



A.A. 2020/21

BASI DI DATI - TEORIA

ALBERTO BELUSSI

FABS :)

NOTA

Questi appunti/sbobinatura/versione “discorsiva” delle slides sono per mia utilità personale, quindi pur avendole revisionate potrebbero essere ancora presenti typos, commenti/aggiunte personali (che anzi, lascio di proposito) e nel caso peggiore qualche inesattezza!

Comunque spero siano utili!  

**Questa sbobina fa parte della mia collezione di sbobinature,
che è disponibile (e modificabile!) insieme ad altre in questa repo:
<https://github.com/fabfabretti/sbominamento-seriale-uniVR>**

TABLE OF CONTENTS

Sistemi Informativi	4
Applicazioni gestionali.....	5
Modello dei dati	7
Modello, schema e istanza	8
Architettura di un DB.....	9
Proprietà.....	9
Progettazione di una base di dati.....	10
Modello Entità Relazione	11
Costrutti del modello ER.....	11
Entità (o TIPO di entità)	11
Relazione (o TIPO di relazione).....	11
Attributo	12
Identificatore di identità.....	13
Vincoli di cardinalità:	15
Attributo opzionale e multivalore	16
Attributo composto	16
Generalizzazione di entità	18
Relazioni ternarie.....	20
Startegie di progettazione	22
Analisi di qualità	22
Implementazione dei dati.....	24
Modello relazionale	24
Domini di base	24
Relazione (o tabella).....	24
Progettazione dei dati nel modello relazionale.....	27
Modello relazionale.....	27
Decomposizione della relazione unica (rimozione delle anomalie).....	28
Proprietà del modello:.....	28
Terminoloigia varia.....	29
Valori nulli.....	30
Vincoli di integrità.....	32
Notazione per vincoli strutturali.....	34
Notazione per integrità referenziale	34
Progettazione logica	36
Fase 1: ristrutturazione	36
1.1 Analisi delle derivabilità.....	36
1.2 Eliminazione delle generalizzazioni	37

1.3	Partizionamento e accorpamento di relazioni	38
1.3	Scelta degli identificatori principali.	39
	Regole di traduzione verso il modello relazionale	40
	Regolette	40
	Algebra Relazionale	44
	Operatori	44
	Operatori insiemistici	45
	Operatori specifici	46
	Operatori di giunzione o JOIN	48
	Algebra con valori nulli.....	53
	Selezione.....	53
	Join naturale	53
	Join esterni	53
	Ottimizzazione di espressioni DML.....	54
	Ottimizzatore.....	54
	Equivalenza.....	54
	Trasformazioni di equivalenza.....	55
	Vista	58
	SQL.....	59
	Forma base di un'interrogazione SQL	59
	Clausola SELECT	60
	Clausola FROM.....	60
	Clausola WHERE	61
	Clausola ORDER BY	61
	Interrogazioni nidificate	62
	Clausula EXISTS.....	63
	Clausula IN e NOT IN.....	64
	Esercizi	64
	Interrogazioni con operatori aggregati	65
	Operatori aggregati standard (SQL92)	65
	Interrogazioni con raggruppamento	66
	Group by	66
	Clausula HAVING	66
	Interrogazione di tipo insiemistico	67
	Viste SQL.....	68

SISTEMI INFORMATIVI

» **Anni '60 :**

- › L'informatica applicata è utilizzata solo in laboratori di ricerca, con enfasi sugli algoritmi

» **Anni '80 :**

- › L'informatica esce dal laboratorio e inizia ad essere usata nei confini gestionali. I requisiti e l'enfasi si spostano dall'algoritmo al dato: si devono svolgere operazioni semplici su grandi moli di dati.
- › È la nascita del *sistema informativo*.

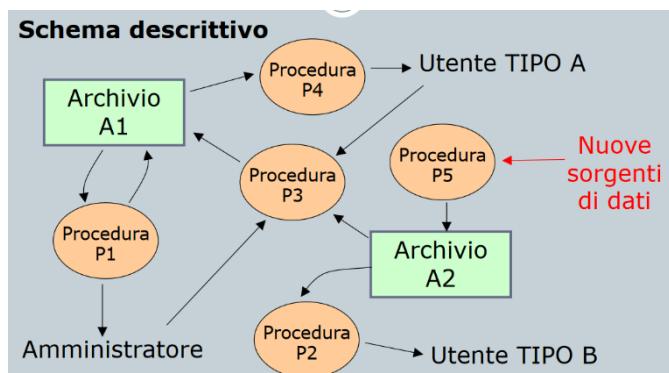
Sistema Informativo

Insieme delle attività umane e dei dispositivi di memorizzazione ed elaborazione che organizza e gestisce le informazioni richieste.

[!] Non contiene necessariamente tecnologia informatica!

Dato: Elemento di conoscenza di base.

Informazione: interpretazione dei dati che permette di ottenere più conoscenza.



I sistemi informativi possono essere descritti informalmente attraverso uno schema descrittivo informale.

Alcuni esempi di "nuove sorgenti di dati" possono essere dati non inseriti manualmente, ma risultanti da particolari sensori o persino dai social media. Queste nuove sorgenti producono nuovi flussi informativi che andranno integrati con il resto del sistema informativo.

Il workflow (o diagrammi di flusso) è lo strumento formale da studiare che permette di:

- » Definire archivi dati, sorgenti dati
- » Definire utenti
- » Definire procedure e processi
- » Definire flussi di dati

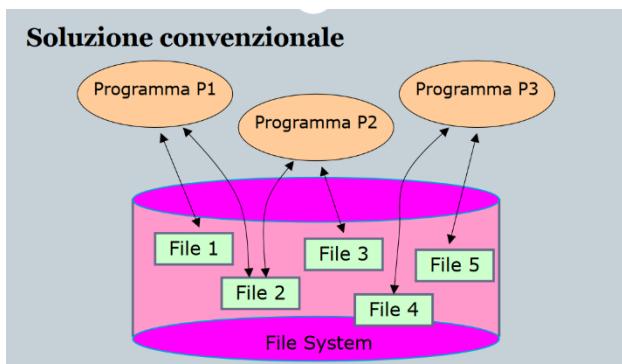
Per introdurre automazione nel sistema informativo bisogna spesso riorganizzare l'intero sistema. Il successo di un progetto dipende, spesso, dal saper individuare bene quali sono i punti su cui agire!

Base di Dati

È una collezione di dati utilizzati per rappresentare con tecnologia informatica le informazioni di interesse per un sistema informativo.

APPLICAZIONI GESTIONALI

SOLUZIONE CONVENZIONALE



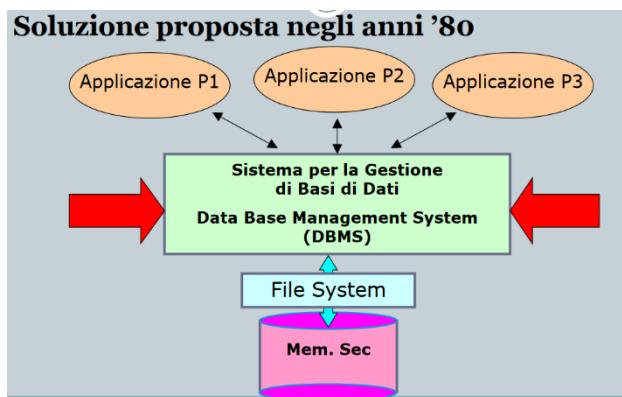
Scrivo programmi in qualche linguaggio che interagiscono con la memoria secondaria via file system dove i file sono organizzati come cartelle e file di record omogenei fra loro. L'unico modo di dialogare col file è aprirlo e scandire i record contenuti in esso.

[Le nuove tecnologie vogliono considerare le situazioni in cui il dato da memorizzare è enorme e non sta in un tipico database; quindi sono ripartite proprio da questa idea :)]

Tuttavia, questa soluzione presenta dei problemi:

- » Il file è ad **accesso sequenziale**, dunque è molto inefficiente.
- » Il dato è spesso **ridondante**: se progetto in modo autonomo moduli di un'applicazione, rischio di avere dati duplicati.
 - Questo produce il fenomeno dell'**inconsistenza**, poiché è facile cadere in aggiornamenti parziali.

SOLUZIONE ANNI '80: DBMS



DBMS

È un sistema che gestisce su memoria secondaria collezioni di dati chiamate Basi di Dati dalle seguenti caratteristiche:

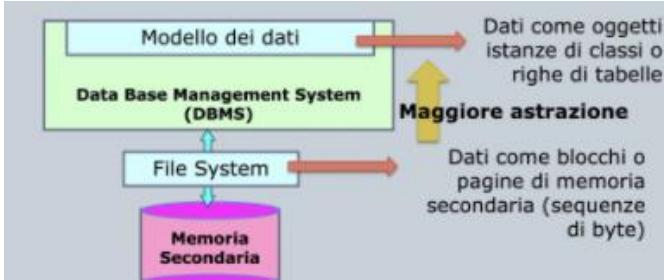
- » **Grandi**
- » **Condivise**
- » **Persistenti.**

Esso deve assicurare:

- » **Affidabilità**: tenta di mettere in atto tecniche che prevengano elaborazioni in corso e dati al verificarsi di problemi tecnici.
- » **Privatezza**
- » **Accesso efficiente**: ottimizzazione e indirizzamento dei dati, in modo da essere più appropriati del puro accesso sequenziale

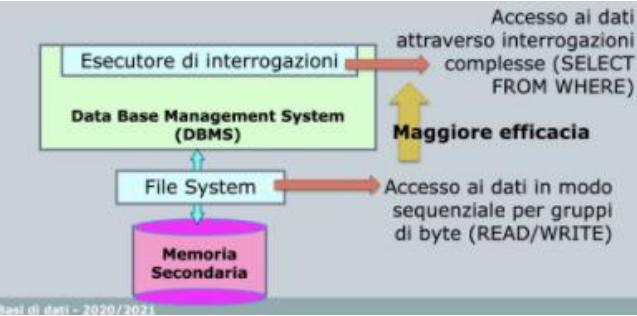
Vantaggi di un DBMS

» Maggiore astrazione



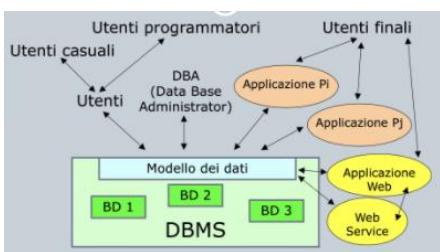
Maggiore espressione per descrivere le caratteristiche del dato rappresentato; si concretizza nella realizzazione di un modello dei dati. Questa astrazione permette di passare da un file visto come blocco su memoria secondaria nel file system a un livello dove i dati sono oggetti, istanze di classi o anche come righe di una tabella, e ci permette di specificare informazioni (es. il tipo).

» Maggiore efficacia



L'accesso ai dati passa da essere puramente sequenziale al poter essere gestito in maniera più appropriata.

Interazione con il sistema



L'interazione con il sistema avviene in più modi; in generale, una DBMS gestisce più di un database; posso gestire la mia personale così come dover interagire con DB condivise. Ad accedere al DBMS possono essere sia utenti effettivi (che usano programmi più o meno avanzati di accesso), sia applicazioni che necessitano di dati per funzionare.

Gli utenti che interagiscono con il DBMS usano dei linguaggi. Sono molto diversi dai normali linguaggi di programmazione, e sono di tipo dichiarativo: non sono una sequenza di operazioni, ma un'espressione che descrive il dato (esempio: tutti gli studenti che fanno Rossi di cognome).

» DML: un linguaggio per l'interrogazione e aggiornamento dei dati

- Linguaggio di interrogazione (estrae)
 - SQL e algebra relazionale
- Linguaggio di manipolazione (popola, modifica e cancella)
 - SQL

Questo tipo di approccio caratterizza il linguaggio SQL, ma ci sono anche altri tipi di interazione, come i DDL: linguaggio per la definizione dei dati.

MODELLO DEI DATI

Modello dei dati

E' l'insieme dei costrutti forniti dal nostro sistema DBMS per descrivere la struttura e le proprietà dell'informazione contenute nei basi di dati.

Gli obiettivi dei costrutti sono:

- » **Descrivere la struttura di dati** che conterrà le informazioni (simile ai costruttori dei linguaggi di programmazione)
- » Specificare le **proprietà** che dovranno soddisfare le istanze di informazione che saranno contenute nelle strutture dati

[!] Attenzione: la parola modello è spesso usata con accezioni diverse! Per esempio, nel mondo della fisica il modello è usato per etichettare la descrizione di un fenomeno fatta attraverso un linguaggio formale. Al contrario, in ambito basi di dati il modello è l'insieme degli strumenti che ho a disposizione per descrivere la base di dati!! (-> qui il modello sarà, per esempio, la matematica)

Come visto precedentemente, il modello dei dati è ciò che caratterizza il DBMS.

I modelli che caratterizzano i DBMS sono:

- » **Modelli del passato:** reticolare e gerarchico
- » Modelli attuali:
 - > **Modello relazionale:** risale agli anni '70 ed è stato poi standardizzato nei '90-'89. E' il nostro modello target del corso, ed è stato poi sostituito da altri approcci che però non hanno sempre avuto successo
 - > **Modello ad oggetti:** ha prodotto la programmazione ad oggetti! L'apporccio ad oggetti è molto più espressivo del relazionale, ma c'era stato un investimento enorme nel relazionale e il passaggio era troppo complesso in quanto molto distante dal relazionale. Inoltre, è anche un po' meno efficiente: insomma, fu un fallimento e si tornò al relazionale
 - Lol: un investimento fatto sui dati è difficile da far evolvere! Investire tanto in un tipo può portare persino a non saper più che caspita ci sia dentro una base di dati.
 - > **Modello object-relational:** è il SQL3 e semplicemente si è aggiuntato qualche ingrediente dell'object al relazionale classico, senza che i due mondi non si possano più parlare come successe con l'object relational.
 - > **Modello semistrutturato:** lo schema non è strutturato: spesso è verboso...
 - JSON e XML
 - > **Modelli noSQL:** mandano a cagare l'SQL perché pur avendo molti pregi, le tecnologie associate all'SQL non erano adeguate a nuovi requisiti (per esempio i big data).
 - Si riparte dal file e costruiscono altre idee come document-based, column-based, key-value, cluster, gestione parallela dell'interrogazione...

MODELLO, SCHEMA E ISTANZA

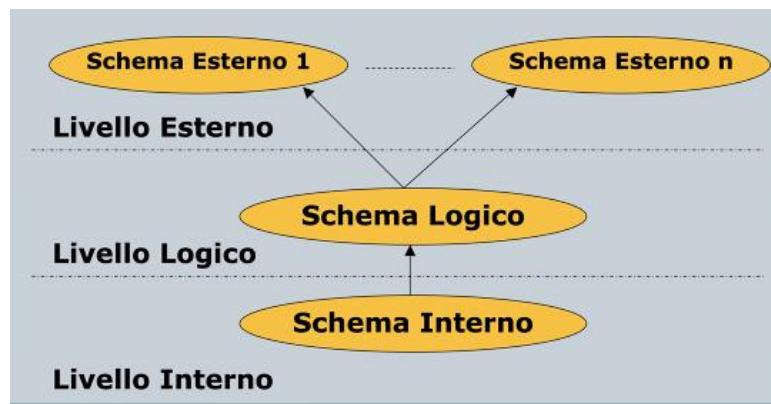
Per descrivere un DBMS abbiamo bisogno di:

- » **Modello** di dati che descriva strutture e proprietà; insieme di strumenti per definire lo schema
- » **Schema**: quello che tipicamente è definito modello in altri contesti; è la struttura delle proprietà di un DB fatta utilizzando i costituti del modello di dati. Descrive le proprietà invarianti nel tempo (ad esempio, "tutti gli studenti di bioinfo hanno matricola nome cognome")
- » **Istanza**: è l'insieme di informazioni che vado a mettere nella struttura dati; sarebbero i valori contenuti effettivamente nella struttura

Esempio		
	Modello dei dati	Schema
Basi di dati	Tabella (o relazione)	P (cognome: VARCHAR(40), nome: VARCHAR(30))
Linguaggi di progr.	Array	<pre>Class Persona { String cognome; String nome; } Class X { ... Persona[] p; p = new Persona[100]; }</pre> <p>The diagram illustrates an array p of type Persona[] pointing to a single Persona object. The Persona object has attributes Nome: Mario and Cognome: Rossi.</p>

ARCHITETTURA DI UN DB

L'architettura di un DB può essere schematizzata nel seguente modo:



Tutte le DB di tipo relazionale sono caratterizzate da questa struttura. Essa divide il mondo del sistema in tre parti, e in ciascuna definisce uno schema (= una descrizione della base di dati).

» *Livello esterno*

- > *Schema esterno*: è quello visibile ad applicazioni e utenti; descrive la porzione di schema logico che è utilizzata da una specifica applicazione o utente. L'idea è che la differenza fra logico ed esterno è minima (uno è porzione dell'altro).

» *Livello logico*

- > *Schema logico*: è il livello centrale, ed è la rappresentazione della struttura e delle proprietà della base di date definita attraverso i costrutti del modello. E' Lo schema della base di dati. Il modello è quello del sistema (es. relazionale).

» *Livello interno*

- > *Schema interno*: rappresentazione del DB per mezzo delle strutture fisiche che il sistema implementa; alla fine tabelle e oggetti saranno blocchi di memoria secondaria, oppure record di un file. Queste strutture fisiche mi dicono come è stata implementata ed è lo schema interno.

PROPRIETÀ

» *Indipendenza fisica*: lo schema logico della base di dati è completamente indipendente dallo schema interno

- > Quindi, mentre il sistema lavora posso cambiare l'implementazione fisica senza che nessuno se ne accorga; le interfacce verso i programmi rimangono invariate. E' importantissimo per fornire accesso al DB senza dover interrompere il servizio!!
Chiaramente, il file system non me lo permette.

» *Indipendenza logica*: gli schema esterni sono indipendenti da quelli logici.

- > Posso variare lo schema logico senza intaccare gli schemi esterni; l'interfaccia utente-dato rimane invariata

PROGETTAZIONE DI UNA BASE DI DATI

E' importante saper progettare il DB in modo indipendente dalla tecnologia.

Si parla di progettazione dell'intero sistema informativo. La soluzione va progettata con una metodologia tipica dell'ingegneria, che parte dallo studio di fattibilità.

- » **Studio di fattibilità:** definisce costi e alternative possibili prima ancora di arrivare in fondo. Ci vuole molta esperienza; noi ccciovani non lo possiamo fare. Non è affar nostro; noi partiamo da quello dopo.
- » **Raccolta e analisi dei requisiti:** individua proprietà e funzionalità del sistema producendo una descrizione completa ma informale. Produce l'insieme dei requisiti da cui partiamo noi progettatori di DB. I requisiti non sono contrattabili, a meno che non siano ambigui...
- » **Progettazione**

Si divide in due parti che producono documentazione e risultati differenti, ma possono comunicare fra loro:

- > **Progettazione dell'applicazione:** produce una SPECIFICA [trattata in Ingegneria del Software]
- > **Progettazione dati:** produce uno SCHEMA.

Metodologia di progettazione:

- *Decomposizione dei passi dell'attività di progetto*
- *Insieme di strategie e criteri di scelta*
- *Insieme di modelli di riferimento (passo ad un linguaggio formale dei requisiti).*
 - **Progettazione concettuale:**
Produce uno schema concettuale che è il punto di partenza della fase successiva e considera i requisiti. Ci concentriamo su questa: vogliamo una descrizione formale ma indipendente dalle operazioni e dal sistema scelto (ovvero astratta).
 - *Schema concettuale*

Schema concettuale

Documento formale che rappresenta il contenuto della base di dati in modo indipendente dall'implementazione.

[!] Non è solo un processo intermedio: diventa il punto di partenza e il risultato dell'intero processo, dato che descrive tutto il contenuto informativo.

- **Progettazione logica**

Parte dallo schema concettuale e crea lo schema logico per traduzione, applicando una serie di regole. Spesso faccio qualche aggiustamento

- Schema logico

- **Progettazione fisica**

Aggiunge parametri relativi alla memorizzazione fisica dei dati, eventualmente indici e altri stratagemmi che velocizzino l'accesso.

- Schema fisico

- *Implementazione su un DBMS*

- *Validazione e collaudo*

Eventualmente si ricomincia in caso di problemi, ma finita questa fase la DB può essere usata da un'applicazione.

MODELLO ENTITÀ RELAZIONE

E' il modello utilizzato storicamente per la progettazione concettuale di una base di dati in maniera totalmente astratta.

Ci fornisce un insieme di strumenti per definire struttura e proprietà di una DB.

E' formale e non ambiguo, ma semplice e indipendente dalla tecnologia .

Ha una sintassi grafica.

COSTRUTTI DEL MODELLO ER

Gli strumenti formalici che contengono il modello si chiamano **costrutti**.

Ogni costrutto viene definito con:

- » **Semantica:** il significato
- » **Sintassi grafica:** come lo inserisco nello schema
- » **Istanze o occorrenze:** come si rappresentano le istanze, ovvero devo precisare che cosa conterrà la base di dati.
 - Sempre in modo indipendente dalla tecnologia! Quindi useremo... la matematica. (*Boooo!!!*)

L'obiettivo è descrivere le caratteristiche del dato usando questi strumenti, che sono fatti apposta per mettere in evidenza le caratteristiche del dato necessarie a una buona implementazione.

ENTITÀ (O TIPO DI ENTITÀ)

» Semantica

Un'entità E rappresenta una **classe di oggetti** (=insieme di oggetti) con le seguenti caratteristiche:

- *Proprietà comuni*
- Hanno *esistenza autonoma* rispetto al resto dell'informazione che metto nel DB, ovvero ciò che stiamo rappresentando ha un insieme di proprietà comuni che nascono e muoiono "tutte insieme" e indipendentemente dal resto
- Hanno *identificazione univoca*, ovvero esiste una chiara corrispondenza tra gli oggetti istanze della nostra entità e i concetti istanziati nel sistema informativo

» Sintassi grafica



- E' un rettangolo che contiene il nome.

