BASI DI DATI
SECONDO SEMESTRE 2023

PROF. SCHETTINI[T2]

Andrea Falbo - Quack

Capitolo 1: Introduzione al corso

Sistema *Informativo*: componente di un'organizzazione che gestisce le informazioni di interesse
Sistema *Informatico*: parte del sistema informativo che gestisce informazioni con tecnologia informatica.

Nei sistemi informatici le informazioni vengono rappresentate attraverso i *dati*.

Database (DB): collezione di dati utilizzati per rappresentare le informazioni di interesse di un sistema informativo.

Data Base Management System (*DBMS*): Sistema software capace di gestire collezioni di dati che siano:

- *grandi*: il limite deve essere solo quello fisico dei dispositivo
- *condivise*: ciascun settore ha un sistema informativo
- *persistenti*: hanno un tempo di vita indipendente dalle singole esecuzioni dei programmi che le utilizzano.
- *affidabilità*: resistenza a malfunzionamenti hardware e software
- *privatezza*: si possono definire meccanismi di autorizzazione

In ogni base di dati esistono:

- lo *schema*, invariante nel tempo, descrive la struttura (intestazione). Aspetto intensionale
- l'*istanza*, i valori attuali, cambiano rapidamente (il corpo). Aspetto estensionale

Due tipi di modelli:

- *concettuali*: cercano di descrivere i concetti del mondo reale (ER)
- *logici*: Adottati nei DBMS esistenti per l'organizzazione dei dati (Relazionale)

Linguaggi per basi di dati:

- Linguaggi per la definizione dei dati (*DDL*)
- Linguaggi di manipolazione dei dati (*DML*)

SQL ha le funzionalità di entrambe le categorie

Capitolo 2: Modello ER

Introduzione

Una metodologia è un'articolazione in fasi ad una attività di progettazione. Si basa su un principio di separazione netta fra cosa fare e come farlo. Prevede 3 fasi:

1. progettazione *concettuale*: traduce i requisiti del sistema informatico in una descrizione formalizzata, integrata delle esigenze aziendali. Si crea uno *schema concettuale*. Modello E-R.
2. progettazione *logica*: consiste nella traduzione dello schema con concettuale nel modello dei dati del DBMS. Il risultato è lo *schema logico*. Modello Relazionale.
3. progettazione *fisica*: completa lo schema logico ottenuto con le specifiche proprie dell'hw/sw scelto. Il risultato e' lo *schema fisico*

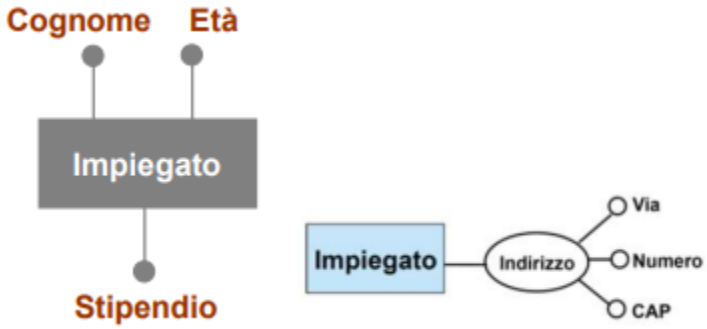
Modello ER

Entità: Classe di oggetti con proprietà comuni e con esistenza autonoma. Nome espressivo e singolare.

Istanza: Oggetto della classe che l'entità rappresenta. Non vengono rappresentate.

Attributo: proprietà locale di un'entità, di interesse ai fini dell'applicazione.

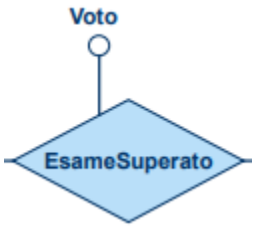
- Attributi *composti*: si ottengono raggruppando attributi di una medesima entità o relazione che presentano affinità nel loro significato.



Relazione: Fatto che descrive una situazione e che stabilisce legami logici tra istanze di entità. Ha un nome al singolare e si usano sostantivi piuttosto che verbi.

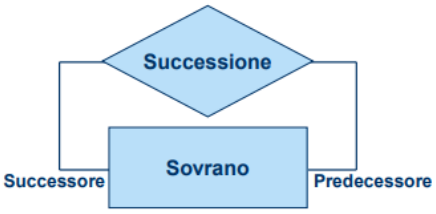
Istanza di relazione: combinazione di istanze di entità che prendono parte all'associazione.

Attributo di relazione: proprietà locale di una relazione di interesse ai fini dell'applicazione.



Associazione *ricorsiva*: un'associazione può coinvolgere "due o piu' volte" la stessa entità.

Ruoli: specifica come l'entità prende posizione in un associazione ricorsiva.



Cardinalità: Coppia di valori che si associa a ogni entità che partecipa a una relazione. Specificano il minimo ed il massimo numero di occorrenze della relazione cui ciascuna entità può partecipare:

- Cardinalità minima: 0 se opzionale, 1 se obbligatoria.
- Cardinalità *massima*: 1 se limitata, N se illimitata.

