

# Appunti Basi di Dati

## Lezione 1

### Sistema Informativo

Componente di una organizzazione che gestisce le informazioni di interesse - non tutte quelle disponibili, ma quelle di interesse! **Cosa fa?**

Gestisce la informazioni:

- acquisire → memorizzare → analizzare (interrogare, per dire che mi servono per fare qualcosa)
- elaborare
- produrre
- conservare
- condividere
- analizzare

Rientra tutto nel verbo **gestire**.

Il nostro sistema è la **BASE DI DATI** (ma in realtà io faccio riferimento a tutti i sistemi nella gestione di dati).

Ogni organizzazione ha un sistema informativo. Un'organizzazione è un termine generale, possiamo rappresentare qualsiasi insieme organizzato che ha necessità di gestire informazioni. Non per forza questo è esplicitato, soprattutto se l'organizzazione è piccola. Ma quasi sempre è di supporto ad altri sottosistemi: alla parte operativa, decisionale, di produzione, ecc.

Il *sistema organizzativo* è l'insieme di risorse e regole per lo svolgimento coordinato delle attività: se sono un'organizzazione avrò sicuramente un **obiettivo**. Le attività che mi permettono di raggiungere degli obiettivi fanno parte dei processi. Come organizzare le risorse? Se ne occupa il sistema organizzativo, non sempre identificabile.

Tra le risorse di un'azienda ci sono le informazioni. È per questo che le organizzazioni hanno un *sistema informativo*, ergo **imparare a gestire l'informazione è importante**.

Il concetto di "Sistema Informativo" non dipende per forza dall'automazione. Nella nostra accezione è sinonimo di **Sistema Informatico**, ma in realtà la gestione dell'informazione può essere indipendente dall'informatica e dall'automazione.



**Il Sistema Informatico è la porzione automatizzata del Sistema Informativo: gestisce informazioni con tecnologia informatica.**

### Gestione delle informazioni

Gestione: quello che posso fare con l'informazione (posso e devo non sono la stessa cosa):

- raccogliere
- acquisire
- conservare
- archiviare
- elaborare
- trasformare
- produrre
- distribuire
- comunicare
- scambiare
- proteggere

Cose che mi interessa sapere: Informazioni. Perché? Sono una risorsa.

Le basi di dati sono il sistema informativo che ci permette di immagazzinare i dati.

Nelle attività umane ci sono per forza delle informazioni, in forme diverse, sono qualsiasi cosa: idee, cose scritte, cose raccontate, è tutta informazione. Formati diversi. Ha vari supporti: mente, carta, dispositivi.

## I dati

Data la diversità dell'informazione, via via si è avuta la necessità di avere metodi di organizzazione e codificazione dell'informazione. Come tenerne traccia? **Estrapolare i dati di interesse e archiviare.**

Sottile differenza tra informazioni e dati: il dato è una rappresentazione dell'informazione. Per rappresentare un'informazione uso un certo insieme di dati: devono essere interpretati per fornire una determinata informazione. L'intuizione che mi porta al significato, ciò non vuol dire però che il dato sia completo.



I dati diventano informazione solo una volta che sono stati interpretati, altrimenti mi servono a poco.

*Per com'è costruito un modello relazionale avremo la spiegazione dei dati inseriti.*

Più dati ho, più risorse ho: rappresentarli bene e viceversa estrapolarli non è banale, quindi impareremo, data una descrizione delle informazioni di interesse, come posso rappresentare i dati in un certo contesto, a interrogare la base di dati.

## Base di dati

Una base di dati è:

- Accezione generica, metodologica:
  - Insieme organizzato di dati, utilizzati per il supporto allo svolgimento delle attività di un ente.
- Accezione specifica, metodologica e tecnologica:
  - Insieme di dati gestito da un Sistema di Gestione di Basi di Dati (DBMS)

### Cos'è un sistema di Gestione dei Basi di Dati?

Gestisce dati grandi, persistenti e condivisi (collezione di dati). Garantisce privacy, affidabilità, efficienza ed efficacia.

Cosa vuol dire che le collezioni sono grandi? Le dimensioni dei dati memorizzate è molto maggiore della memoria centrale dei sistemi di calcoli utilizzati. Il limite alla dimensione dei dati è solo quello fisico dei dispositivi: ho tanti dati, con l'evoluzione dei sistemi siamo arrivati a parlare di BigData, più passa il tempo più aumentano i dati da gestire.

Cosa vuol dire che devono essere persistenti? Hanno un tempo di vita indipendente dalle singole esecuzioni dei programmi che le utilizzano (le devo immagazzinare)

Cosa vuol dire che sono condivise? (spesso usiamo Basi di Dati come sinonimo dei dati stessi) spesso allo stesso insieme di dati accedono persone e settori diversi. Alla stessa base di dati accedono utenti diversi, con funzionalità diverse. Ognuno accede alla porzione che gli è di competenza (privacy).

Ho bisogno di meccanismi di autorizzazione: sembra ovvio ma devo controllare e il sistema deve sapere chi sta accedendo e con quali diritti, dal momento che ho più utenti che accedono alle stesse sezioni devo avere un controllo della concorrenza. Il DBMS mi permette di fare queste cose.

Potrei avere delle parti a cui determinate persone non possono accedere.

Cosa vuol dire che sono affidabili? Anche in caso di malfunzionamenti io non devo perdere dati, né essi devono essere danneggiati: è garantita la persistenza dei dati. È importante garantire affidabilità? Non mi posso permettere che vengano modificati o persi.

Cos'è una transazione? È un insieme di operazioni da considerare indivisibile, corretto, anche in presenza di concorrenza o mal funzionamento e con effetti definitivi. Grazie all'utilizzo delle transazioni ho il concetto di indivisibili: o arrivo a fare tutto o non posso fare nulla. Il concetto di insieme di operazioni indivisibili e atomiche vuol dire che devo per forza farle tutte insieme o tornare al punto di partenza. Le transazioni sono concorrenti: devo gestire le cose che concorrono senza perderne alcuna. Permanenti: la conclusione positiva di una transazione corrisponde ad un impegno, è definitivo. E il **DBMS** deve fare in modo che sia così.

Cosa vuol dire che garantisce efficienza? Cerca di utilizzare al meglio le risorse: spazi di memoria e tempo.

Cosa vuol dire che è efficace? Offre funzionalità articolate, potenti e flessibili.

## Descrizione dei dati nei DBMS

Due **modelli**: dei dati **relazionale** e **entità-relazione**. Si inseriscono a due diversi livelli di astrazione.

I modelli dei dati sono il modo in cui rappresento i dati. È un insieme di concetti che utilizzo per organizzare i dati di interesse e descriverne la struttura.

Lo uso per astrarre e concettualizzare i dati.

## Modello relazionale

Es. tabella relazione: il modello relazionale dei dati prevede il costruttore relazione, che permette di definire insiemi di record omogenei.

Ogni modello di dati fornisce meccanismi di strutturazione (o costruttori tipo) analoghi a quelli dei linguaggi.

Costruito = **RELAZIONE** (dentro la tabella è una rappresentazione tabellare. *Cosa decido delle tabelle? Cosa mi interessa*)

Decido la struttura del record in base ai dati di interesse. Una volta decisa la struttura, è quella. Per questo è importante pensare prima la struttura. A priori devo sapere cosa mi interessa memorizzare e una volta che lo so definisco la relazione. Dentro le relazioni ho **attributi**, che rappresentano la struttura dell'informazione.



Il **modello relazionale** è un modello logico, che usa il costrutto per rappresentare l'informazione. Il modello logico utilizza delle strutture che riflettono una particolare organizzazione.

## Modello entità-relazione

È un modello concettuale, ovvero astrae e si occupa dei concetti che voglio rappresentare.



I **modelli concettuali** descrivono i concetti del mondo reale piuttosto che i dati utili a rappresentarlo. Sono il primo passo per costruire il grafico per avere ben presente l'astrazione grafica delle informazioni.

Usiamo la parola relazione per rappresentare cose diverse: nel modello logico è la tabella, in quello concettuale è l'approccio di un'entità a un'altra.

La potenza di questo modello sta nell'essere sconnesso da ogni forma logica.



Dato il modello logico relazionale dei dati: a cosa si riferisce la relazione? dare una definizione di relazione.  
Cos'è un modello logico e cos'è un modello concettuale?

## Architettura di base

Noi interagiamo con uno schema logico, lo schema logico interagisce con uno schema interno.

**Indipendenza dei dati**: il livello logico è indipendente da quello fisico, che il sistema usi un modello o un altro non ho bisogno di sapere cosa c'è sotto. Il modello concettuale è usato per costruire quello logico (può sempre essere utile come documentazione, ma l'utente non ci interagisce quando usa la base di dati relazionale).

## Schema e Istanza

In ogni base di dati esistono uno schema e un'istanza.

Lo **schema** descrive la struttura della base di dati, sostanzialmente è invariante nel tempo. Modificare lo schema di una base di dati può causare problemi.

L'**istanza** sono i valori attuali di una base di dati: tutto ciò che ci inserisco, cambia continuamente nel tempo, anche molto rapidamente.

## Linguaggi per basi di dati

DDL (linguaggi per definizione di dati) e DML (Linguaggi di manipolazione dei dati)

La prima categoria definisce la tabella per com'è fatta: da una definizione dell'organizzazione dei dati. Permette la definizione di schemi e altre operazioni generali.

Il secondo permette di interrogare la base di dati e aggiornare un'istanza. SQL fa entrambe le cose ed è usato dalla maggior parte delle basi di dati.

## Il modello relazionale

È diventato il modello standard di riferimento, è disponibile dall'81 (quindi in dieci anni si è affermato come standard). Si basa sul concetto matematico di relazione con una variante. È basato sui valori: quindi anche i riferimenti fra i dati contenuti nelle relazioni sono rappresentato per mezzo dei valori stessi.

Le relazioni si rappresentano naturalmente come tabelle.

Duplica caratteristica: parte formale delle relazioni e rappresentazioni intuitive.

### Relazione

3 accezioni diverse.

#### 1. Relazione matematica, come da teoria degli insiemi

Cos'è la relazione matematica?

Dati  $n$  insiemi  $D_1, D_2, D_n$

- il prodotto cartesiano
- relazione matematica è un sottoinsieme del prodotto cartesiano
- Una relazione di  $n$  domini ha grado  $n$

Una relazione è un insieme e quindi: non è definito un ordinamento, sono distinte, ciascuna  $n$ -upla è ordinata. Sono definiti grado e cardinalità.

Ciascuno dei due domini ha un ruolo diverso in base alla posizione in cui si trova: la struttura è posizionale - questo è un po' limitante: per cui Cod associa ad ogni dominio un nome unico (chiamato attributo) che descrive il ruolo del dominio. L'ordine tra gli attributi è ora irrilevante e la struttura non è più posizionale. Ad ogni dominio quindi associo un nome dell'attributo che descrive il significato del dominio, che definisce la struttura della tabella.

#### 2. Relazione (relationship) che rappresenta una classe di fatti, nel modello entity-relationship; tradotto come associazione o correlazione

Una tabella rappresenta una relazione quando i valori di ogni colonna sono omogenei tra di loro e i domini sono unici (le righe sono diverse tra di loro) e l'ordinamento tra righe e colonne è irrilevante. Tabelle diverse nella stessa base di dati possono avere attributi uguali, ma fanno riferimento a tabelle diverse - creo legami tra le tabelle basati sul contenuto degli attributi (SUL VALORE NON SUL NOME)

Il modello relazionale è basato sui valori.

#### 3. Relazione secondo il modello relazionale dei dati



Quando una tabella rappresenta una relazione?

## Lezione 2

### Modello Relazionale

#### Schema Relazione

$R(X)$   $X = \{A_1, A_2, A_3, \dots, A_n\}$  è una relazione  $R$  su un insieme  $X$  di attributi

anche indicato come  $R(A_1, A_2, \dots, A_n)$  dove  $R$  è il nome della mia relazione e  $A_1, A_2, \dots, A_n$  sono un insieme di attributi.

Es. tabella  $\rightarrow$  STUDENTE(matriola, cognome, nome) e questo è lo schema della mia relazione. Non devo definire altro perché i valori di una determinata colonna sono omogenei rispetto al tipo.

Una base di dati è formata da più relazioni, quindi lo schema di una base di dati sarà:

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

dove  $R$  è il nome della base di dati e ogni  $R_i(X_i)$  è una relazione. Lo schema di una base di dati è formato da un insieme di schemi di relazioni.

## Istanza

Ogni riga dell'istanza si chiama **tupla**, una tupla è una **funzione**, definita su un insieme di attributi  $X$ , che mi permette di associare a ciascun attributo  $A$ , un valore del dominio di  $A$ .

$t[\text{matr}] = \text{VR01}$

Ogni riga della tabella si chiama tupla.

L'istanza della mia base di dati:

**(ISTANZA di) BD** (uso solo basi di dati ma faccio riferimento alla stessa cosa) è l'insieme dei valori che fanno riferimento alla mia base di dati.

$r = \{ \{(\text{VR01}, \text{Rossi}, \text{Paolo}), (\text{VR02}, \text{Verdi}, \text{Giorgia})\}(\text{BDBIO}, 6, \text{Basi di dati}, 30), (\text{SIA}, 6, \text{SisInformativi aziendali}, 10)\} \{(\text{VR01}, \text{BDBIO}), (\text{VR01}, \text{SIA}), (\text{VR02}, \text{BDBIO})\}$  questo è lo schema di tutti i valori della mia base di dati, la mia **istanza**.

$R = \{ \text{STUDENTE}(\text{matricola}, \text{nome}, \text{cognome}), \text{CORS}(\text{cod}, \text{CFU}, \text{titolo}, \text{StudFreq}), \text{FREQ}(\text{codStud}, \text{CodCorso}) \}$  e questo è lo **schema** della mia base di dati

Posso avere relazioni sui singoli attributi:

es.  $\text{RAPPRESENTANTE}(\text{MatrStud})$  ad esempio rappresentante è sulla matricola dello studente. La relazione di questa tabella è:

$\text{RAPPRESENTANTE}(\text{MatrStud})$

## VALORE NULLO

### Attenzione!

Posso avere una situazione in cui non tutte le tuple hanno tutti i valori per tutti gli attributi. Io impongo la struttura che devono avere i dati che devono essere inseriti, però non sempre tutti i dati sono disponibili.

*Cosa si fa in questo caso?*

Uso un valore nullo, perché non posso pensare di avere un valore di dominio. Deve essere riconoscibile il fatto che non ho il valore, non posso usare una stringa qualsiasi.



Il valore nullo denota l'assenza di un valore nel dominio e non è un valore del dominio.

Posso rappresentare 3 casi possibili:

#### 1. Il valore è sconosciuto

→ il valore esiste ma io non lo conosco

#### 2. Il valore è inesistente

→ il valore non esiste nella realtà modellata

#### 3. Il valore è senza informazione

→ il valore può esistere o non esistere e se esiste io non lo conosco (è un or dei due casi precedenti)

I DBMS fanno sempre riferimento al 3° caso, non fanno distinzione tra i diversi casi (vuol dire che non so niente!)