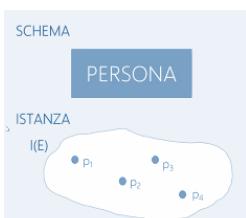
» Istanze o occorrenze

- Un'istanza dell'entità E è un oggetto appartenente alla classe rappresentata da E.
- Insieme delle istanze

L'insieme di tutte l'istante è definito come $I(E)$. L'insieme delle istanze di E è la popolazione dell'entità E ad un certo istante. Inizialmente sarà vuoto, poi sarà un insieme.

ESEMPIO

Posso rappresentare con un'entità il concetto di **persona**.



Lo faccio quando nei requisiti devo gestire le persone, e che 1. Le persone che rappresento nella DB sono tutte descritte con lo stesso insieme di proprietà 2. Hanno esistenza autonoma rispetto al resto: quando inserisco una persona nel DB, la persona deve nascere con tutte le proprietà che la descrivono (= tutte le proprietà nascono in quel momento). Per cancellarla, non devo tirar via o aggiungere altre informazioni.

RELAZIONE (O TIPO DI RELAZIONE)

» Semantica

- > Una relazione R rappresenta il *legame logico* fra due o più entità.
- > Tipicamente sono binarie, ma possono essere ternarie o oltre (*n-arie*).
- > Possono essere *ricorsive*

» **Sintassi grafica**

Si rappresenta con un *ROMBO* a cui si collegano entità coinvolte per mezzo di linee spezzate. Il nome della relazione è solitamente scritto vicino al rombo.



!!! Non ha detto nulla dell'autonomia!!! Wink wink

» **Istanze o occorrenze**

- > Data una relazione R tra n entità E_1, \dots, E_n , un'istanza della relazione R è una *ennupla di istanze* di entità. (e_1, \dots, e_n) dove $e_i \in I(E_i), 1 \leq i \leq n$
La relazione indica qualcosa con *minore autonomia* delle entità: deve esistere una relazione, altrimenti non nasce la relazione xD

» **Istanza della relazione binaria R**

- > L'istanza di R rappresenta **l'insieme delle coppie di entità E e F che sono in relazione**.

$$I(R) = \{ (e_i, f_j) \mid e_i \in I(E), f_j \in I(F) \}$$

Al momento della creazione, $I(R) = \emptyset$

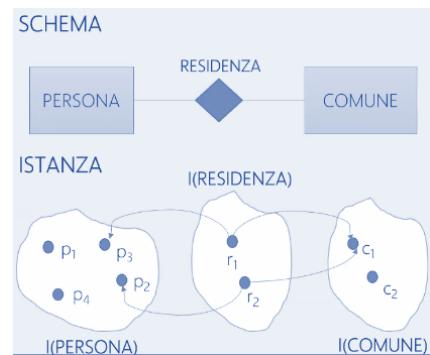
ESEMPIO

Supponiamo di aver inserito alcune entità – *persona* e *comune*.

Ipotizziamo che nei requisiti sia necessaria la residenza.

Rappresento il costrutto come *RELAZIONE di RESIDENZA*. Questo implica che:

- » Per esistere un'istanza di residenza devono esistere almeno un'istanza di persona e una di comune
- » Ogni istanza di residenza richiede sempre di essere rappresentata come coppia di istanze.



!!! Data una relazione R tra n entità, vale sempre la seguente entità sull'insieme delle istanze:

$$I(R) \subseteq I(E_1) \times \dots \times I(E_n)$$

L'insieme delle ennuple è un sottoinsieme del prodotto cartesiano di tutte le ennuple.

Il prodotto cartesiano genera, a partire da un certo numero di insiemi, tutte le coppie ordinate distinte.

CONSEGUENZA: NON POSSO RAPPRESENTARE LA STESSA ENNUPLA PiÙ VOLTE

Se qualcosa non funziona, **cambia schema!!**

ATTRIBUTO

» **Semantica**

Rappresenta una *proprietà elementare* di un'entità o di una relazione. Ogni attributo di entità,

l'attributo associa ad un'istanza **UNO E UNO SOLO** valore appartenente ad un dominio detto insieme dei valori ammissibili.

E' una funzione $f_a : I(E) \rightarrow D$

» Sintassi grafica:

Cerchio vuoto. Se lo coloriamo va male!!!



» Istanze o occorrenze

E' il valore **v** che l'attributo assume su un'istanza di E o di R.

- Attrivuto **a** su E o **b** su R: Data un'istanza **e** dell'entità **E** o una relazione **R**, l'istanza di un suo attrivuto **a** si ottiene dalla funzione f_a applicata a e:

valore di a su e = $f_a(e)$

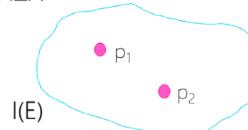
ISTANZA

SCHEMA



$f_{\text{nome}} : I(E) \rightarrow D_{\text{stringhe}}$

$$\begin{aligned}f_{\text{nome}}(p_1) &= \text{'Paolo'} \\f_{\text{nome}}(p_2) &= \text{'Maria'}\end{aligned}$$



$f_{\text{cognome}} : I(E) \rightarrow D_{\text{stringhe}}$

$$\begin{aligned}f_{\text{cognome}}(p_1) &= \text{'Rossi'} \\f_{\text{cognome}}(p_2) &= \text{'Verdi'}\end{aligned}$$

Esempio ER.1

REQUISITI

Progettare una base di dati che contenga le informazioni che descrivono la carriera di uno studente del corso di laurea triennale in informatica dal momento dell'immatricolazione fino alla laurea. In particolare il sistema registra gli esami sostenuti dagli studenti.

Per ogni studente si memorizzano: il nome, il cognome, la data di nascita, la matricola, la data di immatricolazione e la data di laurea. Per ogni insegnamento si memorizzano: la denominazione, l'anno accademico di erogazione, l'anno di corso (I, II o III) e il docente.

Si vuole tener traccia nella base di dati per ogni studente:

- degli insegnamenti frequentati indicando la percentuale di presenza (>0) di uno studente alle lezioni e
- degli esami sostenuti indicando la data dell'esame e il voto finale.

Si vuole inoltre poter calcolare quanti studenti hanno frequentato un insegnamento.

<p><i>PROPOSTA MIA:</i></p> <p>Problema: uno studente potrebbe non essere laureato...</p>	<p><i>SOLUZIONE:</i></p> <p>- Se avessi il voto lo metterei sulla relazione stud-laurea) - Anche la soluzione con attributo docente è ok)</p>

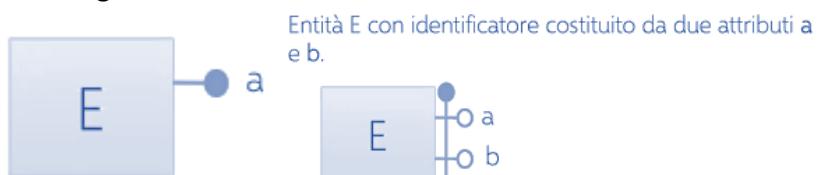
13/10/2020

IDENTIFICATORE DI IDENTITÀ

» Semantica

Insieme di proprietà che identificano univocamente le istanze dell'entità; non ci sono due esemplari che hanno gli stessi valori di queste proprietà.

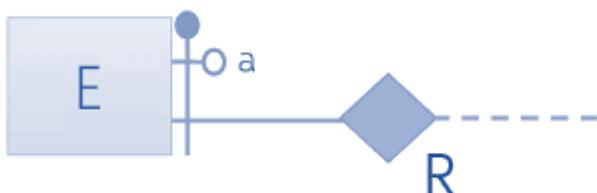
» Sintassi grafica



Trattino= a e b sono identificatori insieme

Pallino pieno su ciascuno=ciascuno è un identificatore univoco a sé

Se una relazione e un attributo fanno entrambi parte:



Esterni: comprendono almeno una relazione

[!!!] OSSERVAZIONE IMPORTANTE

Non è possibile specificare un identificatore sulle relazioni! Ogni istanza di relazione è identificata dalla ennupla di istanze di entità che la definisce.

SE SBAGLI QUESTO SEI BOCCIA

La proprietà di identificazione vale sempre e solo sulle entità, MAI sulle relazioni

ESEMPIO

Esempio di ENTITA'

Rappresento con il costrutto entità il concetto di PERSONA, ipotizzando che nei requisiti sia indicata la necessità di gestire nella base di dati, il codice fiscale, il nome, il cognome e la data di nascita di un gruppo di persone.

Quali identificatori possono essere definiti sull'entità PERSONA?

OSSERVAZIONE IMPORTANTE

L'identificatore di un'entità rappresenta allo stesso tempo:

- Un vincolo di identificazione: limita la popolazione di un'entità
- Una regola di corrispondenza tra le istanze dell'entità e gli oggetti del sistema informativo.

SCHEMA



CONSEGUENZA

La scelta dell'identificatore va sempre fatta considerando proprietà significative per il sistema informativo.

A livello concettuale l'introduzione di nuovi attributi identificatori (ad esempio l'ID) è da evitare.

Istanze o occorrenze: come si rappresentano le istanze, ovvero devo precisare che cosa conterrà la base di dati.

VINCOLI DI CARDINALITÀ:

» Semantica:

Data una relazione R, i vincoli di cardinalità specificano regole su **numero minimo** e **numero massimo** di istanze della relazione R a cui una istanza di E deve o può partecipare.

> Valori per minimo:

- 0: indica che una istanza può anche non partecipare alla relazione. La partecipazione è *opzionale*.
- 1: la partecipazione è *obbligatoria*.
- $n > 1$: l'istanza di entità deve partecipare almeno un certo numero di istanze della relazione.

> Valori per massimo:

- 1: un'istanza di E può al massimo partecipare a una occorrenza della relazione. Se R è binaria, questo indica che R è una funzione.
- 'N': indica che un'istanza può partecipare a un numero illimitato di occorrenze
- $n > 1$: indica che per ogni istanza possono essere presentati al più n occorrenze della relazione R che la coinvolgono -> es. numero massimo di esami da registrare. Però poi non ci possono essere eccezioni, quindi pessima idea.

» Sintassi grafica:



Esempio:



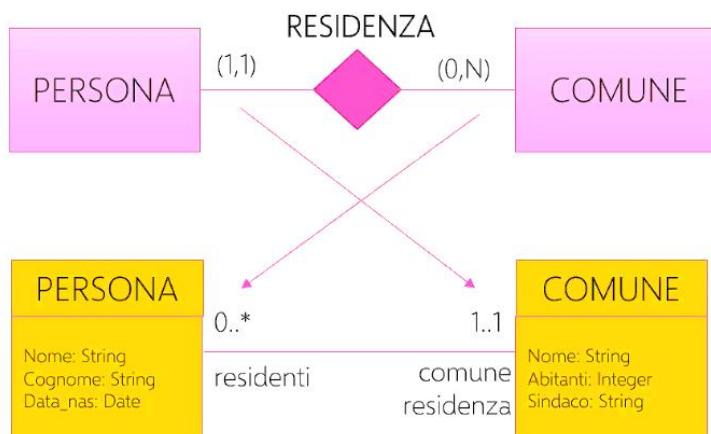
! Preciso il numero minimo o massimo sulla linea spezzata che collega ogni entità alla relazione. Per ogni entità di E si specifica una coppia di simboli (min,max)

- Una persona può avere al massimo una residenza (min 1), e deve averne esattamente una (max)

- 1). In ogni istanza di persona DEVE esserci una residenza.
- Dato il comune di verona,immaginando che esso possa nascere vuoto, è concesso che possa non avere alcuna residenza registrata all'inizio. Poi, ovviamente, non ha limiti.

!!!! Attenzione: è diverso da UML !!!

La posizione in cui metto il vincolo è opposta:



ATTRIBUTO OPZIONALE E MULTIVALORE

Ricordiamo che l'attributo associa uno e un solo valore

» Semantica:

L'attributo opzionale e /o multivalore si ottiene da un attributo normale specificando un vincolo di cardinalità sui valori che l'attributo può assumere. Il default è 1..1.

- **0..1**: attributo *opzionale* (come una patente di guida)
- **1..N**: attributo *multivalore* (come recapiti telefonici multipli)
- **0..N**: attributo *opzionale e multivalore*

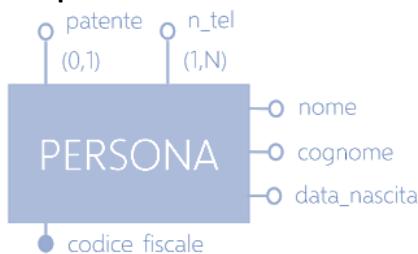
Lo uso solo in *situazioni particolari* dove devo elencare molti valori. Noi non usiamolo sulle relazioni, solo sulle entità talvolta :))

» Sintassi grafica:

Entità E con attributo a MULTIVALORE. Entità E con attributo a OPZIONALE



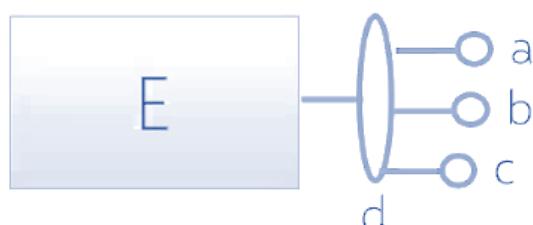
» Esempio



ATTRIBUTO COMPOSTO

Meglio usarlo poco, a lui non piace :)

» **Semantica:** mette insieme e raggruppa un sottoinsieme di attributi che hanno relazione semantica fra loro. Vuole solo rendere più chiaro lo schema.



Esempio

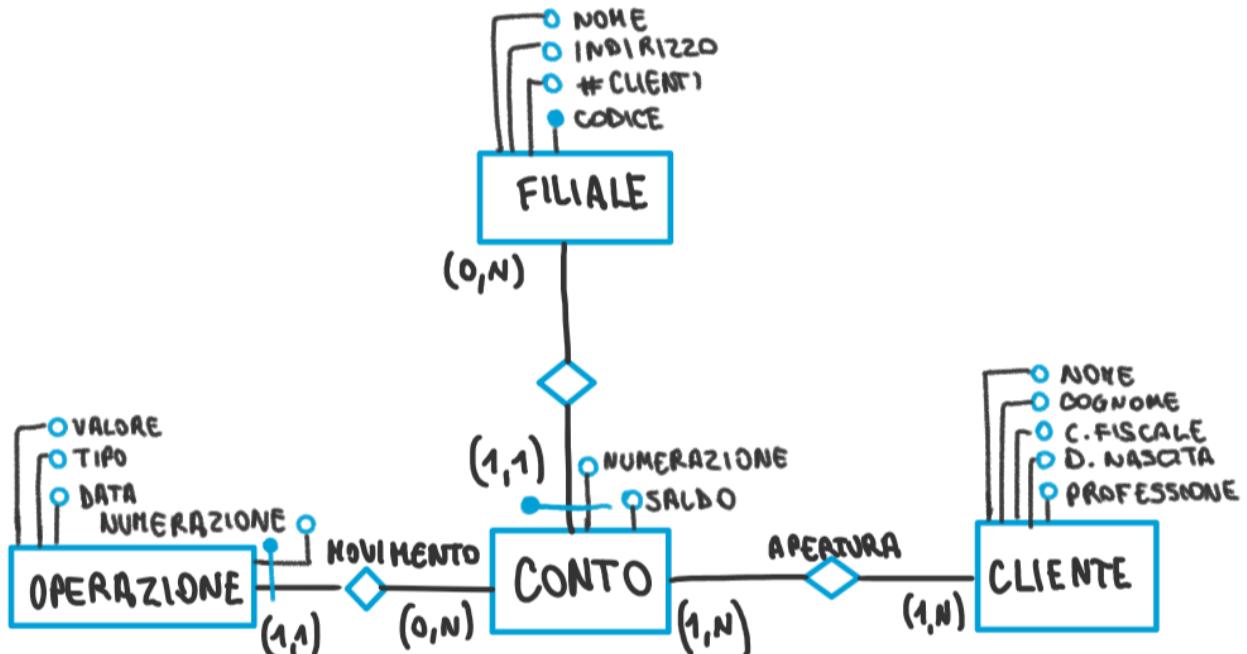


Esempio ER.2

REQUISITI

Si vuole progettare la basi di dati di una banca. La banca è suddivisa in filiali e di ogni filiale si conosce il nome, il codice numerico (univoco), l'indirizzo e il numero di clienti. I clienti della banca sono memorizzati nella base di dati. Per ogni cliente sono memorizzati: il nome, il cognome, il codice fiscale, la data di nascita e la professione. La base di dati contiene inoltre i dati sui conti correnti aperti presso le filiali della banca. Ogni conto corrente viene aperto presso una e una sola filiale e ha una numerazione univoca nella filiale. Ogni cliente può aprire più conti (anche presso la stessa filiale) e un conto può avere più intestatari. Si vuole mantenere traccia nella base di dati di tutte le operazioni (movimenti) eseguite sui conti correnti, indicando per ogni movimento, un numero progressivo univoco per conto corrente, la data, il tipo di operazione e il valore. Per ogni conto corrente si deve produrre il saldo. Si supponga che un cliente possa aprire conti diversi in più filiali della banca.

Proposta mia:



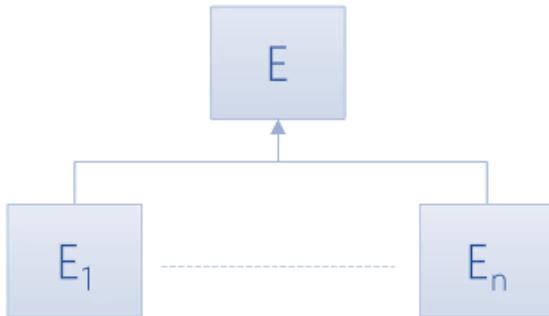
!!!! SE UNA RELAZIONE È IDENTIFICATORE DEVE ESSERCI (1,1) !!!!!

Se non è così allora è un errore grave. Bocciata. Boom.

PS La mia proposta era giusta :))

GENERALIZZAZIONE DI ENTITÀ

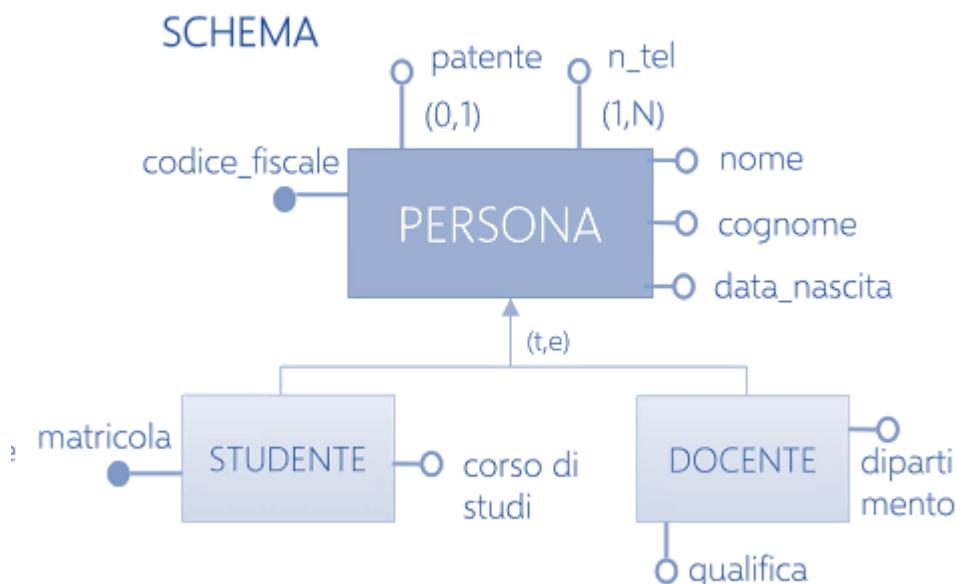
- » **Semantica:** è un legame logico simile all'ereditarietà fra classi, fra un'entità padre E e n entità figlie.
- E rappresenta una classe di oggetti più generale rispetto alle classi di oggetti figlie.
- » **Sintassi grafica:** usiamo una frecciolina



Note su come la generalizzazione influenza le relazioni con le altre entità:

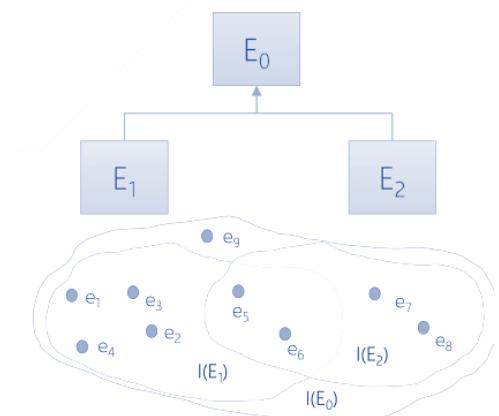
- > Ogni istanza di un'entità figlia E è anche istanza dell'entità padre E.
- > Ogni proprietà di attributi dell'entità padre E è anche proprietà di ogni istanza delle entità figlie.