Relazioni

Classificazione di relazione: in base alla cardinalità massima abbiamo 3 tipi di relazione:

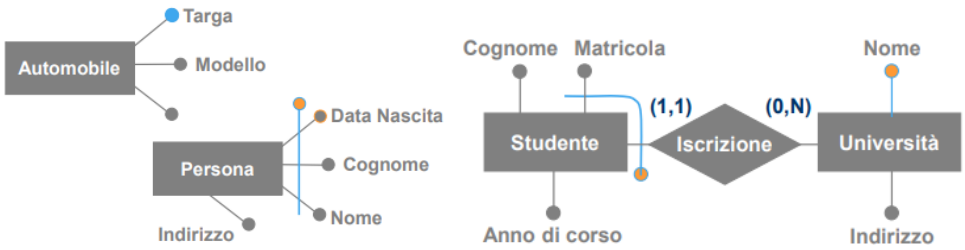
- *Uno a uno*, se entrambe le entità hanno cardinalità massima 1

- *Uno a molti*, se una ha cardinalità 1 e l'altra N

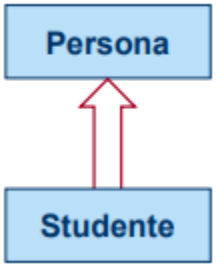
- *Molti a molti*, se entrambe hanno cardinalità massima N

Identificatore: strumento per l'identificazione univoca delle occorrenze di un'entità. Ogni entità deve averne almeno uno. Se l'identificatore è un solo attributo si denota con un pallino nero, se invece abbiamo più attributi si uniscono gli attributi attraverso una linea con un pallino nero. Può essere:

- Identificatore interno: solo attributi interni alla relazione.
- Identificato *esterno*: attributi + entità esterne attraverso relazioni. Possiamo trovarlo solo in relazioni a cui l'entità da identificare ha cardinalità (1,1).

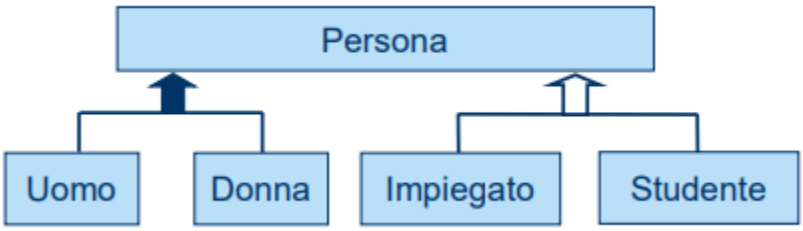


Relazione IS-A: si definisce tra due entità, che si dicono entità padre ed entità figlia, quando un'istanza di una classe lo è anche dell'altra. Si rappresenta nel diagramma dello schema concettuale mediante una freccia dalla sottoentità alla entità padre. Ogni proprietà dell'entità padre è anche una proprietà della sottoentità, e non si riporta esplicitamente nel diagramma. L'entità figlia può avere ovviamente ulteriori proprietà.



Generalizzazione: le sottoentità hanno insiemi di istanze disgiunti a coppie. Può essere:

- Totale: l'unione delle istanze delle sottoentità è uguale all'insieme dell'entità padre (t). Si indica con la freccia nera
- Parziale: l'unione delle istanze delle sottoentità non è uguale all'insieme dell'entità padre (p). Si indica con la freccia bianca.
- Esclusiva: gli elementi di una sottoentità non possono appartenere ad un'altra entità (e)
- Sovrapposta: gli elementi di una sottoentità possono appartenere ad un altra entità (s)



Osservazione: vige la regola che una entità può avere al massimo una entità padre. In altre parole, il modello ER non ammette ereditarietà multipla.

Capitolo 3: Modello Relazionale

Introduzione

Il modello relazionale è il modello di dati più diffuso: permette di definire tipi per mezzo del costruttore relazione che permette di organizzare i dati in insiemi di record a struttura fissa. Rappresenta graficamente un modello ER.

Una relazione è spesso rappresentata da una tabella:

- le *righe*: rappresentano specifici record
- le *colonne*: corrispondono ai campi dei record

In ogni base di dati esistono:

- lo *schema*, sostanzialmente invariante nel tempo, che ne descrive la struttura (aspetto intensionale)
- l'*istanza*, i valori attuali, che possono cambiare anche molto rapidamente (aspetto estensionale)

Chiave

Chiave: insieme di attributi che identificano univocamente le tuple di una relazione

- una *superchiave* è un sottoinsieme di attributi della relazione tale che in nessuna istanza valida della relazione possano esistere due tuple diverse che coincidono su

tutti gli attributi superchiave. Una superchiave quindi identifica univocamente ogni tupla;

- una chiave di una relazione è una superchiave *minimale*, nel senso che se si elimina un attributo, i rimanenti non formano più una superchiave;
- una chiave *primaria* (o primary key) è una delle possibili chiavi, di solito quella con meno attributi.
- una chiave *esterna* è una chiave primaria di una tabella inserita in un altra tabella per collegare le due. Si indica con il nome della tabella a cui si riferisce.

Vincoli di Integrità

Un'informazione incompleta viene rappresentata con valore *nullo*. Può essere:

- valore sconosciuto
- valore inesistente
- valore senza informazione

Vincoli di *integrità*: proprietà che deve essere soddisfatta dalle istanze che rappresentano informazioni corrette per l'applicazione

- valori nulli: valori fuori del dominio di un attributo
- tuple inconsistenti valori di più attributi non simultaneamente assegnabili
- tuple con valori uguali per chiavi
- valori inesistenti in attributi usati per corrispondenze tra relazioni
- vincoli di integrità referenziale tra più relazioni

Vincoli intrarelazionali:

- vincoli su valori (o di *dominio*)
- vincoli di *tupla*
- vincoli di *chiave*

Vincoli interrelazionali:

- vincoli di *integrità referenziale*

Capitolo 4: Progettazione Concettuale

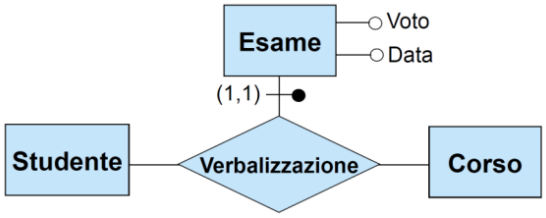
Dalle specifiche al modello ER:

- Se un concetto ha *proprietà significative* e descrive classi di oggetti con esistenza autonoma conviene rappresentarlo con una entità
- Se un concetto ha una *struttura semplice* e non possiede proprietà rilevanti associate conviene rappresentarlo come un attributo di un altro concetto
- Se sono state individuate *due o più entità* e nei requisiti compare un concetto che le associa, questo concetto può essere rappresentato con una relazione
- Se uno o più concetti risultano essere il *caso particolare* di un altro è opportuno rappresentarli facendo uso della generalizzazione

Design pattern è una una soluzione progettuale generale ad un problema ricorrente

Reificazione: il procedimento di creazione di un modello di dati basato su un concetto astratto predefinito

Identificazione esterna: un identificatore esterno può anche non comprendere attributi, e può coinvolgere una sola relazione attraverso un unico ruolo.



Reificazione di relazione ricorsiva: Partita potrebbe essere vista come una relazione binaria tra squadra e se stessa

Strategie di progetto: ciascuna strategia prevede opportune primitive di raffinamento che specificano in che modo sostituire o integrare una parte dello schema con una versione più raffinata della stessa

- *top-down*: Si parte da uno schema iniziale molto astratto ma completo, che viene successivamente raffinato fino ad arrivare allo schema finale
- *bottom-up*: Si suddividono le specifiche in modo da sviluppare semplici schemi parziali ma dettagliati, che poi vengono integrati tra loro
- *inside-out*: Lo schema si sviluppa "a macchia d'olio", partendo dai concetti più importanti, aggiungendo quelli ad essi correlati, e così via.

Capitolo 5: Progettazione Logica

Introduzione

Lo *schema di operazione* descrive i dati coinvolti in un'operazione. Con essi si può fare una stima del costo di un'operazione contando il numero di accessi alle istanze di entità e relazioni. Il risultato può essere riassunto in una *tavola degli accessi*.

Esempio: Stampare il curriculum dello Studente. Si suppone che la media degli esami sostenuti dagli studenti sia 10.

Concetto	*Costrutto*	*Accessi*	*Tipo*
Studente	E	1	L
Esame	R	10	L
Corso	E	10	L

Il tipo distingue gli accessi in scrittura (S) e in lettura (L).

Tavola dei Volumi: specifica il numero stimato di istanze per ogni entità (E) e associazione (R) dello schema. Influenzata da:

- dalle cardinalità nello schema
- dal numero medio di volte che le istanze delle entità partecipano alle relazioni

Attività della ristrutturazione: Abbiamo 5 attività per ristrutturare un modello relazionale attraverso una progettazione logica:

- Analisi delle ridondanze (meno frequente)
- Eliminazione delle generalizzazioni/gerarchie (molto frequente)
- Partizionamento/accorpamento di entità e relationship (meno frequente)
- Eliminazione attributi multivalore (molto frequente)
- Scelta degli identificatori primari (meno frequente)

Le 5 Attività nel Dettaglio

Analisi delle ridondanze: una ridondanza in uno schema E-R è una informazione significativa ma derivabile da altre. Spesso sono attributi *derivabili*, ovvero informazioni che possiamo ottenere da altri attributi della stessa entità o di altre entità. Possiamo derivare anche delle associazioni.

Esempi:

1. attributo derivabile da entità stessa: importo lordo da importo netto + iva.
2. attributo derivabile da altra entità: l'importo di un acquisto è lo stesso del prezzo di un prodotto
3. associazione derivabile: associazione a cascata. Se A è in relazione con B e questo è in relazione con C, la relazione tra A e C potrebbe essere ridondante (a seconda della semantica)

Eliminazione delle generalizzazioni/gerarchie: il modello relazionale non può rappresentare direttamente le generalizzazioni. 3 possibilità:

1. accorpamento delle figlie della generalizzazione nel genitore:
    a. Le entità figlie sono eliminate
    b. Gli attributi delle figlie sono aggiunte al padre
    c. Si aggiunge un attributo per distinguere il tipo
    d. Gli attributi che provengono da una figlia possono essere nulli
    e. Le relazioni che provengono da una sola figlia hanno cardinalità minima pari a 0

    f. conviene se gli accessi al padre e alle figlie sono *contestuali*
2. accorpamento del genitore della generalizzazione nelle figlie:
    a. L'entità padre è eliminata
    b. Gli attributi, le associazioni e l'identificatore del padre sono aggiunti alle figlie
    c. Ogni associazione definita sul padre genera una associazione distinta per ogni figlia
    d. conviene se gli accessi alle figlie sono distinti e solo se la generalizzazione è *totale*
3. *sostituzione della generalizzazione con relazioni*:
    a. Si introduce una relazione uno-a-uno fra l'entità padre e ciascuna entità figlia
    b. Occorre inserire il vincolo che ogni istanza dell'entità padre può partecipare solo ad una relazione di legame con le entità figlie
    c. Se la generalizzazione è totale ogni istanza dell'entità padre partecipa necessariamente ad una (sola) delle relazioni di legame con le figlie
    d. conviene se gli accessi alle entità figlie sono *separati* dagli accessi al padre

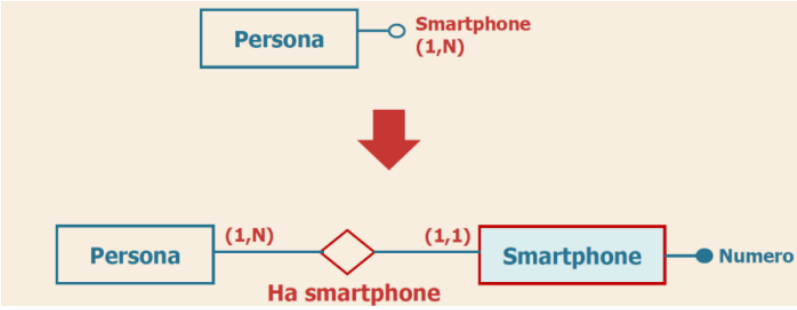
Partizionamento/accorpamento di entità e relationship: Ristrutturazioni effettuate per rendere più efficienti le operazioni. Gli accessi si riducono:

- separando attributi di un concetto che vengono acceduti separatamente
- raggruppando attributi di concetti diversi acceduti insieme

Si può effettuare in 3 modalità:

- *Partizionamento verticale di entità*: Si dividono gli attributi in entità con i relativi attributi più specifici. Es. divido gli attributi di un impiegato in Dati anagrafici e Dati lavorativi
- *Partizionamento orizzontale di relationship*: Stessa situazione ma sulle relazioni.
- *Accorpamento di entità/ relationship*: Scelta contraria alle precedenti. Due entità legate da un'associazione possono essere fuse in un'unica entità contenente gli attributi di entrambe quando le operazioni fanno sempre riferimento a tutti gli attributi delle due entità. Si effettuano per associazioni 1 a 1.

