modo scegliere il driver JDBC appropriato a run-time

Eseguire uno Statement

Lo Statement è l'oggetto che "trasporta" le istruzioni SQL sulla connection verso il DB. È unico per tutta la durata della connessione:

- Per fare una nuova query, si cambia la stringa SQL al suo interno

```
try
{
    Statement st = con.createStatement();
    ResultSet rs = st.executeQuery("SELECT nome FROM studenti");
}
catch(SQLException sqe)
```

```
{  
    //problema  
}
```

Lo scopo principale della classe Statement è eseguire istruzioni SQL

Si utilizza il metodo executeQuery() per gli statement che restituiscono tuple

- Comandi DQL
- Restituisce un ResultSet  
Si utilizza executeUpdate() per gli statement che non restituiscono tuple
- Comandi DDL/DML
- Restituisce il numero di righe modificate  
JDBC 2.0 introduce anche executeBatch() per eseguire più statement in sequenza

---

All'interno dell'IDE

- Creazione progetto
- Importare la libreria per il DBMS
- Caricare il Driver (funziona anche senza dover inserire il Class.forName(), ma è utile per essere sicuri di agganciare il driver corretto)

```
try{  
  
    Class.forName("org.postgresql.Driver");  
}  
catch(ClassNotFoundException e)  
{  
    System.out.println("Class Not Found:\n "+ e);  
}
```

- Ottenere una connessione

```
try{  
Connection conn =  
DriverManager.getConnection("jdbc:postgresql://localhost:5432/airdb");  
    Statement st = conn.createStatement();  
    ResultSet rs = st.executeQuery("SELECT * FROM AAAA");  
    int i = 1;  
    while(rs.next())  
    {  
        System.out.println("Aereo numero: " +i);  
        System.out.println("ID = " + rs.getString("id"));  
        System.out.println("produttore = " + rs.getString("produttore"));  
    }  
}
```

```

        System.out.println("Modello = " + rs.getString("modello"));
        System.out.println("Data immatricolazione = " +
rs.getDate("dataimm"));
        System.out.println("Numeri posti = " + rs.getInt("numposti"));
        i++;
    }
}
catch(SQLException e)
{
    System.out.println("SQL Exception:\n "+ e);
}

```

- Eseguire uno Statement (per comodità le creiamo assieme alla connection)
- Gestire i risultati

- Release delle risorse

```

rs.close();
st.close();
conn.close();
catch(SQLException e)
{
    System.out.println("SQL Exception: \n" +e);
}

```

# Lezione del 10-12

## Design Pattern Creazionali

(singleton)

- Fornisce un'astrazione del processo di instanziazione degli oggetti
- Permettono ad un sistema di essere indipendente da
  - Modalità di creazione, rappresentazione e composizione degli oggetti

Elementi caratterizzanti:

- Capacità di incapsulare la conoscenza delle classi concrete utilizzate nel sistema
- Nasconderne le modalità (di creazione e composizione) delle classi
- Configurare un sistema con oggetti che possono variare nel tempo (sia come struttura che come funzionalità)

I DP creazionali più utilizzati sono:

- Abstract Factory
- Builder
- Factory Method
- Prototype
- Singleton

		Purpose		
		Creational	Structural	Behavioral
Scope	Class	Factory Method	Adapter (class)	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter (object) Bridge Composite Decorator Flyweight Facade Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

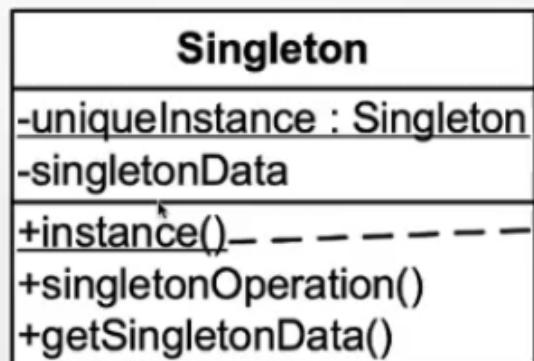
## Singleton

- + Assicura che una classe abbia una sola istanza
- + Fornisce un punto d'accesso globale a quella istanza

La motivazione è garantire l'esistenza di una sola istanza di una classe, con lo scopo di mantenerne la stabilità (prendi come esempio tot stampanti → esiste una sola coda di stampa)

Per far ciò facciamo che sia la classe stessa a gestirne l'istanza

Definisce un'operazione *Instance* che consente ai clienti di accedere all'unica istanza esistente della classe (il Singleton può essere responsabile della creazione della sua unica istanza)



```
public class Singleton{  
  
    private static Singleton istanza = null;  
    private Singleton (){  
        public static Singleton getSingleton() {  
            if (istanza == null)  
                istanza = new Singleton();  
            }  
            return istanza;  
        }  
}
```

### Prepare Statement

- SQL Statement precompilato
- Utilizzabile con query molto simili nella struttura, ma che cambiano spesso i parametri
- Migliora le prestazioni se le query sono eseguite molte volte

```
PreparedStatement updateEsami =  
    con.prepareStatement("UPDATE Studenti SET esami = ? WHERE Matricola  
LIKE ?");  
    updateEsami.setInt(1,13);  
    updateEsami.setString(2, "011/245389");  
    updateEsami.executeUpdate();
```

Per controllare che un eventuale tabella non esista se ne controllano i Metadata.