Esempio



Ulteriori caratterizzazioni: *CLASSIFICAZIONE DELLE GENERALIZZAZIONI*

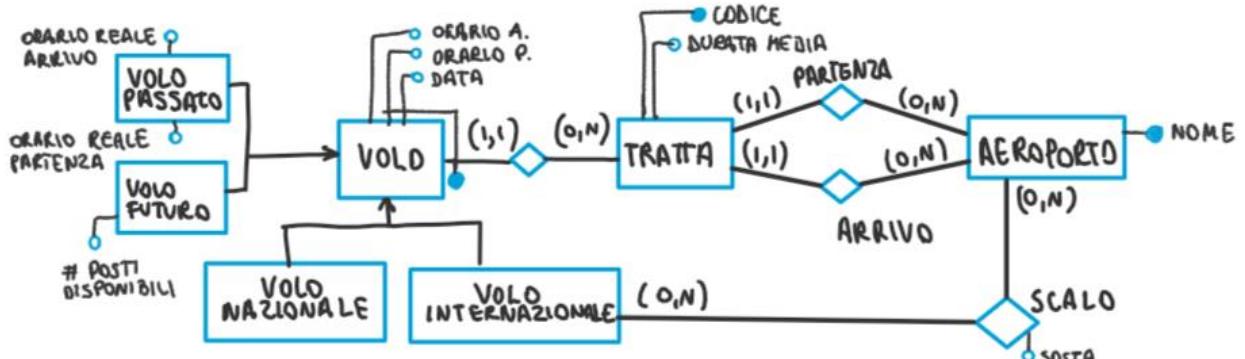
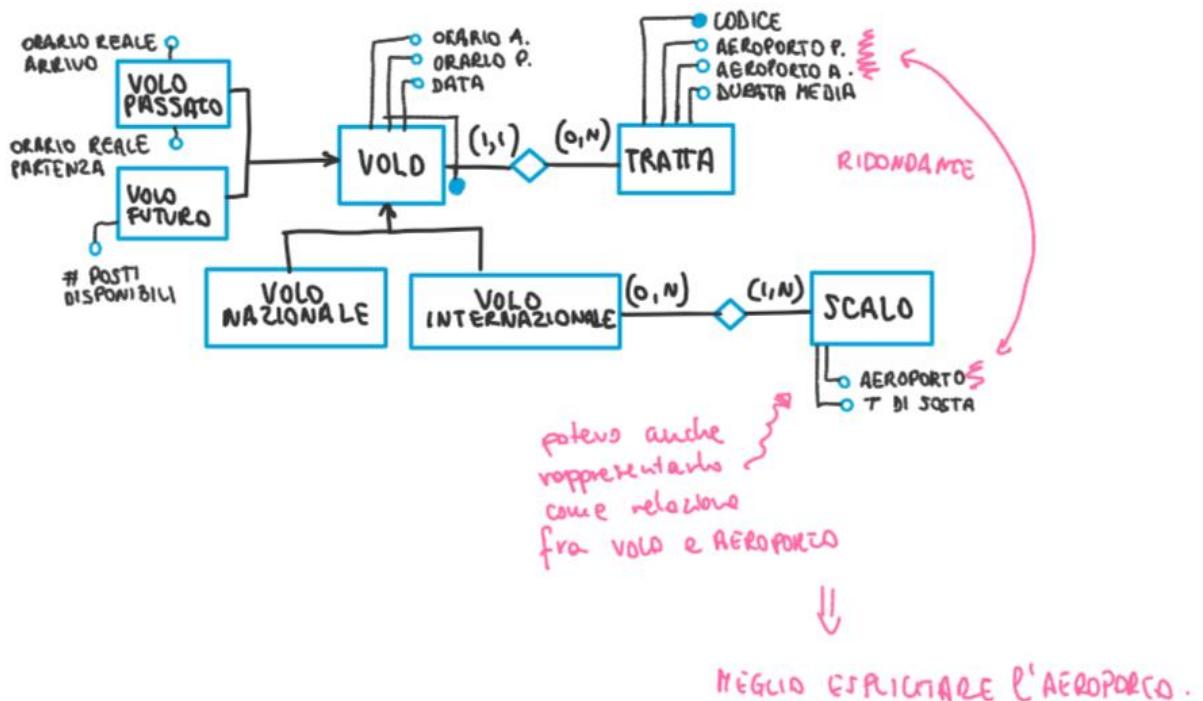
- » Una generalizzazione si dice **totale** se *ogni istanza de padre E è anche istanza di un'entità figlia Ei*. Altrimenti è **parziale**
- » Una generalizzazione è **esclusiva** se *ogni istanza dell'entità padre è istanza al più di un'entità figlia Ei*; altrimenti si dice **sovrapposta**.



Esempio ER.3

REQUISITI

Una compagnia aerea offre voli per diverse destinazioni. Ogni volo si riferisce ad una e una sola tratta ed è caratterizzato da una data, un orario di partenza e un orario di arrivo. Ogni tratta è univocamente identificata da un codice numerico ed è caratterizzata da un aeroporto di partenza, un aeroporto di arrivo e dalla durata media del volo. Ci sono voli nazionali e internazionali. I voli internazionali possono avere zero, uno o più scali. Per ogni scalo si memorizza l'aeroporto e il tempo di sosta dell'aereo. Dei voli passati si vuole inoltre memorizzare l'orario reale di partenza e l'orario reale di arrivo; per i voli futuri si è interessati al numero di posti disponibili.



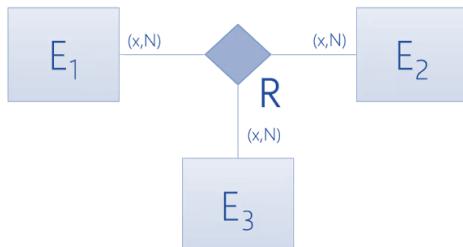
RELAZIONI TERNARIE

» **Semantica:** rappresento un'istanza del concetto che per esistere nel sistema informativo richiede la presenza di tre istanze di entità.

> !!! *Senza eccezioni*

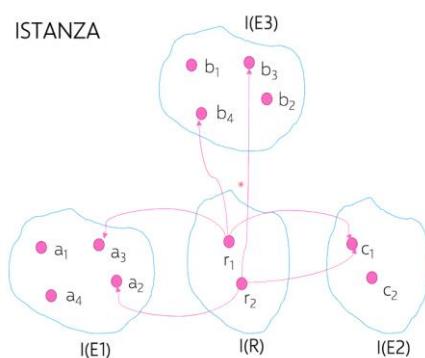
> !!! *Senza situazioni nelle quali la terna di entità debba essere rappresentata più volte.*

» **Sintassi grafica:**



» **Istanza:**

La relazione istanziale proprie occorrenze specificando terne.

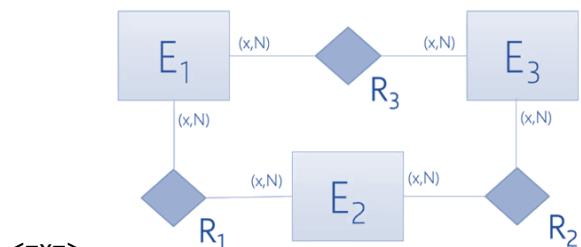
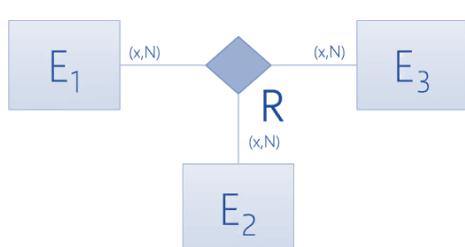


INDIZI DI MODELLAZIONE ERRATA

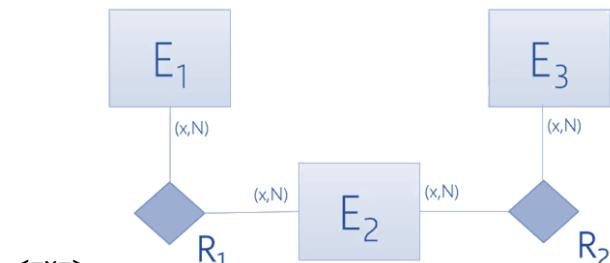
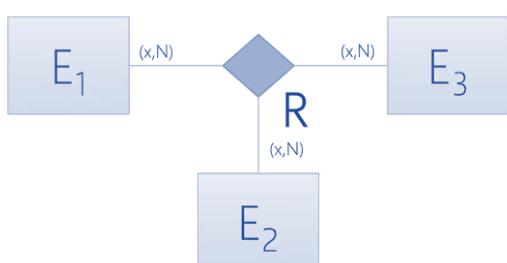
(ovvero cose che mi suggeriscono che ho fatto la scelta sbagliata)

» **Se in alcuni casi mi basterebbe solo una coppia, nope!**

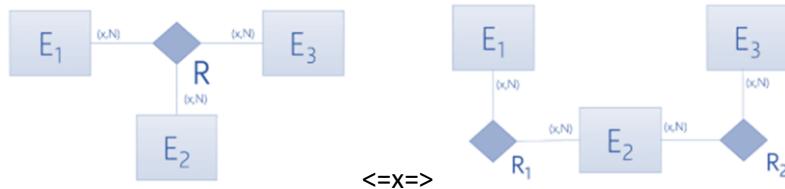
Ad esempio, se nascono legami che rappresentano legami che hanno parti fra loro autonome e che potrebbero essere rappresentate solo nelle coppie. Allora posso trasformare il tutto in *tre relazioni separate!!*



» Se la terza componente può essere derivata dalle altre due, nope!



» Se la relazione ternaria è l'unione di due relazioni binarie indipendenti, nope!

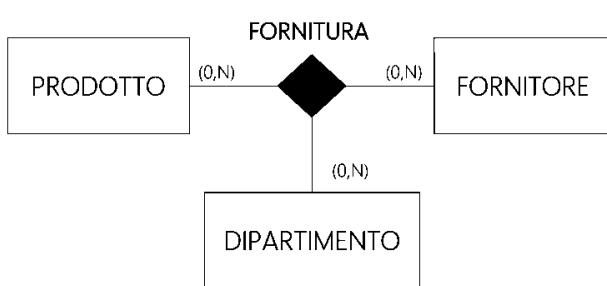


la relazione ternaria è l'unione di due relazioni binarie indipendenti

$$\begin{aligned} I(R) &= \{(a,b,c), (a',b,c'), \dots\} \\ I(R1) &= \{(a,b), (a',b), \dots\} \\ I(R2) &= \{(b,c), (b,c'), \dots\} \\ I(R1 \circ R2) &= \{(a,b,c), (a,b,c'), (a',b,c), (a',b,c')\} \end{aligned}$$

Esempio corretto:

Vogliamo rappresentare il fatto che un prodotto venga fornito ad un dipartimento da un fornitore. Ogni dipartimento può scegliere quali prodotti ordinare dai fornitori selezionando un sottoinsieme dei prodotti forniti dal fornitore. Ogni fornitore vende a più dipartimenti prodotti diversi. Ogni prodotto viene venduto da più fornitori.



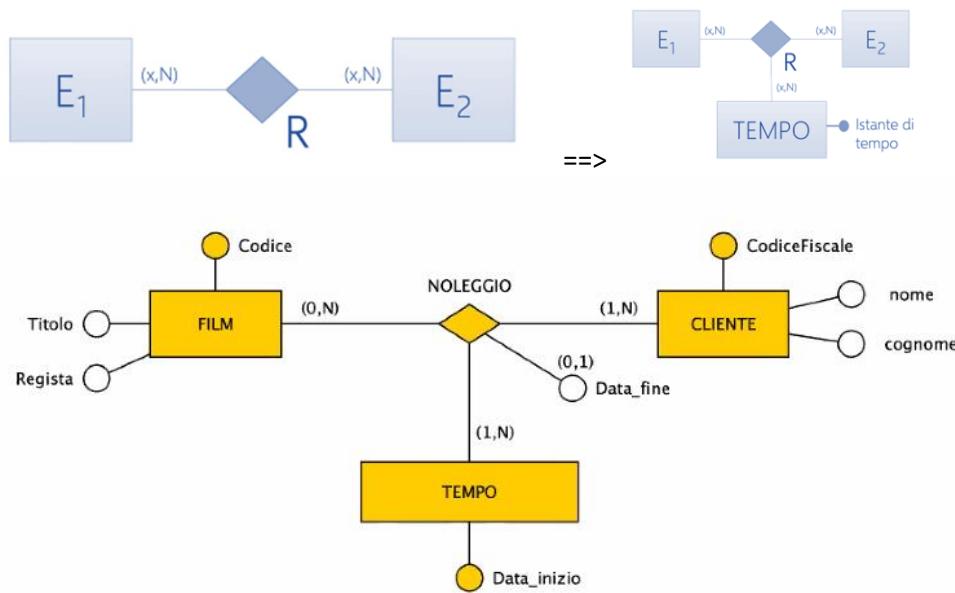
ISTANZA

$$\begin{aligned} I(FORNITORE) &= \{ f1, f2, f3 \} \\ I(PRODOTTO) &= \{ p1, p2, p3 \} \\ I(DIPARTIMENTO) &= \{ d1, d2 \} \\ I(FORNITURA) &= \{ (f1,d1,p1), (f1,d2,p3), (f3,d1,p3), (f3,d2,p2) \} \end{aligned}$$

USO STORICIZZANTE

Nella rappresentazione di una relazione binaria, possiamo aggiungere un terzo attore alla relazione per rappresentare più volte la stessa coppia in tempi diversi. Pertanto, la relazione binaria viene trasformata in una ternaria includendo l'entità tempo come nuovo attore della relazione, in modo da poter rappresentare più volte la stessa coppia in istanti di tempo diverso.

Non suffice aggiungere un attributo data su R; in questo modo non potrei avere più istanze della stessa coppia.



STRATEGIE DI PROGETTAZIONE

Sono sostanzialmente quelle di Ingegneria del software.

» Top down

L'idea è di partire dall'alto, in astratto, e pian piano raffinare il prodotto intermedio per arrivare al prodotto completo dopo una serie di passaggi.

» Bottom-up (*consigliata*)

- > Decompongo i requisiti in sezioni che descrivono una parte, e analizzo la descrizione al fine di identificare le parti.
- > Genero gli schemi per ciascuna delle parti che ho identificato
- > Metto insieme gli schemi che ho prodotto introducendo relazioni o ristrutturando lo schema per aggiungere quanto manca.

» Inside-out (*FORTEMENTE consigliata*)

L'idea è quella di analizzare i requisiti per capire quali siano i concetti più importanti

- > Individuo i concetti guida e ne faccio uno schema
- > Fondo gli schemi e genero lo schema finale

ANALISI DI QUALITÀ

Posso fare verifiche di:

» Verifica di correttezza

Uno schema concettuale è corretto se *utilizza propriamente i costrutti*.

Possibili errori:

- > *Errori sintattici*: si verificano quando non seguo le regole dei costrutti. Ad esempio:
 - ~ Generalizzazione delle relazioni
 - ~ Relazioni non collegate ad entità
 - ~ Identificatori sulle relazioni
- > *Errori semanticici*: sono un po' più complessi da rintracciare. Si verificano quando si utilizzano i costrutti senza rispettare la loro definizione.
 - ~ Uso di una relazione per rappresentare un concetto che ha esistenza autonoma e identificazione univoca
 - ~ Uso di un attributo per rappresentare un concetto che dovrebbe avere una struttura propria

» Verifica di completezza

Devo essere certa di *non aver dimenticato pezzi di requisiti*. Uno schema è completo se tutte le informazioni di interesse e tutte le operazioni descritte nelle specifiche possono essere eseguite a partire dai dati dello schema.

» Verifica di minimalità

Uno schema è minimale quando *tutti i concetti descritti sono rappresentati nello schema una volta sola*. La minimalità porta a un'**analisi di ridondanza**, sulla quale possiamo comunque passare oltre in certi casi.

- > Nel passato la ridondanza era un vero problema, poiché rischia di creare inconsistenze (più che per lo spazio). Le nuove tecnologie hanno tolto enfasi a questo problema in quanto *evitare la ridondanza è costoso*: la ridondanza può aiutare enella performance.

Casi di ridondanza probabile (= è possibile; devo fare un'analisi per capire se c'è o meno)

- > Ereditarietà
- > Cicli di relazione

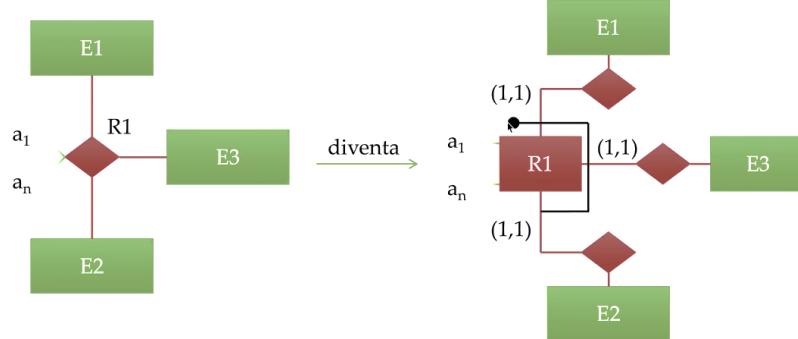


» Verifica di leggibilità

Verifico se lo schema presenta delle caratteristiche leggibili in modo naturale e facilmente comprensibile.

ESEMPI:

- > *Eliminazione delle relazioni teranarie a tre binarie*



- > *Eliminazione delle relazioni ternarie a due binarie*

Posso farlo SOLO quando una delle tre partecipa a una e una sola, e dunque che quella terna può essere vista come un legame con una e una sola terna. Dunque la ternaria non serve; sono due legami.



- > *Esplicitazione delle rgeneralizzazioni sovrapposte.*

IMPLEMENTAZIONE DEI DATI

Sisemi e modelli per i dati

- » Modello **gerarchico reticolare** (vecchio)
- » Modello **relazionale** (anni 80-oggi)
- » Modello ad **oggetti** (anni 90) -> poco successo
- » Modello oggetti relazionale o **object-relational** (anni 2000->oggi)
 - > SQL3: è un buon compromesso perché non ha obbligato i database vecchi a cambiare completamente
- » Modello a **grafo RDF** (2005-oggi)
 - > Neo4j
- » Modelli **noSQL**: vogliono allontanarsi radicalmente dal modello gerarchico, e anziché basarsi su SQL possono basarsi su molte altre tecnologie: document based, key-value, etc.
 - > L'obiettivo principale è quello di **gestire il dato enorme**. MongoDB, etc
 - > Sono rivolti a **contesti nuovi** dove i vecchi modelli diventano problematici: es. i dati devono nascere da sensori o dalla rete, dalla quale estrarre indicazioni o analisi

I relazionali hanno la caratteristica di essere **dichiarativi**: questa è la chiave che distingue il modello relazionale.

MODELLO RELAZIONALE

Anche qui avremo una carrellata di costrutti che descrivono proprietà e struttura dei dati. Sono pochi e molto facili. (*speròm!*)

DOMINI DI BASE

Sono i **valori da cui si scelgono i valori delle proprietà delle istanze di informazioni**. Sono le istanze di valori fra cui io scelgo gli elementi per rappresentare le mie proprietà.

Esempi tipici di domini:

- » Caratteri
- » Stringhe di caratteri
- » Numeri interi
- » Numeri decimali a virgola fissa o mobile
- » ...

RELAZIONE (O TABELLA)

PRESENTAZIONE INTUITIVA

Una relazione può essere vista come una **tabella**. Una tabella è un **contenitore di dati la cui struttura è caratterizzata da una lista di colonne**.

La domanda "cos'è una relazione" E' AMBIGUA!!!! Lui specificherà se si riferisce al modello relazionale o alla relazione del modello ER.

- » I dati sono scritti nelle righe, dove ogni riga descrive le caratteristiche di un'istanza dell'informazione
- » I valori contenuti nelle colonne descrivono sempre la stessa proprietà delle istanze di informazione da rappresentare.

MILANO	20100	1.300.000
VERONA	37100	350.000
BRESCIA	25100	250.000

DEFINIZIONE COME INSIEME DI ENNUPLE (LISTE)

Dati n insiemi di valori (domini) D_1, \dots, D_n con $n > 0$ e indicato con $D_1 \times \dots \times D_n$ il loro prodotto cartesiano:

$$D_1 \times \dots \times D_n = \{(v_1, \dots, v_n) | v_1 \in D_1 \wedge \dots \wedge v_n \in D_n\}$$

una relazione ρ di grado n è un qualsiasi sottoinsieme di $D_1 \times \dots \times D_n$:

$$\rho \subseteq D_1 \times \dots \times D_n$$

dove: (v_1, \dots, v_n) è una ennupla della relazione e $|\rho|$ è la cardinalità della relazione (numero di ennupla)

Si nota che:

- » I **domini** $D_1 \dots D_n$ possono essere a **cardinalità infinita**, mentre le **relazioni** sono sempre a **cardinalità finita** perché devono essere rappresentate fisicamente a sistema. :)
- » Non è definito **nessun ordinamento** sulle ennupla di una relazione, ovvero la relazione è un **insieme** di ennupla elencate con un **ordine qualsiasi**. Se vorrò elencare le ennupla, saranno elencate con un qualche ordine ma esso non avrà importanza né significato.
- » Trattandosi di un insieme, **non possono esserci duplicati** di una ennupla.
- » Considerando la singola ennupla $D_1 \dots D_n$, i valori sono **ordinati (internamente)**.

Esempio di insieme di ennupla:

$$\rho = \{ (\text{MILANO}, 20100, 1.300.000), (\text{VERONA}, 37100, 350.000), \\ (\text{BRESCIA}, 25100, 250.000) \}$$

$$\rho \subseteq D_1 \times D_2 \times D_3$$

D_1 = Stringhe di caratteri

D_2 = Numeri interi

D_3 = Numeri interi

Accesso ai valori di una ennupla:

Se t è una ennupla (v_1, \dots, v_n) il valore posto in i -esima posizione si indica con la notazione:

$$t[i]$$

Questo modo di lavorare sulla base di dati per estrarre informazioni è **poco efficace**, in quanto devo sapere quale proprietà è in quale posizione, e questo non è banale se si hanno tabelle con molte colonne.

Quindi si preferisce assegnare un nome alle colonne

-> *Hence, intruduciamo le tuple!*

DEFINIZIONE COME INSIEME DI TUPLE (MAPPINGS)

Arriva allo stesso risultato ma è somewhat più agevole.

Sia X un insieme di nomi e sia Δ l'insieme di tutti i domini di base ammessi dal modello.
Si definisce la funzione:

$$\text{DOM}: X \rightarrow \Delta$$

Che associa ad ogni nome A di X un dominio $\text{DOM}(A)$ di Δ . I nomi di X si dicono **attributi**.

Una tupla t su X è una funzione: $t : X \rightarrow \bigcup_{A \in X} \text{DOM}(A)$
dove: $t[A] = v \in \text{DOM}(A)$

Una relazione su X è un insieme di tuple su X , dove X è l'insieme di attributi della relazione.

Si noti che:

- » Le relazioni sono piatte: non sono ammessi domini più complicati o liste di valori.
- » Non possono esserci tuple duplicate
- » In generale la base di dati non è fattata da una tabella, ma da più tabelle/relazioni.