Eliminazione attributi multivalore: Un attributo multivalore viene gestito da una relativa entità con relazione 1 a n oppure n a n.

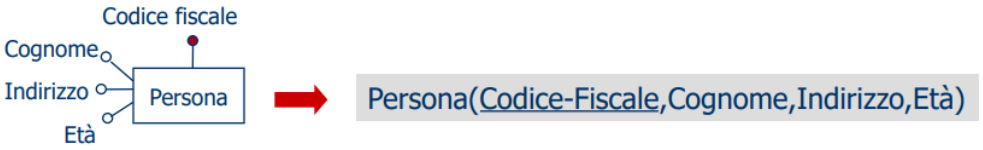


Scelta degli identificatori primari: Criteri:

- assenza di opzionalità (no valori nulli)
- semplicità (preferenza agli identificatori interni, dimensioni ridotte)
- utilizzo nelle operazioni più frequenti o importanti

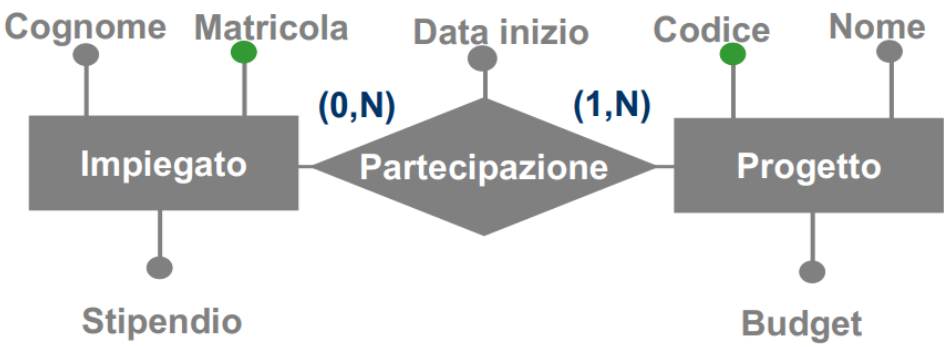
Traduzione verso il modello relazionale

Una entità diviene una relazione definita sugli stessi attributi e con chiave uguale all' identificatore.



Ogni associazione è tradotta con una relazione con gli stessi attributi, cui si aggiungono gli identificatori di tutte le entità che essa collega:

- Gli identificatori delle entità collegate costituiscono una superchiave
- La chiave dipende dalle cardinalità massime delle entità nell'associazione.
- Le cardinalità minime determinano, a seconda del tipo di traduzione effettuata, la presenza o meno di valori nulli



Impiegato(Matricola, Cognome, Stipendio)

Progetto(Codice, Nome, Budget)

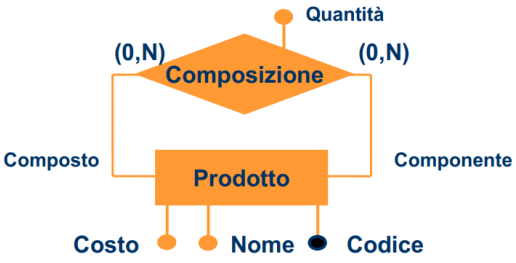
Partecipazione(Matricola, Codice, DataInizio)

- Si possono scegliere nomi più espressivi per gli attributi della chiave della relazione che rappresenta la relationship

Partecipazione(Matricola, Codice, DataInizio)

Traduzione verso il modello molti a molti:

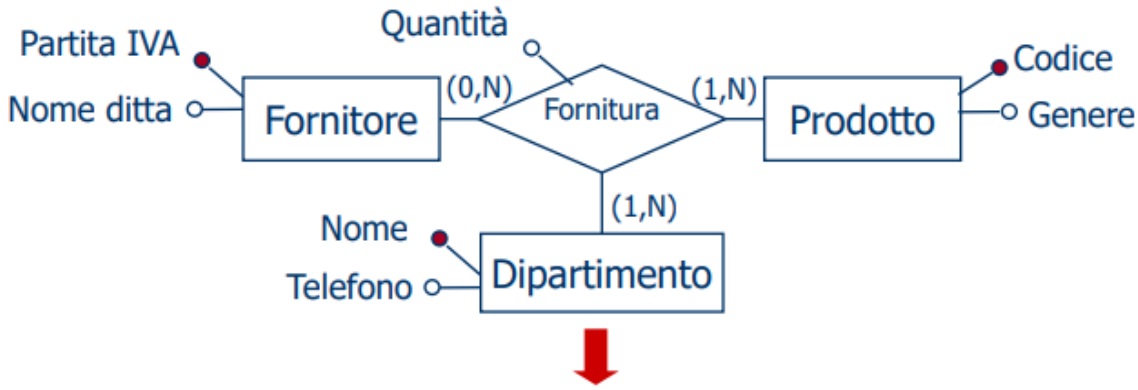
- Le relazioni ricorsive sono un costrutto traducibile, senza trasformazione. Le chiavi della relazione sono i ruoli.



Prodotto(Codice, Nome, Costo)

- **Composizione(Composto, Componente, Quantità)**

- L'associazione prende le chiavi primarie di tutte e 3 le entità. Questa chiave può essere rappresentata attraverso una chiave surrogata che semplifica l'accesso; bisogna ricordarsi però che le 3 chiavi esterne (che ora non sono più chiavi) devono avere dei vincoli di unicità.



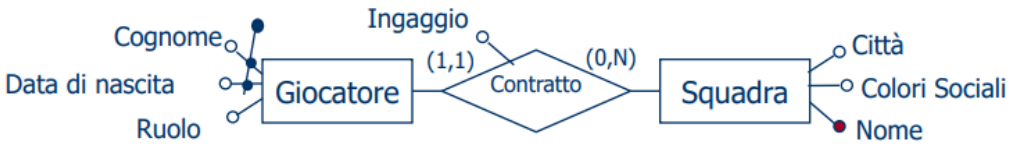
Fornitore(PartitaIVA, NomeDitta)

Prodotto(Codice,Genere)

Dipartimento(Nome, Telefono)

Fornitura(ID, *Fornitore, Prodotto, Dipartimento*, Quantità)

Traduzione verso il modello uno a molti:



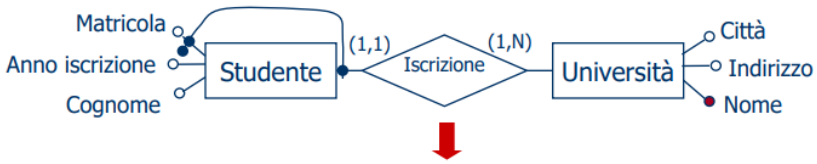
- La chiave per la relazione Contratto è solo l' identificatore dell'entità Giocatore perché la cardinalità verso la relazione è (1,1); mentre l'identificatore di Squadra non è parte della chiave in Contratto in quanto la relazione è (0, N)

Giocatore(Cognome,DataNascita,Ruolo)
Contratto(Giocatore,DataNascitaGiocatore,NomeSquadra,Ingaggio)
Squadra(Nome,Città,ColoriSociali)

- Le relazioni Giocatore e Contratto hanno un'unica chiave e quindi possono essere fuse insieme

Giocatore(Cognome,DataNascita,Ruolo,NomeSquadra,Ingaggio)
Squadra(Nome,Città,ColoriSociali)

Traduzione verso il modello con identificatore esterno:

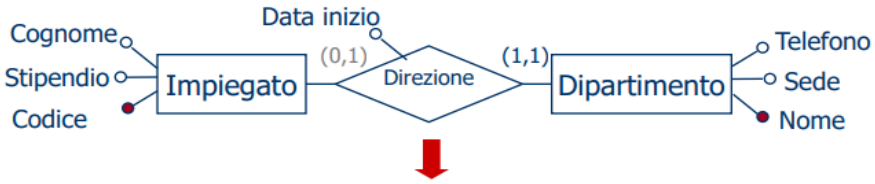


Studente(Matricola,NomeUniversità,Cognome, AnnoIscrizione)
Università(Nome,Città,Indirizzo)

- Rappresentando l'identificatore esterno viene rappresentata anche la relazione fra le due entità

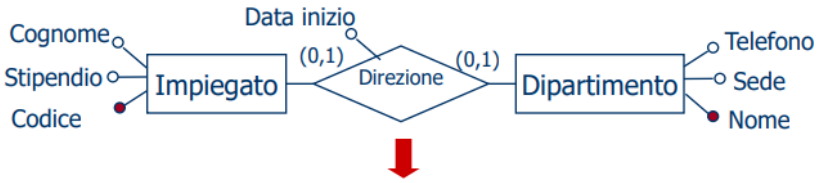
Traduzione verso il modello relazionale relazioni uno-a-uno:

- La relazione è biunivoca e quindi può essere rappresentata in una qualsiasi delle relazioni che rappresentano le entità.
- Preferibile rappresentare la relazione nell'entità che ha cardinalità minima maggiore, evitando così valori nulli frequenti.
- Si potrebbe anche definire un'unica relazione che fonde entrambe ma si perde la distinzione fra le due entità



Impiegato(Codice,Cognome,Stipendio)
Dipartimento(Nome,Telefono,Sede,Direttore,InizioDirezione)

- Se la relazione ha entrambe le entità opzionali è necessario introdurre la relazione come tupla separata.



Impiegato(Codice,Cognome,Stipendio)
Direzione(Direttore,Dipartimento,DataInizioDirezione)
Dipartimento(Nome,Telefono,Sede)

Capitolo 6: Algebra Relazionale

Introduzione

I linguaggi per le basi di dati possono essere:

- Sullo schema - DDL: data definition language
- Sui dati - DML: data manipulation language

Inoltre si dividono in due tipi di linguaggi:

- Procedurali: specificano le modalità di generazione del risultato ("come")
- Dichiarativi: specificano le proprietà del risultato ("che cosa")

I linguaggi per le basi di dati relazionali sono:

- *Algebra relazionale*: procedurale
- Calcolo relazionale: dichiarativo
- *SQL*: (parzialmente) dichiarativo
- QBE (Query by Example): dichiarativo

Capire l'algebra è la chiave per la comprensione dell'SQL.

AR	SQL
Linguaggio prevalentemente formale che forma la base per linguaggi "reali".	Linguaggio più usato per basi di dati relazionali

Linguaggio procedurale: si specifica l'algoritmo con cui ottenere il risultato.	Linguaggio (parzialmente) dichiarativo: si specifica il risultato da ottenere senza preoccuparsi di specificare l'algoritmo.
Istruzioni equivalenti possono differire in termini di efficienza.	Istruzioni equivalenti differiscono solo per leggibilità
Relazioni intese in senso matematico => *Insiemi di tuple*, definite su attributi	Relazioni intese come *tabelle*.
Negli insiemi non ci possono essere elementi uguali.	Possono esserci righe uguali

Operazioni Base

L'algebra relazionale si basa su un insieme di *operatori*:

- su relazioni
- che producono relazioni
- e possono essere composti tra loro a formare nuove interrogazioni

Le operazioni possono essere riassunte in un insieme minimale di operatori che danno l'intero potere espressivo del linguaggio:

- unarie
  - *ridenominazione* ($\rho_{y \leftarrow x}(r)$) con y=nome nuovo e x=nome vecchio e r=relazione
  - *selezione*
  - *proiezione* ($\pi_{listaAttributi}(r)$): produce una relazione sugli attributi specificati che contiene ennuple a cui contribuiscono tutte le ennuple di r.
- binarie
  - *unione* ($\cup$)
  - *intersezione* ($\cap$)
  - *differenza* (-)

- *join*

Una relazione è un insieme quindi:

- non c'è ordinamento tra le diverse tuple
- le tuple sono distinte
- ciascuna tupla è al suo interno ordinata: l'i-esimo valore proviene dall'i-esimo dominio
- è possibile applicare unione, intersezione, differenza solo a relazioni definite sugli stessi attributi

Osservazione: Per poter rilasciare l'ultimo vincolo, abbiamo bisogno di potere uniformare i nomi degli attributi. L'operatore di ridenominazione permette di modificare i nomi degli attributi

Osservazione: Se effettuo proiezione su attributi che non sono una chiave a livello semantico, posso ottenere due righe uguali anche se sono due elementi diversi. Esempio: Proietto Cognome e Nome di Studenti. Non posso avere due righe uguali nella tabella, anche se esistono due studenti con stesso Cognome e Nome ma Matricola diversa, in quanto Matricola, che è la chiave, non è stata proiettata.

La *cardinalità* di una relazione è il numero delle sue ennuple; si indica con |r|

Due espressioni sono *equivalenti* se producono lo stesso risultato qualunque sia l'istanza attuale della base di dati

Operatori Selezione e Join

La *selezione* è l'operatore ortogonale alla proiezione. Se in quest'ultima riduciamo la dimensione dello schema (intensionale), nella selezione teniamo tutti i campi ma diminuiscono le tuple (estensionale)

Sintassi Selezione: $\sigma_{condizione}(r)$

Semantica Selezione: stesso schema dell'operando ma contiene un sottoinsieme delle ennuple dell'operando, ovvero quelle che soddisfano la condizione espressa dall' operatore

L'operatore *Join* è fondamentale in quanto permette di combinare tuple appartenenti a relazioni diverse: produce il sottoinsieme del prodotto cartesiano di due relazioni in cui il valore di determinati attributi coincide.

Esistono diverse varianti dell'operazione di Join:

- Join naturale
- Prodotto Cartesiano
- Theta join
- Join esterno (destro e sinistro)

Join *naturale*: Data la relazione r1(X1) definita sugli attributi X1 e una relazione r2(X2) definita sugli attributi X2, allora ogni tupla del join naturale sarà combinazione di una tupla di r1 con una tupla di r2 sulla base dell'uguaglianza dei valori degli attributi comuni.

Sintassi del join naturale: r1(X1) ⋈ r2(X2)

Semantica del join *naturale*: Produce una relazione definita sull'unione X1X2 degli attributi degli operandi composta da n-ple t che hanno gli stessi valori degli attributi comuni

Osservazione: si tratta di un join *incompleto* in quanto non tutte le tuple degli operandi contribuiscono al risultato. Le tuple che ne stanno fuori sono dette dangling

Osservazione: Se due relazioni sono definite sugli stessi attributi il join naturale equivale all'*intersezione* delle due relazioni

Osservazione: Si parla di *join vuoto* se nessuna tupla contribuisce al risultato

Osservazione: Un join naturale su relazioni che non hanno attributi in comune produce il prodotto cartesiano delle relazioni, perché non essendoci un criterio di combinazione tutte le coppie di ennuple sono combinabili tra loro. sufficiente che il nome dell'attributo sia leggermente diverso perché due attributi siano considerati diversi

Due possibili soluzioni per ottenere un join sensato:

- Ridenominazione degli attributi
- Selezione sul prodotto cartesiano

Per la selezione sul prodotto cartesiano esiste un operatore dedicato: il *theta* join

Sintassi del theta join: $r1(X1) \triangleright\triangleleft_{condizione} r2(X2)$

Semantica del theta join: se r1 e r2 non hanno attributi in comune allora il theta join fa una selezione in base ad una condizione sul prodotto cartesiano delle relazioni.

La condizione del theta join è tipicamente una congiunzione (AND) di espressioni di confronto A θ B dove θ è uno degli operatori di confronto. Se l'operatore utilizzato nella condizione è l'uguaglianza allora si parla di equi-join.

Tipologie di join esterno:

- *Sinistro* (LEFT JOIN): mantiene tutte le ennuple del primo operando, estendendole con valori nulli, se necessario
- *Destro* (RIGHT JOIN): mantiene tutte le ennuple del secondo operando
- Completo (FULL JOIN): mantiene tutte le ennuple di entrambi gli operandi

Sintassi del join esterno : $r1 \triangleright\triangleleft_{LEFT/RIGHT/FULL} r2$

Semantica del join esterno: estende, con valori nulli, quelle ennuple di r1 (opzione left), di r2 (right), o di entrambe (full) che verrebbero escluse dal join tra r1 e r2

Self Join è un join di una relazione con una copia di se stessa per: confrontare tuple, trovare elementi duplicati

Logica a 3 valori: Oltre ai valori di verità Vero (V) e Falso (F), si introduce “Sconosciuto” con ?

Per riferirsi ai valori nulli esistono le apposite condizioni: IS NULL e is NOT NULL.

Per semplificare, soprattutto nel caso di sotto espressioni spesso ripetute è utile avere delle relazioni derivate a partire dalle relazioni definite nello schema di base di dati. A questo scopo, in algebra relazionale è possibile definire delle *viste*, che altro non sono che espressioni a cui viene assegnato un nome.

Interrogazioni in AR

Metodo per produrre interrogazioni in algebra relazionale:

1. Prima cerca di capire se servono dei *join* (e quali join) sulle relazioni in R che devono essere collegate per costruire il risultato finale
2. Poi cerca di capire se vi sono *operazioni insiemistiche*, che non alterano la struttura della relazione ma avvicinano al risultato finale.
3. Poi individua eventuali *Selezioni* necessarie per rappresentare analoghe selezioni esistenti nella interrogazione in linguaggio naturale.
4. Infine definisci su quali attributi fare le *proiezioni*, ed in particolare la Proiezione finale

Capitolo 7: Linguaggio SQL

Query SQL

SQL è il linguaggio per la definizione e la manipolazione dei dati in database relazionali, adottato da tutti i principali DBMS.

A seconda delle features, esistono dei livelli di conformità alla quale si adattano i DBMS:

1. *Entry*, tutti i DBMS rispettano
2. *Intermediate*, versione che soddisfa le esigenze del mercato
3. *Full*, versione con funzioni avanzate che non sono realizzate in DBMS

Ogni espressione dell'algebra relazionale può essere tradotto in SQL ma non viceversa. Adotta la logica a 3 valori.

Istruzioni principali di SQL:

- Operazione di *definizione* schema e modifica:
    - Create: definisce tabelle, database, domini ecc.
    - Alter: modifica attributi e vincoli
    - Drop: elimina database e tabelle
- Operazione di *interrogazione*:
    - Select: Formula query come quelle dell'AR
- Operazioni di *aggiornamento*, possono basarsi sul risultato di una query:
    - Insert: inserisce nuove tuple

    - Delete: elimina tuple
    - Update: modifica tuple

Notazione:

- maiuscolo per termini del linguaggio
- minuscolo per gli argomenti
- parentesi quadre per termini opzionali
- parentesi graffe per termini che possono comparire 0 o più volte
- simbolo | per la selezione tra due argomenti

Select:

- sintassi:
    - Select è presente in ogni query SQL, è una proiezione, scelgo colonne
    - From indica le tabelle da cui vogliamo estrarre le informazioni
    - Where è la selezione, sceglie le righe
- semantica: seleziona tra le ennuple del prodotto cartesiano delle tabelle del FROM, quelle che soddisfano la condizione del WHERE e rappresenta gli attributi del SELECT
- Osservazioni:
    - Se nel FROM si indicano più tabelle, si considera sempre il prodotto cartesiano
    - Se nel SELECT vogliamo tutti gli attributi, usiamo *
    - La SELECT mantiene i duplicati, se non li vogliamo dovremo usare clausola DISTINCT

Condizione *Like* è la selezione di stringhe di caratteri:

- _ indica un singolo carattere arbitrario
- % indica una stringa con un numero arbitrario di caratteri

Esempio: SELECT * FROM Docente WHERE nome like 'A_d%' seleziona i docenti che hanno un nome che inizia per 'A' e ha una 'd' come terza lettera.

La gestione dei valori *nulli* avviene esattamente come in AR

Prodotto cartesiano:

```
SELECT Attributo1,Attributo2,...
FROM Table1, Table2,...
WHERE condizione
```

Mettendo la condizione di uguaglianza tra due tabelle diverse, si crea un *JOIN implicito*

La clausola *PUNTO .* permette di specificare a che relazione appartiene un attributo

Il theta join ha sempre senso, perché si combinano solo le tuple che hanno lo stesso valore per gli attributi specificati nella condizione di join, ma occorre disambiguare attributi omonimi con la clausola punto.

Si possono fare *ridenominazioni* su attributi e anche su relazioni con la clausola AS specificandolo nel FROM

Si può esplicitare il predicato di join direttamente nella clausola FROM utilizzando il costrutto *JOIN ON* anche detto join esplicito:

Esempio:

- Join implicito:

  ```
  SELECT *
  FROM Personale_docente, Stipendio
  WHERE Classe_stipendio=Classe AND Valore>=60000 AND Ruolo='Ricercatore'
  ```

- Join esplicito:

  ```
  SELECT *
  FROM Personale_docente JOIN Stipendio ON Classe_stipendio=Classe
  WHERE Valore>=60000 AND Ruolo='Ricercatore'
  ```

Si può scegliere che join eseguire attraverso i seguenti costrutti:

- INNER JOIN (default)
- LEFT OUTER JOIN (left)
- RIGHT OUTER JOIN (right)
- FULL OUTER JOIN (completo)

Se vogliamo effettuare un self join bisogna effettuare una ridenominazione della tabella con AS.

Operatori Aggregati

L'istruzione SELECT permette di fare più dell'AR.

L'operatore *BETWEEN* permette di specificare il campo in cui deve essere compreso un campo.

L'operatore *ORDER BY* permette di specificare un criterio di ordinamento delle righe. L'ordinamento può essere alfabetico o numerico. Specificato in fondo alla query.

Espressioni aritmetiche nella target list: Attraverso l'uso dei soliti operatori aritmetici: +, -, * , /. Sono calcolate riga per riga.

SQL mette a disposizione anche degli operatori, che non esistono nell'algebra relazionale, che sono valutati su insiemi di tuple, gli operatori aggregati:

- COUNT, restituisce il numero di righe o il numero di valori distinti (con distinct) di un particolare attributo. SELECT COUNT (*| [DISTINCT] ListaAttributi)
- MIN/MAX, restituiscono il minimo o il massimo di attributi numerici o stringhe
- AVG o SUM restituiscono la media o la somma di attributi numerici

COUNT e *valori nulli*:

- Numero di tuple: *
- Numero di tuple senza NULL: count(attributo)
- Numero di tuple distinte senza NULL: count(distinct attributo)

Group By

Negli esempi visti finora le funzioni di aggregazione si applicano al risultato della interrogazione. Le funzioni possono essere applicate a partizioni delle relazioni, cioè a gruppi di tuple con la clausola *GROUP BY*:

- Sintassi:
  SELECT operatoreAggregato (*| [DISTINCT] ListaAttributi)

FROM ...
WHERE ...
GROUP BY listaAttributi

- Semantica:
    - Esegui la interrogazione base senza tener conto del GROUP BY e degli operatori aggregati
    - Raggruppa le righe che hanno stessi valori per gli attributi che compaiono nella lista Attributi della GROUP BY
    - Applica l'operatore aggregato a ciascun gruppo righe

Osservazioni:

- Nonostante si può evitare l'uso dello * per il null, verrà inserito NULL a *0* nell'output.
- La target list non può contenere colonne *non aggregate insieme a colonne aggregate*

Tramite la GROUP BY le righe di una tabella sono raggruppate in sottoinsiemi. Fino ad ora non è possibile esprimere condizioni su sottoinsiemi, in quanto la clausola WHERE va necessariamente indicata prima della clausola GROUP BY. Introduciamo allora *HAVING*, con lo stesso scopo di WHERE ma viene eseguito *dopo* il raggruppamento

HAVING:

- Sintassi:
  SELECT target list
  FROM ...
  WHERE ...
  GROUP BY insieme attributi
  HAVING condizione
- Semantica: La clausola HAVING permette di esprimere condizioni sui gruppi, applicate a ogni insieme di n-ple risultato della applicazione del GROUP BY.

Osservazione: l'operatore aggregato può essere calcolato direttamente nella clausola HAVING

Query Complesse: Quantificatori

Select completo

```
SELECT [operatoreAggregato]([ DISTINCT ] ListaAttributiOEspressioni)
FROM ListaTabelle
[ WHERE CondizioneSelezioneTuple]
[ GROUP BY ListaAttributiDiRaggruppamento ]
[ HAVING CondizioniSelezioneGruppi ]
[ ORDER BY ListaAttributiDiOrdinamento ]
```

Le condizioni atomiche permettono sempre e soltanto di confrontare valori elementari. Ci interessa poter fare un confronto dove la clausola è un'altra query.

Una query dentro ad un'altra query è detta *subquery* o query annidata. Gli operatori di confronto semplici si possono usare solo per subquery che restituiscono un singolo valore. Per usare predicati di confronto con subquery che possono restituire più di una riga, occorre usare le quantificazioni:

- ALL, la condizione è vera se il confronto è vero rispetto ad *almeno* uno degli elementi di una lista
- ANY, la condizione è vera se il confronto è vero rispetto a *tutti* gli elementi di una lista
- [NOT] IN vero se il valore [non] *compare* nell'insieme risultato della interrogazione
- [NOT] EXISTS è invece un quantificatore *esistenziale* che permette di verificare se la query restituisce o meno una tupla

Avremo Semantica Bottom up

Query Complesse: Viste

Esiste un'altro quantificatore esistenziale:

WHERE EXISTS (Sottoespressione) restituisce il valore vero se la interrogazione restituisce un risultato non vuoto. • L'operatore EXISTS ha praticamente senso solo nel caso in cui la query interna faccia riferimento alla query esterna

Esempio: Data la Relazione Studente (Matricola_st, Nome, Cognome) trovare gli studenti che hanno degli omonimi.

```
SELECT *
FROM Studente S1
WHERE EXISTS (SELECT *
                FROM Studente S2
                WHERE S1.Nome = S2.Nome AND S1.Cognome = S2.Cognome AND
                S1.Matricola.st <> S2. Matricola.st)
```

Come faccio a eseguire la interrogazione interna senza assegnare un valore alla variabile S1? La semantica bottom up non è qui possibile.

Quando in un blocco interno si fa riferimento a variabili definite in blocchi più esterni, l'interrogazione nidificata viene valutata separatamente per ogni n-pla prodotta nella valutazione della query esterna. Semantica top down

La visibilità delle variabili in SQL segue la seguente semplice regola: una variabile è visibile nella query che l'ha definita o in una query nidificata in essa, non vale il contrario!

Quando andiamo a creare una subquery nel FROM è come se usassimo la subquery da lista. Obbligatorio ridenominare. Per farlo conviene usare le viste.

Mediante l'istruzione CREATE VIEW si definisce una vista, ovvero una "tabella virtuale":
CREATE VIEW nome Vista AS (query)

Query Complesse: Universali

In SQL non esiste il quantificatore universale, ma le subquery permettono di esprimere la quantificazione universale per mezzo di una doppia negazione: "Non esiste alcun x per cui non vale la proprietà P"

Query Complesse: Operazioni di Aggiornamento

Insert:

- Inserire valori singoli: INSERT INTO Tabella [(Attributi)] VALUES(Valori)
- Inserire insieme di ennuple: INSERT INTO Tabella [(Attributi)] SELECT …
- Regole
    - Se qualche attributo non è specificato, è inserito valore NULL.

- Se è violato un vincolo di not null, l'insert e' rifiutato.
- L'ordinamento degli attributi è significativo
- Le due liste (Attributi, Valori) debbono avere lo stesso numero di elementi.
- Se la lista di attributi è omessa, si fa riferimento a tutti gli attributi della relazione
- Se la lista di attributi non contiene tutti gli attributi della relazione, per gli altri viene inserito un valore di default, o, in assenza di questo, il valore nullo

Delete:

- Eliminazione di ennuple: DELETE FROM Tabella [WHERE Condizione].
- Elimina le ennuple che soddisfano la condizione se la where viene omessa, si intende where true, quindi vengono eliminate tutte le ennuple
- la eliminazione può causare eliminazioni in altre relazioni, se ci sono vincoli di integrità referenziale,

Update:

- UPDATE NomeTabella
  SET Attributo = <Espressione |SELECT … | NULL |DEFAULT>
  [WHERE Condizione]
- Modifica le n-ple che rispettano la (eventuale) condizione. Se non c'è la condizione, le modifica tutte
- Il nuovo valore specificato in SET può essere:
  - Espressione, il risultato della valutazione di una espressione sugli attributi della tabella
  - SELECT, il risultato di una interrogazione
  - 3. NULL, il valore nullo
  - 4. DEFAULT, il valore di default