Alcuni valori nulli non mi creano alcun tipo di problema, ma altri possono creare vari problemi:

- Ad esempio può succedere che una tupla non possa mai essere correlata ad altre tuple (es. medico/paziente)
- oppure risultino del tutto inutili, non fornendo alcuna informazione
- oppure creino confusione tra le tuple

**C'È LA NECESSITÀ DI IMPORRE RESTRIZIONI SULL'USO DEI VALORI NULLI!**

Potrei anche avere dei dati sintatticamente corretti ma non ammissibili per l'applicazione (es. pazienti con lo stesso codice - identificazione univoca, oppure inesistente tra le possibilità che ho)

Per ovviare a questi problemi:

## Vincoli di integrità (referenziale)

Sono proprietà che devono essere soddisfatte dalle istanze che rappresentano informazioni corrette per l'applicazione.

Fanno riferimento a un certo dominio dell'applicazione, non sono universali, io devo sapere le regole da rispettare perché i dati siano inseriti nel mio DB.

A cosa servono i vincoli? Ci permettono di avere a che fare con dati di qualità!

Quali sono i tipi di vincoli di integrità?

Due categorie:

### 1. INTRARELAZIONALI

Riguardano una singola relazione

#### a. vincoli di tupla

la valutazione del vincolo avviene sulla singola tupla

#### b. vincoli su un singolo valore o di dominio

es. il voto non può essere maggiore di 30 (sul singolo valore)

#### c. **vincoli di chiave**

specifico qual è l'attributo o l'insieme di attributi, che mi permettono di identificare in maniera univoca una tupla, **molto importanti**

Quindi ci interessa riflettere su cos'è la chiave di una relazione!

È un insieme di attributi, che può essere anche uno solo, utilizzato per identificare in maniera univoca le tuple di una relazione.

### 2. INTERRELAZIONALI

Sono definiti tra più relazioni

## Lezione 3

## Vincoli di Chiave

Chiave = insieme degli attributi che mi permette di individuare in maniera univoca ogni tupla della relazione e deve garantire due proprietà.

### 1. UNICITÀ

In una qualunque istanza di una relazione, non possono esistere due tuple distinte la cui restrizione alla chiave sia uguale  
→ due tuple diverse non possono avere la stessa chiave.

$$t1[k] = t2[k] \leftrightarrow t1[x] = t2[x]$$

### 2. MINIMALITÀ

Non deve essere possibile sottrarre un attributo alla chiave senza che la proprietà di unicità cessi → la chiave è formata dall'insieme minimo di attributi che identificano la tupla e se ne tolgo uno non identifica più la chiave

## Chiave e Superchiave

Un insieme  $k$  di attributi è **superchiave** per una relazione  $r$ , se  $r$  non contiene due tuple distinte  $t1$  e  $t2$ , con  $t1[k] = t2[k]$

Questo insieme di attributi  $k$  diventa una **chiave** per la mia relazione  $r$ , se è una **superchiave minimale** per  $r$ .

Se so che un attributo è chiave allora qualsiasi insieme di attributi che lo contengono non sarà sicuramente chiave (poiché è superchiave, ma sicuramente non minimale).

## Individuare una chiave

Quando definisco uno schema, devo ricordare che sto definendo la struttura dei dati che devo inserire, e vorrò definire uno schema generale che non sia vincolato dalle informazioni che so per certo ci finiranno dentro → la scelta a priori è una scelta generale (es. scegliere cognome e nome come chiave di una tabella non è opportuno).

Nella definizione dello schema della mia relazione, per individuare una chiave (specificare quali sono i vincoli di chiave che voglio soddisfare), devo:

- considerare le **proprietà dei dati**

- frammento del mondo reale di interesse → tra tutti gli attributi che caratterizzano l'informazione prendo solo quelli importanti per ciò che sto gestendo.
- Cerco di stabilire quali sono i dati che mi interessano e poi cerco ciò che può individuare univocamente una tupla (in generale, non basandosi sulla propria istanza) → quali insiemi di attributi → non esistendo due tuple uguali, di base so per certo che l'insieme di attributi che ho definito è una superchiave.



Cose da tenere presente:

1. Poiché le relazioni sono **insiemi**, una relazione non può contenere due tuple distinte ma uguali
2. Ogni relazione ha come superchiave l'insieme degli attributi su cui è definita. Quindi ha **almeno** una chiave.

### Perché è così importante avere una chiave?

- Permette di accedere alle tuple → garantisce l'accessibilità a ciascun dato della mia relazione e quindi a ciascun dato della BD
  - Ogni valore della base di dati è accessibile tramite il nome della relazione, il valore della chiave e il nome dell'attributo
- Le chiavi ci permettono di correlare i dati in relazioni diverse, poiché il modello relazionale dei dati è basato sui valori.

### Chiavi e Valori Nulli

In presenza di valori nulli, i valori della chiave non ci permettono di identificare le n-uple e non posso realizzare i riferimenti.

Quindi la presenza di valori nulli nelle chiavi deve essere limitata.

Si chiama **Chiave Primaria** la chiave su cui non sono ammessi valori nulli.



notazione:

STUDENTE(matricola, cognome, nome, dataNascita)

se sottolineato → chiave primaria

La chiave primaria ovviamente cambia a seconda della realtà che sto rappresentando, così come il concetto di ciò che sto rappresentando.

## Vincoli di integrità referenziale

### ("Foreign key" o "vincoli di chiave esterna")

Le informazioni in relazioni diverse della mia base di dati sono correlate attraverso valori comuni, in modo coerente (rappresentano la stessa informazione).

In particolare i valori comuni a cui faccio riferimento sono i valori delle chiavi primarie ma non solo.

Si chiama **Vincolo di Integrità Referenziale** fra gli attributi X di una relazione  $r1$  ed  $r2$ , un vincolo che impone ai valori su X di  $r1$  di comparire come valori di chiave primaria di  $r2$ .

→ es. esiste un vincolo di integrità referenziale tra l'attributo codice medico della tabella paziente e la tabella medico:

**Paziente**(Codice, Cognome, Nome, dataNascita, *codMedico*)

**Medico**(*Codice*, Cognome, Nome, Telefono, Specializzazione, *Reparto*)

**Reparto**(*Codice*, Descrizione)

ed esiste quindi un vincolo d'integrità referenziale tra l'attributo Reparto della tabella medico e la tabella Reparto.  **$r1$  è la tabella interna,  $r2$  è la tabella esterna.**

È importante proprio perché il modello relazionale è basato sui valori. L'importante è controllare che quando inserisco qualcosa nella tabella interna, esiste nella tabella esterna, non il contrario. L'insieme da cui traggio (la tabella esterna) può essere più grande, ma mai più piccolo.

**Importanza:**

1. Giocano un ruolo fondamentale nel concetto di modello basato sui valori
2. I riferimenti sono diretti verso le chiavi primarie
3. In presenza di valori nulli, possono essere resi meno restrittivi → potrei avere dei passaggi caricando
4. Vincoli su più attributi, conta l'ordine (primo con primo, secondo su secondo, ...)
5. Si possono definire delle azioni compensative a seguito della loro violazione → devo aver definito dei meccanismi che mi dicano cosa fare in funzione della violazione del vincolo.

## Violazione del vincolo di integrità referenziale

Si possono introdurre violazioni al vincolo di integrità referenziale, **modificando** la tabella interna o la tabella esterna (intendo che modifico l'istanza)

/aggiungere tabelle per vincoli integrità referenziale

1. Violazioni a fronte di una modifica della tabella **interna**
  - a. modificando attributo **referente**
  - b. inserimento di una **nuova** riga che viola il vincolo
    - queste operazioni vengono **impedite**
2. Violazioni a fronte di una modifica della tabella **esterna**
  - a. modifica attributo **riferito**
  - b. **cancellazione** di una tupla
    - queste operazioni hanno bisogno di **definire delle azioni compensative**

## Azioni compensative

- a. Modifica attributo riferito
  - Modifica a cascata
  - Introduzione di NULL
  - Introduzione di un valore di default
- b. Cancellazione di una tupla
  - Cancellazione a cascata
  - Introduzione di NULL (meno pericoloso)
  - Introduzione di un valore di default

## Esercizio

Definire uno schema di basi di dati (con il modello relazionale dei dati) per organizzare le informazioni di una clinica privata:

- infermieri(*codiceFiscale*, nome, cognome, dataNascita)
- medici(*codiceFiscale*, nome, cognome, dataNascita, specializzazione)
- reparti(codicePiano, *caporeparto* (infermiere), *primario* (medico))

Ogni infermiere e ogni medico lavorano in un reparto. Indicare le chiavi e i vincoli di integrità referenziali dello schema e costruire una istanza della base di dati.

### Vincoli di integrità:

C'è un vincolo di integrità referenziale tra l'attributo Codice Fiscale (riferito) della tabella Infermiere e l'attributo caporeparto (rifer

C'è un vincolo di integrità referenziale tra l'attributo Codice Fiscale (riferito) della tabella Medico e l'attributo primario (riferente) della tabella Reparto

C'è un vincolo di integrità referenziale tra l'attributo codicePiano (riferito) della tabella Reparto e codReparto della tabella Inferm

idem per quel che riguarda la tabella Medico

INFERMIERE.codReparto → Reparto (non specifico nulla perché è chiave primaria)



MEDICO.codReparto → Reparto

REPARTO.Caporeparto → Infermiere

REPARTO.Primario → Medico

## Lezione 4

### Operatori Insiemistici

$r_1$  ed  $r_2$  devono essere compatibili, quindi devono avere lo stesso grado (stesso numero di attributi), domini ordinatamente dello stesso tipo in modo da avere tuple compatibili.

#### Unione

L'unione di due relazioni  $r_1$  ed  $r_2$ , definite sullo stesso insieme di attributi  $X$ ,  $r_1 \cup r_2$ , è una relazione su  $X$ . Il risultato contiene tutte le tuple che appartengono ad  $r_1$  oppure tutte le tuple che appartengono ad  $r_2$  oppure ad entrambe.

Ad esempio ipotizzando un'unione tra la tabella docenti e la tabella studenti, e sempre ipotizzando di avere uno studente che è anche docente, avrò una tabella che contiene tutti i docenti e tutti gli studenti e il docente-studente che appare solo una volta.

#### Intersezione

L'intersezione di due relazioni  $r_1$  ed  $r_2$ , definite sullo stesso insieme di attributi  $X$ ,  $r_1 \cap r_2$ , è ancora una relazione su  $X$ . Il risultato contiene solo le tuple che appartengono sia ad  $r_1$  che ad  $r_2$ .

Rifacendoci all'esempio in cui abbiamo uno studente-docente, l'unica tupla presente nella tabella della relazione sarà quello studente-docente.

#### Differenza

La differenza di due relazioni  $r_1$  ed  $r_2$ , definite sullo stesso insieme di attributi  $X$ ,  $r_1 - r_2$ , è ancora una relazione su  $X$ . Il risultato contiene le tuple di  $r_1$ , che non appartengono ad  $r_2$ .

Sempre nello stesso esempio avremo una tabella come quella studenti, ma senza lo studente-docente, poiché questo appartiene anche ai docenti.

### Attributi diversi e ridenominazione

Sarebbe sensato alle volte voler costruire determinate unioni, ma sorge il problema di avere relazioni non definite sullo stesso insieme di attributi. Abbiamo quindi bisogno di quella che si chiama **Ridenominazione** è un operatore unario, cioè che lavora sulla singola relazione, che modifica lo schema lasciando inalterata l'istanza dell'operando, permettendomi di rendere compatibili gli schemi.

Sia  $r$  una relazione su un insieme di attributi  $X$ , e  $Y$  un altro insieme di attributi con la stessa cardinalità.



Siano  $A_1, A_2, \dots, A_k$  e  $B_1, B_2, \dots, B_k$ , rispettivamente un ordinamento per gli attributi di  $X$  e di  $Y$ , allora la  $\rho_{B_1, B_2, \dots, B_k \leftarrow A_1, A_2, \dots, A_k}(r)$

Nell'esempio padre e madre, li ridenominò  $\rho_{\text{GENITORE}} \leftarrow \text{PADRE}(\text{Paternità})$  e

$\rho_{\text{GENITORE}} \leftarrow \text{MADRE}(\text{Maternità})$ .

Risultato: MATERNITÀ(genitore, figlio) e PATERNITÀ(genitore, figlio).

Ora posso fare l'unione delle due tabelle.

### Selezione

È un operatore unario, che produce un risultato che ha lo stesso schema della tabella di partenza, e contiene un sottoinsieme delle tuple della tabella di partenza, quindi dell'operando, che soddisfano una certa condizione.

La selezione produce quindi un risultato sullo stesso schema in cui però son state tagliate alcune tuple, perché non soddisfano una condizione.



Si dice che la selezione genera una decomposizione orizzontale.

$\sigma_{condizione}(operando)$

- **Condizione** di selezione: la condizione è ciò che le tuple devono soddisfare per finire nel risultato
- L'**operando** è la relazione su cui applico la selezione

Esempio: condizione: sulla tabella studenti, solo quelli che hanno età maggiore di 25

$\sigma_{età > 25}(STUDENTI)$

### Condizioni che posso specificare

- Confronti tra attributi  $A_1 O A_2$ , dove  $O$  può essere  $=, \neq, <, >, \leq, \geq, \dots$
- confronto tra un attributo e una costante  $k$ 
  - $A_1 O k$ , dove  $O$  può essere  $=, \neq, <, >, \leq, \geq, \dots$
- Condizioni complesse ottenute combinando condizioni semplici con connettori logici
  - $Cond_1 \wedge Cond_2$
  - $Cond_1 \vee Cond_2$
  - $\neg Cond_1$

$\sigma_{stipendio < 50}(Impiegato)$

### Proiezione

Operatorie unario che produce un risultato, il quale NON ha lo stesso schema dell'operando. Lo schema del risultato è un sottoinsieme degli attributi di partenza.



Si dice che la proiezione genera una decomposizione verticale.

$\Pi_{ListaAttributi}(Operando)$

Il risultato di una proiezione contiene al più tante tuple quanto l'operando ma può contenerne di meno: se due tuple hanno attributi proiettati uguali, avrò una sola tupla (non ho duplicati).

### Cardinalità delle Proiezioni

La cardinalità della proiezione è  $\leq$  alla cardinalità dell'operando.

Esiste un legame tra i vincoli di chiave e le proiezioni.

$\Pi_y r$  contiene lo stesso numero di tuple di  $r$  se e solo se  $y$  è una **superchiave** di  $r$ .

Se  $y$  è una superchiave allora  $r$  non contiene tuple uguali su  $y$  e quindi ogni tupla di  $r$  su  $y$  dà un contributo al risultato.

Nel momento in cui proietto la chiave sono sicura che tutte le tuple finiscano nel risultato.