Esempio:

Relazione delle città:

$$X = \{\text{Nome}, \text{CAP}, \text{Abitanti}\}$$

$\text{DOM}(\text{Nome})$ = Stringhe di caratteri

$\text{DOM}(\text{CAP})$ = Numeri interi

$\text{DOM}(\text{Abitanti})$ = Numeri interi

$$\rho_X = \{ t_1, t_2, t_3 \}$$

$t_1[\text{Nome}] = \text{MILANO}$, $t_2[\text{Nome}] = \text{VERONA}$, $t_3[\text{Nome}] = \text{BRESCIA}$

$t_1[\text{CAP}] = 20100$, $t_2[\text{CAP}] = 37100$, $t_3[\text{CAP}] = 25100$

$t_1[\text{Abitanti}] = 1.300.000$, $t_2[\text{Abitanti}] = 350.000$, $t_3[\text{Abitanti}] = 250.000$

PROGETTAZIONE DEI DATI NEL MODELLO RELAZIONALE

Qui dimentico del tutto il modello ER! Nel modello relazionale, lo schema è costituito da un **insieme di relazioni o tabelle**.

Ma come implemento i requisiti direttamente nello schema relazionale? Quante e quali relazioni definisco? E' fuori dal percorso della metodologia (noi partiremo sempre dalla progettazione concettuale).

MODELLO RELAZIONALE

Lo strumento formale per definire i requisiti direttamente in questo contesto sono le **dipendenze funzionali**: non possiamo introdurlle in quanto *sono ex-corso* ma esistono :) In realtà, comunque, la loro funzione è svolta dalla progettazione concettuale.

Quante e quali relazioni generare? Vediamo un esempio

REQUISITI

Vogliamo rappresentare in una base di dati relazionale le informazioni sulla proprietà degli appartamenti siti nel comune di Verona. Per ogni appartamento vogliamo memorizzare: il codice ecografico (univoco), la categoria catastale e l'indirizzo composto da: via, n.civico, subalterno. Per ogni proprietario si registra: il codice fiscale, il nome, il cognome e la data di nascita. Un appartamento può essere in comproprietà e un proprietario può possedere più appartamenti.

APPROCCIO NAIF: RELAZIONE UNICA

Categoria	Via	Civico	Sub	Codice ECO	Codice Fiscale	Nome	Cognome	Data Nas
A1	Roma	23	A	RMAXXX	RSSMRR75D	Mario	Rossi	1/1/1975
A1	Roma	23	A	RMAXXX	BNCMRA80F	Maria	Bianchi	10/9/1980
A3	Milano	9		MILYYY	BNCMRA80F	Maria	Bianchi	10/9/1980
A3	Milano	9		MILYYY	BNCPLO77A	Paolo	Bianchi	3/1/1977
A1	Torino	2	C	TORZZZ	BNCPLO77A	Paolo	Bianchi	3/1/1977
A2	Garibaldi	33		GARWWW	RSSMRR75D	Mario	Rossi	1/1/1975

!!! Le informazioni sono rappresentate più volte - ridondante!

- » Devo avere più righe se ho più proprietari per lo stesso appartamento.
- » Lo stesso proprietario compare più volte se possiede più appartamenti.
- » Non è possibile rappresentare un appartamento che non abbia un proprietario o viceversa

Informazioni su ripetute: rischio di avere inconsistenza (es. ho due valori di nata di nascita!!)

Potrebbe non essere un problema, soprattutto se ci mettiamo nella visione dei dati proposta dalle nuove tecnologie – ovvero dove l'eliminazione della ridondanza è un problema maggiore della ridondanza stessa.

Anomalia: modo preciso per definire la presenza di errori, ridondanza o eccessiva aggregazione.

La ridondanza provoca le seguenti anomalie:

- » **Anomalie di aggiornamento:** indicano che per aggiornare il valore di un attributo siamo costretti a modificare tale valore su più tuple -> sono "ammessi" valori inconsistenti

- » **Anomalia di inserimento:** ho aggregato troppi concetti insieme, perché per inserire una nuova tupla devo inserire valori sconosciuti (sostituibili da valori nulli) per gli attributi non disponibili.
- » **Anomalia di cancellazione:** per cancellare una tupla è necessario cancellare valori ancora validi o inserirli valori nulli (*ew*)

Insomma fa cagher.

Il criterio da applicare per la creazione del modello sarà *evitare le anomalie*. Vogliamo che le operazioni sulle strutture corrispondano alle operazioni di nascita e morte delle informazioni. Lavorando bene a livello concettuale posso tradurre quasi in automatico in un buon schema logico.

Per togliere le anomalie procediamo alla *decomposizione della relazione unica*.

DECOMPOSIZIONE DELLA RELAZIONE UNICA (RIMOZIONE DELLE ANOMALIE)

Decomposizione = separazione dei concetti indipendenti.

Diminuiamo la ridondanza descrivendo una volta sola ciascuna tupla (= la relazione è un insieme di tuple, il quale per definizione non può rappresentare più volte la stessa tupla).

- » *Decomposizione:*

Decomposizione 1

APPARTAMENTO

PROPRIETARIO

Categoria	Via	Civico	Sub	Codice ECO	Codice Fiscale	Nome	Cognome	Data Nas
A1	Roma	23	A	RMAXXX	RSSMRR75D	Mario	Rossi	1/1/1975
A3	Milano	9		MILYYY	BNCMRA80F	Maria	Bianchi	10/9/1980
A1	Torino	2	C	TORZZZ	BNCPL077A	Paolo	Bianchi	3/1/1977
A2	Garibaldi	33		GARWWW				

SONO SCOLLEGATE!! Non esistono i puntatori! Quindi, per collegare, identifichiamo alcuni attributi che hanno il ruolo di identificare.

- » *Tabella delle coppie:*

APPARTAMENTO

PROPRIETARIO

Categoria	Via	Civico	Sub	Codice ECO	Codice Fiscale	Nome	Cognome	Data Nas
A1	Roma	23	A	RMAXXX	RSSMRR75D	Mario	Rossi	1/1/1975
A3	Milano	9		MILYYY	BNCMRA80F	Maria	Bianchi	10/9/1980
A1	Torino	2	C	TORZZZ	BNCPL077A	Paolo	Bianchi	3/1/1977
A2	Garibaldi	33		GARWWW	RSSMRR75D	Mario	Rossi	1/1/1975

Come ridondanza utile replico codice ECO e fiscale per identificare il legame. Non è una vera ridondanza: indico solo il legame.

Ovviamente dovrò controllare che i valori messi in Proprietà devono essere esattamente quelli presenti in appartamento e proprietario.

> Tutte le istanze di appartamento e proprietario sono tuple singole in una tabella; la relazione fra essi sono una nuova tabella. Questo schema riduce al minimo le anomalie. *È un buon schema :*

Codice ECO	Codice Fiscale	PROPRIETÀ
RMAXXX	RSSMRR75D	
RMAXXX	BNCMRA80F	
MILYYY	BNCMRA80F	
MILYYY	BNCPL077A	
TORZZZ	BNCPL077A	

C'è doppia!

decomposto

PROPRIETÀ DEL MODELLO:

I legami tra relazioni si realizzano attraverso la *replicazione di un insieme di attributi*, dove il legame tra due tuple si intende stabilito quando esse presentano gli stessi valori negli attributi replicanti.

- » **Value based** -> non devo “aggiungere valori”
 - > Completa *indipendenza dall'implementazione dallo schema fisico*.
 - > Facilmente *trasferibile*: non devo materializzare legami nascosti!
 - > È rappresentato solo ciò che è *rilevante per il sistema informativo*.
 - > **Svantaggio:** non è a oggetti, quindi è difficile da mappare col linguaggio di programmazione
- » **Non prevede puntatori**

TERMINOLOGIA VARIA

Terminologia SCHEMA di una relazione

E' costituito dal nome della relazione e da un insieme di nomi per i suoi attributi:

$R(X)$ oppure $R(A_1, \dots, A_n)$ oppure $R(A_1: D_1, \dots, A_n: D_n)$

Terminologia ISTANZA di una relazione di schema R(X)

E' un insieme r di tuple su X.

$$r = \{t_1, \dots, t_k\}$$

Terminologia SCHEMA di una base di dati

E' costituito da un insieme di schemi di relazioni:

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

I nomi delle relazioni devono essere diversi fra di loro per poter essere identificati.

Terminologia ISTANZA di una base di dati di schema

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

E' costituito da un insieme di istanze delle relazioni $R_1(X_1), \dots, R_n(X_n)$:

$$db = \{ r_1, \dots, r_n \}$$

VALORI NULLI

Abbiamo detto che nella progettazione concettuale alcuni tributi possono non ricevere un valore significativo, oppure riceverlo con un ritardo rispetto all'istanziazione dell'entità. Pertanto abbiamo introdotto gli *attributi opzionali*.

La stessa cosa si ritrova nel *modello relazionale*. Considerata una tupla (=riga della tabella) la mancanza di valore si presenta nei seguenti casi:

- » **Il valore di un attributo A è inesistente per quella specifica istanza.**
 - > Esempio: # patente di guida in una persona spantata.
- » **Il valore di un attributo A è sconosciuto: esiste ma non lo conosco**
 - > Esempio: tutti hanno una data di nascita, ma ammetto che eso potrebbe essere sconosciuto alla base di dati.

Nel modello relazionale definito in maniera classica *non si distinguono i due casi*; si rappresentano tutti i casi mancanti con il valore nullo.

Valore nullo

*E' un valore speciale che indica la mancanza di un valore.
Può essere assunto da qualsiasi attributo in qualunque tupla.*

Ciascun attributo della tupla, dunque, potrà assumere il valore nullo oppure uno dei valori del dominio. Alla luce di questa informazione possiamo rivedere la definizione di tupla.

Definizione. Tupla su X con valori nulli

Una tupla su X è una funzione:

$$t : X \rightarrow \{NULL\} \cup (\bigcup_{A \in X} DOM(A))$$

dove:

$$t[A] = v \in DOM(A) \vee t[A] = NULL$$

ACCORTEZZE:

- » Si preferirebbe **evitare l'utilizzo del valore nullo**
 - > Nelle informazioni "nuove" (inteso come tipi di data moderni, come i big data *or smth*) è sempre più comune non avere uno schema conosciuto. Dunque, arriverei a mettere molti più attributi del dovuto con moltissimi valori nulli.
- » Gli attributi che vogliamo utilizzare per generare **legami non possono essere nulli**
 - > Impedirebbero la rappresentazione del legame.
- » Posso accettare valori nulli **solo in alcuni attributi**
 - > Non è possibile avere tuple di soli valori nulli

ESEMPIO: PROGETTAZIONE DIRETTAMENTE IN MODELLO RELAZIONALE.

Attenzione! Non è l'approccio che useremo di solito, ci viene utile solo in questo specifico contesto.

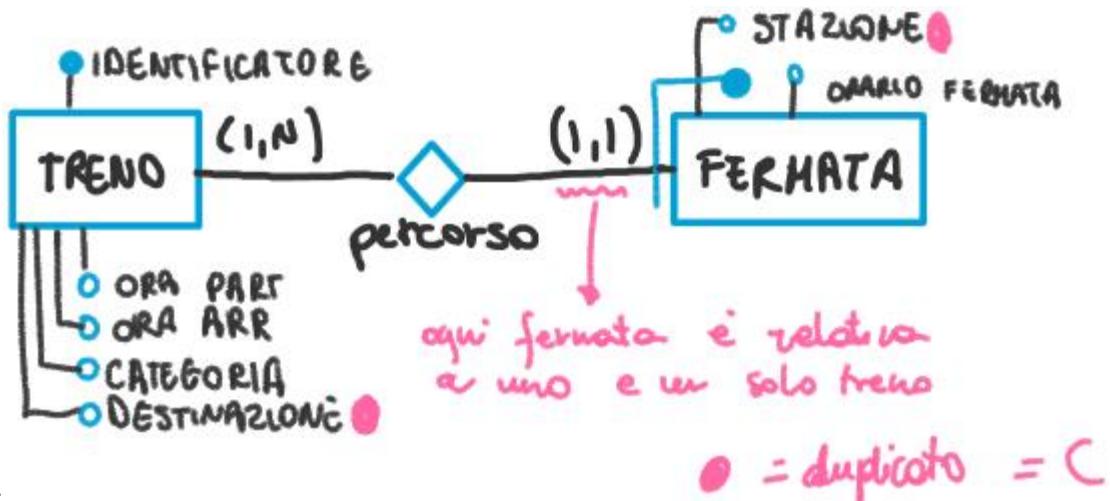
Rappresentare attraverso un insieme di relazioni le informazioni contenute in un orario ferroviario dove si riportino per ogni treno in partenza dalla stazione di Verona Porta Nuova: numero, orario di partenza destinazione finale, categoria, fermate intermedie, orario di arrivo.

Progettazione ER:

TRENO (Numero, Categoria, OraPart, Destinazione, OraArr)

FERMATA (NumTreno, Stazione, Orario)

Progettazione concettuale:



v.1:



v.2:

VINCOLI DI INTEGRITÀ

Spesso è necessario introdurre vincoli sulla popolazione di una base di dati. Allo scopo di indicare quali sono le istanze si introduce il concetto di integri di integrità:

Vincolo di Integrità

È una **condizione** espressa da un **predicato** che deve sempre essere soddisfatta da ciascuna istanza di una base di dati.

Riprendiamo l'esempio di prima e vediamo alcuni esempi di vincolo.

TRENO (Numero, OraPart, MinutoPart, Categoria, Destinazione,
OraArr, MinutoArr)

FERMATA (NumTreno, Stazione, OraFer, MinutoFer)

Quello che facciamo è scrivere condizioni e prediciati basati sulla logica.

$$\forall t, t' \in TRENO : t \neq t' \Rightarrow t[Numero] \neq t'[Numero]$$

$$\forall f \in FERMATA : \exists t \in TRENO : t[Numero] = f[NumTreno]$$

$$\forall t \in TRENO : t[Categoria] = 'regionale' \Rightarrow$$

$$\exists f \in FERMATA : f[NumTreno] = t[Numero]$$

I vincoli si dividono in:

» **Vincoli di dominio**

Impongono una *restrizione di dominio* sull'attributo.

$$\begin{aligned} & \forall t \in TRENO : t[OraPart] \in \{0, 1, \dots, 23\} \quad \forall t \in TRENO : t[MinutoPart] \in \{0, 1, \dots, 59\} \\ & \forall t \in TRENO : t[Numero] > 0 \end{aligned}$$

» **Vincoli di tupla**

Impongono una *restrizione sulle combinazioni*, ovvero riguardano *attributi multipli*

$$\forall t \in TRENO : t[Numero] > 5000 \Rightarrow t[Categoria] = 'regionale'$$

» **Vincoli intrarelazionali, "il piatto forte"! xD**

Impongono una restrizione al contenuto di una relazione e specificano una *condizione che ogni tupla della relazione deve soddisfare rispetto alle altre tuple della medesima relazione*.

Ne deriva che non possono essere verificati sulla singola tupla.

-> **Chiavi**

Sono proprietà di schema, che valgono in quanto **intrinsecamente parte del sistema informativo**.

$$\forall t \in r : t[K] \neq t'[K] \Rightarrow t[A_i] \neq t'[A_i]$$

Data una relazione R(X), un insieme di attributi K è superchiave di R(X) se per ogni istanza di R(X) gli attributi hanno valore univoco.. ovvero vale la condizione

$$\forall t, t' \in r : t \neq t' \Rightarrow t[K] \neq t'[K]$$

dove:

$$t[K] \neq t'[K] \equiv \exists A_i \in K : t[A_i] \neq t'[A_i]$$

Sostanzialmente è quella che identifica la relazione: se due tuple sono diverse, allora sugli

attributi che compaiono in K devono essere diversi. Dato che K è un insieme, esiste una i in K dove le due tuple risultano diverse.

- **Chiave candidata** (*E' una chiave che può essere considerata come possibile identificatore #1*)

K deve essere una *superchiave* e deve essere *minimale*, ovvero non deve esistere un K' sottoinsieme di K tale per cui K' è superchiave di R(X) -> K è una superchiave minimale candidata

$$\neg \exists K' \subset K : K' \text{ è superchiave per } R(X)$$

Possiamo dimostrare che esiste sempre una chiave candidata K per una relazione R(X).

DIM:

- X è superchiave per R(X)
- se X non ha sottoinsiemi propri che sono superchiave, allora X è chiave
- altrimenti si riapplica il medesimo ragionamento ricorsivamente al sottoinsieme di X che è superchiave
- poiché la cardinalità di X è finita e poiché l'insieme vuoto non è superchiave si troverà in un numero finito di passi una chiave per R(X)

- > **Chiave primaria** (*superchiave che decido di usare*).

E' la chiave che ho scelto per identificare le tuple della relazione.

- Non può contenere *valori nulli*
- E' quella su cui il sistema genera una *struttura d'accesso ai dati* per supportare le interrogazioni.

10/11/2020 [integrata ✓]

- > **Esempi**

TRENO (Numero, OraPart, MinutoPart, Categoria, Destinazione,
OraArr, MinutoArr)

- K1=[Numero]
 - ~ ✓ Superchiave
 - ~ ✓ Candidata
- K2=[Numero, categoria]
 - ~ ✓ Superchiave
 - ~ ✗ Candidata : Se prendo Numero da solo è superchiave. Quindi NON è chiave candidata-
- K3 = [OraPart, MinutoPart, Destinazione, Categoria]
 - ~ ✓ Superchiave: sì, sembra inverosimile che due treni siano uguali
 - ~ ✓ Candidata: sì, perché i sottoinsiemi da soli non bastano.

FERMATA (NumTreno, Stazione, OraFer, MinutoFer)

- K1 = {NumTreno} No.
 - ~ ✗ Superchiave
 - ~ ✗ Candidata
- K2 = {NumTreno, Stazione} : Se immagino che il treno non passa due volte nella stessa stazione, allora è superchiave e anche chiave candidata.
 - ~ ✓ Superchiave
 - ~ ✓ Candidata
- K3 = {OraFer, MinutoFer, Stazione} : Dipende dall'interpretazione. Es. se ho un binario unico tale epr cui non posso far fermare 2+ treni insieme, allora più essere superchiave.

» **Vincoli interrelazionali**

Impongono una restrizione al contenuto di una relazione e specificano una condizione che ogni tupla della relazione deve soddisfare rispetto alle tuple di ALTRE relazioni della DB.

->**Vincoli di integrità referenziale, o vincoli sulle chiavi esportate**

Ci aiutano a mantenere corretti ed integri i riferimenti tra tuple, che noi abbiamo rappresentato replicando porzione di informazione.

Vincolo di integrità referenziale.

*Un vincolo di integrità referenziale tra un insieme di attributi $Y=\{A_1 \dots A_p\}$ (soggetti al vincolo) e un insieme di attributi $K=\{K_1 \dots K_2\}$, **chiave primaria di un'altra relazione**, è soddisfatto se per ogni istanza r_1 di R_1 e per ogni istanza r_2 di R_2 vale la condizione*

$$\forall t \in r_1 : \exists s \in r_2 : \forall i \in \{1, \dots, p\} : t[A_i] = s[K_i]$$

Ovvero per ogni riga deve esistere una tupla s in R_2 tale che per ogni indice i considerato devo avere dentro R_2 una tupla s che presenta la combinazione di valori che appare in R_1 .

Se ammetto che possono esserci i valori nulli (ovvero il legame è opzionale), allora aggiungo un pezzetto:

$$\begin{aligned} \forall t \in r_1 : \exists s \in r_2 : (\forall i \in \{1, \dots, p\} : t[A_i] = s[K_i]) \vee \\ (\exists i \in \{1, \dots, p\} : t[A_i] = \text{NULL}) \end{aligned}$$

La chiave primaria di una relazione $R()$ viene indicata sottolineando gli attributi.

NOTAZIONE PER VINCOLI STRUTTURALI

Molto banalmente, si **sottolineano gli attributi che sono chiave primaria**. Tornando al nostro esempio:

TRENO(Numero, OraPart, MinutoPart, Categoria, Destinazione,

OraArr, MinutoArr)

FERMATA(NumTreno, Stazione, OraFer, MinutoFer)

La chiave primaria è una sola, quindi *tutti quelli che sottolineo saranno "uniti"*.

NOTAZIONE PER INTEGRITÀ REFERENZIALE

Quella che il prof consiglia è di riquadrare gli attributi soggetti al vincolo (= la chiave esportata) e collegarli con una freccia alla tabella (!!! Alla tabella! Non alla chiave primaria, basta la tabella!!!)

→ TRENO (Numero, OraPart, MinutoPart, Categoria, Destinazione,
OraArr, MinutoArr)

FERMATA (NumTreno, Stazione, OraFer, MinutoFer)

Esercizio 2.7

Sarà reverse engineering: data la tabella cerca chiavi primarie e vincoli di integrità referenziale. NOI PARTIREMO SEMPRE DAI REQUISITI IN COMPITO!!

- 2.7** Individuare le chiavi e i vincoli di integrità referenziale che sussistono nella base di dati di Figura 2.23 e che è ragionevole assumere siano soddisfatti da tutte le basi di dati sullo stesso schema. Individuare anche gli attributi sui quali possa essere sensato ammettere valori nulli.

PAZIENTI

Cod.	Cognome	Nome
A102	Necchi	Luca
B372	Rossini	Piero
B543	Missoni	Nadia
B444	Missoni	Luigi
S555	Rossetti	Gino

RICOVERI

Paz.	Inizio	Fine	Rep.
A102	2/05/11	9/05/11	A
A102	2/12/11	2/01/12	A
S555	25/04/11	3/05/11	B
B444	1/12/11	2/01/12	B
S555	5/10/11	1/11/11	A

MEDICI

Matr.	Cogn.	Nome	Rep.
203	Neri	Piero	A
574	Bisi	Mario	B
461	Bargio	Sergio	B
530	Belli	Nicola	C
405	Mizzi	Nicola	A
501	Monti	Mario	A

REPARTI

Cod.	Nome	Primario
A	Chirurgia	203
B	Pediatria	574
C	Medicina	530

La coppia paziente-reparto da sola non si può identificare!! Ci sono doppi!! Ma riesco se aggancio con la data.

→ PAZIENTI (Cod, Cognome, Nome)

→ MEDICI (Matr, Cogn, Nome, Rep)

→ REPARTI (Cod, Nome, Primario)

RICOVERI (Paz, Rep, Inizio, Fine)

PROGETTAZIONE LOGICA



L'obiettivo è **produrre uno schema logico che descrive in modo corretto ed efficace tutte le informazioni contenute nello schema concettuale**. Partiremo dalle caratteristiche del modello relazionale (domini di base, relazioni, chiavi primarie, vincoli di integrità) e le useremo per descrivere il nostro schema concettuale.

