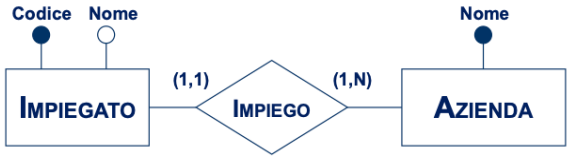
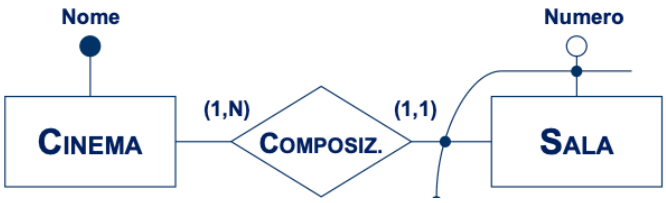
Design pattern

Wednesday, 22 March 2023 11:05

- Quando noi programmiamo, vogliamo poter creare delle funzioni riutilizzabili per il futuro, perché? Così il lavoro che faccio oggi lo posso riciclare anche per domani.
  Un qualcosa simile alle funzioni esiste anche in basi di dati, questo è chiamato design pattern.
- Il design pattern non è altro che, una documentazioni di casi comuni a cui noi possiamo fare riferimento. Why? Così facendo, in futuro, quando ci ritroviamo ad un sistema simile abbiamo praticamente tutto il lavoro già fatto, e dobbiamo solo fare copia ed incolla.
  Quindi, se sei pigro/a come me, fatti tanti casi comuni siccome, così facendo lavorerai di meno in futuro.
  Ecco dei casi comuni:
  - Noi abbiamo una situazione dove
    L'entità Impiegato ha attributo [Codice, nome, azienda]
    Notiamo che azienda non dovrebbe essere un attributo ma un entità
    Allora il nuovo grafico sarà:

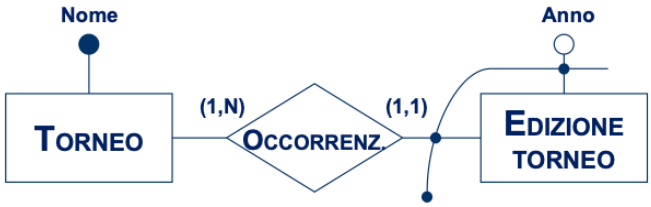


  - Se ci pensiamo, a volte quando noi estraiamo un attributo, possiamo notare delle discrepanze (forse non è la parola giusta però è figa e mi fa sembrare ~~rincoglionito che non sa l'italiano~~ figo)
    Tipo, abbiamo un cinema con attributi [Nome, Sala]
    Notiamo che è possibile estrarre sala e li diamo come chiave "numero stanza"
    Però, possiamo avere più cinema, e più cinema avranno la stanza n^1
    Un modo per sistemare questo è rendere sala dipendente da cinema con "part-of"



    E' come se aggiungessimo una chiave esterna.
    Nota che, part of bisogna metterlo solamente quando abbiamo una dipendenza
  - Simile alla part-of esiste la istance-of

- Simile alla part of esiste la istanza of
  Praticamente è come se facessimo una extend

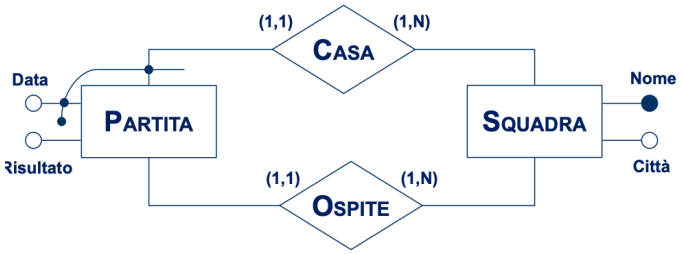


  E quindi c'è tutta la questione di ereditarietà

Esempi partici

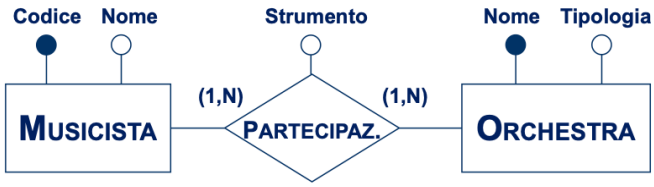
- Relazione = prodotto del piano cartesiano
  Quindi, nella relazione ricorsiva/riflessiva
  Non possiamo avere 2 volte S1 S2