### Combinare selezione e proiezione

Prima vado a verificare in quali tuple applico la selezione ed elimino quelle che non rispettano la condizione. Date le tuple che rispettano la selezione, vado a prendere gli attributi che proietto.

$\Pi_{cognome, matricola}(\sigma_{stipendio > 50}(Impiegato))$

Faccio prima la selezione e poi la proiezione, altrimenti perderei le informazioni su cui operare la selezione.

Posso scriverlo anche in **formato ad albero**:

$\Pi_{matricola, cognome}$

|

$\sigma_{stipendio > 40}$

|

La logica dell'algebra relazionale è che sono io a stabilire quali operazioni fare sulle tabelle per trovare le cose che sto cercando.

Nell'esame ci sono 3 operazioni di algebra relazionale

## Esercizi

### Domande di ripetizione

Per associare più cose ad una sola ho bisogno di una relazione esterna

1. **Cos'è il modello relazionale?** Un modello di dati logico, che si basa su un tipo di costruito che è la relazione
2. **Perché è importante?** Perché ci permette di organizzare i dati in una struttura
3. **Cos'è la schema di una relazione?** È dato dal nome di una relazione e l'insieme degli attributi, con nomi tutti diversi. All'interno di una base di dati posso avere più relazioni che hanno attributi con lo stesso nome (uso anche la relazione per riconoscere l'attributo)
4. **Il modello relazionale è basato sui valori. Che vuol dire?** I collegamenti tra relazioni diverse si basano sui valori degli attributi, il valore dell'attributo è UNO SOLO, non posso avere più valori.
5. Per avere la **garanzia che i dati sono coerenti** usiamo i vincoli di integrità referenziale, specificando quando modelliamo il data base quali vincoli vogliamo che vengano rispettati. L'attributo di tot. tabella fa riferimento ad un'altra (chiave primaria) della tabella esterna. Tutto ciò che appare nell'attributo referente nella tabella interna, deve essere un valore di una chiave di una tabella interna. Per evitare i problemi, specifico delle azioni di gestione della violazione dei vincoli referenziali.

### Domande da esame:

1. Dato il modello relazionale dei dati, cos'è una relazione?
2. Cos'è lo schema e cos'è l'istanza di una relazione?
3. Cos'è la chiave di una relazione?
4. Quando posso parlare di chiave? (quando ho una superchiave minimale)
5. Quando una superchiave è primaria? (quando non può avere valori nulli)



Esercizi svolti sul quaderno

## Lezione 5

### Join

simbolo:  $\bowtie$

Permette di correlare i dati che sono contenuti in relazioni diverse, confrontando i valori contenuti in esse

Sfrutta le proprietà del modello di essere basato sui valori.

#### Join Naturale

- operatore binario
- collega dati in relazioni diverse sulla base di valori uguali in attributi con lo stesso nome

Produce un **risultato**

- sull'unione degli attributi delle due relazioni di partenza (gli operandi)

- le tuple sono costruite ciascuna a partire da una tupla di ognuno degli operandi, combinando le tuple con valori uguali negli attributi comuni



Il grado ottenuto con un join è  $\leq$  della somma dei gradi degli operandi

## Join Completo

Ciascuna tupla di ciascun operando contribuisce ad almeno una tupla del risultato

## Join Non Completo

Alcune tuple (dangling) degli operandi non contribuiscono al risultato

- Perché?  
Nell'altra relazione non ha il valore corrispondente nell'attributo con lo stesso nome  
esempio: in una tabella è citato qualcosa che nell'altra non c'è e quindi quella tupla non partecipa al join

## Join Vuoto

Nessuna tupla degli operandi è compatibile

## Cardinalità

- **Join completo:**  $|r_1 \bowtie r_2| \geq \max(|r_1|, |r_2|)$
- **Join non completo:**  $|r_1 \bowtie r_2| < |r_1 \times r_2|$
- **Join Vuoto:** 0
- **Join Naturale Completo:**  $|r_1 \bowtie r_2| < |r_1 \times r_2|$

Il join naturale di  $r_1$  ed  $r_2$  contiene un numero di tuple compreso tra 0 e il prodotto delle cardinalità dei due operandi

$$0 < r_1 \bowtie r_2 < r_1 \times r_2$$

- In particolare, se il join è completo, allora contiene almeno un numero di tuple pari al massimo tra  $r_1$  ed  $r_2$
- se dati:  $r_1(A, B)r_2(B, C)$ ,  $r_2B$  è chiave allora  
 $0 \leq |r_1 \bowtie r_2| \leq |r_1|$
- Se c'è anche il vincolo di integrità referenziale  $r_1B \rightarrow r_2$  allora  
 $|r_1 \bowtie r_2| = |r_1|$

## Join Esterno

Posso prevedere che tutte le tuple degli operandi finiscono nel risultato:  
per farlo estende con valori nulli le tuple che vengono tagliate fuori.

3 tipi di join esterni:

- sinistro: mantiene tutte le tuple del primo operando  $r_1 \bowtie_{LEFT} r_2$
- destro: mantiene tutte le tuple del secondo operando  $r_1 \bowtie_{RIGHT} r_2$
- completo: mantiene tutte le tuple  $r_1 \bowtie_{FULL} r_2$

## Proprietà del join naturale

- è commutativo  $r_1 \bowtie r_2 = r_2 \bowtie r_1$
- è associativo:  $r_{\{1\}} r_1 \bowtie (r_2 \bowtie r_3) = (r_1 \bowtie r_2) \bowtie r_3 = r_1 \bowtie r_2 \bowtie r_3$

$r_1(x_1) \bowtie r_2(x_2)$  due casi estremi:

- $x_1 = x_2$   
Quando è questo il caso il Join corrisponde con l'intersezione:  
 $r_1(x_1) \bowtie r_2(x_2) = r_1(x_1) \cap r_2(x_2)$

- $x_1 \cap x_2 \neq 0$

In questo caso, degenera in una condizione sempre vera, quindi il risultato del join contiene tutte le tuple ottenute combinando in tutti i modi possibili le tuple degli operandi.

$$x_1 \cup x_2$$

$$|r_1(x_1) \bowtie r_2(x_2)| = |r_1(x_1)| \times |r_2(x_2)|$$

diventa un join naturale sulle relazioni senza attributi in comune:  
per trovare le tuple correlate da un attributo, faccio seguire una **selezione**

$$\sigma_{\text{reparto}=\text{codice}}(\text{Impiegato} \bowtie \text{Reparto})$$

## Theta Join

Definito come un prodotto cartesiano seguito da una selezione

- è un operatore derivato dagli operatori



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

La condizione di selezione diventa la condizione sul join

I DBMS utilizzano il theta join e non il join naturale!

Quando nella condizione uso l'uguaglianza parlo di **EquiJoin**

### Esercizio:

date le tabelle IMPIEGATO e SUPERVISIONE, ottenere le matricole dei capi degli impiegati che guadagnano più di 40

$$\Pi_{\text{capo}}(\text{SUPERVISIONE} \bowtie_{\text{impiegato}=\text{matricola}} (\sigma_{\text{stipendio}}(\text{IMPIEGATO})))$$

## Lezione 6

### Equivalenza delle espressioni algebriche

Un'espressione potrebbe essere ottimizzata rispetto ad un'altra, avendo un'equivalenza posso sostituire ed ottimizzare tutte le espressioni.

Logica: eliminare il prima possibile tutto ciò che posso eliminare ed evitare di portarmi dietro.



Due espressioni sono equivalenti se producono lo stesso risultato.

Come nell'algebra normale

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

Abbiamo due tipi di equivalenza nell'algebra relazionale:

- **Equivalenza assoluta**

- Indipendente dallo schema delle relazioni

$$\text{exp}_1 \approx \text{exp}_2 \text{ se } e_1 \approx e_2 \text{ per un insieme } R$$

Esempio:

$$\Pi_{ab}(\sigma_{a>0}(R)) \approx \sigma_{a>0}(\Pi_{ab}(R))$$

- **Equivalenza che dipende dallo schema**

$$\text{exp}_1 \approx \text{exp}_2 \text{ se } e_1 r \approx e_2 r \text{ per ogni istanza } r \text{ di } R$$

Esempio:

$$\Pi_{ab}(R_1) \bowtie \Pi_{ac}(R_2) \approx \Pi_{abc}(R_1 \bowtie R_2)$$

Vale solo se l'intersezione tra gli attributi di  $R_1$  ed  $R_2$  è uguale ad  $a$ .

### Anticipazione della selezione rispetto al join - pushing selection down

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

Il punto è evitare di fare il join su più tuple di quante non ne siano necessarie → mi riduce il numero di tuple che mi porto dietro.

La selezione sulla condizione  $F$  coincide con il join di  $E_1$  e la selezione sulla condizione  $F$  di  $E_2$

Questo ovviamente vale se la selezione  $F$  fa riferimento ad attributi di  $E_2$ .

### Anticipazione della proiezione rispetto al join - pushing projection down

$$\Pi_{x_1, y_2}(E_1 \bowtie E_2) \approx E_1 \bowtie \Pi_{x_1, y_2}(E_2)$$

### Valori nulli

Le condizioni possono essere vere solo per i valori diversi da NULL, le tuple con NULL vengono ovviamente escluse dalle selezioni.

Il join di tuple su cui è avvenuto questo tipo di selezione, ritornano tuple diverse dal totale delle tuple che potrei volere.

Possiamo selezionare attributo is NULL!

Esercizio:

**STUDENTE**(Matricola, Cognome, Nome, DataNascita)

**ESAME**(MatricolaStudente, CodiceInsegnamento, Voto, Data)

**INSEGNAMENTO**(Codice, Titolo, CodDocente, AnnoAccademico)

**DOCENTE**(CodFiscale, Cognome, Nome, DataNascita)

Vincoli di integrità referenziale:

- MatricolaStudente<sub>ESAME</sub> → Matricola<sub>STUDENTE</sub>
- CodiceInsegnamento<sub>ESAME</sub> → Codice<sub>INSEGNAMENTO</sub>
- CodDocente<sub>INSEGNAMENTO</sub> → CodFiscale<sub>DOCENTE</sub>

Es.1

Trovare cognome, nome e data di nascita di docenti e studenti:

$$\Pi_{cognome, nome, datanascita}(DOCENTE) \cup \Pi_{cognome, nome, datanascita}(STUDENTE)$$

Es.2

Trovare cognome e nome degli studenti nati nel 1985

$$\Pi_{cognome, nome}(\sigma_{datanascita >= 1/01/1985 \wedge datanascita <= 31/12/1985}(STUDENTE))$$

## Lezione 7

### Linguaggio SQL - Structured Query Language

Definito nel 1973, linguaggio di interrogazione per definizione del modello relazionale → Linguaggio universale dei sistemi relazionali.

Varie funzionalità:

- **DDL - Data definition language**
- **DML - Data manipulation language**

L'istruzione per la definizione dei dati (definizione dello schema della tabella) è l'istruzione `create table` definire lo schema della tabella e crearne un'istanza vuota, posso specificare quali sono gli attributi, i domini e i vincoli.

Es:

```
CREATE TABLE Impiegato (  
    Matricola CHAR(6),  
    Nome VARCHAR(20),  
    Cognome VARCHAR(20),  
    Qualifica VARCHAR(20),  
    Stipendio FLOAT);
```

Abbiamo definito: IMPIEGATO(Matricola, Nome, Cognome, Qualifica, Stipendio)

Quindi: nome + insieme degli attributi con il loro tipo.

In SQL per la definizione dei domini possiamo usarne alcuni elementari predefiniti, che sono analoghi a quelli dei linguaggi di programmazione (Caratteri CHAR, Stringe di lunghezza fissa CHAR(lenght), Stringhe di lunghezza variabile VARCHAR(lenghtmax), ecc.), posso avere tutti i tipi numerici, i tipi per le date, booleani, ecc.

Posso usare anche domini definiti dall'utente.

## Vincoli

- **Intrarelazionali**

- interni, soddisfatti da tutte le istanze
- In SQL:

```
/* NOT NULL --il valore nullo non è ammesso per l'attributo  
UNIQUE --il valore è superchiave  
PRIMARY KEY --il valore è chiave primaria  
CHECK  
*/  
CREATE TABLE Impiegato (  
    Matricola CHAR(6) UNIQUE,  
    Nome VARCHAR(20),  
    Cognome VARCHAR(20),  
    Qualifica VARCHAR(20),  
    Stipendio FLOAT,  
    UNIQUE(Nome, Cognome)); --la coppia nome cognome è unica  
  
--non ci devono essere due righe che hanno lo stesso nome e cognome:  
CREATE TABLE Impiegato (  
    Matricola CHAR(6) UNIQUE,  
    Nome VARCHAR(20) NOT NULL,  
    Cognome VARCHAR(20) NOT NULL,  
    Qualifica VARCHAR(20),  
    Stipendio FLOAT,  
    UNIQUE(Nome, Cognome));  
--è diverso da:  
    Nome VARCHAR(20) NOT NULL UNIQUE,  
    Cognome VARCHAR(20) NOT NULL UNIQUE,  
/*  
sopra sono i singoli valori a dover essere unici  
non possono esistere due persone con lo stesso nome,  
anche se hanno cognomi diversi  
*/  
  
--VINCOLO DI CHIAVE PRIMARIA  
CREATE TABLE Impiegato (  
    Matricola CHAR(6) PRIMARY KEY,  
    Nome VARCHAR(20),  
    Cognome VARCHAR(20),  
    Qualifica VARCHAR(20),  
    Stipendio FLOAT,  
    UNIQUE(Nome, Cognome)  
)  
/*  
--posso ovviamente usarlo solo una volta  
--implica un vincolo not null  
--possiamo definirla su un solo attributo  
--o su più attributi  
*/  
PRIMARY KEY(Nome, Cognome)  
  
--FINALE:  
CREATE TABLE Impiegato (  
    Matricola CHAR(6) PRIMARY KEY,  
    Nome VARCHAR(20) NOT NULL,  
    Cognome VARCHAR(20) NOT NULL,  
    Qualifica VARCHAR(20) NOT NULL,
```

```

    Stipendio FLOAT DEFAULT 100,
--nel caso in cui non sia specificato non inserisce NULL ma 100
    UNIQUE(Nome, Cognome)
    CHECK(Stipendio>=100)
)

/*
il valore dell'attributo stipendio deve essere sempre >= a 100

*/

```

## Come inserire i Dati

Istruzione: INSERT

```

INSERT INTO Impiegato(Matricola, Nome, Cognome, Qualifica)
VALUES( NR3468, 'Giovanni', 'Neri','Manager')
INSERT INTO Impiegato(Matricola, Nome, Cognome, Stipendio)
VALUES( VR3289, 'Maria', 'Gialli', 200.00)

```

### • Interrelazionali

Coinvolgono più di una tabella, sono i vincoli di **integrità referenziale**.

Costrutto: **Foreign Key** (*chiave esterna*)

Crea un legame tra i valori dell'attributo della tabella interna e i valori dell'attributo della tabella esterna.

```

IMPIEGATO(Matricola,Nome,Dipartimento)
DIPARTIMENTO(NomeDip, Descrizione, Indirizzo)

```

- Con Dipartimento → NomeDip
- L'attributo a cui faccio riferimento è o chiave primaria o essere specificato con un vincolo di UNIQUE.