L'output sarà uno **schema relazionale di una DB**, aka un *insieme di tabella* $R_1(A_{1,1}, \dots, A_{1,n_1}), \dots, R_k(A_{k,1}, \dots, A_{k,n_k})$

La progettazione logica parte da uno schema concettuale e da un modello logico. Potrei avere info anche sul carico di lavoro (es. operazioni più frequenti), ma non è il nostro caso.

Dovrò fare una **ristrutturazione dello schema concettuale** per aggiungere informazioni necessarie alla traduzione. Sullo schema concettuale ristrutturato applicherò una serie di regole **di traduzione quasi automatica**.

FASE 1: RISTRUTTURAZIONE

Noi la finalizziamo alla traduzione dunque dedichiamo meno attenzione.

Motivazioni:

- » **Semplificare** la fase successiva di traduzione
- » **Ottimizzare le strutture rispetto al carico di lavoro.**
 - > *Dimensione* dei dati
 - > Caratteristiche delle operazioni: *costo e frequenza*.

Fasi:

1. **Analisi delle ridondanze dovute a presenza di dati derivabili**
2. **Eliminazione delle generalizzazioni:** ha solo lo scopo di semplificare la traduzione verso il modello relazionale.
3. **Accorpamenti o partizionamenti di entità e relazioni:** divido popolazioni di istanze perché necessario per le operazioni. Noi lo malcaghiamo perché non abbiamo il carico di lavoro!!!!
4. **Scelta degli identificatori da usare come chiave primaria**

1.1 ANALISI DELLE DERIVABILITÀ

Possono essere presenti dati derivabili da altri, ad esempio:

- » **Dalla stessa entità:** età e data di nascita.
- » **Da altre entità:** operazioni e saldo conto
- » **Dal conteggio di occorrenze:** numero di conti correnti intestati al cliente.
- » **Dalla composizione di altre relazioni** (*di solito non ci succede o comunque non lo guardiamo*)

Non è sempre necessario eliminarli ma magari può esserlo in base al costo delle operazioni. Comunque è bene discuterne. Devo decidere se :

- » **Lasciare il dato derivabile**
 - > E' vantaggioso se *costo calcolo > costo aggiornamento*
- » **Calcolare il dato derivabile quando serve**
 - > E' vantaggioso se *costo calcolo < costo aggiornamento* (es. mi serve molto spesso).

Chiaramente dipende dal carico di lavoro. Il costo si calcola in *numero di accessi su unità di tempo*, in quanto vengono considerate le frequenze di calcolo e di aggiornamento del dato.

1.2 ELIMINAZIONE DELLE GENERALIZZAZIONI

Sostituire le generalizzazioni è utile in quanto **non sono direttamente rappresentabili in modello relazionale**.

Ci sono tre soluzioni; con ciascuna opzione perdo e guadagno qualcosa, dunque devo scegliere di conseguenza.

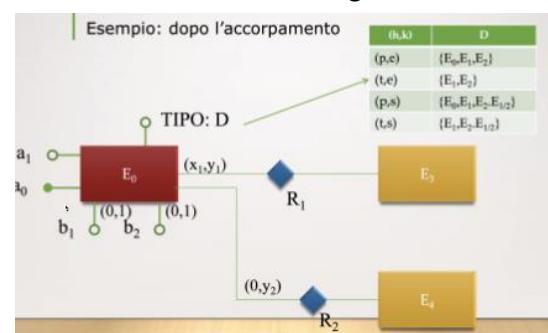
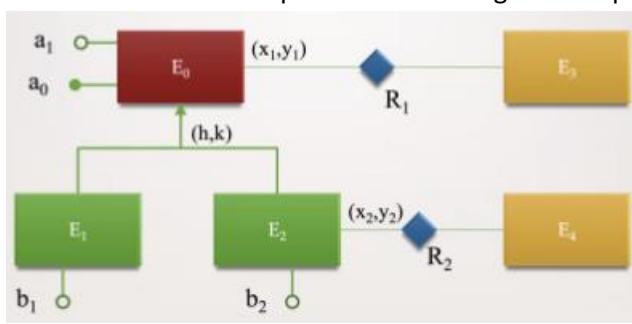
» ACCORPAMENTO DELLE ENTITÀ FIGLIE NEL PADRE

> Uso:

- Si preferisce questa applicazione quando *introduco pochi attributi opzionali*.
- E' *sempre possibile* utilizzarla

> Passi:

- Elimino entità figlie
- Accordo nel padre tutti gli attributi specifici delle entità figlie come attributi opzionali
- Accordo nel padre delle relazioni che coinvolgono le entità figlie assegnando cardinalità minima 0 (lato entità padre) perché diventano opzionali (= sarebbe solo per alcune sottoclassi, quindi unendole insieme è opzionale)
- Per mantenere la distinzione fra le istanze aggiunto un attributo TIPO che mi permette di distinguere nel padre le occorrenze delle entità figlie eliminate.



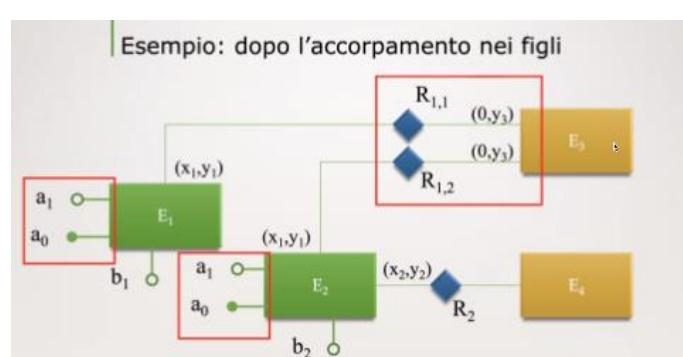
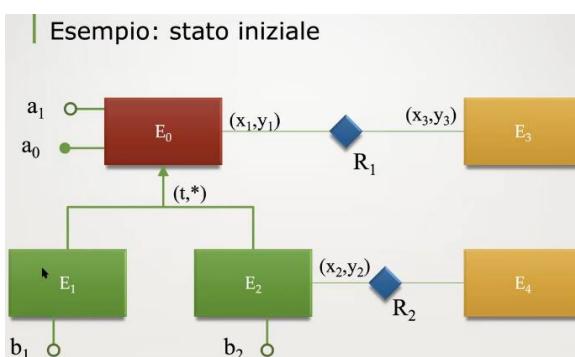
» ACCORPAMENTO DEL PADRE NELLE ENTITÀ FIGLIE

> Uso:

- Si preferisce questa applicazione quando *la maggior parte degli attributi sono sui figli*.
- **PRECONDIZIONE:** posso farlo solo se *tutte le istanze sono figlie*, ovvero la generalizzazione è TOTALE

> Passi:

- Eliminazione dell'entità padre
- Replicazione degli attributi dell'entità padre su ciascuna entità figlia
- Partizionamento sui figli delle relazioni che coinvolgono l'entità padre assegnando cardinalità minima uguale a zero (lato entità esterne).

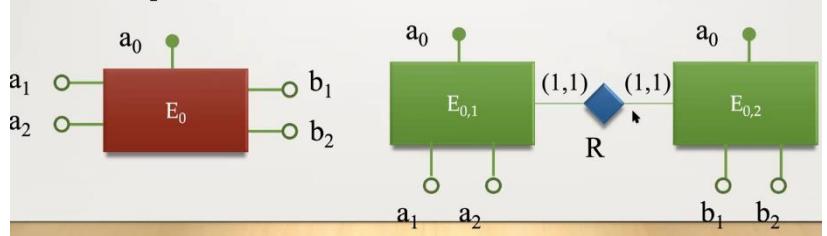
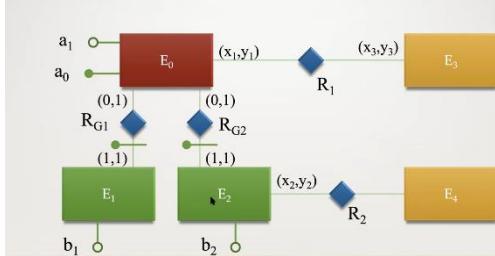


» SOSTITUZIONE DELLA GENERALIZZAZIONE CON RELAZIONI

> Uso:

- Voglio *Mantenere le popolazioni intatte*: utile quando voglio poter gestire la popolazione del padre ma anche quella dei figli, senza perderne nessuna: mantengo tutto.
 - Posso *sempre applicarlo*.
- > Passi:
- Eliminazione della generalizzazione
 - Inserisco n relazioni tra il padre e ciascuna delle n entità figlie
 - Tutti gli attributi conservano la loro collocazione.
 - *! Potrebbero essere aggiunti ulteriori per precisarla meglio (???)*

Esempio: stato iniziale

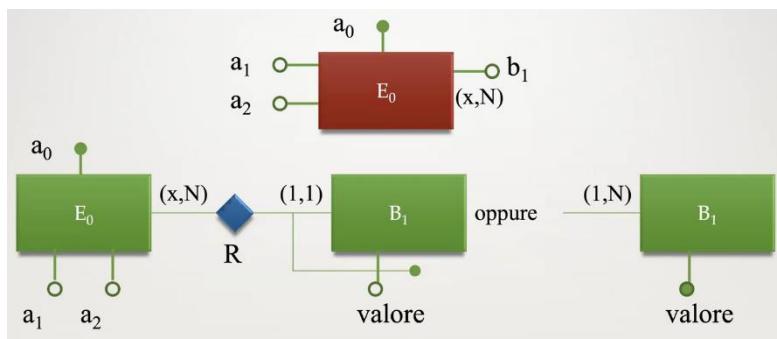


1.3 PARTIZIONAMENTO E ACCORPAMENTO DI RELAZIONI

(noi non lo applicheremo mai tranne che per eliminare gli attributi multivaleure!!)

Sostituisce un'entità dello schema con una **coppia di entità con lo stesso identificatore e una relazione uno a uno che le lega fra loro**.

Si esegue solo nel caso in cui esistano *operazioni frequenti* che trattando solo un sottoinsieme degli attributi dell'entità da partizionare.



L'unico caso in cui lo faremo è per **eliminare gli attributi multivaleure**, che *non sono rappresentabili* in una BD in modello relazionale.

B1 saranno, ad esempio, i numeri di telefono e valore sarà il numero effettivo.

Ho due possibilità:

- » Posso ipotizzare che i valori siano **assolutamente esclusivi**; allora li *collego a una e una sola istanza di E0*, e l'identificazione è il legame con E0 e il valore. Se ho dei numeri di telefono in comune li rappresento più volte.
- » Se invece voglio mantenere esplicita questa condivisione l'Identificatore va **solo sul valore e lo collego a 1,N**. E' più flessibile ma richiede una **tabella in più**.

1.3.1 PARTIZIONAMENTO DI RELAZIONI

Noi non ne vedremo. Si realizza sostituendo una relazione R con due relazioni R1 e R2, dove le occorrenze di R si partizionano fra le due. Conviene quando **esistono operazioni che si riferiscono a un sottoinsieme delle occorrenze di R e altre operazioni che si riferiscono alle altre**.

1.3 SCELTA DEGLI IDENTIFICATORI PRINCIPALI.

In generale ogni entità può avere più identificatori (*!!!! CE NE POSSONO ESSERE PiÙ DI UNO. NON ANDATE IN ESCANDESCENZA*). Poi, nella trasformazione, non vanno persi! Uno rimane identificatore e l'altro è un controllo di integrità.

Criteri:

- » Non voglio identificatori che includono **attributi opzionali**
- » Meglio avere **pochi attributi**
- » Meglio identificatori **usati nelle operazioni**

*!!! Si incappa se a livello **concettuale** parliamo di righe, attributi nulli e così via.*

REGOLE DI TRADUZIONE VERSO IL MODELLO RELAZIONALE

La traduzione è un processo automatico o quasi che si realizza applicando allo schema concettuale ristrutturato un insieme di regole di traduzione. Ogni regola si applica ad un costrutto del modello concettuale e produce una o più strutture del modello relazionale.

- » **Istanza di entità** -> *Tupla*
- » **Istanza di relazione** -> *legame fra tuple o tuple esplicite*
- » **L'identificatore principale** diventa la *chiave primaria*

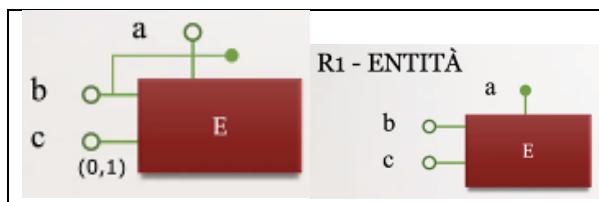
CLASSIFICAZIONE DELLE RELAZIONI BINARIE DEL MODELLO ER

Classifichiamo le relazioni binarie in tre categorie:

- » **Uno a uno** se entrambe le entità coinvolte nella relazione hanno cardinalità massima uguale a UNO.
- » **Uno a molti:** se una delle entità coinvolte nella relazione ha cardinalità massima uguale a uno e l'altra ha cardinalità massima > 1.
- » **Molti a molti:** entrambe le entità coinvolte hanno cardinalità massima > 1.

REGOLETTE

» Entità

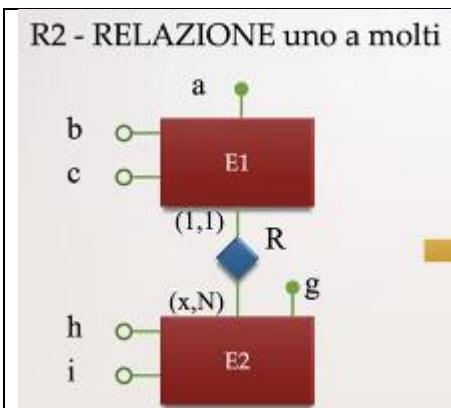


E(a, b, c) **E(a, b, c*)**

- » Nel secondo caso *sottolineo entrambi* perché è una chiave fatta da *due attributi*.
- » L'asterisco è la notazione che usiamo per indicare che l'attributo può contenere valori nulli.

» Relazioni

» **Uno a molti:**

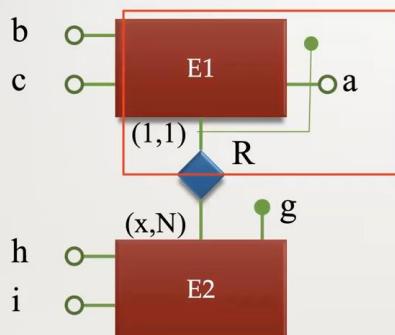


- » Aggiungo un attributo, che è il contenitore che nella tupla di E1 permette di memorizzare ???? sul fatto che l'attributo debba contenere esattamente una chiave esportata di E2. Posso chiamarlo come voglio!!
- » Aggiungo un vincolo di integrità referenziale che dice che G può contenere solo attributi presenti dentro E2
- » !! Rappresenta solo un verso!! Fa niente: rappresentarla twice significherebbe avere ridondanza. Ci sono operazioni che consentono di navigare queste relazioni.

E₁(a, b, c, g)

E₂(g, h, i)

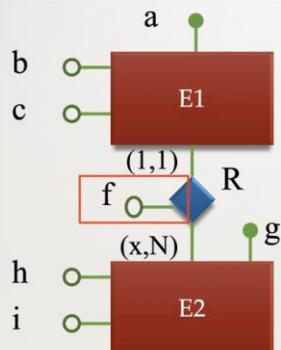
R2.a - RELAZIONE uno a molti
(identificatore esterno)



$E_1(a, g, b, c)$

$\rightarrow E_2(g, h, i)$

R2.b - RELAZIONE uno a molti
(con attributo sulla relazione)



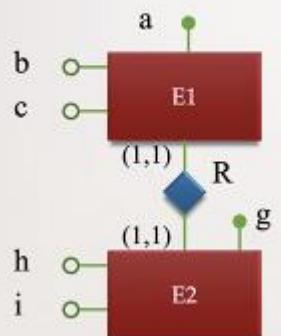
$E_1(a, b, c, g, f)$

$\rightarrow E_2(g, h, i)$

Dato che ho assorbito la relazione su E1, posso assorbire anche l'attributo.

> Uno a uno:

R3 - RELAZIONE uno a uno



Abbiamo diversi casi, che dipendono non solo dalla cardinalità massima ma anche della massima.

Posso assorbire la relazione in ciascuno dei due; rappresento la relazione esportando la chiave di uno sull'altro.

» Posso rappresentarlo su uno

$E_1(a, b, c, g)$

$\rightarrow E_2(g, h, i)$

Prima era obbligatorio, ora è una scelta.

» In alternativa:

$E_1(a, b, c, g)$ $E_1(a, b, c)$

R

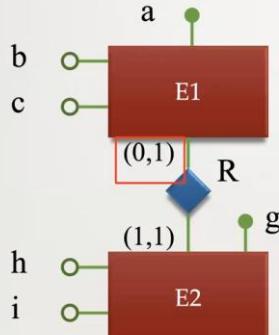
oppure

$E_1(a, b, c)$ $E_1(a, b, c, g)$

R

Quale scegliere dipende dalle operazioni e da quale dei due versi di navigazione del legame è più usato.
NO IN ENTRAMBI o avrei ridondanza!

R3.a - RELAZIONE uno a uno
(con una cardinalità minima = 0)

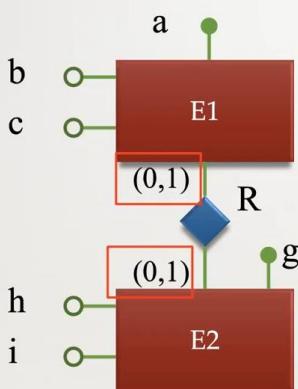


Qui è meglio assorbire la relazione su E2, dato che la prima può mancare!!

$E_1(a, b, c)$

$E_2(g, h, i, a)$

R3.b - RELAZIONE uno a uno
(con due cardinalità minime = 0)



$E_1(a, b, c, g^*) E_1(a, b, c)$

oppure

$E_2(g, h, i, a^*)$

Per eliminare completamente i valori nulli devo fare una relazione esterna. IN QUESTO CASO NON CI VA L'ASTERISCO

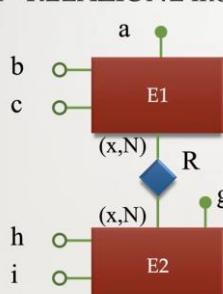
$R(a, g)$

$\rightarrow E_1(a, b, c) \quad E_2(g, h, i) \leftarrow$

Qui posso sfruttare che il vincolo di cardinalità massima è uno, quindi posso usare solo uno dei due come identificatore.

> Molti a molti:

R4 - RELAZIONE molti a molti



Qui non posso assorbire la relazione da nessuna parte. Quindi genero una struttura per contenere le istanze di relazione, che sono coppie di istanze di tuple, sempre realizzate esportando una coppia di chiavi.

$\rightarrow E_1(a, b, c)$

$R(a, g)$

$\rightarrow E_2(g, h, i)$

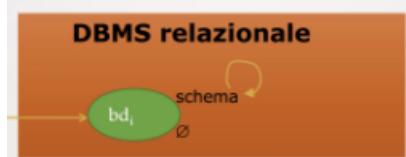
<p>R4.a - RELAZIONE molti a molti (con attributi)</p>	<p>Intuitiva</p> <p>$\rightarrow \mathbf{E}_1(\underline{a}, b, c)$</p> <p>$\rightarrow \mathbf{R}(\underline{a}, \underline{g}, \underline{f})$</p> <p>$\rightarrow \mathbf{E}_2(g, h, i)$</p>
<p>R4.a' - RELAZIONE molti a molti (con identificatori con più attrib.)</p>	<p>$\rightarrow \mathbf{E}_1(\underline{a}, b, c)$</p> <p>$\rightarrow \mathbf{R}(\underline{a}, \underline{b}, \underline{g}, f)$</p> <p>$\rightarrow \mathbf{E}_2(g, h, i)$</p>
<p>R4.b - RELAZIONE molti a molti (relazione ternaria)</p>	<p>$\rightarrow \mathbf{E}_1(\underline{a}, b, c)$</p> <p>$\rightarrow \mathbf{R}(\underline{a}, \underline{g}, \underline{s}, f)$</p> <p>$\rightarrow \mathbf{E}_2(g, h, i)$</p> <p>$\rightarrow \mathbf{E}_3(\underline{s}, t)$</p>

ALGEBRA RELAZIONALE

Esistono due tipi di linguaggi che permettono di interagire con un database:

» Data Definition Language (DDL)

Consente di
*creare e
modificare lo
schema* della
base di dati.
Creiamo le strutture vuote.



» Data Manipulation Language (DML)

Dopo che la BD
è creata,
permette di
*aggiornare i
valori e di
inserire o
cancellare tuple nelle relazioni.* Infine, permette di
interrogare le relazioni per estrarre informazione.



I linguaggi sono la parte più importante del sistema. Scelta una tecnologia, oltre ad analizzare il modello dei dati è essenziale studiare COME il sistema consente di accedere ai dati.

Si dividono in due tipi:

- » **Linguaggio procedurale:** il linguaggio *specifica il procedimento necessario a ottenere il risultato.*
 - > *Algebra relazionale:* E' adottata internamente dai sistemi.
- » **Linguaggio dichiarativo:** *specifico solo le proprietà del risultato* senza specificarne la modalità di ottenimento.
Questo approccio ha il vantaggio di liberare chi vuole accedere dall'onere di dover pensare a una sequenza di trasformazioni.
 - > *Calcolo relazionale e SQL (Structured query language):* è il sistema, date le proprietà, a decidere come mettere assieme l'algebra relazionale.

L'algebra relazionale è un insieme di operazioni su relazioni. Si parte da una BD già popolata!

OPERATORI

SEGNATURA DI UN OPERATORE

In generale un operatore può essere unario o binario.

» **Operatore unario:**
agisce su *una sola relazione*
 $op_{\text{parametri}}(r1) \rightarrow r2$

» **Operatore binario**
Agisce su *due relazioni*
 $r1 op_{\text{parametr}} r2 \rightarrow r3$

CLASSIFICAZIONE DI OPERATORI

» Classificazione funzionale

Distingue gli operatori in base all'‘mondo’ di provenienza

- > Operatori *insiemistici*
- > Operatori *specifici*: legati al modello relazionale
- > Operazioni di *giunzione o join*, che consentono di unire

» Classificazione in base alla derivabilità

Distingue gli operatori in base a se essi possono essere derivati da altri o meno.