  Per risolvere



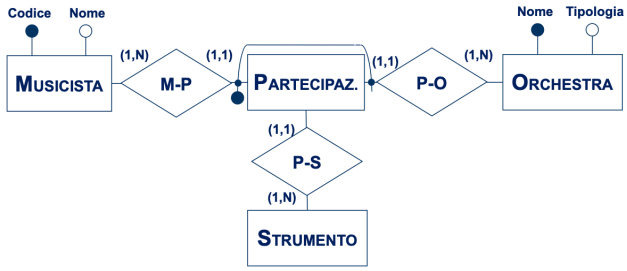
  Perché?
  P1S1S2->P2S1S2->P3S1S2

-

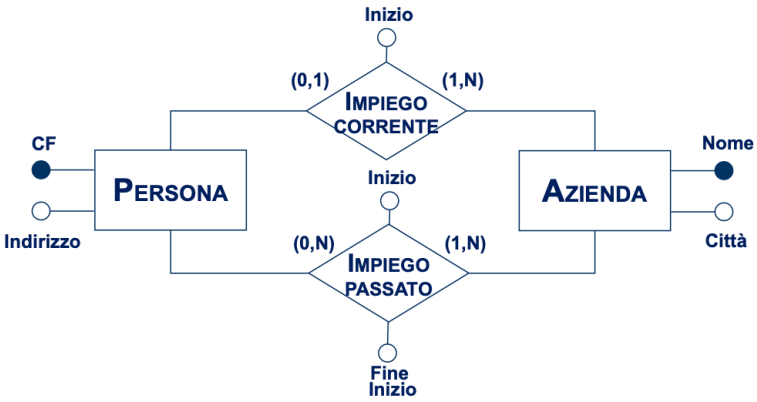


  M1O1
  Quindi musicista 1 partecipa ad orchestra 1
  Però, mettiamo caso il musicista partecipa nella stessa orchestra con 2 strumenti
  Questo non può succedere siccome
  M1O1S1 == M1O1S2
  Quindi dobbiamo per forza rendere partecipazione una entità e anche

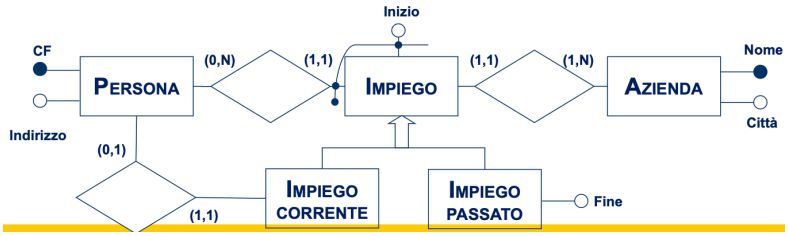
strumento



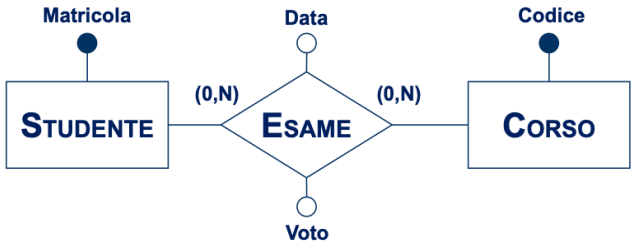
- Un altro problema



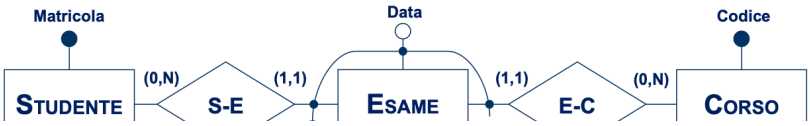
Qui mettiamo caso io sono impiegato di A
Poi me ne vado, ritorno, e me ne vado dinuovo
Come posso dire che io sono stato impiegato 2 volte? Qui non posso
P1A1 Data 50 == P1A1 Data 100
Quindi dobbiamo creare impiego una entità.
E siccome già che ci siamo, facciamo una generalizzazione di impiego



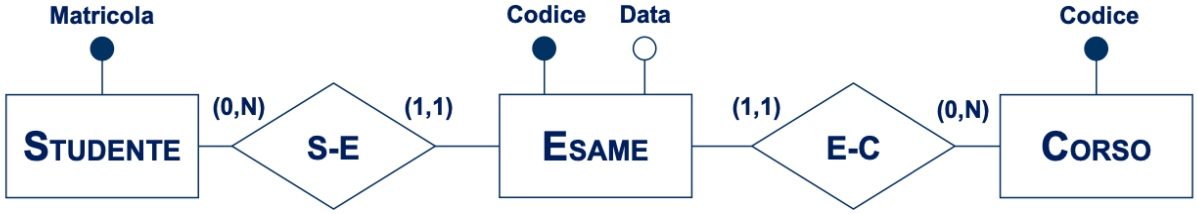
-



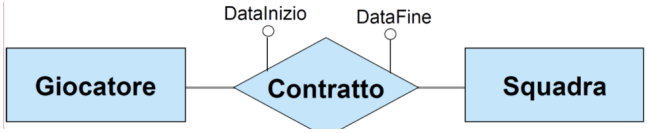
Noi qui abbiamo il problema che
S1C1E1 == S1C1E2
Aka uno studente non può effettuare più esami per uno stesso corso.
La soluzione di questi problemi è rendere esame una entità,



Ora guardiamo attentamente questa soluzione
Noi stiamo dicendo che, in esame abbiamo come chiave primaria Data
E che è strettamente collegato con studente e corso, quindi abbiamo 2 chiavi esterne
S1D1C1
Questo potrebbe fare sorgere un problema: E se uno studente facesse 2 esami in 1 stessa data?
S1D1C1E1 == S1D1C1E1
In questo caso l'idea migliore è dare una chiave primaria ad esame, codice, così facendo però possiamo notare che, un esame non è più dipendente da studente e corso.
Quindi dobbiamo togliere anche le chiavi esterne (part of)



- E' corretto?



Risposta: No siccome un giocatore non può fare parte della stessa squadra 2 volte
G1S1D1 == G1S1D2
Quindi, design pattern e si trasforma la cosa in entità