Oltre a foreign key posso usare anche **References**:

Quando il vincolo è definito su un unico attributo

```

CREATE TABLE Dipartimento (
    NomeDip VARCHAR(15) PRIMARY KEY,
    Sede VARCHAR(20) NOT NULL,
    Telefono VARCHAR(15)
)

CREATE TABLE Impiegato (
    Matricola VARCHAR(6) PRIMARY KEY,
    Nome VARCHAR(20) NOT NULL,
    Cognome VARCHAR(20) NOT NULL,
    Dipartimento VARCHAR(15) REFERENCES Dipartimento(NomeDip)
)
--all'interno di impiegato specifico il vincolo di integrità referenziale

```

```

CREATE TABLE Dipartimento (
    NomeDip VARCHAR(15) PRIMARY KEY,
    Sede VARCHAR(20) NOT NULL,
    Telefono VARCHAR(15)
)

CREATE TABLE Anagrafica (
    CF CHAR(11) PRIMARY KEY,
    Nome VARCHAR(20) NOT NULL,
    Cognome VARCHAR(20) NOT NULL,
    Indirizzo VARCHAR(15),
    UNIQUE(Nome, Cognome)
)

CREATE TABLE Impiegato (
    Matricola VARCHAR(6) PRIMARY KEY,
    Nome VARCHAR(20) NOT NULL,
    Cognome VARCHAR(20) NOT NULL,
    Dipartimento VARCHAR(15) REFERENCES Dipartimento(NomeDip)
)
FOREIGN KEY(Nome, Cognome) REFERENCES Anagrafica(Nome, Cognome)

```



## Politiche di reazione alla violazione di un vincolo

Possiamo introdurre delle violazioni modificando il contenuto della tabella interna - il sistema impedisce l'inserimento

- modificare il valore dell'attributo referente
- inserire una nuova riga

Se tocco la tabella esterna:

- posso inserire delle violazioni per modifica dell'attributo riferito
- cancello una riga

Politiche per modifica dell'attributo riferito: `cascade`

Il nuovo attributo viene riportato su tutte le righe corrispondenti della tabella interna.

Es. cambio il nome del dipartimento da Informatica a Informatica e Matematica

La modifica ricade su tutti i riferimenti a quel nome della tabella interna, quindi tutti quelli che facevano riferimento al nome del dipartimento Informatica adesso lo fanno ad Informatica e Matematica.

## Lezione 8

## SQL - Modificare gli schemi delle tabelle

Create table crea lo schema di una relazione.

- se dobbiamo aggiungere un attributo, l'istruzione da utilizzare è `ALTER TABLE NomeTabella`
  - per aggiungere una colonna `ADD COLUMN NomeAttr Tipo`

```
ALTER TABLE Impiegato
ADD COLUMN Nazionalità VARCHAR(10)
```

- per rimuovere un attributo `DROP COLUMN NomeAttributo`
- per aggiungere un nuovo vincolo

```
ALTER TABLE NomeTabella
ALTER COLUMN NomeAttributo
ADD CONSTRAINT DefVincolo
```

- per rimuovere un vincolo

```
ALTER TABLE NomeTabella
ALTER COLUMN NomeAttributo
DROP CONSTRAINT Vincolo
```

- cancellare una tabella

il comando è `DROP TABLE NomeTabella`

## Operazioni sui dati

- Inserimento di tuple `INSERT`
- Cancellazione di tuple `DELETE`
  - Devo stare attento a non violare vincoli cancellando delle righe nella tabella

```
INSERT INTO
NomeTabella [(ElencoAttributi)]
VALUES (Elenco valori)
DELETE FROM
NomeTabella
[WHERE Condizione]
```

- Aggiornamento di valore nelle tuple **UPDATE**

```
UPDATE NomeTabella
SET NomeAttr1 = Esp1
    NomeAttr1 = Esp2
WHERE Condizione
```

## Interrogazioni in SQL - Query

- Tramite **SELECT**

- Sintassi:

```
SELECT Attributo {Attributo}
FROM NomeTabella{, NomeTabella}
WHERE Condizione
```

- La lista di attributi si chiama anche Target List ed è l'insieme di attributi che voglio nel risultato.
- Invece sotto ho una lista di tabelle da cui andare a cercare le informazioni
- La condizione invece è quella da rispettare perché compaiano nel risultato
- SQL è un linguaggio dichiarativo: dichiaro la serie di attributi, le tabelle da cui prenderle e la condizione da rispettare, non mi interessa il modo in cui questo viene fatto:
  - Opera il prodotto cartesiano tra le tabelle nella clausola **FROM**
  - Considera solo le righe che soddisfano la clausola **WHERE**
  - Per ogni riga valuta la clausola **SELECT**
  - L'asterisco (\*) rappresenta la selezione di tutti gli attributi

```
SELECT *
FROM Corso

--

SELECT Matricola, Nome
FROM Studente
```

Ad esempio se vogliamo visualizzare il codice e il nome dei corsi da tre crediti:

```
oSELECT Codice, NomeCorso
FROM Corso
WHERE NumeroCrediti = 3
```

Nella clausola **WHERE** posso usare anche **LIKE** per il confronto tra stringhe, che si comporta come un operatore di uguaglianza, con in più i caratteri speciali **\_** e **%**

- Con **\_** rappresento un carattere arbitrario
- Con **%** una stringa con numero arbitrario di caratteri, anche zero

Es. Visualizzare la matricola e il nome di tutti gli studenti che abitano in una città che ha una A in seconda posizione e finisce per A

```
SELECT Matricola, Nome
FROM Studente
WHERE Città LIKE '_a%a'
```

Posso invece usare la parola chiave **BETWEEN** per stabilire un intervallo numerico (o **NOT BETWEEN**)

Es. Visualizzare il nome e lo stipendio dei docenti che guadagnano tra i 2000 e i 3000 euro

```
SELECT Nome, Stipendio
FROM Docente
WHERE Stipendio BETWEEN 2000 AND 3000

--oppure

SELECT Nome, Stipendio
FROM Docente
WHERE Stipendio>=2000 AND Stipendio<=3000
```

Un ulteriore modo per selezionare le righe è `IN` e `NOT IN`

```
SELECT NomeAttributo
FROM Tabella
WHERE NomeAttributo IN (valore1, valore2, ...)
```

Posso anche usare `IS NULL` o `IS NOT NULL`, per visualizzare gli attributi che hanno valori nulli, o non visualizzarli

Posso anche visualizzare i risultati in un certo ordine: DISCENDENTE o ASCENDENTE (Default)

Es. Visualizzare ordinati per stipendio il nome e numero di telefono dei docenti

```
SELECT Nome, Telefono
FROM Docente
ORDER BY Stipendio
```

Posso ordinarli anche rispetto a più cose, basta inserire: `ORDER BY Attributo1, Attributo2`

## Esercizio

Visualizzare le città di studenti maschi, presentando eventualmente più volte lo stesso valore

```
SELECT Città
FROM Studente
WHERE Sesso=M
```

Per non visualizzare più volte lo stesso valore, uso `DISTINCT`:

```
SELECT DISTINCT Città
FROM Studente
WHERE Sesso=M
```

# Lezione 9

## Operatori di Aggregazione

### Operatori che vengono applicati ad un insieme di righe.

Prima viene eseguita la query SQL, `Select From Where`, considerando solo le parti `FROM` e `WHERE` → l'operatore allegato viene applicato alla tabella contenente il risultato dell'interrogazione

Due gruppi:

- Count
- Max, Min, AVG, Sum

**ATTENZIONE:** se l'operatore allegato è argomento della `SELECT` allora non possono apparire espressioni a livello di riga, come il nome degli attributi.

## Count

Operatore che permette di contare il numero di righe: posso contare quante righe soddisfano le condizioni che ho inserito nella mia query.

#### 💡 Sintassi:

```
COUNT(<*>) -- conto il numero di righe
COUNT(<* | [DISTINCT | ALL ] | ListaAttributi>)
/*
Se inserisco distinct voglio contare il numero di diversi valori
nella lista attributi
Se uso ALL restituisce il numero di righe con valori diversi da ALL
anche ripetuti
Con più di un attributo e distinct, mi conta la coppia unica,
quindi le combinazioni:
-attributo1, attributo2
-attributo1, attributo2'
-attributo1', attributo2
-attributo1', attributo2'
*/
```

- Quanti corsi ci sono nella tabella CORSO?

```
SELECT COUNT(*)
FROM CORSO

SELECT COUNT(Codice)
FROM CORSO
```

- Quali sono le città di provenienza degli studenti? diventa: Da quante città provengono gli studenti?

```
--Avevamo scritto:
SELECT DISTINCT Città
FROM Studente
--Se vogliamo contare quante sono le città:
SELECT COUNT(DISTINCT Città)
FROM Studente
```

## Sum, Min, Max e AVG

### Sum

L'operatore **SUM** restituisce la somma dei valori posseduti dall'attributo su tutte le righe.

### Min e Max

Restituiscono rispettivamente il minimo e il massimo dei valori posseduti dall'attributo su tutte le righe

Su Min e Max, **DISTINCT** e **ALL** non hanno effetto: in ogni caso trovo il minimo e massimo

### AVG

Restituisce la media dei valori dell'attributo.

#### 💡 Sintassi:

```
<SUM | MAX | MIN | AVG>([DISTINCT | ALL] AttribExpress)
```

- Stipendio massimo dei docenti

```
SELECT MAX(Stipendio)
FROM Docente
```

- Minimo stipendio dei docenti

```
SELECT StipMin=MIN(Stipendio) --cambio intestazione
FROM Docente
```

- Trovare la somma degli stipendi dei docenti che guadagnano più di 2000

```
SELECT SUM(Stipendio)
FROM Docente
WHERE Stipendio>2000
/*
prendo la tabella docente -> applico la clausola WHERE
il risultato di questa operazione diventa l'argomento del mio SUM
*/
```

- Trovare la media degli stipendi dei docenti che guadagnano

```
SELECT AVG(Stipendio)
FROM Docente
WHERE Stipendio>2000
```

## Interrogazioni con raggruppamento

Ci consentono di applicare l'operatore allegato a sottoinsiemi di righe.

La clausola è **GROUP BY** e ci permette di specificare come dividere la tabella in sottoinsiemi.

I sottoinsiemi vengono formati raggruppando le righe della tabella di partenza, rispetto al valore di un attributo.

**Attenzione:** Quando uso **GROUP BY**, nella **SELECT** ci possono andare solo un sottoinsieme degli attributi rispetto cui ho raggruppato.

### Sintassi:

```
SELECT Attributo1, OP_AGGR
FROM Tabella
WHERE Condizione
GROUP BY Attributo1
```

Se metto qualcosa in **GROUP BY** può apparire in **SELECT**: se non appare in **GROUP BY**, di sicuro non può apparire in **SELECT**!

- Visualizzare il numero di studenti che provengono da città diverse (visualizzare anche le città)

```
SELECT Città, COUNT(*)
FROM Studente
GROUP BY Città
```

## Predicati sui gruppi

La clausola **HAVING** permette di descrivere condizioni che si devono applicare al termine della esecuzione di una query che fa uso di **GROUP BY**

- Ogn sottoinsieme di righe farà parte del risultato solo se il predicato di **HAVING** risulta soddisfatto

### Sintassi:

```
SELECT Attributo1, OP_AGG
FROM Tabella
[WHERE Condizione]
GROUP BY Attributo1
HAVING Predicato
```

Visualizzare le città in cui abitano almeno due studenti e il numero di studenti relativo

```
SELECT Città, Count(*)
FROM Studente
GROUP BY Città
HAVING Count(*)>=2
```

## Join

Per selezionare informazioni da due o più tabelle.

Posso usare la sintassi:

```
NomeTabella.NomeAttributo
```

### Sintassi:

```
SELECT ListaAttributi
FROM Tabella1, Tabella2
WHERE Tabella1.Attributo1 = Tabella2.Attributo2 --esempio di una condizione
```

Esempio:

STUDENTE(Matricola, Nome, Indirizzo, Città, CAP, Sesso)

CORSO(Codice, NomeCorso, NumCrediti)

DOCENTE(Matricola, Nome, Telefono, Stipendio)

ESAME(CodiceCorso, Studente, Voto)

INSEGNAMENTO(CodiceCorso, Docente, NumStudenti)

Visualizzare il codice dei corsi insegnati da ogni docente (visualizzando anche il nome del docente):

```
SELECT Docente.Nome, Insegnamento.Corso -- in realtà non serve questa notazione
FROM Docente, Insegnamento
WHERE Docente.Matricola = Insegnamento.Docente
```

Visualizzare il numero di studenti che frequentano i diversi corsi visualizzando anche il nome del corso

```
SELECT NomeCorso, NumStudenti
FROM Corso, Insegnamento
WHERE Corso.Codice = Insegnamento.CodiceCorso
```

Visualizzare i voti presi dagli studenti di Verona, insieme a nome studente e codice del corso

```
SELECT Nome, CodiceCorso, Voto
FROM Studente, Esame
WHERE Studente.Città = Verona
AND Studente.Matricola = Esame.Studente
```

Visualizzare la stessa query ma in ordine decrescente rispetto al nome degli studenti

```
SELECT Nome, CodiceCorso, Voto
FROM Studente, Esame
WHERE Studente.Città = 'Verona'
AND Studente.Matricola = Esame.Studente
ORDER BY Nome DESC
```

Visualizzare il nome dei corsi insegnati da ogni docente

```
SELECT D.Nome AS Docente, C.NomeCorso
FROM Docente D, Insegnamento I, Corso C --do un alias alle tabelle
WHERE D.Matricola = I.Docente --agevolò la notazione puntata
      AND I.CodiceCorso = C.Codice
```

Visualizzare i voti presi dagli studenti di verona visualizzando anche il nome del corso

```
SELECT S.Nome, C.NomeCorso, E.Voto
FROM Studente AS S, Esame AS E, Corso AS C
WHERE S.Città = 'Verona'
      AND S.Matricola = E.Studente
      AND E.CodiceCorso = C.Codice
```


Visualizzare i voti presi dagli studenti di verona, visualizzando anche il nome del corso e il nome del relativo professore

```
SELECT S.Nome, C.NomeCorso, E.Voto, I.Docente, D.Nome
FROM Studente AS S, Esame AS E, Corso AS C, Insegnamento AS I, Docente AS D
WHERE S.Città = 'Verona'
      AND S.Matricola = E.Studente
      AND E.CodiceCorso = C.Codice
      AND C.Codice = I.CodiceCorso
      AND D.Codice = I.Docente
```

Visualizzare il numero di studenti maschi e femmine che hanno superato l'esame di Analisi 1

```
SELECT S.Sesso, COUNT(S.Matricola)
FROM Studente AS S, Esame AS E, Corso AS C
WHERE C.NomeCorso = 'Analisi 1'
      AND S.Matricola = E.Studente
      AND E.CodiceCorso = C.Codice
GROUP BY Sesso
```

SQL ci mette a disposizione anche un altro modo per scrivere i Join

 **Sintassi alternativa:**

```
SELECT : ListaAttributi
FROM Tabella1 TipoJoin Tabella2 --effettuo il join nel from
      ON CondizioneJoin
WHERE AltreCondizioni
```

TipoJoin può essere:

- **INNER JOIN**

è il join interno, rappresenta il theta join algebrico

prende solo le righe che soddisfano la condizione di join

**OUTER JOIN** → viene eseguito il join mantenendo tutte le righe che fanno parte di una o di entrambe le tabelle, a seconda del join esterno

- **RIGHT OUTER JOIN**
- **LEFT OUTER JOIN**
- **FULL OUTER JOIN**

**Esempi di utilizzo:**

Visualizzare il nome dei docenti che tengono un corso con più di 100 studenti

```
SELECT Docente.Nome
FROM Docente INNER JOIN Insegnamento
      ON Docente.Matricola = Insegnamento.Docente
WHERE Insegnamento.NumStudenti >= 100
```

Visualizzare il nome degli studenti che hanno preso 30 nell'esame con codice uguale a 'BD'

```
SELECT Nome
FROM Studente AS S INNER JOIN Esame AS E
ON(S.Matricola = E.Studente)
WHERE E.Voto=30
AND E.CodiceEsame = 'BD'
```

## Alias

S è uno **pseudonimo**

```
SELECT S.Nome
FROM Studente AS S -- S si chiama Alias
WHERE S.Città = 'Verona'
```

**Attenzione:** se scrivo

```
SELECT S1.Cognome
FROM Studente AS S1, Studente AS S2
WHERE S1.Cognome = S2.Cognome
AND S2.Città = 'Padova'
```

Sto trovando il cognome degli studenti che hanno lo stesso cognome di uno studente che abita a Padova. In questo caso

## Interrogazioni di tipo insiemistico

SQL ci mette a disposizione degli operatori di tipo insiemistico, che posso usare solo al livello più esterno di una query: la Select, ad esempio posso fare l'unione di più select.

Operatori:

- **UNION**
- **INTERSECT**
- **EXCEPT (MINUS)**

Come default assumono di eliminare i duplicati ( a meno dell'uso di **ALL** )

 **Sintassi:**

```
SelectSQL <UNION|INTERSECT|EXCEPT>[ALL] SelectSQL
```

- Ovviamente devo avere lo stesso tipo tra i risultati.
- Select non richiede la denominazione - usa il nome del primo

Determinare i cognomi di docenti e studenti

```
SELECT Cognome
FROM Studente
UNION
SELECT Cognome
FROM Docente
```

Determinare i cognomi di studenti che sono anche cognomi di docenti

```
SELECT Cognome
FROM Studente
INTERSECT
SELECT Cognome
FROM Docente
```

Determinare i cognomi e le città degli studenti



```
SELECT Cognome
FROM Studente
UNION
SELECT Città
FROM Studente
```

Determinare i cognomi degli studenti che non sono anche città

```
SELECT Cognome
FROM Studente
EXCEPT
SELECT Città
FROM Studente
```

Determinare i cognomi e le città degli studenti mantenendo anche i duplicati

```
SELECT Nome
FROM Studente
UNION ALL
SELECT Città
FROM Studente
```

## Lezione 10

### Interrogazioni nidificate

```
SELECT Attr
FROM Tabella
WHERE condizione ( SELECT
FROM
WHERE)
```

Il risultato della nidificazione è ciò che uso come argomento di quella esterna.



SQL ammette il confronto di un valore ottenuto come risultato di una espressione sulla singola riga con il risultato di una interrogazione SQL

- Definito direttamente nella clausola **WHERE**
- Abbiamo il problema di confrontare qualcosa (valore) con un'insieme di tuple (un risultato di un'interrogazione) → **Problema di Disomogeneità**

Per **risolvere** questo problema SQL mette a disposizione due parole chiave:

- ANY**
- ALL**

Il risultato deve essere opportunamente esteso con le parole chiave

Esempio: **visualizzare il nome dei docenti che guadagnano meno dello stipendio medio**

```
SELECT Nome
FROM Docente
WHERE Stipendio < ( SELECT AVG(Stipendio)
FROM Docente )
```

Esegui l'interrogazione interna, tiro fuori il valore e poi eseguo quella esterna in cui uso il valore estratto.

**Visualizzare il cognome di tutti gli studenti di Verona che hanno lo stesso cognome di uno studente che abita a Padova**

```
SELECT Cognome
FROM Studente
WHERE Città = 'Verona'
AND Cognome = ANY( SELECT Cognome
                    FROM Studente
                    WHERE Città = 'Padova')
```

L'interrogazione nidificata darà come risultato un insieme di cognomi, che io confronterò con un insieme di valori → devo inserire la clausola **ANY**

Visualizzare l'elenco degli studenti che hanno almeno un voto maggiore o uguale del voto medio di ogni studente

```
SELECT Cognome
FROM Studente S, Esame E
WHERE S.Matricola = E.Studente
AND Voto >= ALL ( SELECT AVG(Voto)
                  FROM Esame
                  GROUP BY Studente)
```

## Interrogazioni nidificate complesse

L'interrogazione nidificata fa riferimento al contesto dell'interrogazione che la racchiude

- C'è un PASSAGGIO DI BINDING cioè nell'interrogazione esterna viene definita una variabile che sarà usata nell'ambito dell'interrogazione interna.

```
SELECT ...
FROM ...
WHERE ... (SELECT
           FROM
           WHERE )
```

Non vale più l'interpretazione secondo cui devo eseguire prima l'interrogazione interna e poi quella esterna: le devo eseguire insieme.

### Cosa succede?

Vediamo passo passo quello che succede durante l'interrogazione:

**Passo 1.** Viene eseguito il prodotto cartesiano delle tabelle

**Passo 2.** Le condizioni che appaiono nella clausola **WHERE** vengono applicate a ciascuna riga del prodotto cartesiano

**Passo 3.** Vengono mantenute solo le righe che soddisfano le condizioni

**Passo 4.** L'interrogazione **NIDIFICATA** che appare nella clausola **WHERE** viene valutata separatamente per ogni riga, prodotta dalla valutazione dell'interrogazione esterna

Ogni volta che mi serve l'interrogazione interna, nel mentre che eseguo quella esterna, allora eseguo quell'interna (per verificare cosa accade)

La parola chiave in questo caso è **EXISTS** che ammette come parametro una interrogazione nidificata e restituisce vero solo se l'interrogazione restituisce un risultato non vuoto.

### Sintassi:

```
SELECT Matricola, Cognome
FROM Studente
WHERE EXISTS (SELECT COUNT(*)
              FROM Esame
              WHERE Studente = Matricola
              HAVING COUNT(*)>1)
```

Visualizzare la matricola e il cognome degli studenti che hanno superato più di un esame.

Le due interrogazioni, ogni volta che valuto una riga della tabella esterna, eseguo la tabella interna.

Visualizzare matricola e cognome degli studenti che non hanno superato alcun esame

```
SELECT Matricola, Cognome
FROM Studente
WHERE NOT EXISTS (SELECT *
                  FROM Esame
                  WHERE Studente = Matricola)
```

Posso usare anche `IN`

## Viste SQL

- Sono delle tabelle che mi costruisco con il risultato di Query SQL.
- Tabelle "virtuali" il cui contenuto dipende dal contenuto di altre tabelle della BD.
- Sono definite associando un nome e una lista di attributi (uno Schema) al risultato di un'interrogazione.

### Sintassi:

```
CREATE VIEW NomeVista ListaAttr
AS QUERY SQL
```

Ovviamente la Query dovrà avere come risultato una lista di attributi che coincide con la mia Vista.

Posso definire dei meccanismi per verificare quello che succede nelle tabelle venga riportato nella Vista.

Posso anche usarla per fare interrogazioni, come un'altra tabella.

Esempio: Definire la vista che contiene la matricola e il cognome dei docenti che tengono un corso con più di 100 studenti

```
CREATE VIEW Docenti100Studenti(Matricola, Cognome)
AS SELECT Docente.Matricola, Docente.Cognome
FROM Docente, Insegnamento
WHERE Docente.Matricola = Insegnamento.Docente
AND Insegnamento.NumStudenti > 100
```

Definire la vista che contiene il cognome dei docenti e il nome del corso che insegnano

```
CREATE VIEW DocenteCorso(Cognome, NomeCorso)
AS SELECT D.Cognome, C.Nome
FROM Docente D, Corso C, Insegnamento I
WHERE D.Matricola = I.Docente
AND C.Codice = I.CodiceCorso
```

## Perché sono utili le viste?

Mi possono aiutare a risolvere le interrogazioni.

Esempio: Determinare qual è il docente che tiene corsi con la massima somma di crediti

```
CREATE VIEW DocentiCrediti(Cognome, SommaCFU)
AS SELECT D.Cognome, SUM(C.NumCrediti)
FROM Docente D, Insegnamento I, Corso C
WHERE D.Matricola = I.Docente
AND I.CodiceCorso = C.Codice
GROUP BY D.Cognome

SELECT *
FROM DocentiCrediti
WHERE SommaCFU = SELECT MAX(SommaCFU) FROM DocentiCrediti
```

Ci sono casi in cui creare una vista può aiutare a risolvere un'interrogazione in maniera più semplice.

## Esercizi 2

Trovare la matricola degli studenti che hanno preso un 30 e un 30 e lode

```

SELECT MatricolaStudente
FROM Esame
WHERE Voto = 30
AND MatricolaStudente IN (SELECT MatricolaStudente
                          FROM Esame
                          WHERE Voto = 33 )

```

--oppure

```

SELECT MatricolaStudente
FROM Esame
WHERE Voto = 30
INTERSECT
SELECT MatricolaStudente
FROM Esame
WHERE Voto = 33

```

Non devo mai dire `WHERE Voto = 30 AND Voto = 33` , perché in ogni tupla c'è un solo valore!

Trovare gli studenti che hanno preso 30 O 30 e lode

```

SELECT MatricolaStudente
FROM Esame
WHERE Voto = 30 OR 33

```

--oppure

```

SELECT MatricolaStudente
FROM Esame
WHERE Voto = 30
UNION
SELECT MatricolaStudente
FROM Esame
WHERE Voto = 33

```

Trovare gli studenti che hanno preso almeno due 30

```

SELECT MatricolaStudente
FROM Esame E1, Esame E2
WHERE E1.MatricolaStudente = E2.MatricolaStudente
AND E1.Voto = 30 AND E2.Voto = 30
AND E1.CodiceCorso <> E2.CodiceCorso

```

--

```

SELECT MatricolaStudente
FROM Esame
GROUP BY MatricolaStudente
HAVING Count(Voto = 30)>1

```

```

SELECT MatricolaStudente
FROM Esame
GROUP BY MatricolaStudente
HAVING Count(Voto) WHERE Voto = 30

```

Elencare il voto medio e il numero di esami sostenuti da ogni studente. Visualizzare anche la matricola dello studente

```

SELECT MatricolaStudente, AVG(Voto), NumEsami = COUNT(*)
FROM Esame
GROUP BY MatricolaStudente

```

Elencare il voto medio e il numero di esami sostenuti da ogni studente con voto medio superiore a 25. Visualizzare anche la matricola dello studente.

```

SELECT MatricolaStudente, AVG(Voto), NumEsami = COUNT(*)
FROM Esame
GROUP BY MatricolaStudente
HAVING AVG(Voto) > 25 //non WHERE

```

Elencare nome, cognome e voto degli studenti che hanno superato il corso con codice IC01

```
SELECT Nome, Cognome, Voto
FROM Studente
WHERE (SELECT *
       FROM Esame
       WHERE Matricola = MatricolaStudente
       AND CodiceCorso = IC01
      )
--troppo complicata

SELECT Nome, Cognome, Voto
FROM Studente
WHERE CodiceCorso = IC01
```

Elencare cognome e nome dei docenti, e nome e codice dei corsi che insegnano

```
SELECT D.Nome, D.Cognome, I.CodiceCorso, C.Nome
FROM Docente AS D, Insegnamento AS I, Corso AS C
WHERE D.Matricola = I.MatricolaDocente
      AND I.CodiceCorso = C.Codice
```

Trovare il nome dei corsi in cui qualche studente maschio ha preso 30

```
SELECT C.Nome
FROM Corso AS C, Esame AS E, Studente AS S
WHERE E.MatricolaStudente = S.Matricola
      AND E.CodiceCorso = C.Codice
      AND S.Sesso = M
      AND E.Voto = 30

--idea:
SELECT C.Nome
FROM Corso AS C, Esame AS E
WHERE E.Voto = 30
      AND E.MatricolaStudente IN (SELECT Matricola
                                  FROM Studente
                                  WHERE Sesso = M )
```

Quanti studenti hanno superato l'esame ddi 'Basi di Dati' insegnato dalla prof. Letizia Tanca?

```
SELECT Count(MatricolaStudente)
FROM Esame AS E, Corso AS C, Insegnamento AS I, Docente AS D
WHERE E.CodiceCorso = C.Codice
      AND I.MatricolaDocente = D.Matricola
      AND D.Nome = 'Letizia'
      AND D.Cognome = 'Tanca'
      AND C.Nome = 'Basi di Dati'
      AND C.Codice = I.CodiceCorso
```

## Esercizi 3

STUDENTE (matricola, nome, indirizzo, città, CAP, M/F)

DOCENTE (matricola, nome, cognome, città, telefono, stipendio)

CORSO (codice, nome, facoltà, numeroCrediti)

ESAME (codiceCorso, matricolaStudente, voto)

INSEGNAMENTO (codiceCorso, matricolaDocente, numeroStudenti)

## Interrogazione 18

quanti sono gli studenti che seguono i corsi di ogni professore? Visualizzare anche i dati del professore.

```
SELECT Matricola, Cognome, Nome, SUM(NumeroStudenti)
FROM Docente AS D, Insegnamento AS I
WHERE D.Matricola = I.MatricolaDocente
GROUP BY Matricola, Cognome, Nome
```

## Interrogazione 19

Trovare la matricola di tutti gli studenti che hanno superato almeno due esami

```
SELECT DISTINCT Matricola
FROM Esame AS E1, Esame AS E2
WHERE E1.matricolaStudente = E2.matricolaStudente
AND E1.CodiceCorso <> E2.CodiceCorso
```

## Interrogazione 20

Trovare la matricola ed il nome degli studenti che hanno superato esami per un totale di almeno 20 crediti

```
SELECT Matricola, Cognome, Nome
FROM Esame AS E, Corso AS C, Studente AS S
WHERE E.matricolaStudente = S.Matricola
AND C.CodiceCorso = E.CodiceCorso
GROUP BY S.Matricola, S.Cognome, S.Nome
HAVING SUM(C.numeroCrediti) >= 20
```

## Interrogazione 21

Quanti studenti e quante studentesse hanno superato l'esame di Basi di Dati?

```
SELECT M/F, COUNT(S.Matricola)
FROM Studente AS S, Esame AS E, Corso AS C
WHERE C.Codice = E.CodiceCorso
AND C.Nome = 'Basi di Dati'
AND E.MatricolaStudente = S.Matricola
GROUP BY M/F
```

## Interrogazione 22

Trovare i nomi dei professori e degli studenti disponibili nella base di dati

```
SELECT DISTINCT Nome, Matricola
FROM Studente
UNION
SELECT DISTINCT Nome, Cognome, Matricola
FROM Docente
```

## Interrogazione 25

Trovare i cognomi dei professori che sono anche cognomi di studenti di milano

```
SELECT D.Cognome
FROM Studente AS S, Docente AS D
WHERE S.Cognome = D.Cognome
AND S.Città = Milano
```

## Interrogazione 26

Trovare cognomi dei professori che non sono anche cognomi di studenti

```
SELECT Cognome
FROM Docente
EXCEPT
SELECT Cognome
FROM Studente
```

## Interrogazione 27

Trovare tutte le facoltà che non prevedono il corso di 'Informatica C' di 5 crediti

```

SELECT Facoltà
FROM Corso
EXCEPT
SELECT Facoltà
FROM Corso
WHERE Corso.Nome = 'Informatica C'
AND Corso.NumeroCrediti = 5

```

## Interrogazione 28

Quali studenti hanno ricevuto 30 in qualche esame?

```

SELECT Nome, Cognome
FROM Studente
WHERE Matricola IN (SELECT MatricolaStudente
                     FROM Esame
                     WHERE Voto = 30)

```

## Interrogazione 29

Quali professori guadagnano meno della media?

```

SELECT Nome, Cognome
FROM Docente
WHERE Stipendio < (SELECT AVG(Stipendio)
                  FROM Docente)

```

## Interrogazione 30

Selezionare le matricole degli studenti che hanno tra i voti qualche 30 ma non hanno un 30 e lofe

```

SELECT MatricolaStudente
FROM Esame
WHERE Voto = 30
AND MatricolaStudente
EXCEPT
SELECT MatricolaStudente
FROM Esame
WHERE Voto = 33

```

## Interrogazione 32

Selezionare le matricole degli studenti che hanno sostenuto almeno 10 esami ma non hanno superato basi di dati

```

SELECT MatricolaStudente
FROM Esame
GROUP BY MatricolaStudente
HAVING COUNT(*) >= 10
EXCEPT
SELECT E.MatricolaStudente
FROM Esame AS E, Corso AS C
WHERE E.CodiceCorso = C.Codice
AND C.NomeCorso = 'Basi di Dati'

```

## Interrogazione 33

Selezionare il codice degli esami superati da un solo studente

```

SELECT E.CodiceCorso
FROM Esame
EXCEPT
SELECT CodiceCorso
FROM Esame AS E1, Esame AS E2
WHERE E1.CodiceCorso = E2.CodiceCorso
AND E1.MatricolaStudente <> E2.MatricolaStudente

```

## Interrogazione 36

Trovare nome e cognome degli studenti che hanno preso qualche voto maggiore di 20 in un esame con codice che inizio con IN riferito ad un corso non di 5 Crediti

```
SELECT S.Nome, S.Cognome
FROM Studente AS S, Corso AS C, Esame AS E
WHERE S.Matricola = E.MatricolaStudente
AND C.Codice = E.CodiceCorso
AND E.Voto > 20
AND C.Nome LIKE 'IN%' NOT IN (SELECT C.Nome
FROM Corso
WHERE numeroCrediti = 5)
```

## Lezione 11

# Progettazione concettuale delle Basi di Dati: Modello Entità-Relazione

## Cosa vuol dire progettare una base di dati?

Vuol dire definirne la struttura e quindi lo schema (il modo in cui i dati devono essere strutturati per poter essere memorizzati nella base di dati), le caratteristiche e il contenuto.

La progettazione di una base di dati si può inquadrare come attività del processo di sviluppo dei sistemi informativi.

## Studio di Fattibilità

Fase in cui verifichiamo se siamo in grado di progettare un sistema informativo. Entrano in gioco i requisiti dell'utente, il tempo che mette a disposizione e quanto è disposto a pagare. Noi dobbiamo verificare se siamo in grado di portare avanti il progetto o meno.

## Raccolta dei Requisiti

La prima cosa da fare, prevede lo studio di tutte le caratteristiche del sistema: dati da comunicare, piattaforme a disposizione, carico di lavoro, creazione dei team ecc. Vi sono tutta una serie di approcci che fanno parte dell'ingegneria dei requisiti.

## Progettazione

Progettiamo sia la parte dati che la parte di funzioni (si vedrà in ingegneria del software)

## Realizzazione

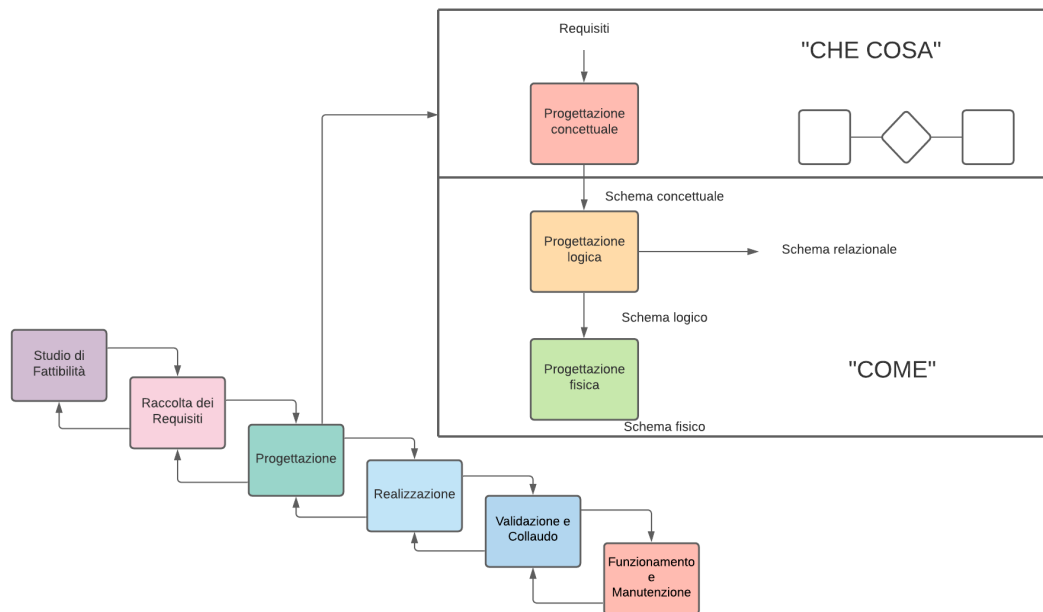
## Validazione e Collaudo

## Funzionamento e Manutenzione

Ogni fase prevedere l'eventuale ritorno alla precedente per eventuali modifiche o richieste: tanto più la strada di ritorno inizia in basso nel ciclo di vita, tanto più è dispendiosa: il tempo è denaro, più ne investo meno ci guadagno.

Il corso si concentra sulla fase di progettazione, le altre si vedono in ingegneria del Software.





## Fase di Progettazione

Parte dai requisiti, che si raccolgono e analizzano. La prima cosa da fare sarà la progettazione concettuale, producendo lo schema concettuale: schema entità relazione → fase successiva è la progettazione logica. Parte dallo schema concettuale e produce lo schema logico: nel modello relazionale dei dati → progettazione fisica che a partire dallo schema relazionale, crea lo schema fisico (di cui non ci occupiamo).

Nella parte "alta" vado ad analizzare **che cosa** voglio rappresentare, nella parte "basse" decido **come** rappresentarlo.

## Il modello entità relazione - Modello Concettuale

Un modello dei dati ci mette a disposizione dei costrutti per poter rappresentare l'informazione di interesse.

Il modello dei dati logico, che già conosciamo e usiamo in fase di progettazione logica è il Modello Relazionale, che ci mette a disposizione le relazioni.

Il modello concettuale ci mette a disposizione dei costrutti per modellare dal punto di vista concettuale (in maniera più astratta) i miei dati: cosa voglio mettere i dati!



Il **modello entità relazione** è il modello concettuale più diffuso, ma non è l'unico: ci permette di descrivere il dominio di interesse a un certo livello di astrazione. Infatti ci permette di rappresentare cosa vogliamo rappresentare.

È un modello grafico che fornisce una descrizione sintetica e visiva.

I principali costrutti sono:

- **Entità**

Una entità è una classe di oggetti (fatti, persone, cose) dell'applicazione di interesse (quindi nel contesto considerato) con proprietà comuni ed esistenza autonoma.

Rettangolo con nome che rappresenta il concetto.  
Nomi al singolare.

- **Relazione** (non quella del modello relazionale) → collegano le entità

Rappresenta un legame logico tra due o più entità.  
Questo legame logico è rilevante nell'applicazione di

interesse.

L'istanza di una relazione è data dalla coppia di entità che sono legate dalla relazione.

Sostantivi non verbi.

- **Relazione ricorsiva**

È la relazione di un'entità con sé stessa.

Può essere simmetrica o non simmetrica: ovvero possono avere lo stesso ruolo oppure no.

Ad esempio: la relazione successore non è simmetrica (x è successore di y, ma non è vero il contrario)

La relazione conoscenza è simmetrica (x conosce y, ma anche y conosce x).

- **Relazione ternaria**

È una relazione tra triplette di entità.

La relazione ricorsiva può sempre essere trasformata in **3 relazioni binarie**, inserendo un'ulteriore entità in mezzo alle 3.

L'istanza della relazione ternaria è formata dalle singole occorrenze delle relazioni coinvolte

Esempio della relazione fornitura:

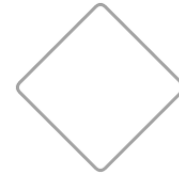
un certo grossista fornisce un certo prodotto ad un certo negozio.

- **Attributo**

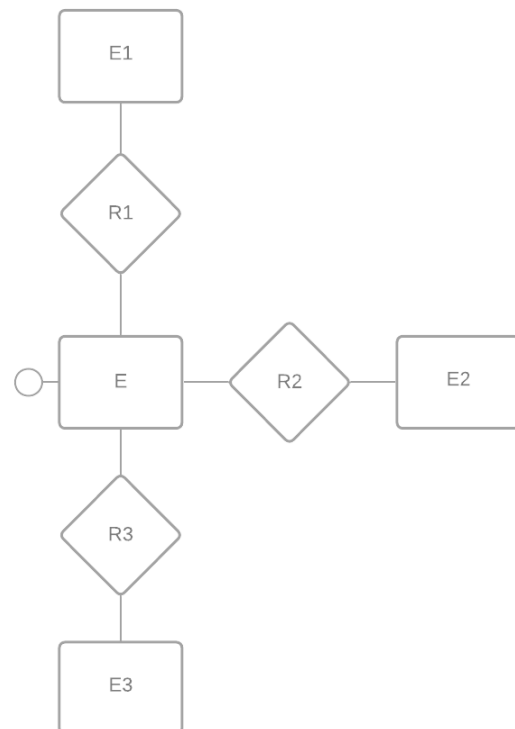
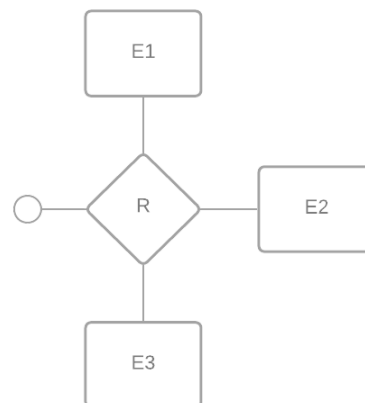
Una proprietà elementare di una entità o di una relazione, di interesse ai fini della relazione.

La sua rappresentazione grafica è il pallino e associa ad ogni occorrenza di un'entità o di una relazione, un valore appartenente al dominio.

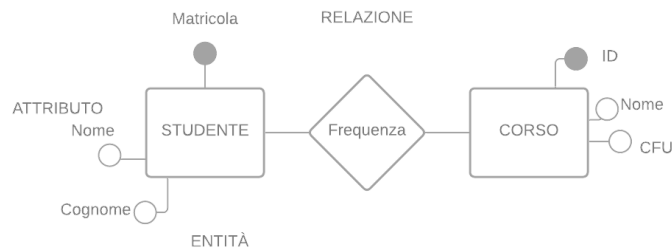
Un attributo può avere altri attributi.



Simboli dei costrutti principali



Relazione ternaria



Esempio

Il modo in cui rappresentiamo le informazioni è astratto: bisogna fare astrazione a partire dai requisiti, per riuscire a darne una rappresentazione grafica.

## Altri costrutti

### • Cardinalità

#### ◦ Di relazione

È una coppia di valori che posso associare ad ogni entità che partecipa ad una relazione. Specifica il numero minimo e il numero massimo di occorrenze della relazione, a cui ciascuna occorrenza di un'entità può partecipare.

Es. Se io ho l'entità Impiegato, in relazione Assegnamento con l'entità Incarico: la cardinalità minima e massima è:

(1,3) dal lato Impiegato (un impiegato può avere tra 1 e 3 incarichi assegnati)

(0,10) dal lato Incarico (un incarico può essere assegnato ad un massimo di 10 persone)

(1,n) notazione per dire infiniti possibili

(1,1) UNIVOCO

- Uno a Uno
- Uno a Molti
- Molti a Molti

#### ◦ Di attributo

- indicare **opzionalità**: di fatto l'informazione può essere incompleta

- Attributi **multivalore**

Quando inserisco (1,1) è obbligatorio e unico, ed è ciò che è impostato di default (quindi solitamente non lo inserisco)

Quando inizia con (0, vuol dire che è opzionale: può anche non esserci

La cardinalità (1,n) invece è alitatoria, ma obbligatoria.

### • Identificatore

Di entità: permette di identificare in modo univoco le istanze e le occorrenze di un'entità.

#### ◦ Interno

Costituito da uno o più attributi dell'entità

Identificato da un pallino pieno (quando più attributi identificano, si fa un attributo pallino pieno che li attraversa)

**Le relazioni non hanno identificatori!**

#### ◦ Esterno

È costituito da attributi dell'entità, più entità esterne attraverso le relazioni.

Ad esempio: il prodotto non è sufficientemente identificato solo dal suo nome: l'attributo identificatore viene incrociato anche con la relazione Produzione, con il produttore.

Questo genere di entità è un'entità debole, ed è importante che abbia una cardinalità 1 a 1.

Dobbiamo fare in modo, nel nostro ER, che tutte le entità abbiano **ALMENO** una chiave: tutte le entità devono essere **identificabili**.

- **Generalizzazione**

## Lezione 12

### Generalizzazione

Rappresenta i legami logici tra entità, in particolare abbiamo una entità  $E$ , che chiamiamo **padre**, e una o più entità **figlie**.



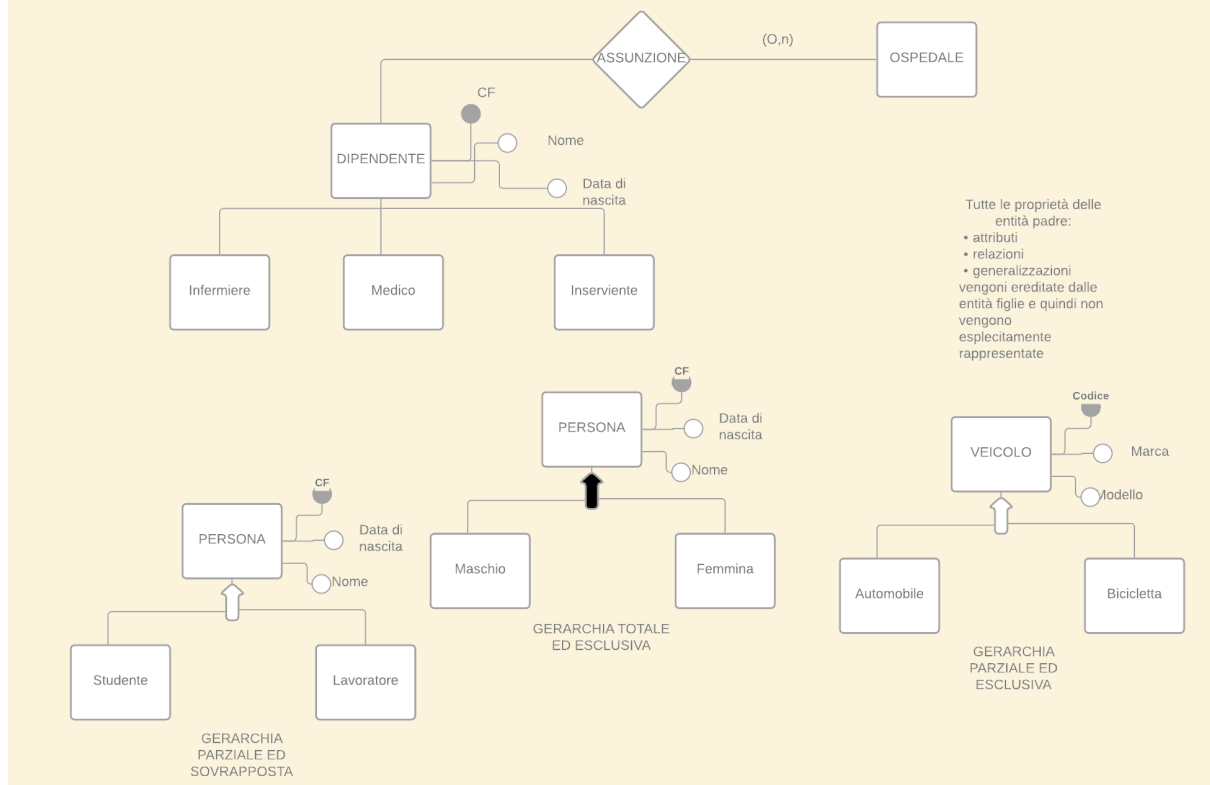
L'entità  $E$ , il padre, è più generale e comprende le entità figlie come casi particolari.

Si dice che le entità  $E$  è la generalizzazione delle entità  $E_1, E_2, \dots, E_n$  e che le entità  $E_1, E_2, \dots, E_n$  sono specializzazioni di  $E$ .

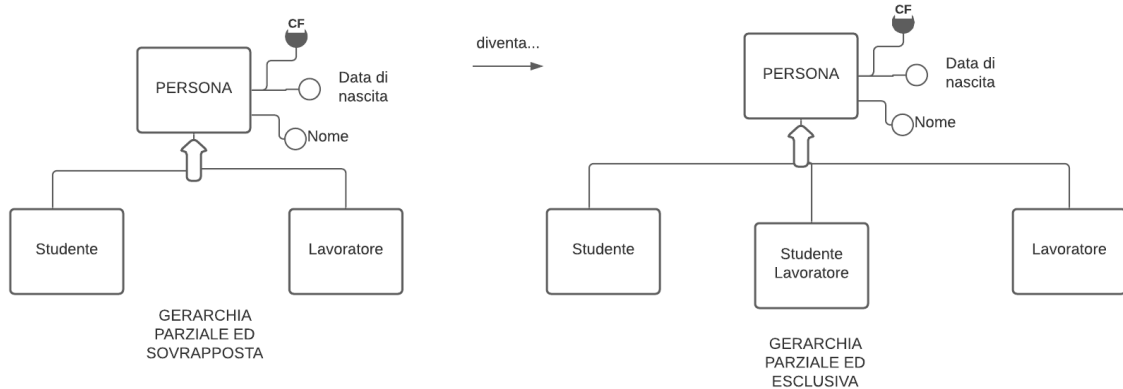
Possiamo dire che ogni occorrenza dell'entità figlia è occorrenza dell'entità padre, e ogni proprietà dell'entità padre è anche proprietà delle figlie.

### Tipi di generalizzazione

- La generalizzazione è **TOTALE** se ogni occorrenza dell'entità padre p occorrenza di almeno una delle entità figlie
  - Se questo non è vero, la generalizzazione è **PARZIALE**
- La generalizzazione è **ESCLUSIVA** se ogni occorrenza dell'entità padre, è occorrenza di al più una delle entità figlie
  - Se questo non è vero, la generalizzazione è **SOVRAPPONSTA**



Noi consideriamo sempre Generalizzazioni ESCLUSIVE (eliminando la generalizzazione sovrapposta aggiungendo un'entità → modelliamo l'intersezione con una nuova entità)



Le classifichiamo come:

- Totali (con freccia piena)
- Parziali (freccia vuota)

Esercizio:

- Le persone hanno CF, nome ed età.
- Le femmine hanno anche il numero di gravidanze
- I dipendenti hanno lo stipendio e possono essere inservienti, infermieri o medici.
- I medici possono essere responsabili di reparto
- I religiosi che non possono essere impiegati hanno un ordine di appartenenza
- Esistono persone che non sono né impiegati né religiosi



Entità	Descrizione	Attributi	Identificatore
Paziente	Paziente sottoposto a visite di controllo periodiche	CodSanitario Nome Età ...	CodSanitario

Relazione	Descrizione	Attributi	Entità coinvolte
Terapia	Associa un paziente agli eventuali farmaci prescritti	Inizio Fine Dosaggio	Paziente Farmaco

- Un'altra cosa che possiamo utilizzare per documentare lo schema R sono delle **annotazioni** testuali che ci permettono di descrivere eventuali vincoli non esprimibili con lo schema R.

esempio:

VINCOLI:

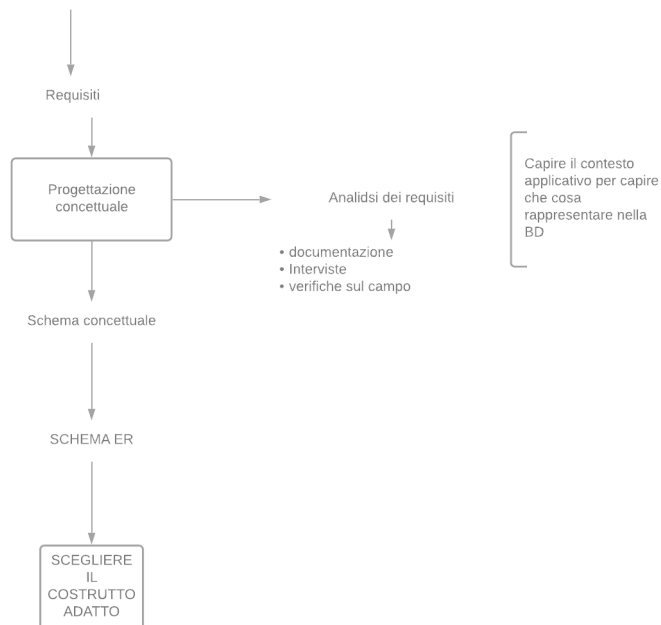
"La fine della terapia deve essere successiva al suo inizio"

"Un paziente non può avere una terapia se non è stato sottoposto a visita"

"La pressione minima deve essere minore della pressione massima"

Esprimo a lato del mio schema R, i vincoli di integrità referenziale che non posso esprimere altrimenti.

## Progettazione concettuale



Per costruire il mio schema ER devo scegliere il costrutto adatto per il concetto considerato.

Per capire parto da come abbiamo descritto i costrutti

- Se il concetto che voglio rappresentare ha delle proprietà significative e descrive oggetti con esistenza autonoma
  - Entità**
- Se il concetto è semplice e non ha esistenza autonoma
  - Attributo**
- Se il concetto lega due o più concetti
  - Relazione**
- Se il concetto è il caso particolare di un altro
  - Gerarchia di generalizzazione**

## Strategia di progettazione

- Top Down**

Faccio un'analisi dei requisiti generale, costruisco una gerarchia generale e la vado a raffinare finché non ottengo quello che mi serve .> se la situazione è molto estesa è un approccio complicato.

- **Bottom Up**

Suddivido il contesto applicativo in frammenti e di questi frammenti costruisco lo schema (capendo un pezzettino alla volta) e poi integro i vari schemi.

- **Mista**

La strategia che si usa effettivamente è una combinazione degli approcci Top Down e Bottom Up. Cerco di raffinare lo schema, partendo da punti diversi e integrando.

## Qualità dello schema ER

Lo schema concettuale deve avere determinate qualità:

1. **Correttezza**

Posso dire che è corretto quando usa correttamente i costrutti messi a disposizione.

2. **Completezza**

Uno schema è completo quando rappresenta tutti i dati di interesse.

3. **Leggibilità**

Uno schema concettuale è leggibile quando rappresenta tutti i quesiti e concetti in maniera naturale e comprensibile.

4. **Minimalità**

Non deve essere ridondante, quindi tutte le specifiche sono rappresentate una sola volta

Esercizio di esempio:

Si vuole rappresentare una base di dati per una società che eroga corsi, di cui vogliamo rappresentare i dati dei partecipanti e dei docenti.

Partecipanti identificati da un codice, rappresentiamo il CF, il cognome, l'età, la città di nascita, i nomi degli attuali datori di lavoro e di quelli precedenti (insieme alla data di inizio e fine rapporto), le edizioni dei corsi che stanno frequentando e di quelli che hanno frequentato nel passato, insieme alla relativa votazione.

Per i partecipanti che sono liberi professionisti rappresentiamo l'area di interesse e se lo possiedono il titolo professionale.

Per i partecipanti che sono dipendenti rappresentiamo il loro livello e la posizione ricoperta.

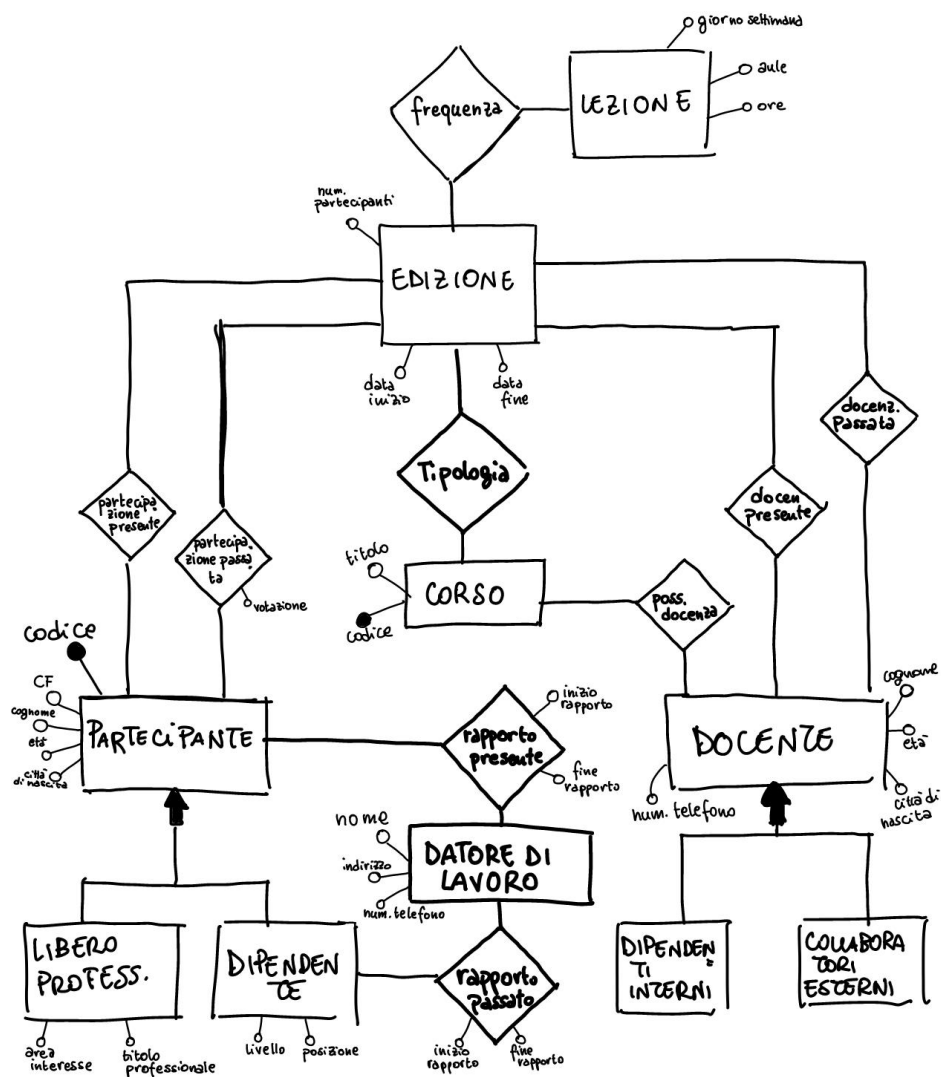
Relativamente ai datori di lavoro presenti e passati, rappresentiamo il nome l'indirizzo e il numero di telefono.

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

Per i docenti rappresentiamo il cognome, l'età, la città di nascita, tutti i numeri di telefono, il titolo del corso che insegnano di quelli che hanno insegnato nel passato e di quelli che possono insegnare

I docenti possono essere dipendenti interni della società di formazione o collaboratori esterni.





## Lezione 13

### Progettazione logica

A questo punto quello che dobbiamo fare dallo schema concettuale (ER) dobbiamo produrre lo schema relazionale. Per farlo dobbiamo ristrutturare lo schema ER, che non è direttamente mappabile.

#### 1. Effettuare l'analisi delle eventuali ridondanze

Decidere in base alle operazioni che verranno fatte sulla base di dati, se è vantaggioso tenere la ridondanza, oppure no.

#### 2. Eliminare le gerarchie

Tutte le generalizzazioni vanno eliminate e sostituite con altri costrutti, con entità e relazioni.

#### 3. Partizionamento o accorpamento di entità

Dovremmo decidere se concetti separati nel nostro schema vanno accorpati, oppure il contrario.

#### 4. Se per una certa identità ho più identificatori, bisogna scegliere gli identificatori primari

Es. Ho codice fiscale e matricola, decido quale usare come identificativo

## Ridondanza

Un'informazione significativa che può essere derivabile da altre informazioni presenti nello schema.

### Vantaggi:

- Semplificare le interrogazioni

### Svantaggi:

- Occupo più spazio
- Aggiornamenti appesantiti

Gli attributi derivabili potrebbero essere derivabili da altri attributi della stessa entità o relazione, da attributi di altre entità o relazioni o da operazioni di conteggio.

Relazioni derivabili da composizione di altre relazioni.

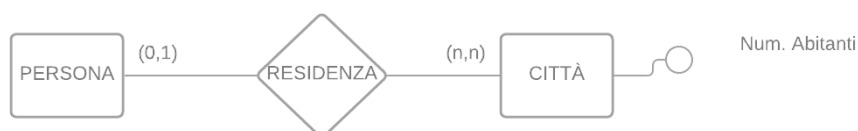
*Es. il numero di abitanti di una città è derivabile dalle occorrenze della relazione cittadino*

Relazioni che generano la presenza di cicli.



**Attenzione:** non sempre la relazione che genera la presenza di cicli, non è detto che per forza generi una relazione ridondante.

## Come capire se è utile mantenere una ridondanza o meno?



CONCETTO	TIPO	VOLUME
CITTÀ	E	200
PERSONA	E	1 mln
RESIDENZA	R	1 mln

**Operazione 1.** Memorizzare una persona con la relativa città di residenza

effettuata 500 volte al giorno

**Operazione 2.** Stampa tutti i dati di una città, compreso il numero di abitanti

effettuata 2 volte al giorno

## Devo capire se mi conviene o no mantenere la ridondanza

### Ipotesi 1: Mantengo la ridondanza

- Per l'operazione 1 - che tipo di accesso facciamo al nostro schema?

Per aggiungere una nuova persona, dovrò accedere una volta in scrittura a persona e una volta in scrittura a residenza.

Una volta fatta questa cosa dovrò accedere anche a città per incrementare il numero di abitanti

CONCETTO	COSTRUTTO	INGRESSO	TIPO
PERSONA	E	1	S
RESIDENZA	R	1	S
CITTÀ	E	1	L
CITTÀ	E	1	S

- Per l'operazione 2 - che tipo di accesso facciamo al nostro schema?

Accedo solo in lettura al numero degli abitanti

CONCETTO	COSTRUTTO	INGRESSO	TIPO
CITTÀ	E	1	L

**Totale:** (3 scritture + 1 lettura ) x 500 volte al giorno + (1 lettura) x 2 volte al giorno = 1500 in scrittura + 2 accessi in lettura = circa 3500 di costo

## Ipotesi 2: Tolgo la ridondanza

- Per l'operazione 1 - che tipo di accesso facciamo al nostro schema?

Per aggiungere una nuova persona, dovrò accedere una volta in scrittura a persona e una volta in scrittura a residenza.

CONCETTO	Costrutto	Ingresso	Tipo
PERSONA	E	1	S
RESIDENZA	R	1	S

- Per l'operazione 2 - che tipo di accesso facciamo al nostro schema?

Accedo alla città e poi accedo a tutte le relazioni di residenza nella città (ipotizziamo siano 5000

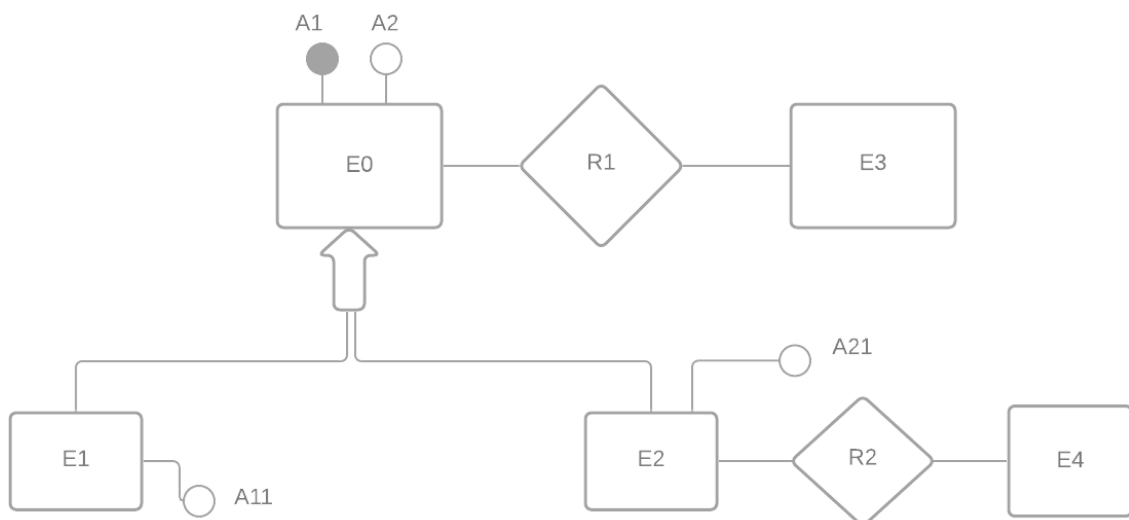
CONCETTO	Costrutto	Ingresso	Tipo
CITTÀ	E	1	L
RESIDENZA	R	5000	L

**Totale:** (2 scritture) x 500 volte al giorno + (5001 lettura) x 2 volte al giorno = 1000 in scrittura + 10 002 accessi in lettura = circa 11 000 accessi

È chiaro che sia più conveniente mantenere la ridondanza.

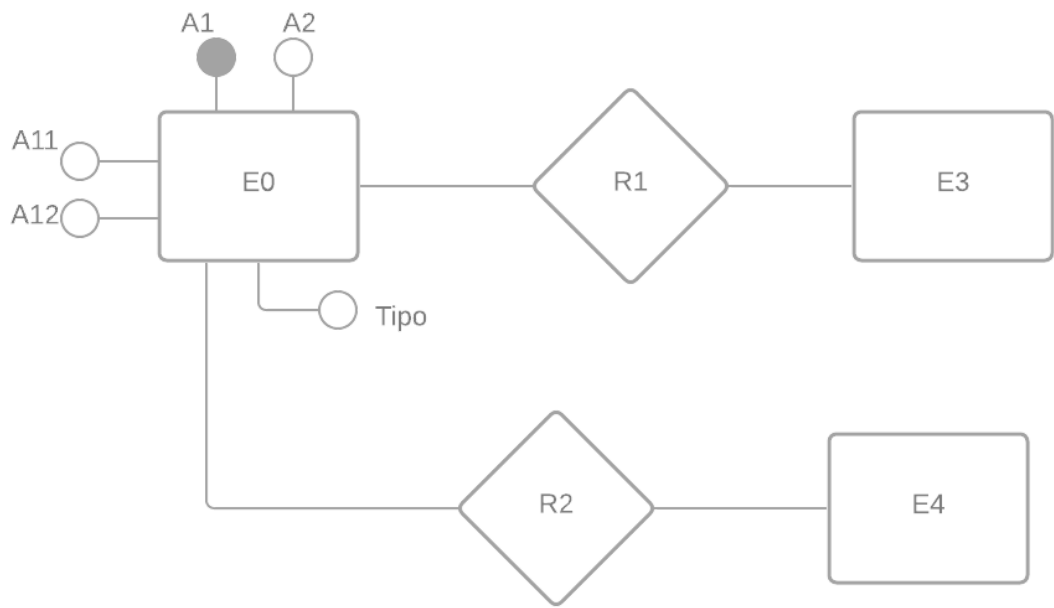
## Eliminare le gerarchie

Vogliamo sostituire la gerarchia con entità-relazioni.



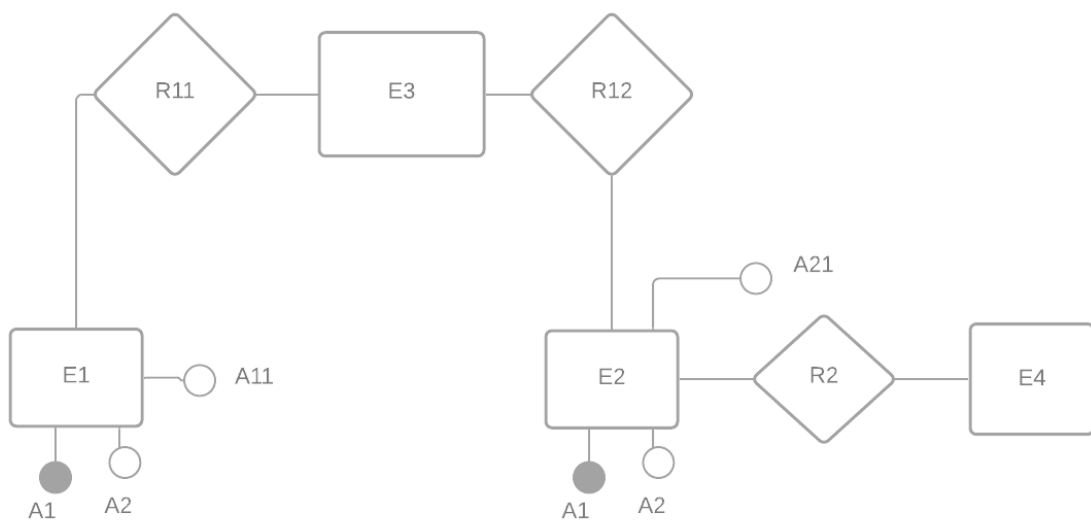
- Accorpamento delle figlie nel padre

Mi conviene se gli accessi alle entità figlie e al padre sono contestuali: quando accedo ai dati delle entità figlie di fatto accedo anche a quelli del padre.



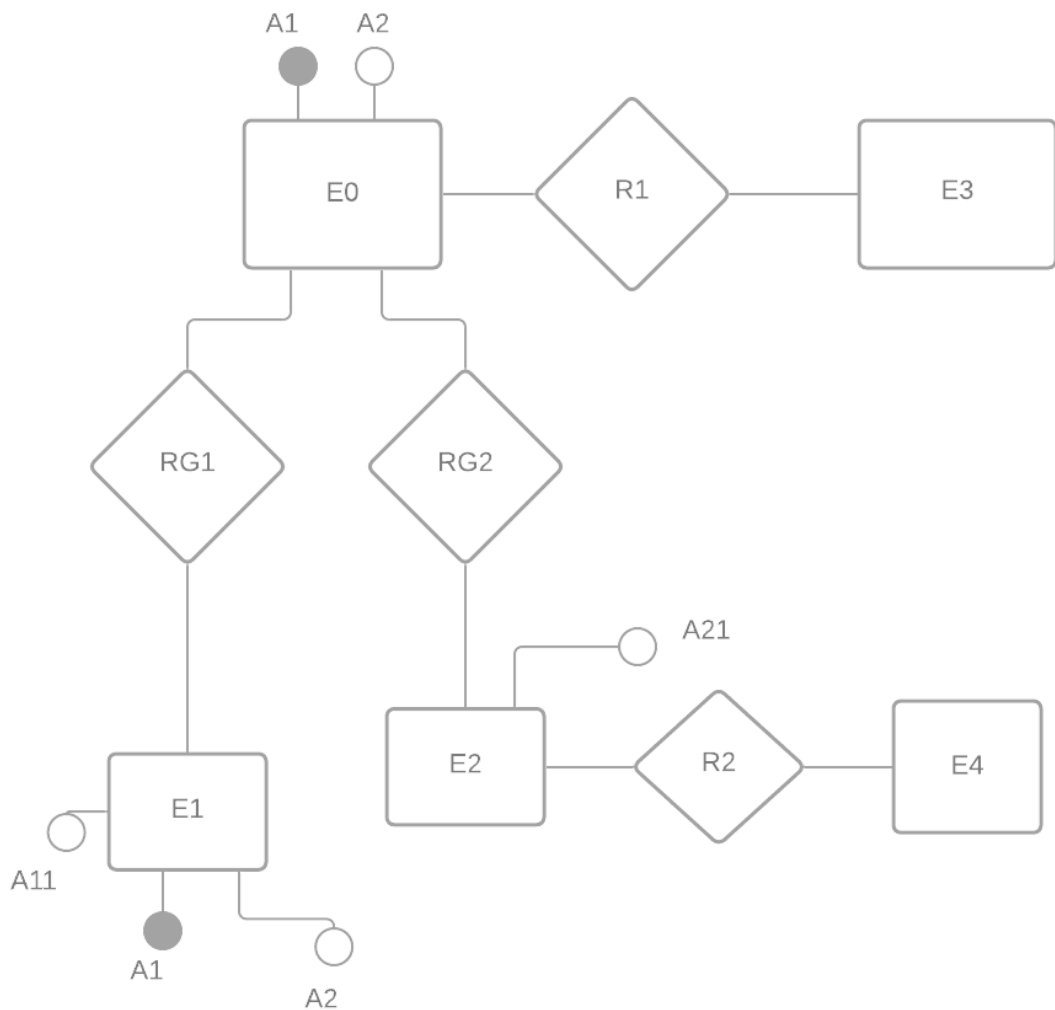
- Accorpamento del padre nelle entità figlie

Se gli accessi alle figlie sono distinti, mi conviene accorpare il padre nelle figlie



- Sostituzione della generalizzazione con associazioni (quindi con relazioni)

Mi conviene se ho accessi separati a padre e figlie



## Partizionamento o accoppiamento di entità

Ad esempio se ho molti attributi su un'entità, posso creare una relazione con un'entità formata dai dati (es. dati lavorativi e dati anagrafici vengono separati) → questo conviene quando abbiamo query separate tra i due tipi di dati.

Dobbiamo evitare anche gli attributi multivalore.

## Traduzione verso il relazionale



L'idea di base è: **le entità del nostro schema diventano relazioni nel modello relazionale dei dati sugli stessi attributi.**

*Ad esempio se io ho PERSONA(CF, Nome, Cognome) nel modello ER, lo trasformo in relazionale con CF come chiave primaria.*

La **relazione del modello ER**, diventa una relazione del modello relazionale sulle chiavi delle entità coinvolte più eventuali attributi propri.

Se nello schema c'è una **relazione ricorsiva**, ci comportiamo nello stesso modo: la relazione ricorsiva diventa una relazione, in questo caso le entità coinvolte sono due volte le stesse.

Le relazioni ternarie seguono la stessa idea.

Casi particolari: **relazioni 1 a molti**

È conveniente importare nell'entità la chiave primaria dell'altra entità cui è associata, così da eliminare la relazione (che avrebbe una entry per ogni entità in ogni caso).

Nella **relazione 1 a 1** vale lo stesso ragionamento, se la **partecipazione è opzionale** importo le chiavi nella tabella dove c'è uno come cardinalità minima (non dalla parte dell'opzionale).

La **partecipazione opzionale da entrambe le parti** la si traduce come da molti a molti, quindi facendo la terza tabella.