- > Operatori di *base*
- > Operatori *derivati*

OPERATORI INSIEMISTICI

Vengono dalle operazioni sugli insiemi. Possiamo farlo poiché osserviamo che le relazioni sono *insiemi di tuple*, e dunque ha senso applicarvi le operazioni insiemistiche.

Le relazioni sono insiemi di tuple **OMOGENEE** (=con gli stessi attributi). Pertanto le operazioni devono *conservare questa caratteristica*: gli operatori insiemistici **SI POSSONO APPLICARE SOLO A RELAZIONI CON LO STESSO SCHEMA**.

Base

» UNIONE

$$r_1 \cup r_2 = r_3$$

- > Schema = X
- > Istanza
 - istanza: $r_3 = \{t \mid t \in r_1 \vee t \in r_2\}$
- > Cardinalità
 - $\max(|r_1|, |r_2|) \leq |r_1 \cup r_2| \leq |r_1| + |r_2|$

» DIFFERENZA

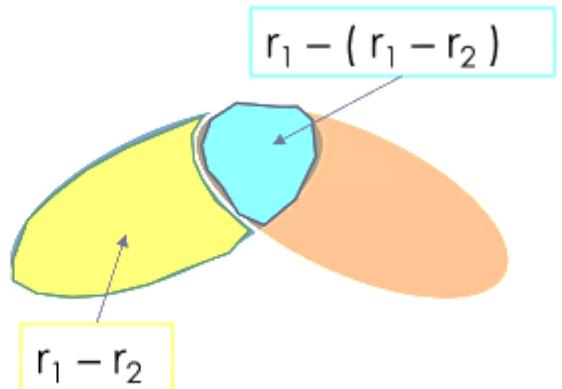
$$r_1 - r_2 = r_3$$

- > Schema = X
- > Istanza
 - istanza: $r_3 = \{t \mid t \in r_1 \wedge t \notin r_2\}$
- > Cardinalità
 - $0 \leq |r_1 - r_2| \leq |r_1|$

Derivati

» INTERSEZIONE:

$$r_1 \cap r_2 = r_1 - (r_1 - r_2)$$



- > Cardinalità:
 - $0 \leq |r_1 \cap r_2| \leq \min(|r_1|, |r_2|)$

OPERATORI SPECIFICI

Sono tutti *operatori di base*.

» RIDENOMINAZIONE

Consente di cambiare il nome degli attributi di una relazione.

Poiché esiste, esso è utile e va usato: infatti esistono operatori dell'algebra che hanno una semantica che dipende dallo schema. Cambiare lo schma durante un'espressione permette di avere un risultato diverso. *Di solito elenchiamo solo gli attributi che cambiano.*

Sia $Y = \{B_1 \dots B_n\}$ un insieme di attributi tali per cui $|Y| = |X|$;

$$\rho_{A_1, \dots, A_n \rightarrow B_1, \dots, B_n}(r_1) = r_2$$

schema: Y
istanza: $r_2 = \{t \mid \exists t' \in r_1 : \forall i \in [1..n] : t[B_i] = t'[A_i]\}$

In pratica cambia i nomi $A_1..A_n$ in $B_1..B_n$ (vecchi->nuovi). *NON FARE LA RIDENOMINAZIONE è UN ERRORE GRAVE!!! :(*

> **Cardinalità:** rimane invariata (ovviamente, sto solo cambiando nome....)

> *Esempietto tattico:*

$CAPOLUOGO_PROV(Nome, Popolazione, CAP)$

$COMUNI_MINORI(Nome, Abitanti, CAP)$

Per poter unire capoluogo e comune devo prima essere *certa che gli attributi abbiano lo stesso nome* – non posso fare l'unione di schemi diversi! *ERRORE GRAVE – 1.5.* Quindi rinomino.

- Producere la lista di tutti i comuni

$\rho_{Popolazione \rightarrow Abitanti}(CAPOLUOGO_PROV) \cup COMUNI_MINORI$

> *Notazione abbreviata:* Se ci serve ridenominare tutti gli attributi posso scrivere

$\rho_{A_1, \dots, A_n \rightarrow A'_1, \dots, A'_n}(r_1) = \rho_{X \rightarrow X'}(r_1)$

= il nome nuovo si ottiene mettendo il pedice a quello vecchio

» SELEZIONE

Consente di estrarre da una relazione le tuple che soddisfano una certa condizione.

F è una certa condizione che deve essere soddisfatta sulla tuple. NON modifica lo schema.

$$\sigma_F(r_1) = r_2$$

schema: X
istanza: $r_2 = \{t \mid t \in r_1 \wedge F(t)\}$
dove $F(t)$ indica che la tupla t rende vera F .

$F(t)$ si ottiene combinando connettivi logici, not e condizioni atomiche del tipo $A \theta B$ o $A \theta c$, con A, B attributi e

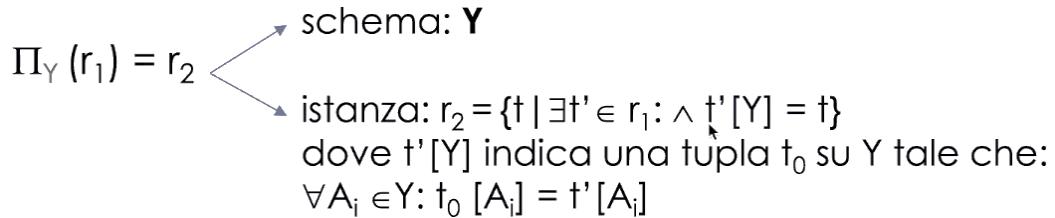
- $A, B \in X$
- $\theta \in \{<, =, >, \neq, \geq, \leq\}$
- c è una costante appartenente a $DOM(A)$ o compatibile con esso

> **Cardinalità:**

$$0 \leq \sigma_F(r_1) \leq |r_1|$$

» PROIEZIONE

Riduce le tuple di una relazione in verticale, ovvero elimina attributi.



> Cardinalità:

!!! Eliminando cose dalle tuple può capitare che tuple diverse diventino uguali. E rischio di "sovrascrivere" altre tuple: quindi le tuple di arrivo sono meno.

Al minimo, si potrebbe lasciare un solo parametro e tutti si schiacciano su una tupla. Inoltre, potrebbe essere 0 se la relazione è vuota. Al massimo, tutte le tuple rimangono distinte.

$$\min(|r_1|, 1) \leq |\Pi_Y(r_1)| \leq |r_1|$$

Inoltre se rispetto la seguente sono certa di non generare alcun duplicato.

$$|\Pi_Y(r_1)| = |r_1| \text{ se } Y \text{ è superchiave per } r_1$$

> Altro esempio tattico:

TRENO(Numer, PartOra, PartMin, Cat, Dest, ArrOra, ArrMin)

FERMATA(NumTreno, Stazione, Ora, Min)

- Producere la lista di tutti i treni non regionali che partono alle 12 o dopo le 12 o prima delle 16
 $\sigma_{PartOra \geq 12 \wedge PartOra < 16 \wedge Cat \neq 'REGIONALE'}(TRENO)$
- Restituire il numero e l'ora di partenza dei treni freccia rossa con destinazione milano centrale
 $\Pi_{Numer, PartOra}(\sigma_{Dest='Milano Centrale' \wedge Cat = 'FR'}(TRENO))$

OPERATORI DI GIUNZIONE O JOIN

Gli operatori di join sono i più importanti!! Le interrogazioni più interessanti usano le join.

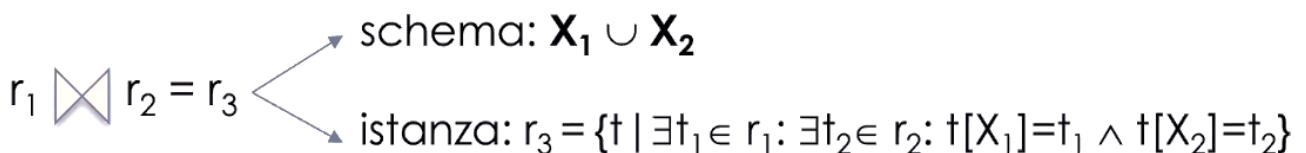
Si tratta di un *operatore binario*. In modo intuitivo generano tuple a partire da coppie di tuple che soddisfano una condizione e fatte come (t_1, t_2) , pescando le coppie dal prodotto cartesiano delle due relazioni r_1 ed r_2 di schema X_1 e X_2 .

Operatori di base

» JOIN NATURALE

E' un join dove il predicato per la produzione delle tuple è una condizione implicita, ovvero dipende dallo schema; in questo caso si producono gli attributi COMUNI alle due relazioni.

Questo rende evidente l'utilità dell'operatore ridenominazione! La semantica dipende dallo schema.
!! Attenzione: gli attributi comuni sono rappresentati una volta sola.



› *Esempietto tattico:*

$\text{DOCENTE(CFDocente, Nome, Cognome)}$

$\text{CORSO(Nome, CFDocente)}$

- *Produc i l'insieme dei corsi riportando nome del corso e cognome del docente.*

Attenzione: se faccio il join diretto fra le due tabelle, anche il nome verrà considerato una condizione di join!!! E' difficile che nome del corso = nome docente, quindi al 100% otterrò un risultato vuoto. :(DEVO RINOMINARE!!! - rinomino, faccio la join, proietto

$\Pi_{\text{NomeCorso}, \text{Cognome}}($

$\rho_{\text{Nome} \rightarrow \text{NomeCorso}}(\text{CORSO}) \bowtie \text{DOCENTE })$

DOCENTE

CFDocente	Nome	Cognome
BLSLRT	Alberto	Belussi
QNTLSA	Elisa	Quintarelli
MGLSRA	Sara	Migliorini

CORSO

Nome	CFDocente
Basi di dati TEORIA	BLSLRT
Basi di dati LAB	MGLSRA
Programmazione	QNTLSA
Data integration	QNTLSA

$\Pi_{\text{NomeCorso}, \text{Cognome}}($

$\rho_{\text{Nome} \rightarrow \text{NomeCorso}}(\text{CORSO}) \bowtie \text{DOCENTE })$

NomeCorso	Cognome
Basi di dati TEORIA	Belussi
Basi di dati LAB	Migliorini
Programmazione	Quintarelli
Data integration	Quintarelli

> **Proprietà:**

- Il join naturale si dice **completo** se *tutte le tuple di r1(r2) contribuiscono a generare almeno una tupla nel join.*
 $\forall t_1 \in r_1: \exists t \in r_1 \bowtie r_2: t[X_1] = t_1 \wedge \forall t_1 \in r_1: \exists t \in r_1 \bowtie r_2: t[X_1] = t_1$
- Dato un join naturale non completo, le tuple che non contribuiscono al risultato si dicono **dangling tuples**.
- **Commutativo:**
 $r_1 \bowtie r_2 = r_2 \bowtie r_1$
- **Associativo:**
 $r_1 \bowtie (r_2 \bowtie r_3) = (r_1 \bowtie r_2) \bowtie r_3$
- Se r_1 e r_2 hanno lo stesso schema, è **equivalente all'intersezione**
 $r_1 \bowtie r_2 = r_1 \cap r_2$
- Se r_1 e r_2 non hanno attributi comuni, allora è **equivalente al prodotto cartesiano**
 $r_1 \bowtie r_2 = r_1 \times r_2$

> **Cardinalità:**

E' utile considerare la cardinalità per capire in che ordine conviene procedere, o anche per valutare il costo. Al minimo avrà 0 tuple (nessun attributo in comune), mentre al più avrà il prodotto cartesiano

$$0 \leq |r_1 \bowtie r_2| \leq |r_1| \bullet |r_2|$$

Altre possibilità:

- > Se il **join è completo**, tutte le tuple partecipano; quindi al minimo avrà il massimo delle cardinalità
 $\max(|r_1|, |r_2|) \leq |r_1 \bowtie r_2| \leq |r_1| \bullet |r_2|$
- > Se l'intersezione dei due schemi è **superchiave per r_2** [##### 58.22 12/01]
 $0 \leq |r_1 \bowtie r_2| \leq |r_1|$
- > Se l'intersezione dei due schemi è una superchiave per r_2 **esiste un vincolo di integrità referenziale**

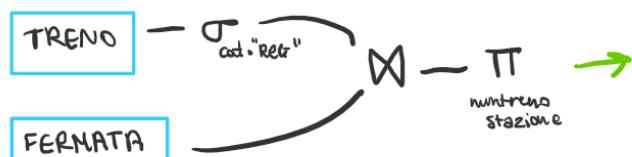
$$|r_1 \bowtie r_2| = |r_1|$$

> *Esempio tattico*

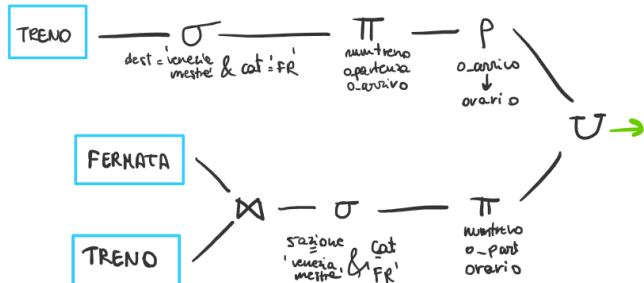
TRENO(NumTreno, OrarioPart, Cat, Dest, OrarioArr)

FERMATA(NumTreno, Stazione, Orario)

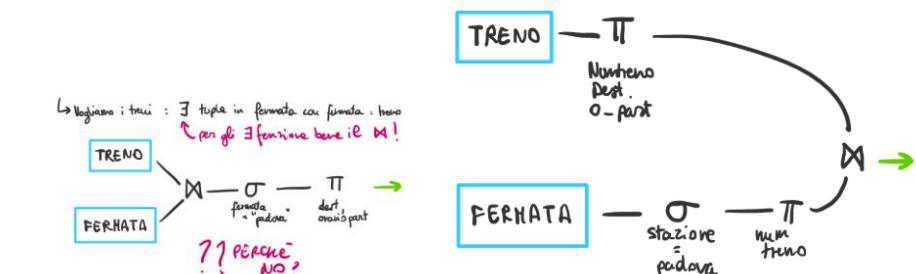
- Tutte le fermate dei treni regionali.
 select treno cat=reg, join treno-fermata,
 proietta solo numtreno e la roba di fermata



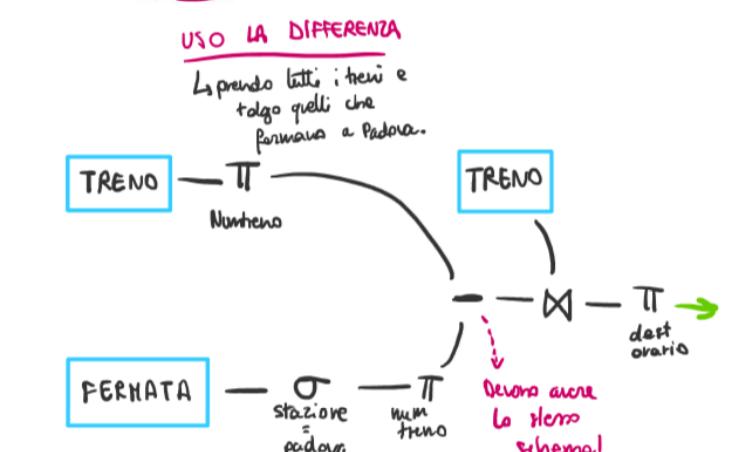
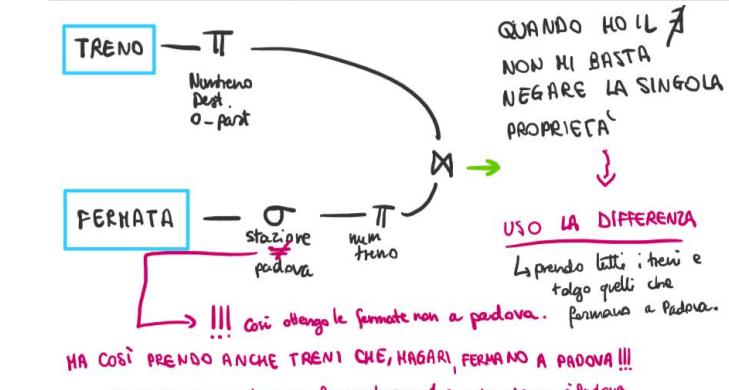
- Q2: Trovare l'elenco di tutti i treni Freccia Rossa per Venezia Mestre (hanno come destinazione Venezia Mestre o fermano a Venezia Mestre) riportando: il numero del treno, l'orario di partenza e l'orario di arrivo a Venezia Mestre.



- Q3: Trovare la destinazione e l'orario di partenza dei treni che fermano a Padova.



- Q4: Trovare la destinazione e l'orario di partenza dei treni che non fermano a Padova.



» THETA JOIN

E' un join dove il predicato per la produzione delle tuple è viene indicato esplicitamente, ed è indipendentemente dallo schema

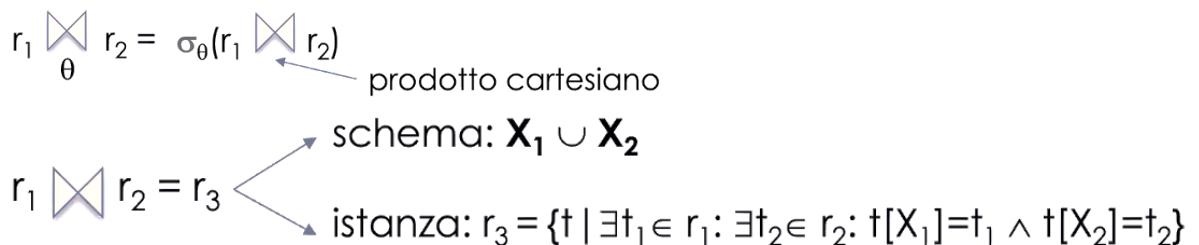
> *Precondizione*

Al contrario del join naturale (che può lavorare in ogni condizione), abbiamo la condizione che gli schemi siano disgiunti (ovvero no attributi in comune).

$$\mathbf{X}_1 \cap \mathbf{X}_2 = \emptyset$$

Corrisponde a un join naturale (che sarà = a un prodotto cartesiano dato che sono disgiunti) e applicare una selezione con la condizione esplicitata.

A livello di implementazione, chiaramente, sono applicate tecniche furbe per non produrre tutte le coppie $\wedge\wedge$



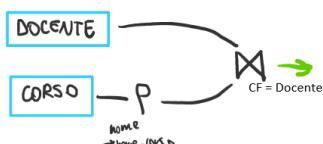
> *Esempietto tattico:*

→ DOCENTE(CF, Nome, Cognome)

CORSO(Nome, Docente)

Attenzione!! Docente.nome e Corso.nome hanno lo stesso nome ma sono due cose ben diverse; in particolare, ciò significa che docente e corso NON SONO DISGIUNTI E NON POSSO APPLICARE LA THETA JOIN. bisognerà usare la ridefinizione.

Si richiede di produrre l'elenco di tutti i corsi riportando:
nome del corso e cognome del docente.



> *Proprietà*

- Si dice **equi-join** se la condizione theta è una congiunzione di uguaglianze.

$$\begin{array}{ccc} r_1 & \bowtie & r_2 \\ A_1 = B_1 \wedge \dots \wedge A_n = B_n & & \end{array}$$

ATTENZIONE: Non esiste un operatore misto (= prende quelli in comune e applica la condizione. NON SE PUO')

Teorema: date due relazioni con degli attributi comuni, vale l'equivalenza

$$r_1 \bowtie r_2 = \Pi_{X_1 \cup X_2} (r_1 \bowtie r_2) \quad \text{C}_1 = C'_1 \wedge \dots \wedge C_m = C'_m$$

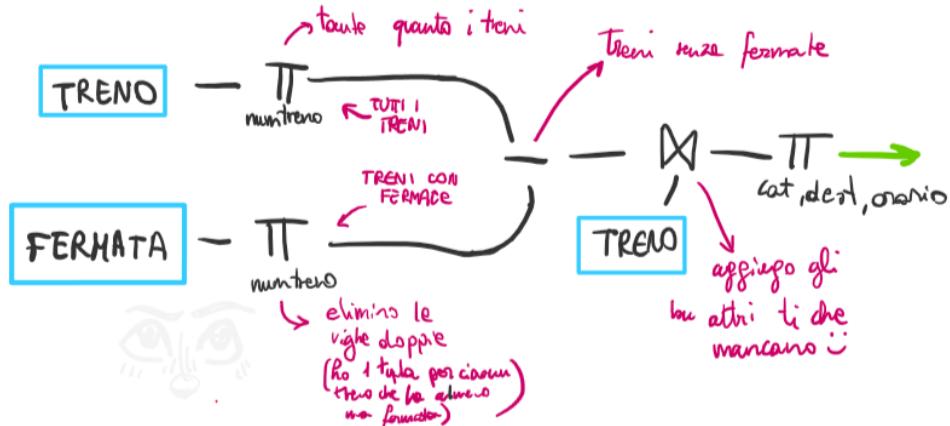
Ovvero il join naturale equivale a un equijoin, dove rinomino lo schema per avere solo attributi disgiunti, fare una theta join su tutti gli attributi e applicare una select.

» Ulteriori esempi tattici:

→ TRENO(NumTreno, OrarioPart, Cat, Dest, OrarioArr)

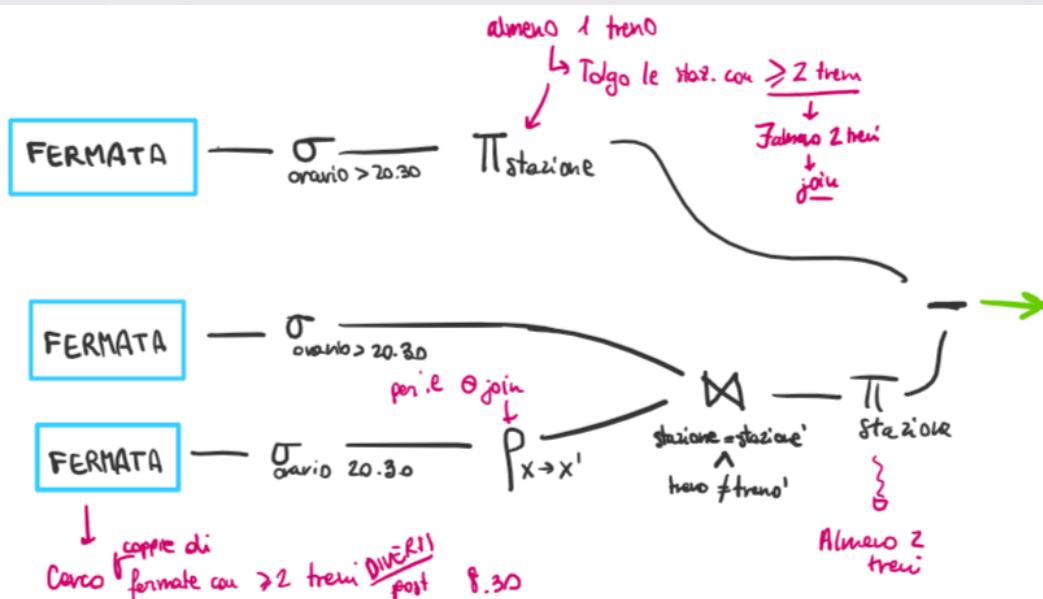
FERMATA(NumTreno, Stazione, Orario)

- Q5: Trovare la categoria, la destinazione e l'orario di partenza
dei treni che non hanno fermate.



Q6: Trovare l'elenco di tutte le stazioni dove dopo le 20.30

- ferma uno e un sol treno.



Q7: Trovare l'elenco di tutte le stazioni dove fermano tutti i treni con destinazione 'Venezia SL'

- Prima costruisco l'insieme di tutte le coppie stazione – treno che destina a venezia. Da questo insieme tolgo le fermate vere.

Q8: Trovare per ogni treno regionale il treno freccia rossa immediatamente successivo con la stessa destinazione, riportando nel risultato il numero e l'orario di partenza dei due treni insieme alla loro destinazione.

ALGEBRA CON VALORI NULLI

Attenzione: e se ho un valore nullo cosa perdincibacco faccio? Gli operatori coinvolti nel casino saranno selezione e join naturale.

Sulle altre operazioni basta fare una piccola osservazione e si nota che l'operatore si adegua.

SELEZIONE

Bisogna dire come si valuta la condizione in presenza di valore nullo: il valore nullo può comparire anche nel predicato di selezione.

→ Sia per A theta B che A theta costante, se uno dei due valori è null, il risultato è falso qualsiasi sia il confronto.

! Attenzione ! Con questa regola NULL == NULL è valutato falso.

Inoltre, aggiungiamo due nuove condizioni atomiche: A is NULL e A is not NULL.

JOIN NATURALE

La condizione di uguaglianza sugli attributi comuni alle due relazioni è falsa sulle tuple t1 e t2 se almeno uno degli attributi comuni è NULL.

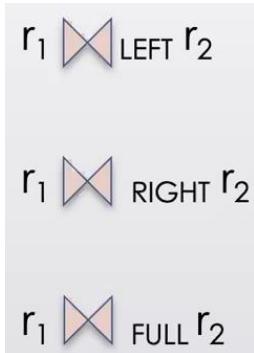
Anche qui, NULL == NULL è valutato falso.

JOIN ESTERNI

La introduciamo ma non vogliamo usarla nelle interrogazioni: sono varianti del join naturale che potremo usare per gestire alcune situazioni col null, ma non sono strettamente necessarie e non le vuole negli esercizi.

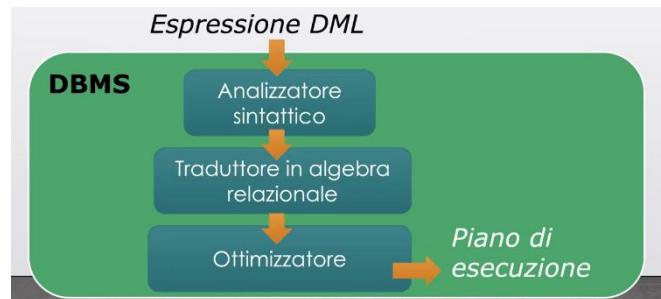
Consentono di otteener nel risultato del join anche le tuple con valori nulli e le dangling tuples.

- › Left join : conserva tutte le tuple della relazione; nel risultato ci saranno tutte le tuple di r1; quelle che non si combinano con nessuna tupla di r2 comunque partecipano al risultato, ma avranno valori nulli negli attributi che sarebbero di r1
- › Right join: stessa cosa ma su r4
- › Full join: stessa cosa ma si conservano entrambe



OTTIMIZZAZIONE DI ESPRESSIONI DML

Ogni espressione DML (di solito in linguaggio dichiarativo tipo SQL) viene ricevuta dal DBMS e soggetta a un processo di rielaborazione.



OTTIMIZZATORE

L'ottimizzatore genera un'espressione equivalente all'interrogazione di input, ma con un costo inferiore.

Il costo viene valutato in termini di dimensione dei risultati intermedi: l'idea è che se durante l'esecuzione parto da una struttura in memoria secondaria (e quindi inizialmente devo inevitabilmente accedervi). Prima riduco la dimensione dei dati da trattare, prima riduco il numero di accessi alla memoria secondaria!

L'ottimizzatore esegue trasformazioni di equivalenza allo scopo di ridurre la dimensione degli stati intermedi.

EQUIVALENZA

Possiamo definire diversi tipi di equivalenze:

- » Equivalenza dipendente dallo schema

$E1 \equiv_R E2$ se $E1(r) = E2(r)$ per ogni istanza r di schema R

- » Equivalenza assoluta

È uguale ma indipendente dallo schema

E' indipendente dallo schema

$E1 \equiv E2$ se $E1 \equiv_R E2$ per ogni schema R compatibile con $E1$ e $E2$

TRASFORMAZIONI DI EQUIVALENZA

Data E un'espressione di schema X, possiamo definire le seguenti trasformazioni.

Alcune non sono ottimizzanti in sé, ma permettono di arrivare a un'ottimizzazione insieme alle altre

» ATOMIZZAZIONE DELLE SELEZIONI

Non ottimizza di per sé ma serve per applicare le altre.

Parte da una selezione con più condizioni in and e le atomizza l'una nell'altra.

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



» IDEMPOTENZA DELLE PROIEZIONI

Non ottimizza di per sé ma serve per applicare le altre.

Similmente a prima, permette di proiettare "un pezzo per volta".

$$\Pi_Y(E) \equiv \Pi_Y(\Pi_{YZ}(E)) \text{ dove } Z \subseteq X$$



» ANTICIPAZIONE DELLE SELEZIONI RISPETTO AL JOIN

(!!! Very important)

Il join può produrre un aumento della quantità di dati da produrre! Quindi vorrei farlo il più tardi possibile, cosicché io possa diminuire il più possibile la cardinalità del risultato.

Con l'atomizzazione delle selezioni potrei favorire l'applicazione di questa regola.

$$\sigma_F(E1 \bowtie E2) \equiv E1 \bowtie \sigma_F(E2)$$



» ANTICIPAZIONE DELLA PROIEZIONE RISPETTO AL JOIN

!!! Devo fare attenzione a non togliere gli attributi che partecipano al join, ovvero attributi in comune.

$$\Pi_{X1Y}(E1 \bowtie E2) \equiv_R E1 \bowtie \Pi_Y(E2)$$

Applicabile solo se $Y \subseteq X2$ e $(X2 - Y) \cap X1 = \emptyset$

» COMBINAZIONE DI ANTICIPAZIONE DELLA PROIEZIONE E IDEMPOTENZA

$$\Pi_Y(E1 \bowtie_F E2) \equiv \Pi_Y(\Pi_{Y1}(E1) \bowtie_F \Pi_{Y2}(E2))$$

$$\Pi_Y(E1 \bowtie E2) \equiv \Pi_Y(\Pi_{Y1}(E1) \bowtie \Pi_{Y2}(E2))$$

dove:

- $Y1 = (X1 \cap Y) \cup J1$
- $Y2 = (X2 \cap Y) \cup J2$
- $J1/J2$ sono gli attributi di $E1/E2$ coinvolti nel join (vale a dire presenti in F per il theta-join, mentre in caso di join naturale $J1=J2=X1 \cap X2$)

> Esempio tattico:

TRENO(NumTreno, OrarioPart, Cat, Dest, OrarioArr)
 FERMATA(NumTreno, Stazione, Ora)

Q: Trovare il numero e l'ora di partenza dei treni freccia rossa che fermano a Vicenza.

$$\Pi_{\text{NumTreno}, \text{OrarioPart}} (\sigma_{\text{Cat}=\text{'FR'}} \wedge \sigma_{\text{Stazione}=\text{'Vicenza'}} (\text{TRENO} \bowtie \text{FERMATA}))$$

Ad ora, la join avrà cardinalità **esattamente** di fermata dato che treno è chiave esportata!

1. Devo **dividere in due la selezione**, così posso dividere quelle che riguardano fermata da quelle che riguardano treno

$$\Pi_{\text{NumTreno}, \text{OrarioPart}} (\sigma_{\text{Cat}=\text{'FR'}} (\sigma_{\text{Stazione}=\text{'Vicenza'}} (\text{TRENO} \bowtie \text{FERMATA})))$$

2. **Anticipo le selezioni:** a questo punto, dato che ciascuna selezione riguarda solo uno dei due, posso anticipare le selezioni sui rispettivi "rami"

$$\Pi_{\text{NumTreno}, \text{OrarioPart}} (\sigma_{\text{Cat}=\text{'FR'}} (\text{TRENO}) \bowtie \sigma_{\text{Stazione}=\text{'Vicenza'}} (\text{FERMATA}))$$

Ora non so esattamente la cardinalità del join, ma di certo sarà minore (poiché ho meno fermate e meno treni) Anticipo le proiezioni

3. **Anticipo le proiezioni:** stessa cosa :)

$$\Pi_{\text{NumTreno}, \text{OrarioPart}} (\Pi_{\text{NumTreno}, \text{OrarioPart}} (\sigma_{\text{Cat}=\text{'FR'}} (\text{TRENO}) \bowtie \Pi_{\text{NumTreno}} (\sigma_{\text{Stazione}=\text{'Vicenza'}} (\text{FERMATA}))))$$

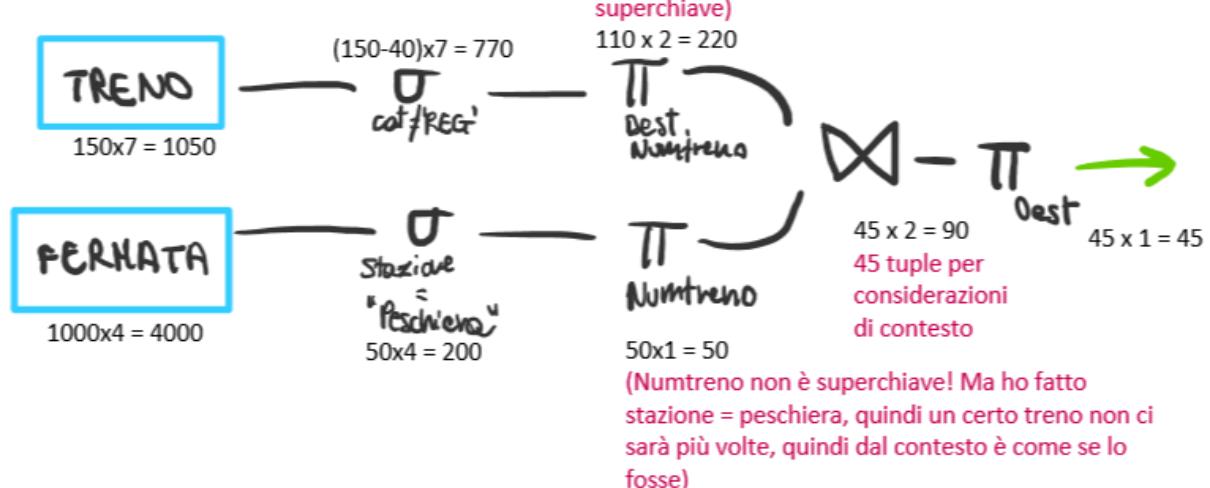
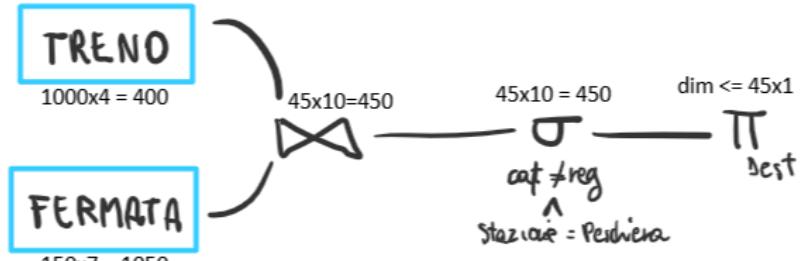
> Esempio tattico con le cardinalità:

TRENO(NumTreno, Cat, OraPart, MinPart, OraArr, MinArr, Dest)
 FERMATA(NumTreno, Stazione, Ora, Min)

Q: Trovare la destinazione dei treni non regionali che fermano a "Peschiera".

Dimensioni dei dati:

TRENO:	FERMATA:
#attributi = 7	#attributi = 4
#tuple = 150	#tuple = 1000
#tuple treni regionali = 40	#tuple staz peschiera = 50
	#tuple staz peschiera e treno non regionale = 45



» **INGLOBAMENTO DI UNA SELEZIONE IN UN PRODOTTO CARTESIANO**

Attenzione! Devo essere certa che $X_1 \cap X_2 = \emptyset$.

Astrattamente dovrei gestire l'intero prodotto cartesiano ma l'implementazione reale del theta join è ottimizzata, quindi meglio usare il theta join.

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



!!! Si usa solo dopo aver verificato che non è possibile anticipare selezioni rispetto al join.

» **ULTERIORI TRASFORMAZIONI** (Non da applicare in esame :D)

Il senso di applicazione e la sua utilità sono più sfumate: dipendono dal contesto ed eventuali statistiche.

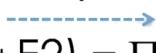
> **Commutativa e associativa di unione, prodotto cartesiano, intersezione**

> **Distributiva**

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



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



$$\Pi_Y(E_1 \cup E_2) \equiv \Pi_Y(E_1) \cup \Pi_Y(E_2)$$

$$E_1 \bowtie (E_2 \cup E_3) \equiv (E_1 \bowtie E_2) \cup (E_1 \bowtie E_3)$$

> **Varie ed eventuali**

$$\sigma_{F1 \vee F2}(E) \equiv \sigma_{F1}(E) \cup \sigma_{F2}(E)$$

$$\sigma_{F1 \wedge F2}(E) \equiv \sigma_{F1}(E) \cap \sigma_{F2}(E) \equiv \sigma_{F1}(E) \bowtie \sigma_{F2}(E)$$

$$\sigma_{F1 \wedge \neg F2}(E) \equiv \sigma_{F1}(E) - \sigma_{F2}(E)$$

Vista

DEF

È una relazione derivata.

Si specifica l'espressione che genera il suo contenuto; esso dipende dalle relazioni che compaiono nell'espressione, e non persiste nella base di dati.

In generale, scrivendo un'interrogazione, può essere necessario definire delle relazioni derivate temporanee (risultati parziali) e poi unirli. Il concetto di vista mi permette proprio questo: divide il lavoro in parti temporanee (che non diventano parte della base di dati).

Osservazione: dando totale libertà per la definizione di queste relazioni derivate, potrei definire viste che sono funzione di altre viste; posso farlo purché esista un ordinamento in grado di guidare il calcolo delle relazioni derivate: non posso avere liste cicliche o che dipendono da sé stesse.

Utilità delle viste:

- » Personalizzazione dell'interfaccia utente: per esempio posso far vedere sottoinsiemi diversi a utenti diversi.
 - > Facilito la gestione della privacy dei dati
 - > Rendo l'interfaccia delle applicazioni indipendente dallo schema logico: **indipendenza logica**.
- » Memorizzazione nel DBMS di interrogazioni complesse condivise
 - > Condivido l'interrogazione fra più applicazioni.

MODALITÀ DI VISTA

» VISTA VIRTUALE

È il meccanismo usato più spesso: memorizzo solo l'espressione di visualizzazione, e la calcolo solo quando mi serve (es. in un'interrogazione).

Conviene se:

- + Dati aggiornati di frequente
- + Interrogazione che genera la vista è semplice

» VISTA MATERIALIZZATA

Una vista è materializzata se viene calcolata e memorizzata esplicitamente nella base di dati. Ha iniziato ad essere usata da poco.

Ogni volta che si aggiornano elementi appartenenti alla vista dovrà ricalcolarla!

Conviene se:

- + Aggiornamenti rari
- + Interrogazione che genera la vista è molto complessa.

SQL

Nasce anche come definition language, ma noi lo consideriamo solo per le interrogazioni.

Poiché quest'ultimo linguaggio basato sulla logica non è parte del programma del corso, la semantica di SQL verrà presentata in modo informale e quando possibile verrà assegnata utilizzando espressioni dell'algebra relazionale.

SQL (Structured Query Language) è stato definito negli anni '70 e standardizzato negli 80-90. È ad oggi il linguaggio più diffuso per i DBMS relazionali.

Anche le nuove tecnologie si ispirano alla sua struttura, quindi è fondamentale.

L'SQL è un **linguaggio di interrogazione dichiarativo**. La sua semantica fa riferimento al calcolo relazionale.

FORMA BASE DI UN'INTERROGAZIONE SQL

```
SELECT<ListaAttributi>
FROM<ListaTabelle>
[WHERE<Condizione>]
```

Il risultato dell'interrogazione sarà una relazione risultato.



Semantica

La *relazione risultato* avrà le seguenti caratteristiche:

» Schema

È costituito da tutti gli *attributi indicati in <ListaAttributi>*. Quindi Select assomiglia alla *proiezione*

» Contenuto

È costituito da tutte le *tuple t ottenute proiettando sugli attributi di <ListaAttributi> le tuple t' appartenenti alle tavole indicate in <ListaTabelle> che soddisfano l'eventuale condizione <Condizione>*.

> Esempietto tattico

TRENO(Num, Cat, Part, Arrivo, Dest)

FERMATA(Treno, Stazione, Orario)

Numero, categoria e destinazione di tutti i treni che partono dopo le 14.

```
SELECT Num, Cat, Dest
FROM TRENO
WHERE Part > '14.00'
```

Stazione e orario di tutte le fermate del treno 123.

```
SELECT Stazione, Orario
FROM FERMATA
WHERE Treno = 123
```

CLAUSOLA SELECT

<[DISTINCT]> <espr> [[AS] <alias>] {, <espr> [[AS] <alias>] } | *

Dove:

- » **<espr>** è un'espressione che coinvolge gli attributi della tabella (anche, semplicemente, il numero di attributi)
- » * è una stenografia per indicare tutti gli *attributi delle tabelle coinvolte*.
- » **<alias>** è il *nome assegnato all'attributo che conterrà il risultato dell'espressione <espr>* nel risultato.
- » **DISTINCT**: se presente, richiede *l'eliminazione nella relazione risultate delle tuple duplicate*. In algebra relazionale è "sottointeso", ma dato che essa richiede lavoro SQL la fa solo se richiesto.

Esempietto tattico

Tutti gli attributi di treno dei treni frecciarossa.

```
SELECT *
FROM TRENO
WHERE Cat = 'FR'
```

Tutte le destinazioni distinte dei treni regionali che partono dopo le 9.

```
SELECT DISTINCT Dest
FROM TRENO
WHERE Cat ='Reg' AND Part > '9:00'
```

CLAUSOLA FROM

Può essere esercitata con diverse varianti; noi consideriamo solo quella classica, e vedremo le altre in lab.

FROM<ListaTabelle>, dove <ListaTabelle> è una lista di tabelle con la seguente sintassi:

<tabella> [[AS] <alias>] {, <tabella> [[AS] <alias>] }

Attenzione alla semantica:

- » Se sono presenti due o più tabelle, la semantica prevede che si esegue il *prodotto cartesiano* fra tutte le tabelle e successivamente si applichi una selezione basata sulla condizione specificata nella WHERE. Non è richiesto che le tabelle abbiano schemi particolari; possono essere fatte in qualunque modo.
- » **Se ci sono attributi con lo stesso nome in più tabelle, è possibile permettere al nome dell'attributo il nome della tabella come segue:** <NomeTabella>.<NomeAttributo>

Esempietto tattico

TRENO(NumTreno, Cat, Part, Arrivo, Dest)

FERMATA(NumTreno, Stazione, Orario)

```
SELECT T.NumTreno, F.Stazione
FROM TRENO AS T, FERMATA AS F
WHERE T.NumTreno = F.NumTreno
```

```
SELECT NumTreno, Cat, Part
FROM TRENO
WHERE (Cat ='Freccia' OR Cat='IC')
      AND Part > '12.00'
Treni freccia o intercity che
partono dopo le 12
```

```
SELECT NumTreno, Cat, Part
FROM TRENO
WHERE (Cat ='Freccia' OR Cat='IC')
Treni freccia e intercity
```

CLAUSOLA WHERE

Where<ListaCondizioni>

Sove <ListaCondizioni> è un'espressione booleana ottenuta combinando condizioni semplici <Cond> con i connettivi AND, OR, NOT.

<cond> può essere:

- » <espr> θ <espr>
- » <espr> θ <costante>

Con:

- » $\theta \in \{=, <, >, <>, \leq, \geq\}$
- » <espr> è l'espressione che contiene riferimenti agli attributi delle tabelle che compaiono in FROM
- » <const> è un valore

INTERPRETAZIONE ALGEBRICA

Possiamo dividerla in casi:

Una sola tabella nella clausola FROM

Corrisponde a una selezione e proiezione sulle tabelle del FROM.

$$\Pi_{\{A_1, \dots, A_n\}}(\sigma_{\text{cond}}(T))$$

Più tabelle nella clausola FROM

Corrisponde a una proiezione e proiezione sul prodotto cartesiano fra le tabelle del from

$$\Pi_{\{A_1, \dots, A_n\}}(\sigma_{\text{cond}}(T_1 \bowtie \dots \bowtie T_m))$$

prodotto cartesiano (nessun attributo comune)

VARIABILI TUPLA

Le variabili tupla, o alias di tabella, sono usate con due obiettivi:

- » Risolvere le ambiguità sui nomi degli attributi
- » Gestire il riferimento a più esemplari della stessa tabella.

Esempio tattico

Trovare il numero e l'orario di partenza dei treni 'FR' che hanno la stessa destinazione di almeno un treno 'Reg'.

```
SELECT T1.NumTreno, T1.Part  
FROM TRENO T1, TRENO T2  
WHERE T1.Cat = 'Fr'  
AND T2.Cat = 'REG'  
AND T1.Dest = T2.Dest
```

CLAUSOLA ORDER BY

Consente di ordinare; non esiste in algebra (dato che in algebra sono insiemi di tuple!)

ORDER BY <Attributo> [<ASC|DESC>]

{ ,<Attributo> [<ASC|DESC>]}

Il default è l'ordine crescente. Tutti i domini che stiamo considerando sono ordinati, quindi è sempre possibile applicare l' ORDER BY!

Esempietto tattico:

Trovare il numero e

l'orario di partenza dei

treni regionali ordinati per

orario di partenza.

```
SELECT T.Part, T.NumTreno  
FROM TRENO AS T  
WHERE T.Cat = 'Reg'  
ORDER BY Part
```

INTERROGAZIONI NIDIFICATE

Sono una variante della clausola where che introducono le interrogazioni che, tipicamente, in algebra corrispondono alla differenze. Si ottiene quando nella condizione where appare un predicato complesso, ovvero che contiene un'altra interrogazione SQL.



Dovremo introdurre nuovi operatori di confronto!

Il predicato confronta il valore di un attributo con il risultato di un'altra interrogazione; Il caso tipico è che la clausola select del predicato complesso abbia un solo attributo, dunque produca un insieme di valori.

Poiché il risultato è un insieme di valori, l'operatore di confronto deve essere adatto a confrontare un valore con un insieme di valori. Quindi il theta' sarà nuovo.

OPERATORI

Si ottengono combinando uno degli operatori di confronto già visti con le parole chiave ALL e ANY, che richiamano gli identificatori esistenziale e universale!

- » **A op ANY:** soddisfatto se nella tupla t esiste almeno un valore v contenuto nel risultato che soddisfa la condizione $t[a] \text{ op } v \rightarrow$ È l'esiste
 - =ANY si può scrivere come **IN**
- » **A op ALL:** soddisfatto dalla tupla t se per ogni valore v è soddisfatta la condizione $t[a] \text{ op } v \rightarrow$ È il per ogni
 - <> ALL si può scrivere come **NOT IN**

Esempio tattico

TRENO(NumTreno, Cat, Part, Arrivo, Dest)

FERMATA(NumTreno, Stazione, Orario)

Trovare la destinazione dei treni che non fermano a Brescia → infatti mi serve la differenza

Mi serve DISTINCT o avrò ventimila volte le città

```
SELECT DISTINCT T.Dest  
FROM TRENO,TERMATA  
WHERE NumTreno NOT IN  
( SELECT F.NumTreno  
FROM TERMATA AS F  
WHERE F.Stazione = 'Brescia')
```

Le interrogazioni nidificate si dividono in:

Interrogazioni nidificate INDIPENDENTI

L'interrogazione nidificata può essere valutata una volta in quanto *non dipende dalla tupla esterna*.

La riconosco dal fatto che non ci sono variabili tupla condivise fra interrogazione esterna e interna.

Interrogazioni nidificate DIPENDENTI

Ci sono variabili condivise, ovvero c'è *almeno una variabile presente sia nell'esterna che nell'interna*.

Essa realizza il cosiddetto passaggio di binding: questo implica che l'interrogazione nidificata deve essere valutata per ogni tupla dell'interrogazione esterna. In generale se ne occupa il sistema (ma di solito, semplicemente, non lo fanno :P)

Esempio tattico (NON È TRENTO FERMATA!!! T_T)

```
CLIENTE(CF, Nome, Cognome, Prof, DataN, Città)
FILIALE(Codice, Nome, Indirizzo, Città)
CONTO(Filiale, Numero, Saldo)
INTESTAZIONE(FilialeCC, NumeroCC, Cliente)
MOVIMENTO(FilialeCC, NumeroCC, Num, Tipo, Data, Imp)
```

Trovare il nome e il cognome degli intestatari dei conti dove tutti i movimenti eseguiti sono stati di importo inferiore a 1000 euro

```
SELECT C.Nome, C.Cognome
FROM CLIENTE as C, INTESTAZIONE as I
WHERE C.CF = I.CF
AND 1000 > ALL(
    SELECT MOVIMENTO.Imp
    FROM MOVIMENTO
    WHERE FilialeCC = I.Filiale
    AND NumeroCC = I.NumeroCC
) #tutti gli importi di quel cliente/conto
```

CLAUSOLA EXISTS

È una clausola utilizzabile nei predicati complessi, come ANY e ALL.. È efficace solo se la q è un'interrogazione nidificata **che dipende da quella esterna**: altrimenti avrei che se q esiste quella esterna è sempre vera o sempre falsa.

EXISTS (SQLquery)

SEMANTICA:

È del tutto equivalente all'identificatore esistenziale della logica.



Esempietto tattico

```
CLIENTE(CF, Nome, Cognome, Prof, DataN, Città)
FILIALE(Codice, Nome, Indirizzo, Città)
CONTO(Filiale, Numero, Saldo)
INTESTAZIONE(FilialeCC, NumeroCC, Cliente)
MOVIMENTO(FilialeCC, NumeroCC, Num, Tipo, Data, Imp)
```

Trovare il nome e il cognome degli intestatari di conti correnti sui quali non sono stati eseguiti prelievi BANCOMAT dal 1/4/2010 a oggi.

- » In algebra useremmo la differenza
- » Servono due condizioni perché la chiave di conto corrente è doppia.

```
SELECT C.Nome, C.Cognome
FROM CLIENTE as C, INTESTAZIONE as I
WHERE C.CF = I.CLIENTE
AND NOT EXISTS (
    SELECT 1 #FilialeCC
    FROM MOVIMENTO as M
    WHERE M.Data > '1/4/2010'
    AND M.Tipo = 'Bancomat'
    AND M.NumeroCC = I.NumeroCC
    AND M.FilialeCC = I.FilialeCC
)
```

CLAUSULA IN E NOT IN

Utilizzando l'operatore tupla A1,12... (oppure ROW(A,A2...)) è possibile (in un'interrogazione che usa IN o NOT IN) confrontare una tupla con un insieme di tuple, e non solo un valore con un insieme di valori.

ESERCIZI

```
CLIENTE(CF, Nome, Cognome, Prof, DataN, Città)
FILIALE(Codice, Nome, Indirizzo, Città)
CONTO(Filiale, Numero, Saldo)
INTESTAZIONE(FilialeCC, NumeroCC, Cliente)
MOVIMENTO(FilialeCC, NumeroCC, Num, Tipo, Data, Imp)
```

<p>1. Trovare numero filiale e saldo dei conti che non hanno intestatari residenti a Verona.</p>	<pre>SELECT CC.FilialeCC, CC.Saldo, CC.NumeroCC FROM CONTO as CC WHERE NOT EXISTS(SELECT 1 FROM CLIENTE C, INTESTAZIONE I WHERE C.CF = I.CF AND C.Città = 'Verona' AND I.FilialeCC = CC.Filiale AND I.NumeroCC = CC.Numero)</pre>
<p>2. Trovare per ogni filiale il nome e il cognome del cliente correntista più giovane, riportando anche il codice della filiale.</p>	<pre>SELECT CL.Nome, CL.cognome, I.FilialeCC FROM Cliente as CL, Intestazione as I WHERE CL.CF = I.Cliente AND NOT EXISTS (SELECT 1 FROM Cliente as CL2, Intestazione as I2 WHERE CL2.DataN < CL.DataN AND I.FilialeCC = I2.FilialeCC) #non esiste un altro cliente nella stessa filiale più giovane di lui Prima di tutto riporto in select quello che voglio nel finale, ovvero nome cognome e filiale. Poiché filiale farà parte del finale, in FROM mi servono cliente e intestazione. Vado a risultato se non esiste nessun altro risultato più giovane. → TUTTE LE VOLTE CHE MI SERVE MAX O MIN LO FACCIO CON UNA "DIFERENZA". NIDIFICO LA CONDIZIONE SU UN INSIEME DI TUPLE LEGATE A QUESTA.</pre>
<p>3. Trovare i clienti intestatari di un conto insieme ad un altro cliente di cognome rossi.</p> <p>!! Ricorda alla fine di verificare che non stai controllando il cliente su sé stesso</p>	<pre>SELECT Cliente.CF FROM CLIENTE CL, INTESTAZIONE I WHERE(CL.CF = I.CLIENTE AND EXISTS (SELECT 1 FROM CLIENTE C1, INTESTAZIONE I1 WHERE C1.CF = I1.Cliente AND C1.Cognome = 'Rossi' AND I1.FilialeCC = I.FilialeCC AND I1.NumeroCC = I.NumeroCC AND I1.Cliente <> I.Cliente) #esiste un Rossi che ha un conto con lui</pre>

INTERROGAZIONI CON OPERATORI AGGREGATI

Salto fuori dal contesto di algebra relazionale.

Sono operatori che ci consentono di **costruire condizioni nella clausola select**. Vengono applicati non alla singola tupla, ma all'insieme di tuple risultato e possono agire su uno o più attributi di queste tuple. Producendo un solo valore: posso quindi, per esempio, calcolare massimo o minimo.

OPERATORI AGGREGATI STANDARD (SQL92)

I sistemi possono arricchire questo set di base.

VINCOLO SINTATTICO: Se la clausa SELECT contiene operatori aggregati NON PUO' contenere attributi singoli o espressioni su attributi.

```
SELECT FILIALE, NUMERO, MAX(saldo)
FROM CONTO
```

NO!!!! Non saprei che valore mettere su filiale e numero.

» **COUNT:** produce un conteggio.

» **Sintassi**

COUNT(*|[DISTINCT | ALL**] <listaAttributi>)**

» **Semantica**

- > **COUNT(*)** conta il numero di righe/tuple.
- > **COUNT(ALL <lista attributi>)** considera le combinazioni dei valori di attributi (es. se metto nome cognome conta solo nome cognome) e **restituisce il numero di combinazioni che non contengono valori nulli**; conteggio i valori diversi da null. Quindi sto assumendo che non siano tutti null =>P
- > **COUNT(DISTINCT <listaAttributi>)** conta come ALL ma considera solo le combinazioni distinte.

» **SUM, MAX, MIN, AVG**

» **Sintassi**

(SUM|MAX|MIN|AVG) ([DISTINCT | ALL**] <espressione>)**

» **Semantica:**

- > *****(**ALL <exp>**)**: Se uso ALL calcolo l'espressione sulle varie tuple, tolgo i valori nulli e poi applico l'operazione.
- > *****(**DISTINCTS <exp>**)**: elimina anche i duplicati prima di fare le cose.

Esempio tattico

```
CLIENTE(CF, Nome, Cognome, Prof, DataN, Città)
FILIALE(Codice, Nome, Indirizzo, Città)
CONTO(Filiale, Numero, Saldo)
INTESTAZIONE(FilialeCC, NumeroCC, Cliente)
MOVIMENTO(FilialeCC, NumeroCC, Num, Tipo, Data, Imp)
```

Trovare il saldo medio e il saldo massimo dei conti correnti delle filiali di Verona.

Mi serve Filiale nella select perché mi serve la città

```
SELECT MAX(C.Saldo), AVG (C.Saldo)
FROM Conto as C, Filiale as F
WHERE C.Filiale = F.Codice and F.Città = 'Verona'.
```

INTERROGAZIONI CON RAGGRUPPAMENTO

Consente di generare dei sottoinsiemi di tuple per poi applicare separatamente a questi sottoinsiemi gli operatori aggregati.

Come costruisco il sottinsieme?

GROUP BY

La suddivisione viene eseguita raggruppando insieme tutte le tuple che presentano gli stessi valori per un insieme di attributi assegnato

SINTASSI

```
SELECT <listaAttributi> FROM ... WHERE  
GROUP BY <Attributo> {, <Attributo>}  
ORDER BY ...
```

VINCOLO SINTATTICO:

<listaAttributi> può contenere **solo operatori aggregati applicati a espressioni**, oppure **attributi indicati nella clausola group by**.

SEMANTICA

- » Eseguiamo l'interrogazione di base
- » Si generano i gruppi di tuple in base agli attributi indicati nella clausola GROUP BY
- » Si applicano gli operatori aggregati ad ogni gruppo, producendo una tupla della relazione risultato per ogni gruppo.

CLAUSOLA HAVING

Dopo aver selezionato i gruppi mi permette di selezionarli (=toglierne qualcuno). Prende solo condizioni sul gruppo! Tipicamente immaginiamo di avere condizioni su operatori aggregati. (Es. solo gruppi >20).

```
SELECT <listaAttributi> FROM ... WHERE  
GROUP BY <Attributo> {, <Attributo>}  
HAVING <condizione_sel_gruppi>  
ORDER BY ...
```

<condizione_sel_gruppi> è un'espressione booleana dove le formule sono:

- » Operatore aggregato(<expr>) θ <costante>
- » Operatore aggregato(<expr>) θ <operatore aggregato(<expr2>)

Esempio tattico

```
CLIENTE(CF, Nome, Cognome, Prof, DataN, Città)  
FILIALE(Codice, Nome, Indirizzo, Città)  
CONTO(Filiale, Numero, Saldo)  
INTESTAZIONE(FilialeCC, NumeroCC, Cliente)  
MOVIMENTO(FilialeCC, NumeroCC, Num, Tipo, Data, Imp)
```

1. **Trovare il saldo medio e massimo dei conti correnti delle filiali della banca, riportando anche il nome della filiale.**

Per fare i gruppi devo utilizzare un meccanismo classico di ER: uso la superchiave! **Per mettere una cosa dentro SELECT esso deve apparire anche in GROUP BY.**

```
SELECT F.Nome, MAX(C.Saldo), AVG(C.Saldo)  
FROM Conto as C, Filiale as F  
WHERE C.Filiale = C.Codice  
GROUP BY F.Codice
```

<p>2. Trovare il nome e il cognome dei clienti che possiedono conti che complessivamente presentano un saldo medio < 1000 euro.</p> <p>Mi servono tutte e tre le tabelle: conto per fare il conteggio del saldo medio, cliente per avere nome e cognome, intestazione per la relazione fra cliente e conto.</p> <p>Dovrò fare join fra conto e cliente e conto e intestazione. !!! Conservo la cardinalità di intestazione! Tante righe tante quante sono le intestazioni. Per ogni cliente avrò tante righe quanti sono i conti.</p>	<pre>SELECT CL.Nome, CL.Cognome FROM Cliente as CL, Intestazione as I, Conto as C WHERE I.Cliente = CL.CF and I.FilialeCC = C.Filiale and I.NumeroCC = C.Numero GROUP BY CL.CF, CL.Nome, CL.Cognome HAVING AVG(C.Saldo)<100</pre>
<p>3. Trovare nome cognome e numero di conti aperti dai clienti di professione avvocato.</p> <p>Ci vuole count :)</p> <p>Quale variante? Count(*):</p>	<pre>SELECT CL.Nome, CL.Cognome, COUNT(*) as NumeroConti FROM Cliente as CL, Intestazione as I WHERE I.Cliente = CL.CF and CL.Prof = 'AVVOCATO' GROUP BY Cliente.CF, CL.Nome, CL.Cognome CLIENTE(CF, Nome, Cognome, Prof, DataN, Città) FILIALE(Codice, Nome, Indirizzo, Città) CONTO(Filiale, Numero, Saldo) INTESTAZIONE(FilialeCC, NumeroCC, Cliente) MOVIMENTO(FilialeCC, NumeroCC, Num, Tipo, Data, Imp)</pre>
<p>4. Trovare per ogni filiale di Milano con più di 500 conti correnti aperti il saldo massimo, riportando anche codice e nome della filiale.</p>	<pre>SELECT F.Nome, F.Codice, MAX(C.Saldo) FROM Filiale as F, Conto as C, Intestazione as I WHERE F.Codice = C.Filiale and F.Città = 'MILANO' GROUP BY F.Codice HAVING (COUNT(*)>500)</pre>
<p>5. Trovare il numero di conti aperti per ogni categoria professionale.</p> <p>Speriamo che chi gestisce il DB abbia scritto la stringa CLIENTE sempre allo stesso modo. :) In caso contrario avrò più gruppi (uno per ogni spelling).</p>	<pre>SELECT CL.Prof, COUNT(*) AS N_Conti FROM CLIENTE CL, INTESTAZIONE I WHERE CL.CF = I.Cliente GROUP BY CL.Prof</pre>

INTERROGAZIONE DI TIPO INSIEMISTICO

In SQL abbiamo già visto degli operatori di selezione/proiezione/etc. Tuttavia c'è anche una parte non del tutto coperta dall'SQL, ovvero quella degli operatori insiemistici!

Come simularle?

Possiamo riuscirci con gli operatori già visti (es. la join è un'intersezione, la differenza con operatori nidificati). **Manca solo l'unione.**

Dunque, il linguaggio è stato esteso per includerle anche direttamente

SINTASSI:

```
<selectSQL> { < UNION | INTERSECT | EXCEPT > [all]
                <selectSQL> }
```

Immagino di avere più interrogazioni SQL e di concatenarle con degli operatori nuovi. La parola chiave ALL mantiene i duplicati.

Intersect e except non aggiungono potere espressivo al linguaggio, poiché sono sostituibili via interrogazioni nidificate e join; la vera novità è l'unione.

VINCOLO SINTATTICO:

In algebra la union era solo fra schemi uguali. Qui il limite non è così forte, ma non posso legare tuple con schema troppo diverso... si chiede che abbiano **almeno lo stesso numero di attributi e che siano di tipo compatibile** (es. stringhe con stringhe, numeri con numeri).

Esempietto tattico

1. **Trovare l'elenco complessivo delle città di residenza dei clienti e di ubicazione delle filiali senza duplicati.**
2. **Trovare il saldo medio dei conti correnti intestati a clienti nati prima del 1/1/1970 e il saldo medio dei conti correnti intestati a clienti nati dopo l'1/1/1980.**

```
SELECT Città FROM CLIENTE  
UNION  
SELECT Città FROM FILIALE
```

```
SELECT 'Prima 1970', AVG(C.Saldo)  
FROM CONTO C, INTESTAZIONE I, CLIENTE CL  
WHERE C.Filiale = I.FilialeCC AND C.Numero=I.NumeroCC  
      AND CL.CF = I.Cliente AND CL.DataN < '1/1/1970'  
UNION  
SELECT 'Dopo 1980', AVG(C.Saldo)  
FROM CONTO C, INTESTAZIONE I, CLIENTE CL  
WHERE C.Filiale = I.FilialeCC AND C.Numero=I.NumeroCC  
      AND CL.CF = I.Cliente AND CL.DataN > '1/1/1980'
```

VISTE SQL

Consentono di **salvare delle interrogazioni** e renderele disponibili come se fossero tabelle virtuali. Si usano per definire le interfacce esterne: ad ogni specifico utente do accesso solo alla lista. Questo permette di gestire i permessi.

Inoltre, è anche un modo di cristallizzare i dati e rendere la struttura indipendente or smth (1.07)

SINTASSI

```
CREATE VIEW <nomeVista> [<lista attributi>]  
AS <selectSQL>
```

Note:

- » Le viste aggiungono potere espressivo poiché permettono di **applicare in cascata più operatori aggregati**
- » **Sempre virtuali:** Non è memorizzata esplicitamente: il sistema la genera ogni volta. (Almeno in SQL standard). In generale le liste non sono aggiornabili in quanto non sono fisiche.
- » **Non posso definire liste ricorsive**
- » Posso avere viste definite con altre viste purchè non vi sia dipendenza circolare

1. **Elenco delle città dove ho filiali.**

```
CREATE VIEW V1 AS (SELECT DISTINCT Città FROM
```

2. **Trovare nome e cognome dei clienti con il numero massimo di conti aperti.**

```
CREATE VIEW NumConti as  
SELECT C.CF, C.nome, C.cognome, count(*) AS conti  
FROM INTESTAZIONE I, CLIENTE C  
WHERE I.CLIENTE = C.CF  
GROUP BY C.CF, C.nome, C.cognome;
```

→ Dovrei applicare in cascata COUNT e MAX.

CLIENTE(CF, Nome, Cognome, Prof, DataN, Città)

FILIALE(Codice, Nome, Indirizzo, Città)

CONTO(Filiale, Numero, Saldo)

INTESTAZIONE(FilialeCC, NumeroCC, Cliente)

MOVIMENTO(FilialeCC, NumeroCC, Num, Tipo, Data, Imp)

3. Trovare per ogni filiale la città dove risiede il maggior numero di correntisti della filiale.

Calcolo count dei correntisti e faccio il max. Il problema è che qui il gruppo dipende dalla filiale E dalla città.

```
SELECT nome, cognome
FROM NumConti
WHERE
    conti = (SELECT MAX(conti) FROM NumConti);
```

```
CREATE VIEW NumCorr AS (SELECT I.FilialeCC, CL.Città, count(DISTINCT
I.Cliente) AS NCorr
FROM INTESTAZIONE I, CLIENTE CL
WHERE CL.CF = I.Cliente GROUP BY I.FilialeCC, CL.Città);
SELECT FilialeCC, Città FROM NumCorr T
WHERE NCorr IN (SELECT MAX(NCorr) FROM NumCorr Tbis
WHERE Tbis.FilialeCC = T.FilialeCC)
```

Approcci NoSQL

Queste soluzioni nascono a fronte di requisiti nuovi che si manifestano nei contesti applicativi con cui hanno a che fare le varie aziende. L'avvendo di Internet e di sistemi in grado di produrre su scala planetaria e con frequenze del tutto nuovi.

Esiste ancora la necessità di gestire dati “tradizionali” e le soluzioni viste vanno benissimo. Tuttavia esistono anche nuovi contesti, dove SQL non funziona a causa del fatto che il dato ha volume enorme e deve essere distribuito.

Nascono quindi nuovi approcci.

- Sistemi Key-Value: