Database: dalla modellazione del dominio all'implementazione in PostgreSQL 8.3

Ing. Eufemia Tinelli

SisInf Lab

POLITECNICO DI BARI

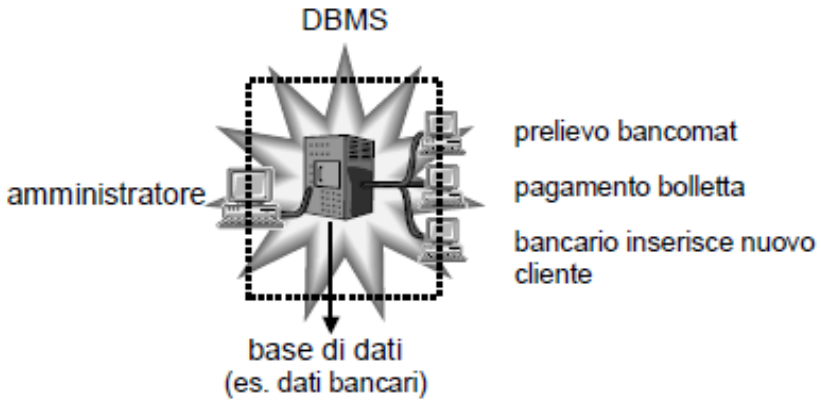
DEE

Argomenti

- Basi di dati & DBMS
  - Concetti generali e proprietà fondamentali
- Come si progetta una base di dati?
  - Progettazione concettuale → logica → fisica
- PostgreSQL 8.3
  - Caratteristiche principali
  - PostgreSQL come ORDBMS
  - PostgreSQL al lavoro…
- Come si interroga una base di dati?
  - Cenni ad SQL come DML (Data Manipulation Language)

Definizioni

- **Sistema Informativo** - I sistemi informativi gestiscono, manipolano, elaborano, archiviano, in generale fanno operazioni su dati. Un Sistema Informativo è chiamato a gestire le informazioni ed i processi informativi in un Sistema Organizzativo
- **Data Base Management System (DBMS)** – Un DBMS è un sistema SW in grado di gestire collezioni di dati che siano *grandi*, *condivise* e *persistenti*, assicurando la loro *affidabilità* e *privatezza*. Un DBMS deve garantire una gestione dei dati:
  - **Efficiente**
  - **Concorrente**
  - **Affidabile**
  - **Integra**
  - **Sicura (protetta)**

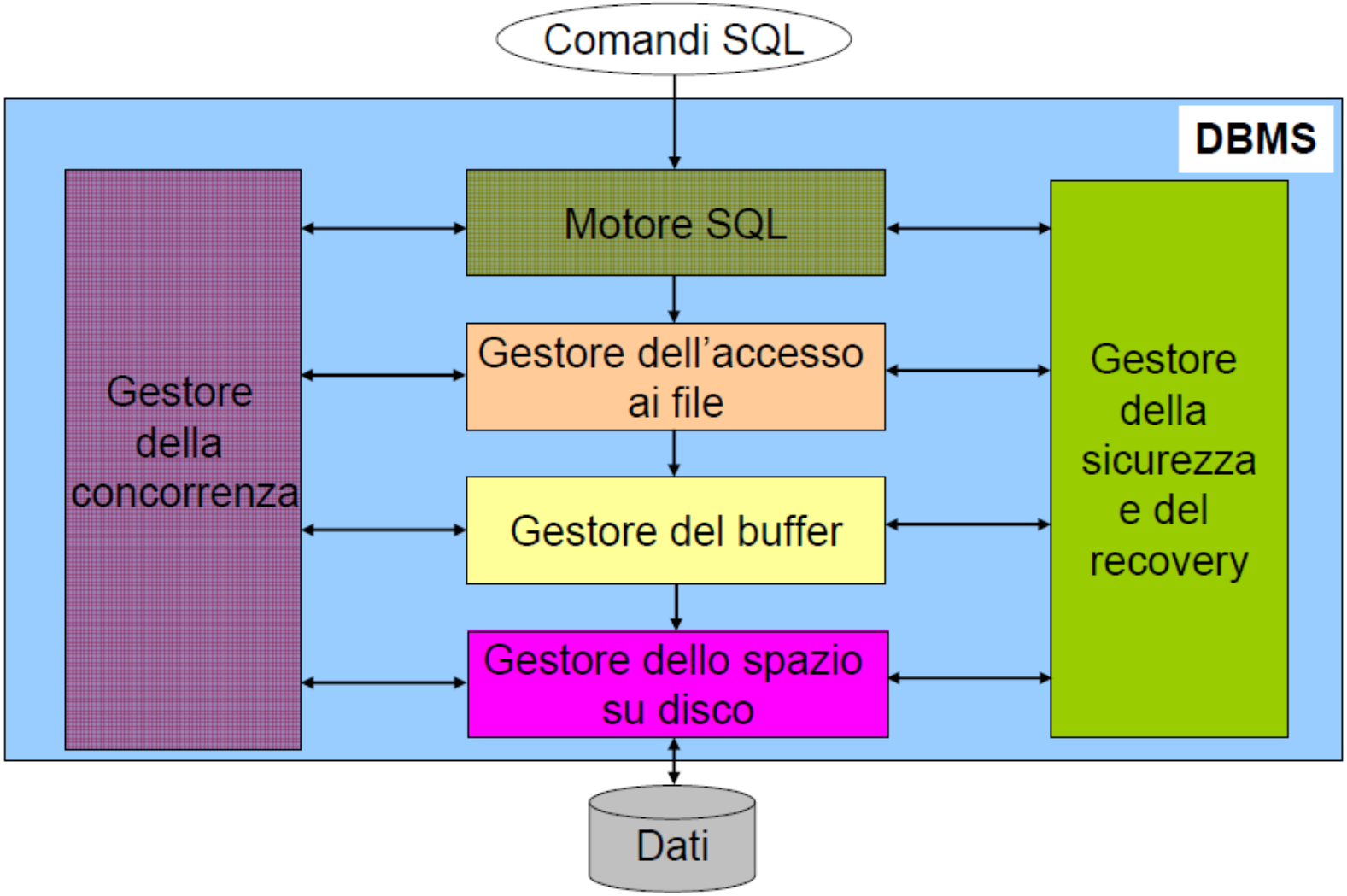


- **Base di Dati** – Una base di dati è una collezione di dati gestita da un DBMS

I principali modelli dei dati

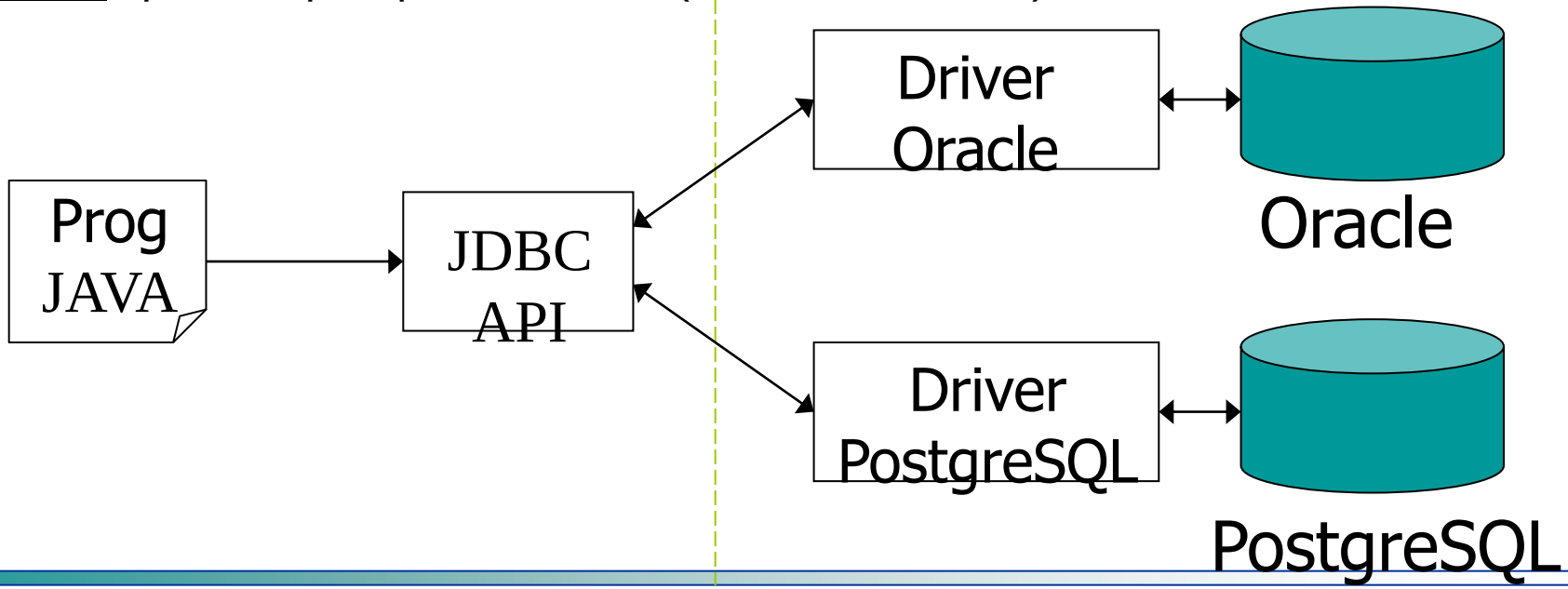
- Il **modello gerarchico**, basato sull'uso di strutture ad albero (e quindi gerarchiche, da cui il nome), definito durante la prima fase di sviluppo dei DBMS (anni Sessanta), ma tuttora ampiamente utilizzato.
- • Il **modello reticolare** (detto anche modello CODASYL, dal comitato di standardizzazione che lo definì con precisione), basato sull'uso di grafi, sviluppato successivamente al modello gerarchico (inizio anni Settanta).
- Il **modello relazionale dei dati**, attualmente il più diffuso, permette di definire tipi per mezzo del costruttore relazione, che consente di organizzare i dati in insiemi di record a struttura fissa.
- Il **modello a oggetti**, sviluppato negli anni Ottanta come evoluzione del modello relazionale, che estende alle basi di dati il paradigma di programmazione a oggetti.

DBMS - Architettura



Come Interfacciarsi ad un database: evoluzione delle CLI

- Utilizza dei driver specifici per ciascun database
- E' uno strato software che si interpone tra l'applicazione e i driver specifici del database. Consente quindi ai programmi applicativi di usare query standard SQL, che accederanno al database, senza necessità di conoscere la particolare interfaccia proprietaria.
- Un modulo del linguaggio di programmazione fornisce una API (serie di funzioni o classi se OO) che permette di interfacciare qualsiasi database tramite un driver specifico per quel database (es. ODBC, JDBC)



Problema della concorrenza

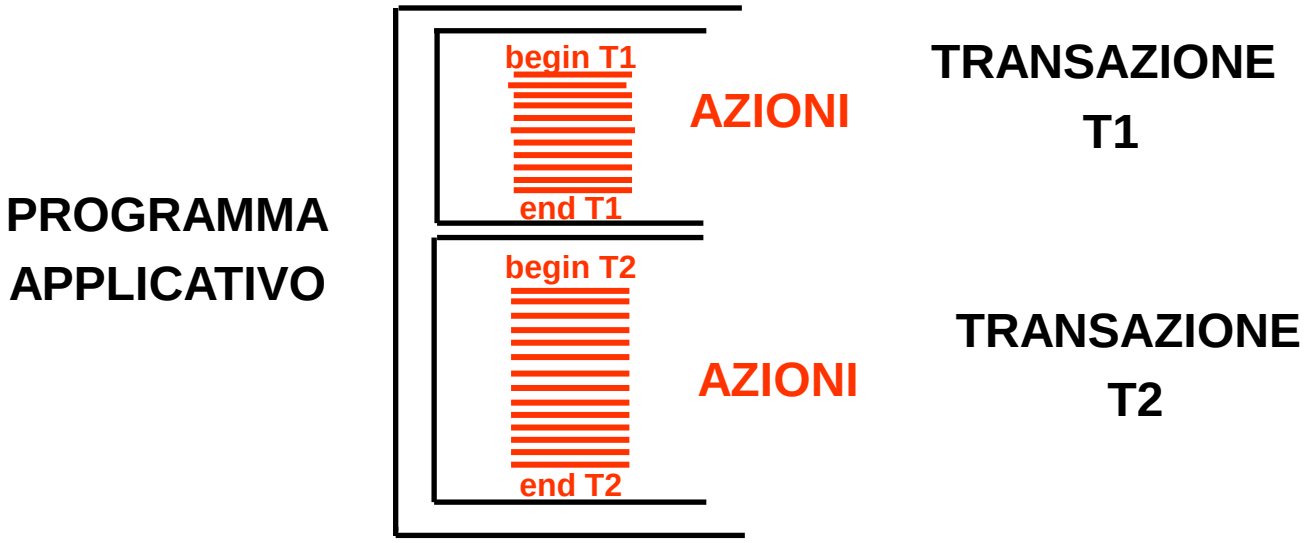
- In un sistema informativo possiamo avere un elevato numero di utenti che accedono anche in maniera concorrente ai dati
- Il sistema deve garantire affidabilità e prestazioni
  - la base di dati deve essere sempre in uno stato consistente
- La gestione della concorrenza e della affidabilità nell'accesso ai dati è una delle proprietà fondamentali che ci offrono i moderni DBMS
- Un concetto cardine è quello di transazione
- Informalmente una **transazione** è una sequenza di operazioni, che può concludersi con un successo o un insuccesso
  - in caso di successo, il risultato delle operazioni deve essere permanente
  - in caso di insuccesso si deve tornare allo stato precedente all'inizio della transazione

Definizione di Transazione

- Transazione
  - parte di programma caratterizzata da un inizio (**begin-transaction,** `start transaction` in SQL), una fine (**end-transaction,** non esplicitata in SQL) e al cui interno deve essere eseguito una e una sola volta uno dei seguenti comandi:
  - **commit work** per terminare correttamente
  - **rollback work** per abortire la transazione

- Un **sistema transazionale** (**OLTP**) e' in grado di definire ed eseguire transazioni per conto di un certo numero di applicazioni concorrenti

Transazione e proprietà ACIDE



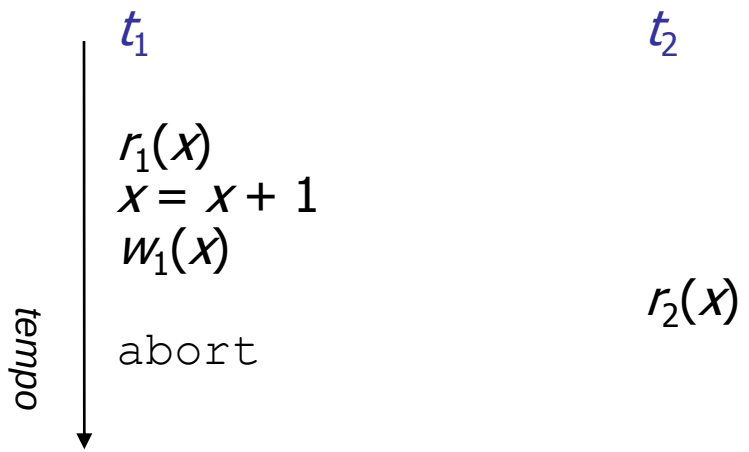
- Una transazione è una unità di elaborazione che gode delle proprietà "ACIDE"
  - Atomicità
  - Consistenza
  - Isolamento
  - Durabilità (persistenza)

Controllo di concorrenza

- Se la proprietà di isolamento non è completamente soddisfatta, due (o più) transazioni effettuate su un database consistente possono portare il database in uno stato inconsistente
- Due azioni sullo stesso oggetto si dicono essere **in confitto** se almeno una delle due è una operazione di scrittura
- Date due transazioni T1 e T2 in conflitto, si possono verificare tre tipologie di situazioni anomale
  - conflitto Write-Read (W-R):
    T2 legge un dato precedentemente scritto da T1
  - conflitto Read-Write (R-W):
    T2 scrive un dato precedentemente letto da T1
  - conflitto Write-Write (W-W):
    T2 scrive un dato precedentemente scritto da T1

Lettura sporca

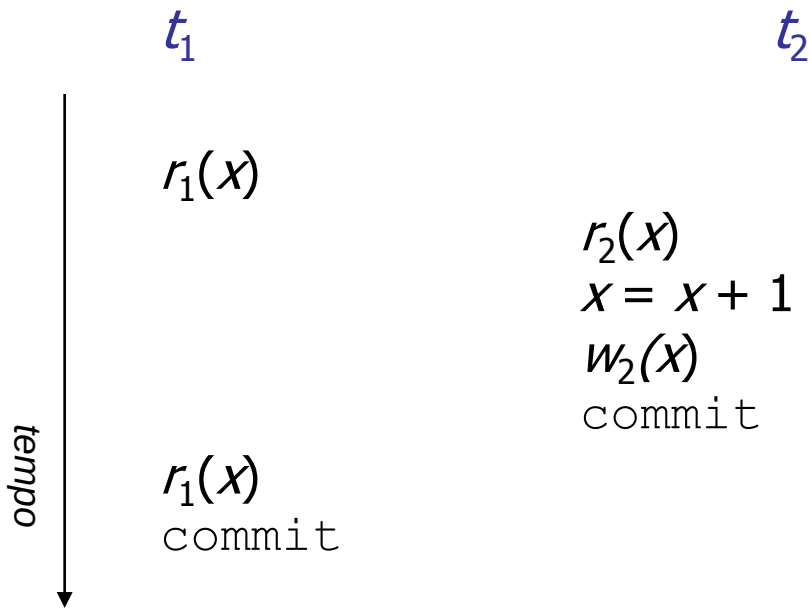
Si può verificare una *anomalia* se una transazione t_2 legge un dato che è stato modificato da una transazione t_1 non ancora conclusa (quindi *uncommitted*)



- t_2 ha letto uno stato intermedio ("sporco") e lo può comunicare all'esterno

Letture inconsistenti

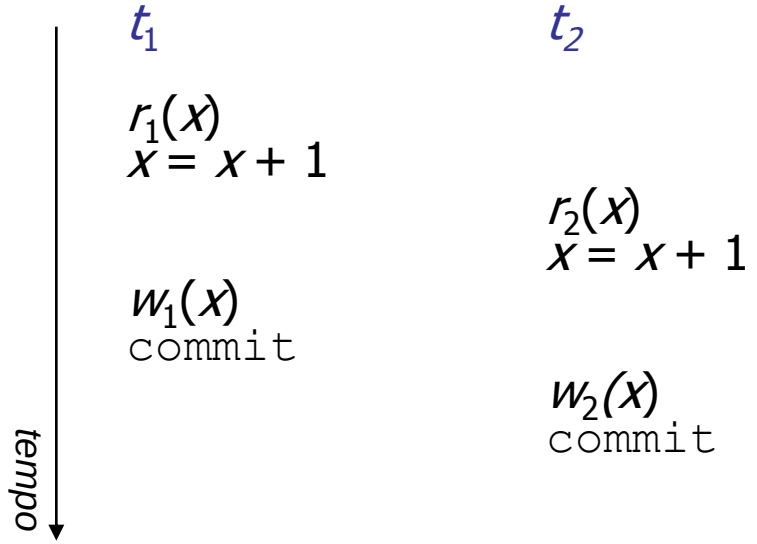
- Si possono verificare diverse anomalie se una transazione t_2 cambia il valore di un dato che è stato letto da una transazione t_1 mentre t_1 è ancora in esecuzione
  - t_1 legge due volte x :



  - t_1 legge due valori diversi per x !

Perdita di aggiornamento

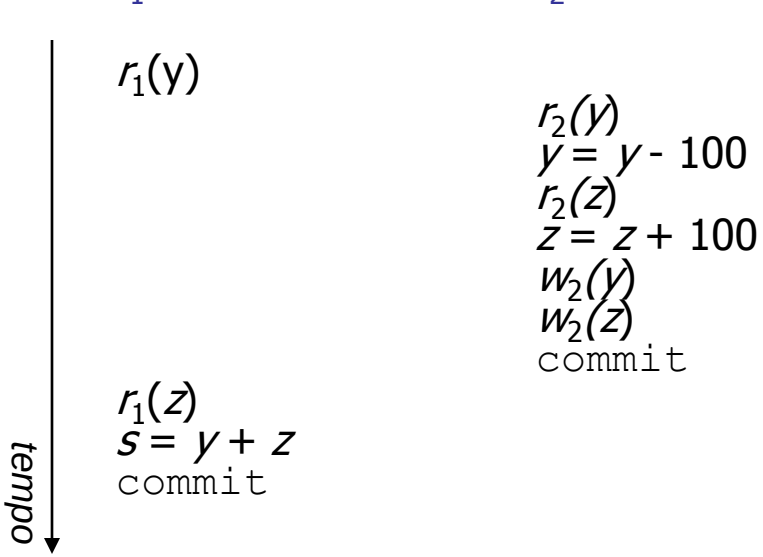
- Due transazioni identiche:
  - t_1: $r(x), x = x + 1, w(x)$
  - t_2: $r(x), x = x + 1, w(x)$
- Inizialmente x=2; dopo un'esecuzione seriale x=4
- Un'esecuzione concorrente:



- Un aggiornamento viene perso: x=3

Aggiornamento fantasma

- Supponiamo che ci sia il vincolo $y + z = 1000$;



- $s = 1100$: ma la somma $y + z$ non è mai stata "davvero" 1100:
  - t_1 vede uno stato non esistente (o non coerente)

Gestione di transazioni mediante lock

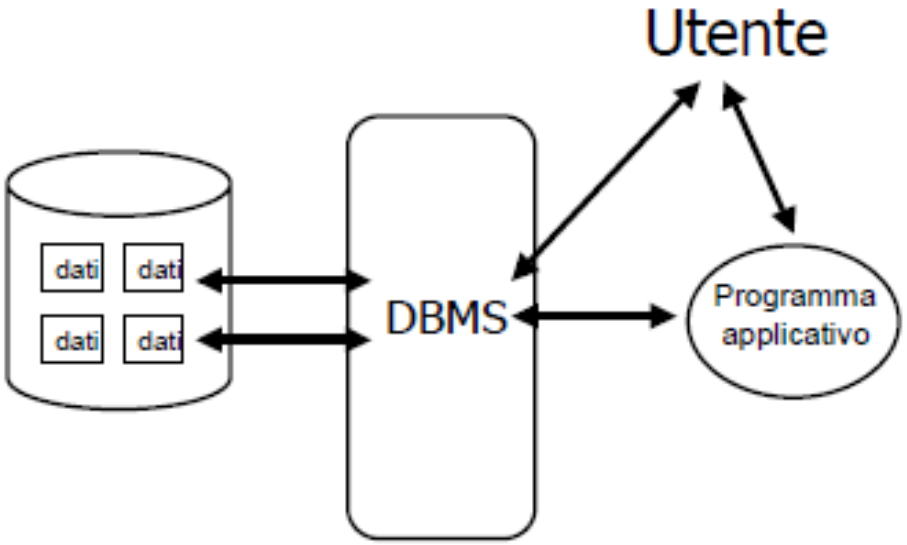
- I DBMS per evitare anomalie nelle transazione concorrenti usano diverse tecniche
- Una delle più comuni è basata su **lock**
- Il lock è un meccanismo che blocca l'accesso ai dati ai quali una transazione accede ad altre transazioni
  - lock a livello di riga, tabella, database
  - lock in operazioni di scrittura/lettura
- Quando una risorsa è bloccata, le transazioni che ne richiedono l'accesso vengono messe in coda

Livelli di isolamento

- Idealmente vorremmo avere sempre garantita la proprietà di isolamento delle transazioni
- Ma questa proprietà ha dei costi che possono limitare le prestazioni del sistema
- I DBMS offrono diversi livelli di isolamento:
  - maggiori restrizioni, minori prestazioni
- Il programmatore deve conoscere i livelli di isolamento e scegliere quello sufficiente ai propri obiettivi
  - Aggiornamento delle disponibilità di magazzino per gli ordini on line?

Livelli di isolamento in SQL

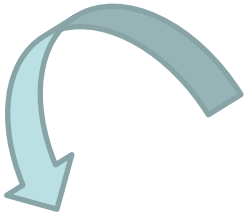
- Il livello di isolamento può essere scelto per ogni transazione che chiede lock in lettura
  - `read uncommitted` permette letture sporche, letture inconsistenti, aggiornamenti fantasma e inserimenti fantasma (transazioni read-only)
  - `read committed` evita letture sporche ma permette letture inconsistenti, aggiornamenti fantasma e inserimenti fantasma
  - `repeatable read` evita tutte le anomalie esclusi gli inserimenti fantasma
  - `serializable` evita tutte le anomalie



Progettazione di una base dati ...

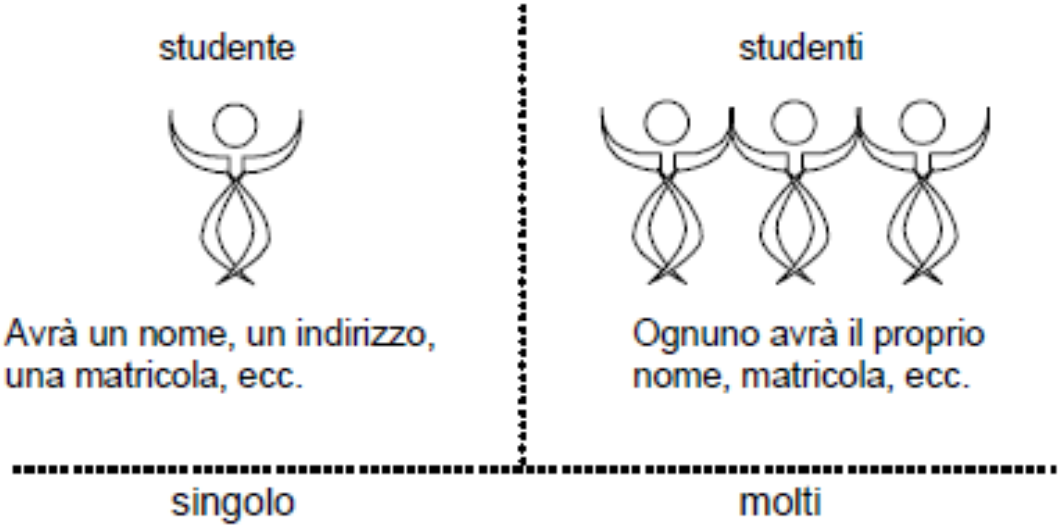
Problema affrontato

- Obiettivo: raccogliere, organizzare, conservare e gestire dati omogenei e strutturati
- ESEMPIO



corso

Avrà un titolo, un docente, dei crediti

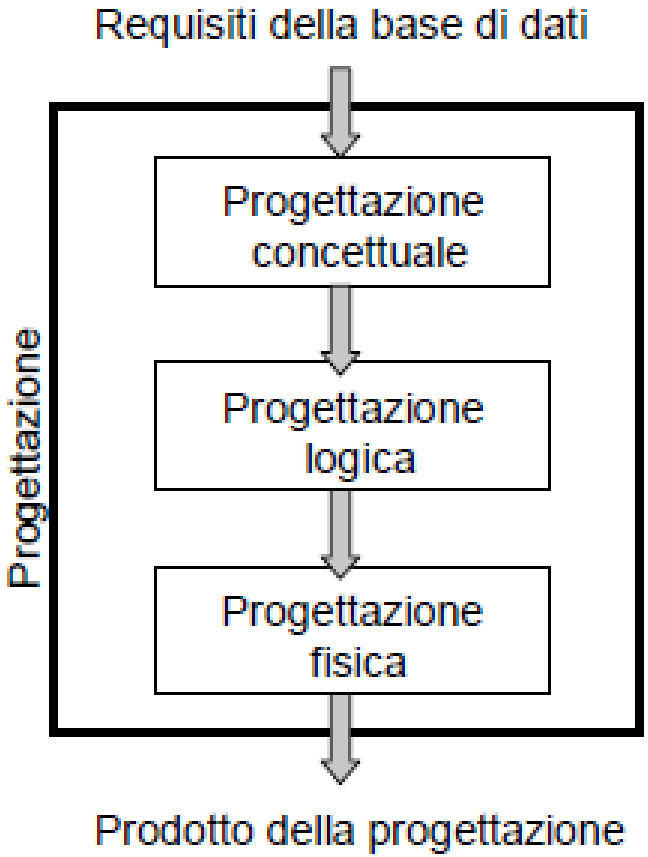


Problema affrontato

- Data una realtà da modellare (es. studenti e corsi)
- Capire quali informazioni sono utili (es. "matricola" è utile per rappresentare gli studenti)
- Capire come le informazioni utili sono correlate (es. chi ha sostenuto quale esame)
- Sapere chi può accedere a quali informazioni per eseguire quali azioni
- Avere strumenti per lavorare sui dati (es. quanti esami ha sostenuto Rossi nel 2002? Con quale media?)

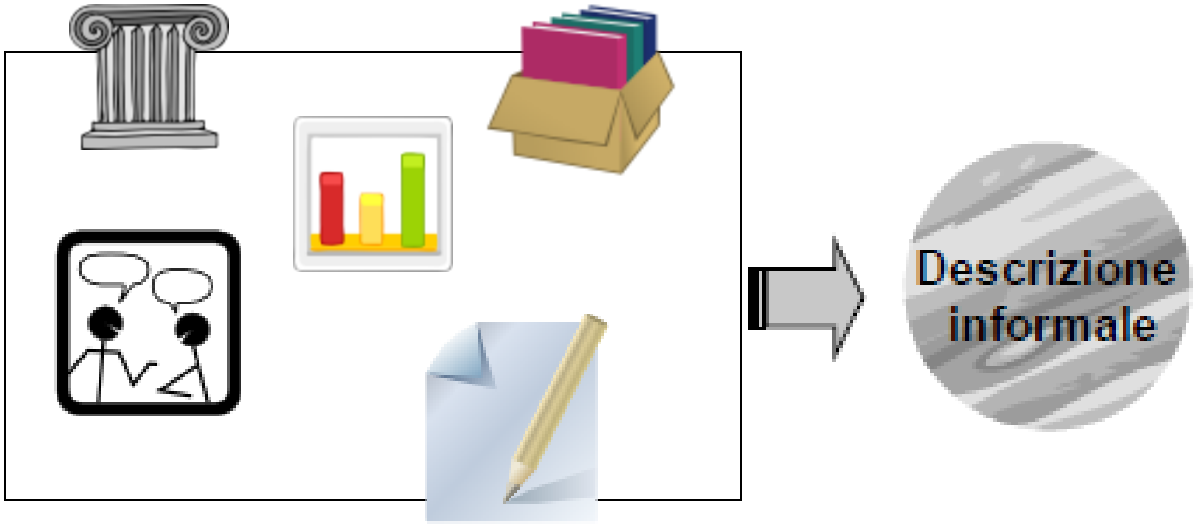
Progettazione di una base di dati

- **Analisi dei requisiti**
  - individuare e studiare le funzionalità che il sistema dovrà fornire
- **Progettazione**
  - (a) concettuale
  - (b) logica
  - (c) fisica
- **Collaudo**
  - verifica del corretto funzionamento del sistema



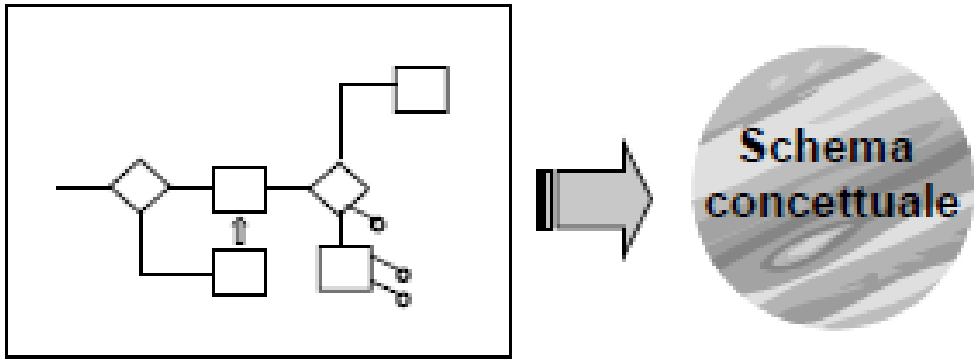
Analisi dei requisiti

- Raccolta e studio delle funzionalità che il sistema dovrà avere. Comporta l'interazione con gli utenti del sistema e si conclude in una descrizione informale dei suoi requisiti



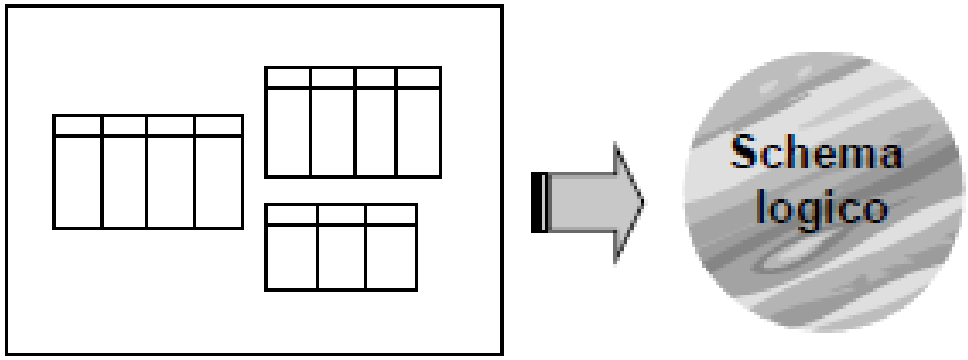
Progettazione Concettuale

- Ha lo scopo di rappresentare la realtà di interesse in termini di una descrizione precisa e completa ma indipendente dai criteri di rappresentazione usati dal sistema informatico scelto per gestire la base di dati (rappresentazione astratta)



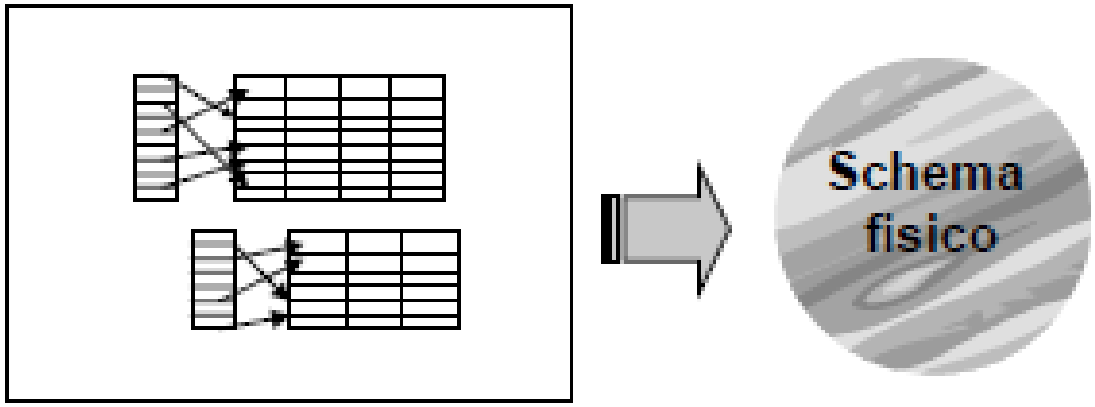
Progettazione Logica

- Ha lo scopo di rappresentare la realtà di interesse in termini di una descrizione ancora indipendente dai dettagli fisici ma concreta, in quanto presente nei sistemi di gestioni delle basi di dati. Lo schema concettuale definito nella fase precedente viene tradotto nello schema logico



Progettazione Fisica

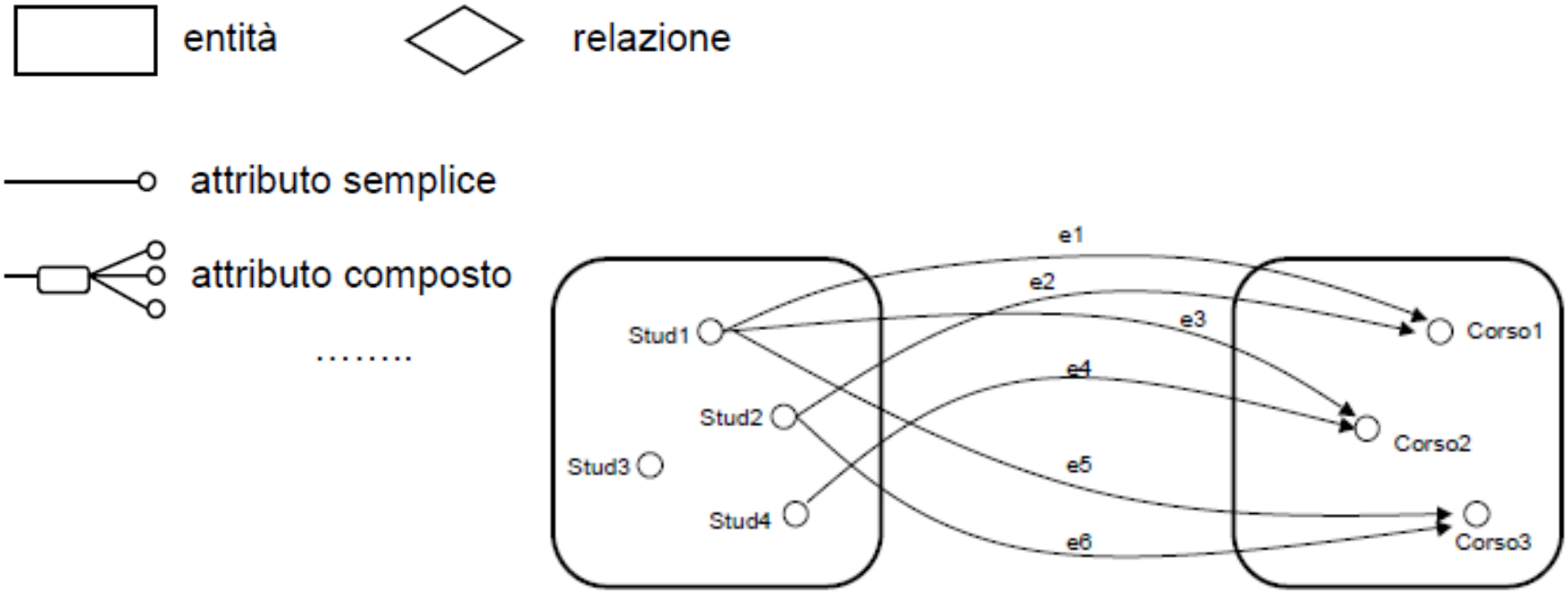
- Lo schema logico viene completato con le specifica dei parametri fisici di memorizzazione dei dati (organizzazione dei file e degli indici). Si definisce lo schema fisico dei dati che dipende dal sistema di gestione di basi di dati scelto



Progettazione Concettuale ...

Il Modello Entità-Relazioni (E-R)

- Consente di rappresentare la realtà di interesse tramite un insieme di costrutti
- Ogni costrutto ha una rappresentazione grafica corrispondente. Ad esempio:



Entità ed Attributi

- Le **Entità** sono ciò che esiste all'interno della realtà che si vuole modellare e di cui ci interessa rappresentare alcuni fatti (le proprietà)
- Le Entità possono essere:
  - **Persone**
  - **Cose**
  - **Eventi**
- **Le proprietà (o attributi)** descrivono caratteristiche di specifiche entità. Esempi:
  - Cognome e nome del cantante
  - Titolo del libro
  - Casa Editrice del libro
  - Data e voto di un esame

Entità ed attributi

- Negli schemi ER le entità (tipi di entità) vengono rappresentate graficamente con rettangoli contenenti all'interno il nome dell'entità. Una entità (nello schema concettuale denota un insieme (o classe) di oggetti "simili". Una istanza dell'entità denota un oggetto dell'insieme rappresentato dall'entità. *Che cosa rappresentiamo nel modello ER?*
- Gli attributi sono le proprietà delle entità:
  - **Attributi semplici:**
    - Sono le proprietà non strutturabili in proprietà più atomiche
  - **Attributi composti o aggregati:**
    - Sono costituiti dall'aggregazione di altri attributi (semplici o composti)
  - **Attributi multipli:**
    - Sono costituiti da un elenco di attributi semplici dello stesso tipo

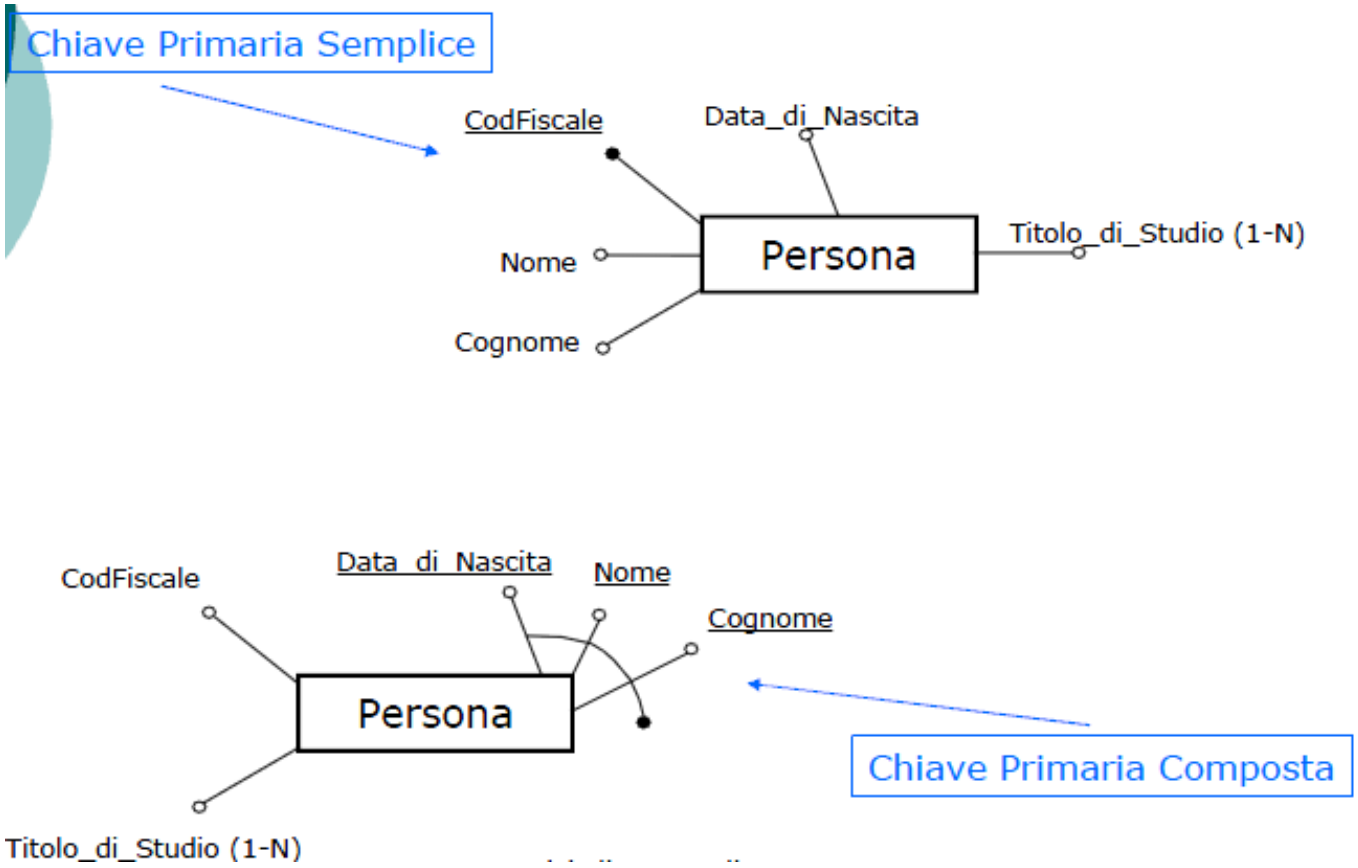
Quando Modellare con un attributo e quando con un'entità

- Nel modellare una realtà di interesse ci troveremo spesso a decidere se modellare un particolare concetto come entità o come attributo.
- La scelta verrà fatta:
  - A seconda del contesto
  - A seconda dell'uso che ne vogliamo fare
- Esempio:
- Nel Conteso Anagrafe il concetto di "Comune" ci porta a pensare al Comune di nascita di una persona → Comune è un **attributo dell'entità persona.**
- Nel Contesto di gestione del territorio il concetto di "Comune" ci porta a pensare ad un'entità che a sua volta avrà i suoi attributi → Comune è un'**entità**

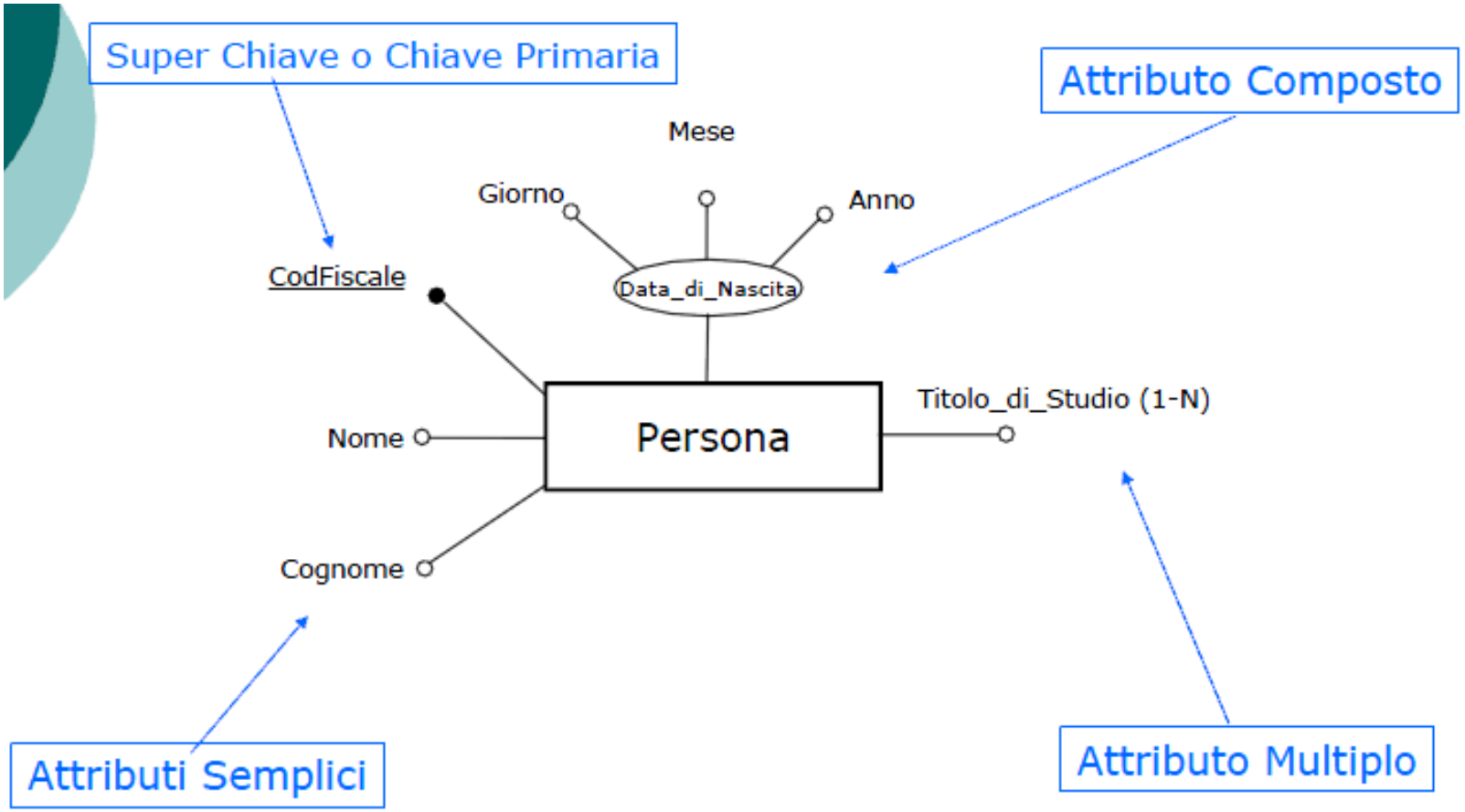
Attributi chiave

- **Chiave:** Insieme di uno o più attributi
- **Chiave candidata:** Una chiave che consente di distinguere in modo univoco un'istanza di entità dall'altra
- **Chiave primaria (primary key) o Identificatore o Super Chiave:** La chiave candidata che si è scelto di prendere come chiave per distinguere univocamente le varie istanze di entità
  - La scelta generalmente viene fatta prendendo la chiave candidata costituita dal minor numero di attributi
  - A volte è possibile aggiunge un **attributo di tipo contatore** che costituisce da solo un buon Identificatore (Chiave Primaria)
- Una chiave primaria semplice viene rappresentata graficamente con il *pallino annerito o sottolineando l'attributo.*
- Una chiave primaria composta viene rappresentata graficamente con un **segmento che unisce gli attributi ed un pallino nero**

Attributi Chiave: Esempi



Attributi atomici, compositi e multivalore



Vincoli di integrità

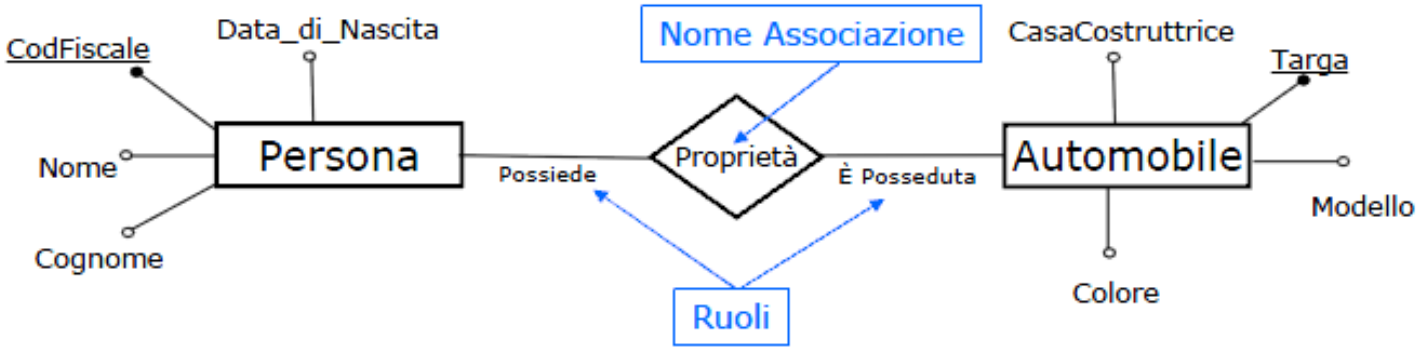
- Con i modelli grafici visti fino ad ora (entità, attributi, associazioni, cardinalità) si possono modellare gran parte delle strutture dati di interesse
- Vi sono alcuni aspetti del mondo reale che potrebbero richiedono l'introduzione di informazioni aggiuntive: **Vincoli di integrità**
- I vincoli di integrità sono delle **restrizioni sui possibili valori relativi ai fatti specifici che si vogliono** rappresentare e **sui modi in cui essi possono evolvere nel tempo**
- Si definiscono vincoli di integrità perché devono essere necessariamente rispettati pena l'integrità stessa dei dati.

Vincoli d'integrità

- I vincoli di integrità sono dei **predicati** che devono essere soddisfatti dalle istanze delle varie entità.
- **Vincoli Impliciti** → imposti dalla stessa struttura dei dati progettata e non richidono predicati specifici se già inseriti nel diagramma ER
  - **Vincoli di Chiave Primaria** → istanze di una categoria devono avere chiave primaria tutte diverse fra di loro
  - **Vincoli Referenziali (vincolo sulla totalità)**→ data un'associazione totale non è possibile inserire un'istanza di un'entità senza inserire almeno un'istanza dell'altra entità associata
- **Vincoli Espliciti** → sono vincoli che impongono il modo in cui i dati possono cambiare
  - Esempio: Il valore dell'attributo Età di un'entità persona non può essere ne negativa ne maggiore di 120.

Relazione

- Una **relazione (o associazione)** si definisce su due o più entità, e rappresenta un **legame fra tali entità.** Ogni relazione ha un nome che la identifica in modo univoco nello schema, ed è rappresentata nel diagramma ER da un rombo collegato alle entità sulle quali è definita la relazione
- Esempio:
  - Tra l'entità "**Persona"** e l'entità **"Automobile"** esiste un'associazione "**Proprietà**" che descrive il fatto che "una persona possiede una o più automobili" e viceversa "un'automobile e posseduta da una persona"



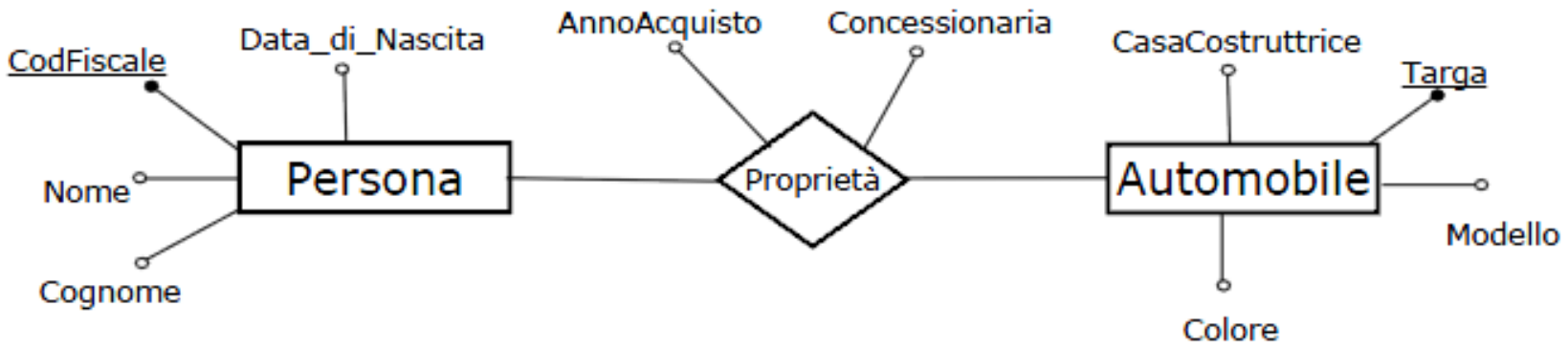
Traduzione di una relazione

- Nel tradurre una relazione ER in una tabella relazionale, gli attributi della tabella devono includere:
  - chiavi per ciascun insieme di entità partecipante (come chiavi esterne)
    - questo insieme di attributi forma la superchiave per la relazione
  - tutti gli attributi descrittivi

```
CREATE TABLE Lavora_In(
cf CHAR(15),
rid INTEGER,
dal DATE,
PRIMARY KEY (cf, rid),
FOREIGN KEY (cf)
REFERENCES Impiegati(cf),
FOREIGN KEY (rid)
REFERENCES Reparti(rid))
```

Associazioni con attributi

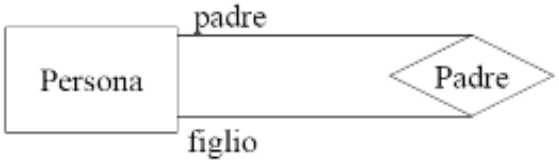
- Un'associazione può avere anche delle **proprietà** analogamente a quanto visto per le entità
- **N.B.** *Non si parla di chiavi di associazioni come invece avviene per le entità!!*



Grado di un'associazione

- **Grado di un'associazione** → Numero di Classi di entità che partecipano ad una associazione

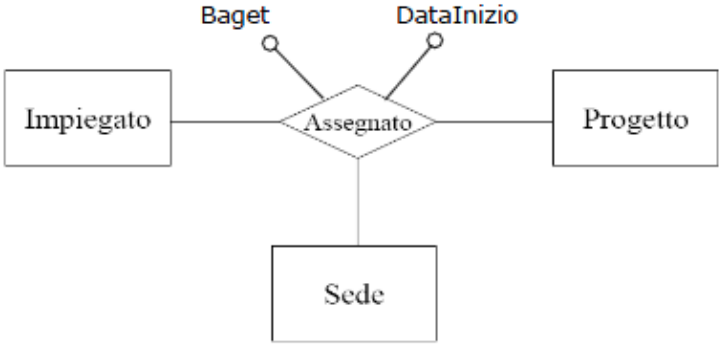
- Associazione Unaria: **Grado 1**



Nel caso di associazione unaria il ruolo è obbligatorio

- Associazione Binaria: **Grado 2**

- Associazione n-aria (multiple): **Grado n > 2**



Associazione totale e parziale, Molteplicità

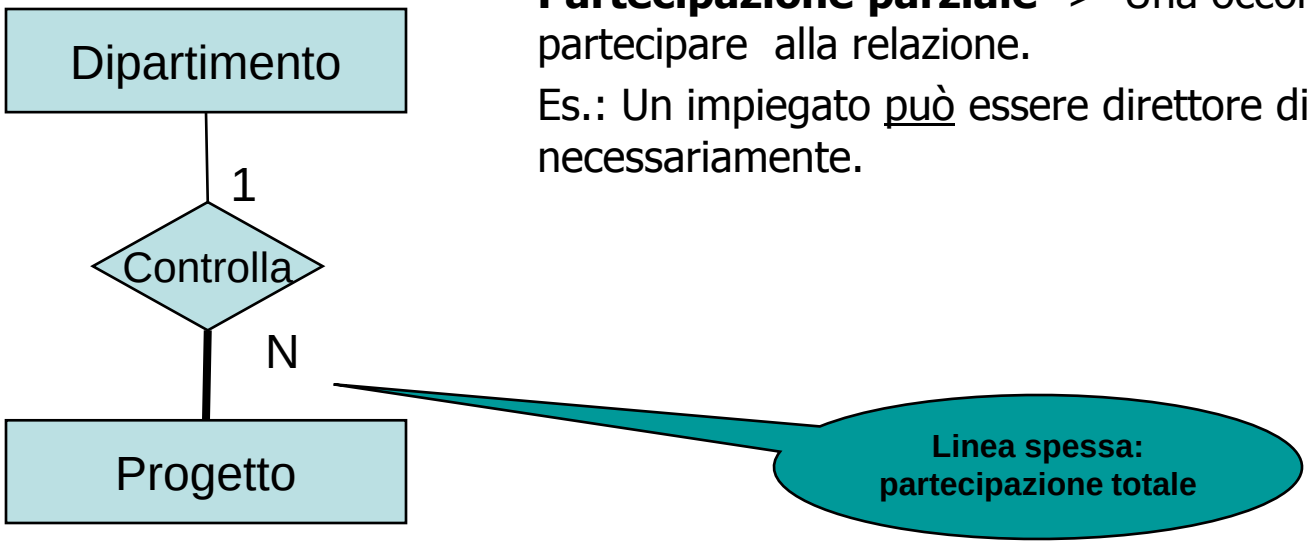
- La cardinalità definisce il massimo (minimo) numero di istanze della relazione cui partecipa una istanza dell'entità
- La **molteplicità** di un'associazione rappresenta il numero massimo di entità che partecipano all'associazione
- Per la cardinalità minima:
  - 0 significa "partecipazione opzionale"
  - 1 significa "partecipazione obbligatoria"
- Un'associazione **è totale** quando il legame tra le entità è sempre presente, cioè ad ogni di X deve corrispondere almeno un elemento di Y, altrimenti si dice **parziale**
- In base alla cardinalità viene quindi fatta una classificazione fondamentale delle associazioni:
  - Associazioni 1:1 (uno a uno) o biunivoca
  - Associazioni 1:N (uno a molti) o semplice
  - Associazioni N:N (molti a molti) o complessa

Esempi

- Si considerano due tipi di partecipazione: totale e parziale.
  - **Partecipazione totale** -> dipendenza esistenziale. Ogni occorrenza di entità partecipa alla relazione

    Es.: I requisiti dichiarano che un progetto (una occorrenza della entità Progetto) deve essere gestito da un dipartimento, altrimenti non ha senso che esista.
  - **Partecipazione parziale** -> Una occorrenza di entità può partecipare alla relazione.

    Es.: Un impiegato può essere direttore di un dipartimento, ma non necessariamente.

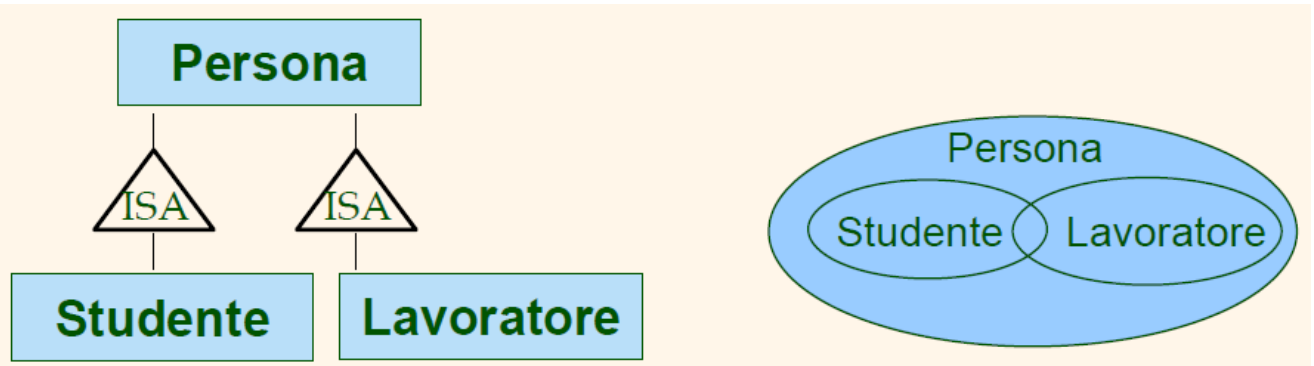


Entità deboli

- Una entità debole può essere indentificata univocamente solo considerando la chiave primaria di un'altra entità (proprietario)
- L'insieme di entità proprietarie e l'insieme di entità deboli devono partecipare in un insieme di relazioni uno-a-molti (1 proprietario, molte entità deboli) oppure uno-a-uno (1 proprietario, 1 entità debole)
- L'insieme di entità deboli deve avere partecipazione totale in questo insieme di relazioni identificanti

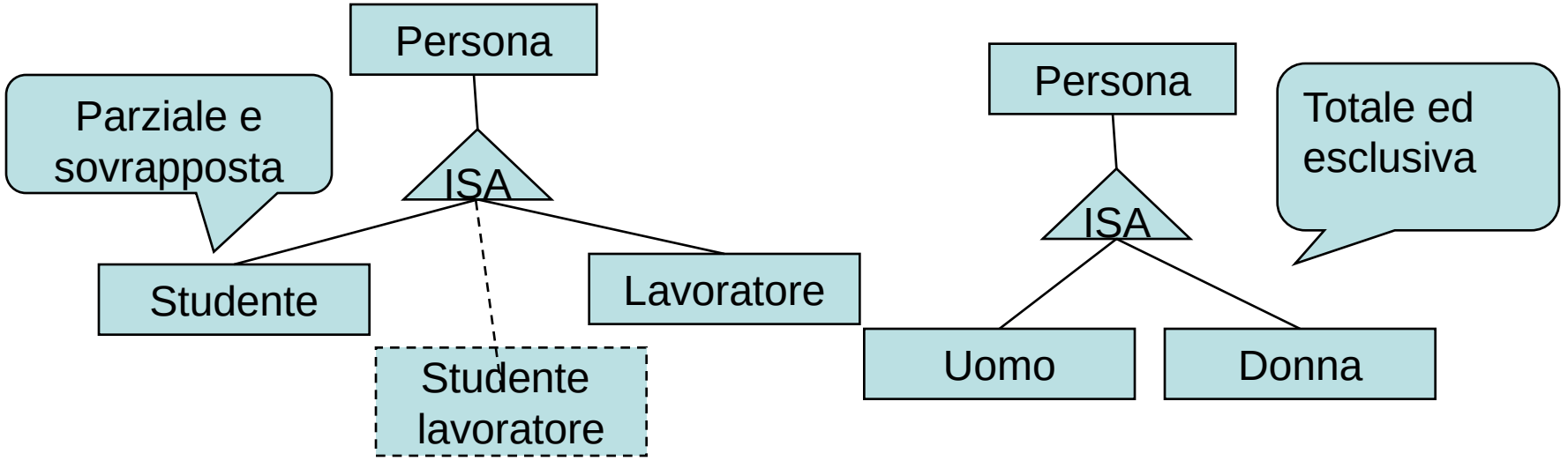
Relazioni ISA

- Se dichiariamo E1 ISA E2, ogni istanza dell'entità E1 è anche un'istanza dell'entità E2 → ogni proprietà dell'entità padre è anche una proprietà delle entità figlie
- La relazione ISA si rappresenta nel diagramma dello schema concettuale mediante una freccia dalla sottoentità alla entità padre
- Una entità può avere al massimo una entità padre. In altre parole, il modello ER **non ammette** ereditarietà multipla!!
- Una entità può avere diverse entità figlie
- Le istanze di due entità che sono figlie della stessa entità possono avere istanze in comune



Generalizzazione

- La relazione ISA stabilisce che l'entità padre è più generale della/e sottoentità
- Tipi di classificazione:
  - **totale**: se ogni occorrenza della classe padre è una occorrenza di almeno una delle entità figlie, altrimenti è **parziale**
  - **esclusiva**: se ogni occorrenza della classe padre è al più una occorrenza di una delle entità figlie, altrimenti è **sovrapposta**



Si progetti uno schema concettuale Entità-Relazioni indicando le cardinalità delle relazioni e un identificatore per ciascuna entità.

- Una base di dati deve essere utilizzata per *gestire le attività di uno scavo archeologico*. Durante la fase di scavo è necessario memorizzare i dati identificativi dei vari strati individuati (chiamati *unità stratigrafiche*).
- Bisogna, pertanto, tener traccia dei seguenti dati:
  - del *personale di scavo* caratterizzato da una matricola univoca, nome, cognome e telefono;
  - delle *unità stratigrafiche* individuate nello scavo caratterizzate dai seguenti dati: numero identificativo di unità stratigrafica, datazione, stato di conservazione, genere (positivo, negativo, rivestimento) ed il responsabile di riferimento (individuato tra il personale di scavo);

- ciascuna unità stratigrafica può essere classificata in una delle seguenti tipologie: *Rivestimento*, *Taglio* e *Deposizione*. Per il Rivestimento bisogna tener traccia anche della tecnica di rivestimento e della dimensione della superficie; per il Taglio anche della forma e dell'orientamento; mentre per la Deposizione bisogna tener traccia anche del tipo di tomba, del numero di ossa trovate e del Taglio a cui è associata. Si osservi che è necessario verificare che un Rivestimento sia un'unità stratigrafica di genere 'rivestimento', che un Taglio sia un'unità stratigrafica di genere 'negativo' e che Deposizione, invece, sia di genere 'positivo'.

Progetto di Laboratorio – I parte

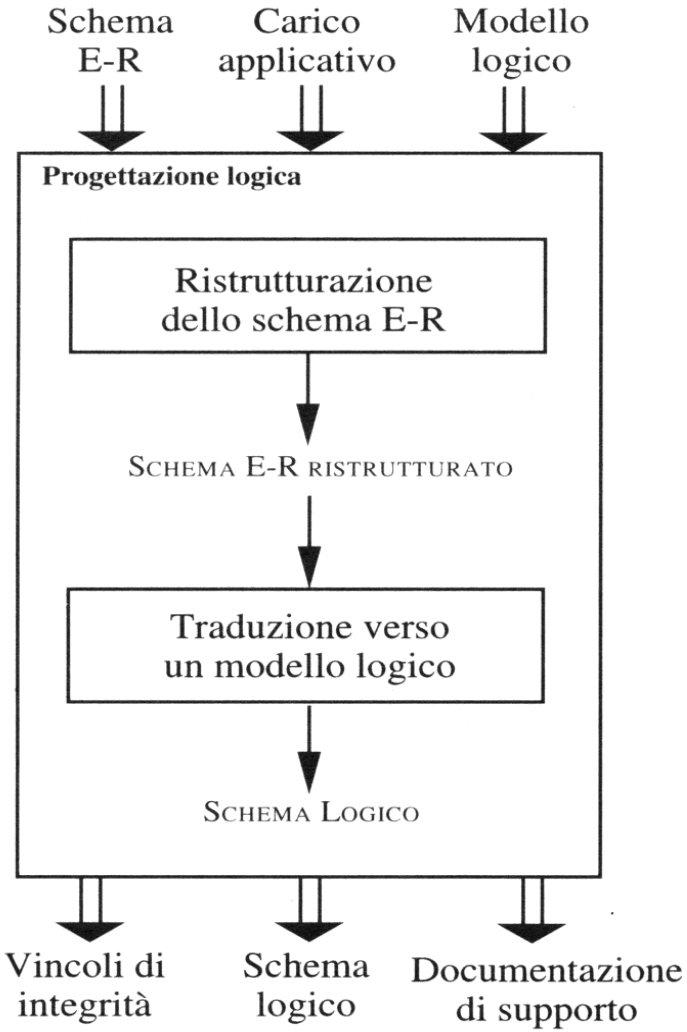
- Ciascun rivestimento ha associati degli *strati preparatori* caratterizzati da un numero, che dipende dal rivestimento a cui è legato, spessore minimo e spessore massimo. Bisogna tener traccia, inoltre, dei reperti associati ad un Rivestimento caratterizzati da un numero di inventario univoco, una descrizione, il colore dominante, il materiale principale di cui è composto e le foto associate (caratterizzate da codice univoco, nome e data di riferimento).
- Infine, per tutte le unità stratigrafiche bisogna tener traccia degli eventuali restauratori coinvolti (individuati sempre tra il personale di scavo).

Progettazione Logica ...

Progettazione Logica

- Obiettivo della **Progettazione Logica** e' quello di costruire uno schema logico ,in un determinato *modello* (ad es. *relazionale*), che descriva in maniera *corretta* ed *efficiente* tutte le informazioni contenute nello schema E-R prodotto dalla progettazione concettuale.
- Non si tratta di una semplice *traduzione*
- La progettazione logica si articola in due fasi:
  - **Ristrutturazione dello schema E-R:** e' una fase indipendente dal modello logico e si basa su criteri di *ottimizzazione* dello schema e di successiva *semplificazione.*
  - **Traduzione verso il Modello Logico:** fa riferimento ad un modello logico (ad es. relazionale) e puo' includere ulteriore *ottimizzazione* che si basa sul modello logico stesso (es. normalizzazione).

Progettazione Logica



Per determinare il **carico applicativo** vengono definiti degli **indici di prestazione** per la valutazione di schemi E-R. Tali indici sono due:

- **Costo di un'operazione**: in termini di numero di occorrenze di entita' ed associazioni che mediamente vanno visitate per rispondere a quella operazione sulla base di dati
- **Occupazione di memoria**: viene valutata in termini dello spazio di memoria (misurato in byte) necessario per memorizzare i dati del sistema.

Ristrutturazione

- Una volta determinato il carico applicativo, ed avendo dalla progettazione concettuale lo schema E-R, tutti gli input della ristrutturazione sono noti.
- Si passa quindi alla ristrutturazione vera e propria, per determinare lo schema ristrutturato (output)
- La ristrutturazione consta di quattro fasi:
  - **Analisi delle Ridondanze**: si decide se eliminare o no eventuali ridondanze.
  - **Eliminazione delle Generalizzazioni**: tutte le generalizzazioni vengono analizzate e sostituite da altro.
  - **Partizionamento/Accorpamento di entità ed associazioni**: si decide se partizionare concetti in piu' parti o viceversa accorpare.
  - **Scelta degli identificatori primari**: si sceglie un identificatore per quelle entità che ne hanno più di uno

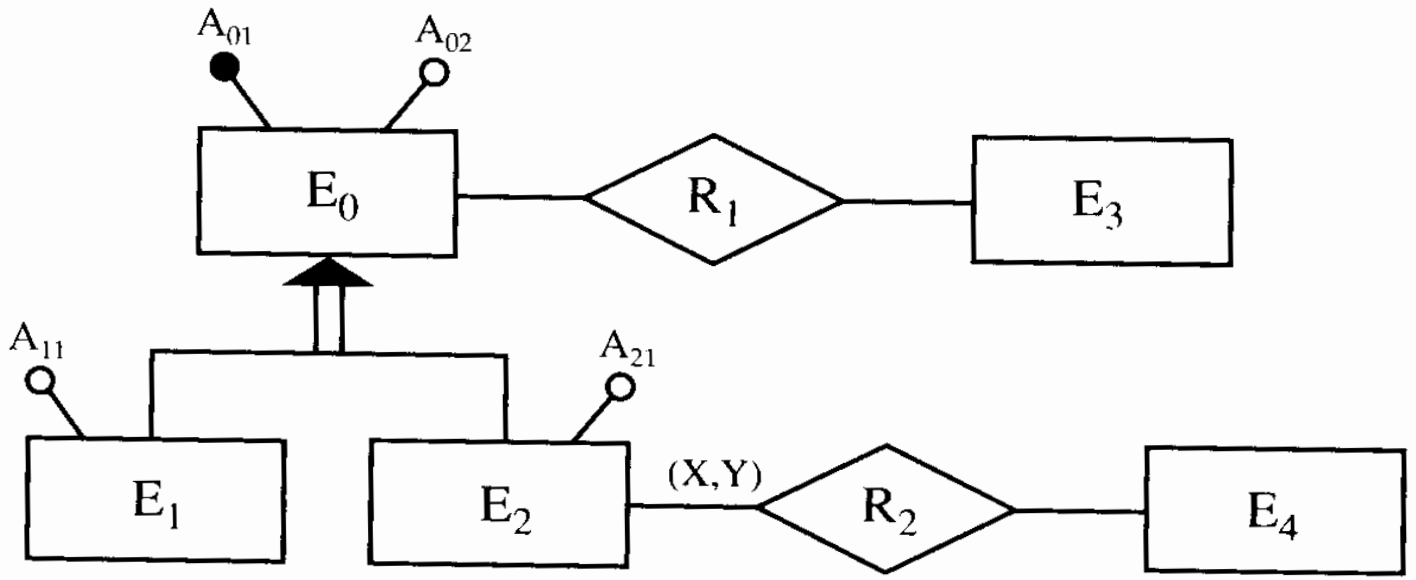
Analisi delle ridondanze

Una **Ridondanza** in uno schema E-R è la presenza di un attributo che può essere ricavato da altri. In particolare, (vedi esempi nelle tabelle seguenti):

- *Attributi derivabili da altri attributi della stessa entità* (importo lordo nella tabella fattura)
- *Attributi derivabili da attributi di altre entità* (o associazioni) (Acquisto: Importo totale da Prezzo)
- *Attributi derivabili da operazioni di conteggio* (Città: Numero abitanti contando il numero di Residenza)
- *Associazioni derivabili dalla composizione di altre associazioni in presenza di cicli* (Docenza da Frequenza ed Insegnamento). Tuttavia i cicli non necessariamente generano ridondanze (Tesi al posto di Docenza).

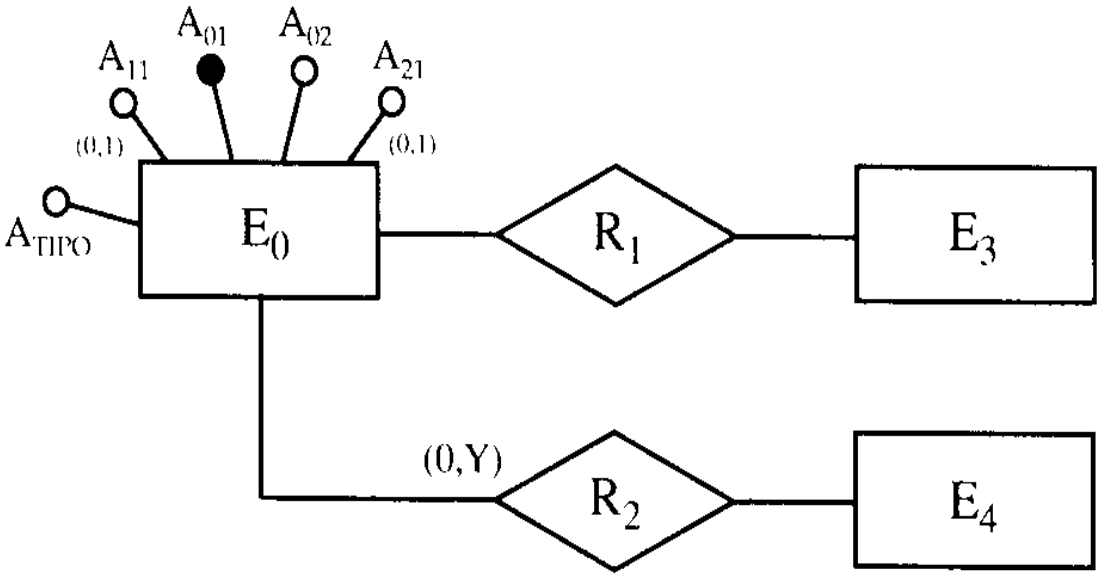
Eliminazione delle generalizzazioni

- Il modello relazionale **non permette di rappresentare gerarchie** quindi conviene trasformarle in altri concetti.



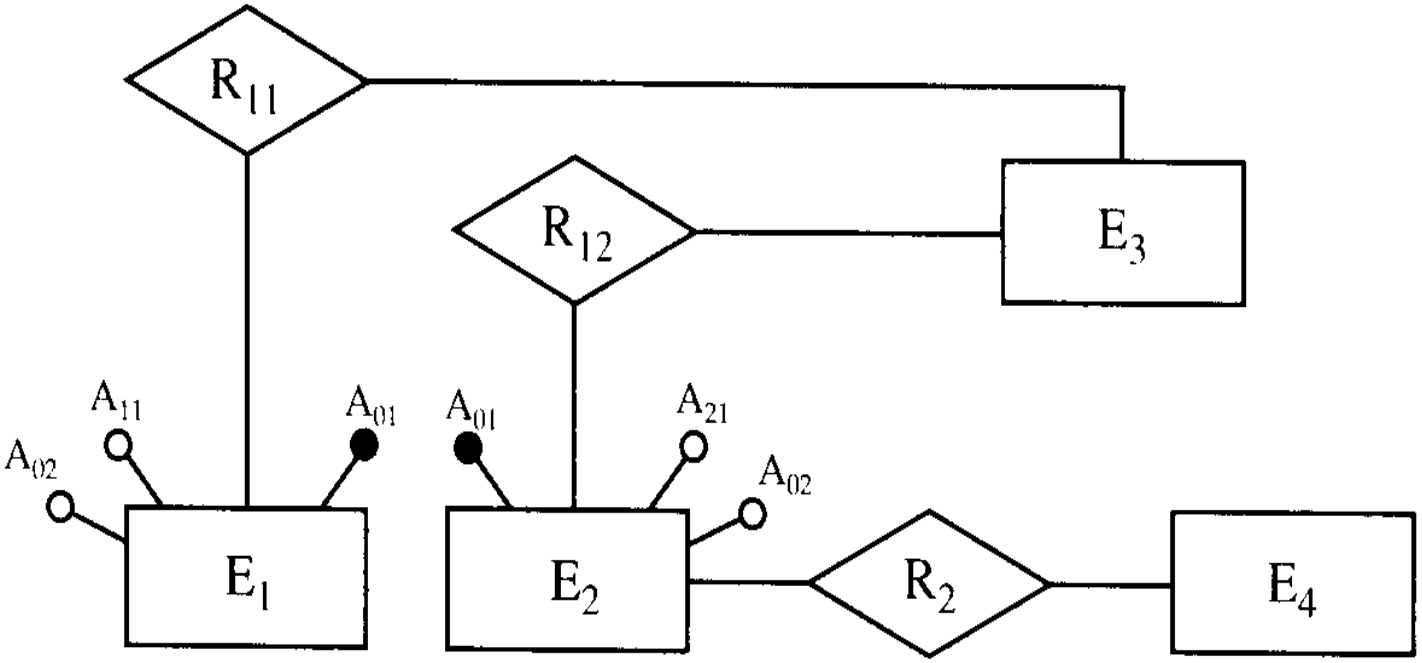
Eliminazione delle generalizzazioni - Prima soluzione

La **prima soluzione** prevede l'accorpamento dei figli nel padre: le entità figlie vengono eliminate e le loro proprietà (attributi, associazioni, generalizzazioni) vengono aggiunte al padre, che deve avere in più un attributo (TIPO) per distinguere le entità figlie.



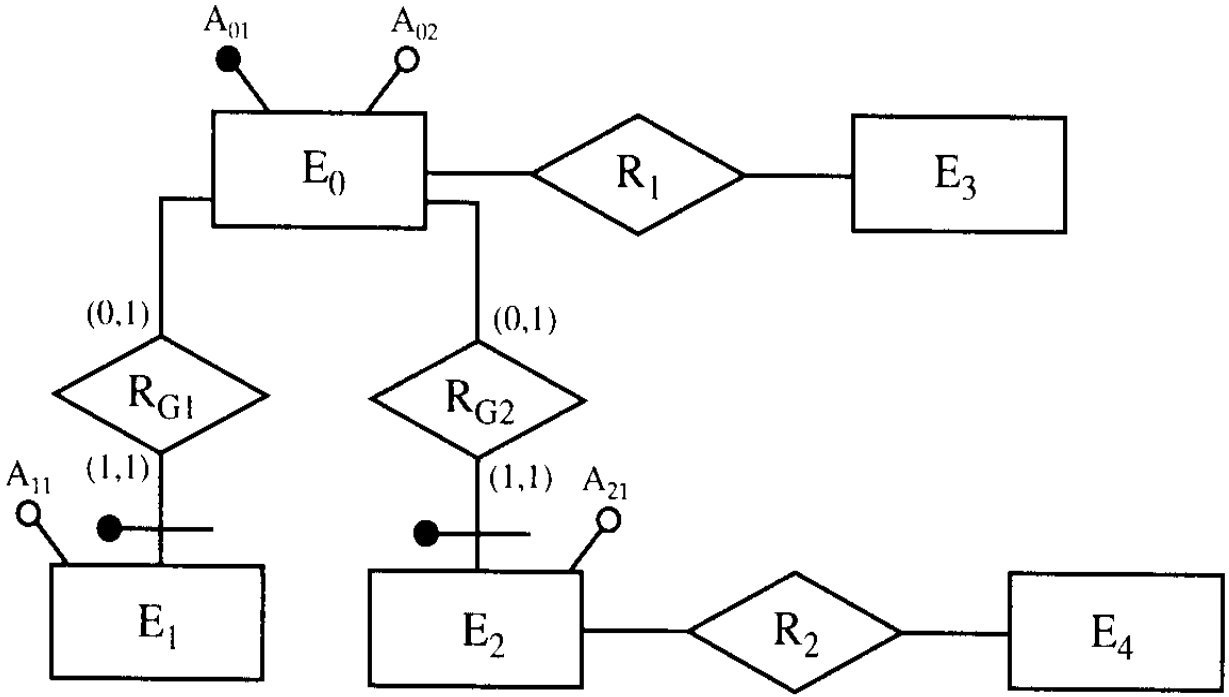
Eliminazione delle generalizzazioni - Seconda soluzione

La **seconda soluzione** prevede l'accorpamento del padre nei figli: l'entità padre viene eliminata e per ereditarietà i suoi attributi, identificatori e relazioni vanno ai figli.



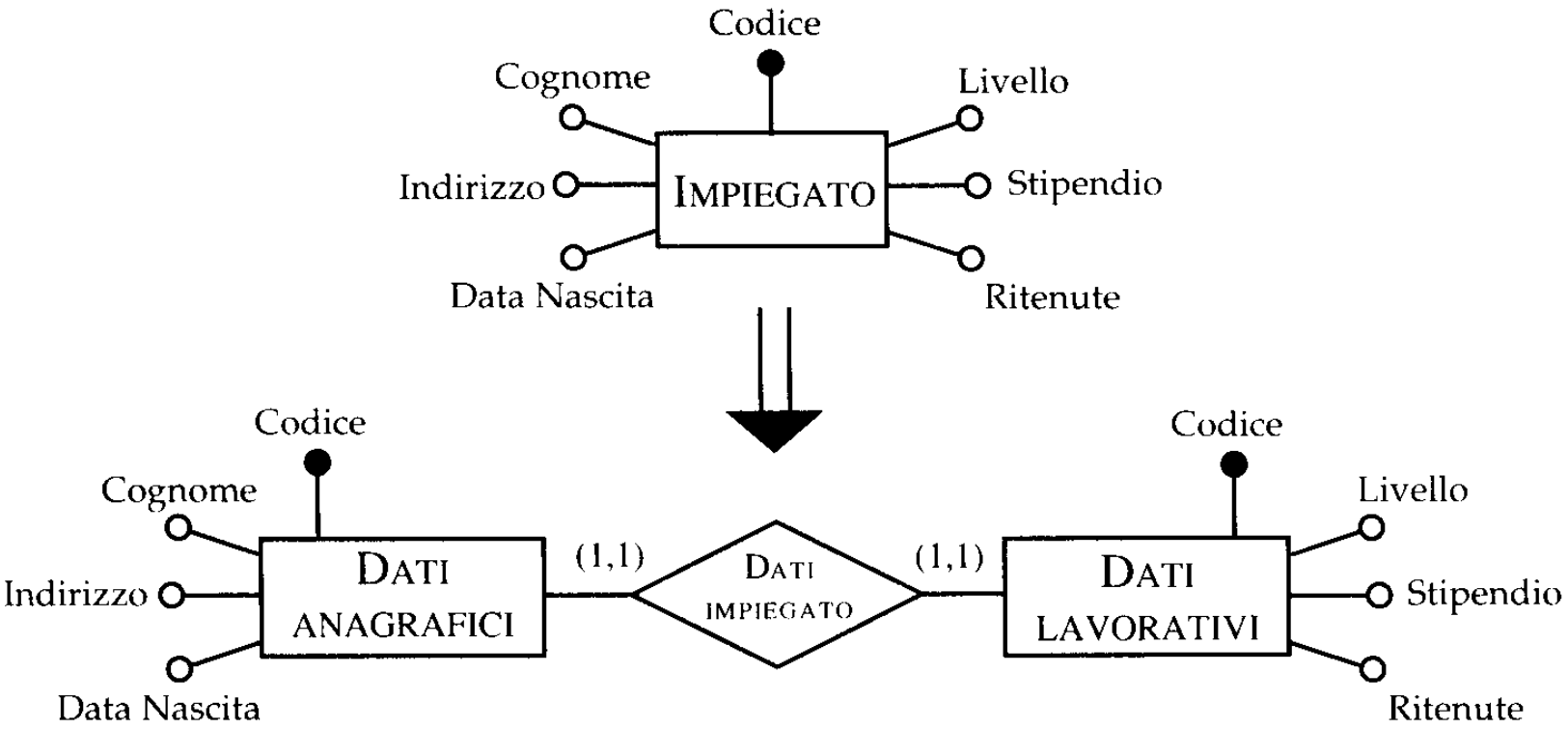
Eliminazione delle generalizzazioni - Terza soluzione

La **terza soluzione** prevede la sostituzione: la generalizzazione si trasforma in associazioni uno ad uno che legano padre e figli



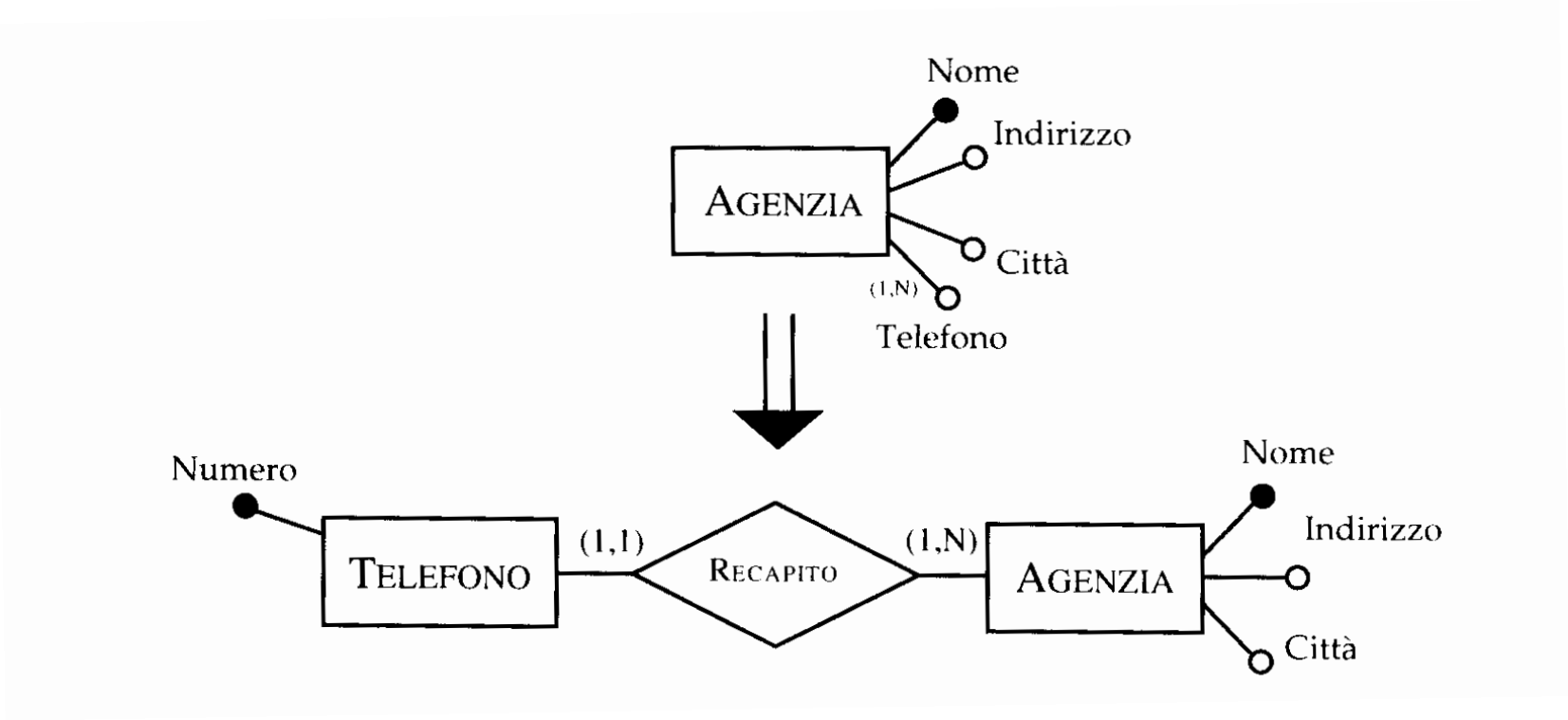
Partizionamento/accorpamento

Esempio di partizionamento "verticale":

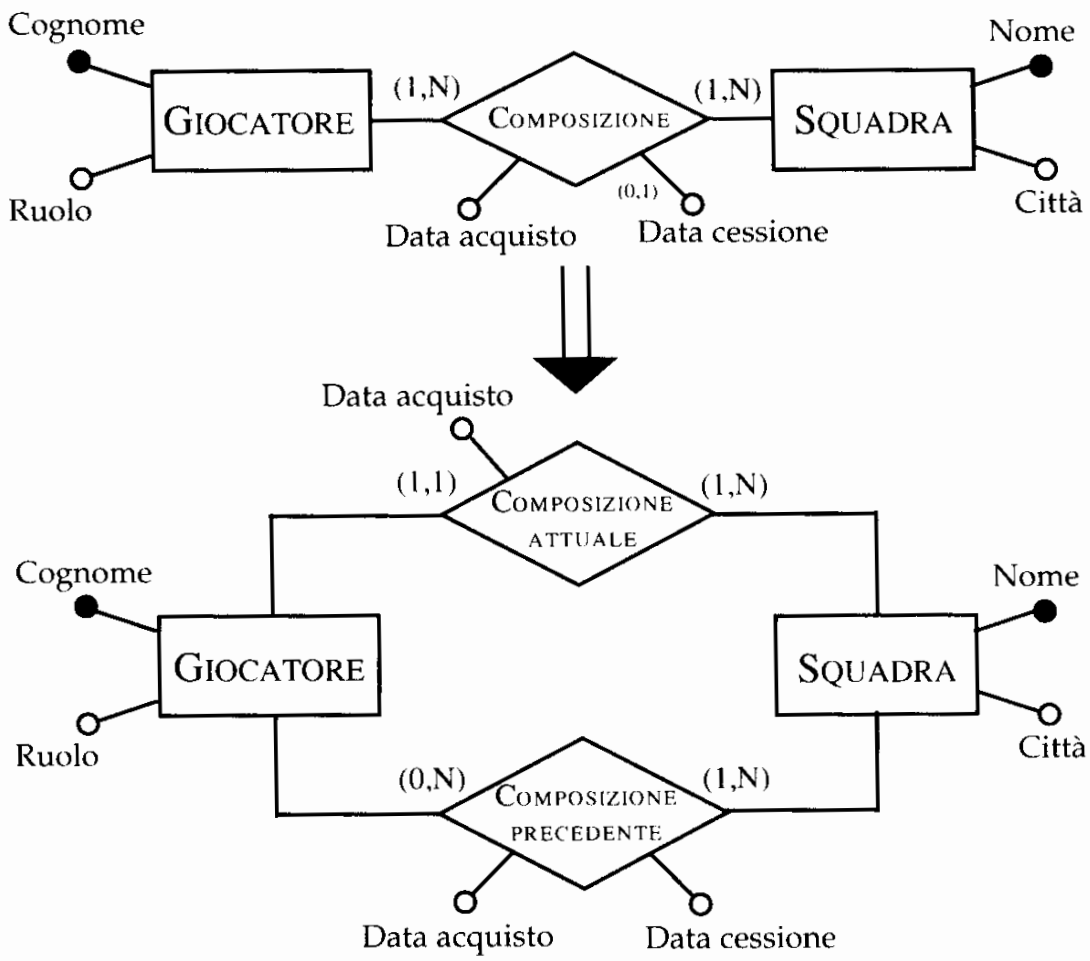


Attributi Multivalore

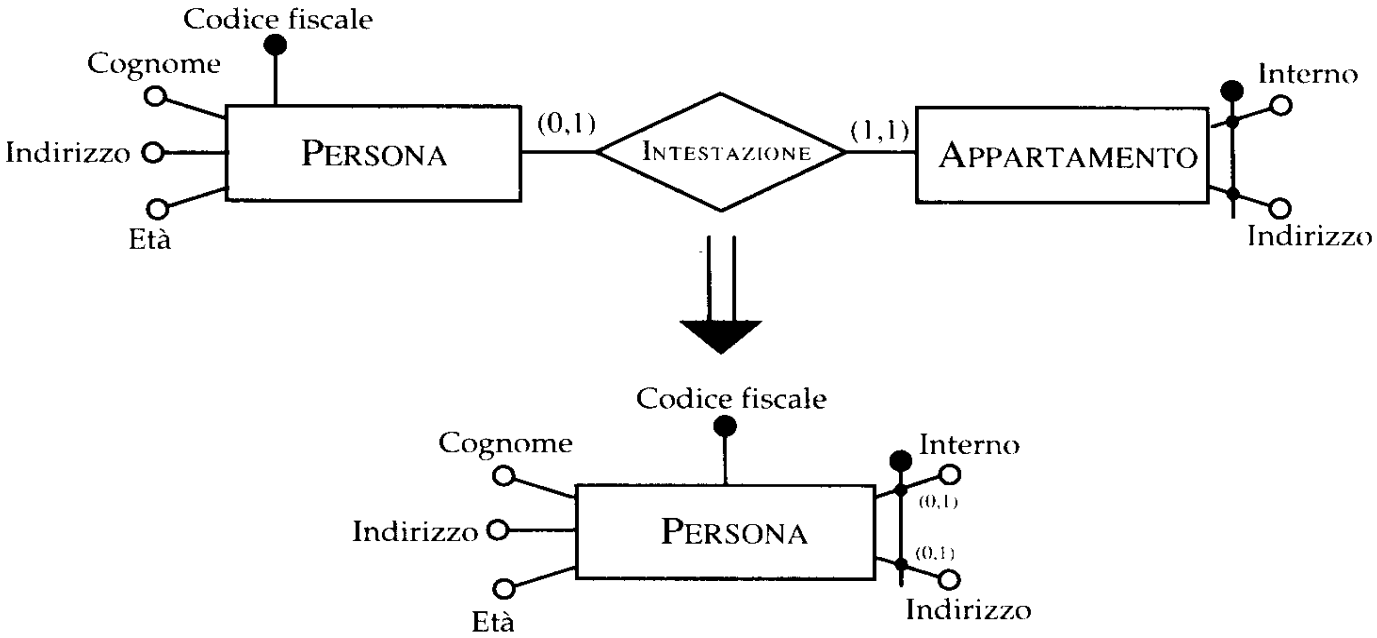
- Un caso particolare di partizionamento è quello necessario per l'eliminazione di attributi multivalore, generalmente non supportati dal modello relazionale



Esempio di partizionamento di associazione



Esempio di accorpamento



L'accorpamento e' giustificato se le operazioni più frequenti su Persona richiedono sempre i dati relativi all'appartamento e quindi vogliamo risparmiare gli accessi alla relazione che li lega. Normalmente gli accorpamenti si fanno su relazioni uno ad uno, raramente su uno a molti mai su molti a molti.

Associazione molti a molti

- Nel caso di **associazione molti a molti**, la traduzione segue le regole:
  - Ogni **entita'** coinvolta si traduce in una relazione (tabella) con lo stesso nome dell'entità, avente i suoi stessi attributi e per chiave il suo identificatore.
  - La **relazione** si traduce in una relazione (tabella) con lo stesso nome avente gli stessi attributi e come *chiave* l'insieme di tutti gli *identificatori* delle entita' coinvolte.

- Associazione molti a molti *ricorsiva*? Es. composizione di un prodotto con la relativa quantità

Associazione uno a molti

- Nel caso di **associazione uno a molti**, la traduzione segue le regole:
  - L'**entita'** partecipante con cardinalità (1,1) si traduce in una relazione (tabella) con lo stesso nome dell'entità, avente i suoi stessi attributi, per *chiave* il suo *identificatore* e che ingloba anche la chiave della entità che partecipa con cardinalità (1,n) (tale attributo diventa quindi una chiave esterna), più tutti gli eventuali attributi della associazione (che non avrà quindi una tabella distinta come nel caso m-n).
  - L'**entita'** partecipante con cardinalità (1,n) si traduce in una relazione (tabella) con lo stesso nome dell'entità, avente i suoi stessi attributi, per *chiave* il suo *identificatore*
- Entità con *identificatore esterno* - la regola ed il risultato sono ancora gli stessi? Es. matricola-università

Associazione uno a uno

- Nel caso di **associazione uno a uno con partecipazione obbligatoria per entrambe le entità (non (0,1) ma (1,1))**, la traduzione segue le regole:
  - Una delle **entita'** si traduce in una relazione (tabella) con lo stesso nome dell'entità, avente i suoi stessi attributi, per *chiave* il suo *identificatore* e che ingloba anche la chiave della altra entità (tale attributo diventa quindi una chiave esterna), più tutti gli eventuali attributi della associazione (come nel caso 1-n, l'associazione non avrà una tabella distinta).
  - L'**entita'** restante si traduce in una relazione (tabella) con lo stesso nome dell'entità, avente i suoi stessi attributi, per *chiave* il suo *identificatore*

Associazione uno a uno: altri casi

Qualora una delle entità partecipi alla associazione con cardinalità (0,1) ossia è **un'entità con partecipazione opzionale**, delle due soluzioni si preferisce quella in cui la associazione viene inglobata dall'altra entità (quella che partecipa con cardinalità (1,1)), questo per evitare i valori nulli che figurerebbero negli attributi relativi all'associazione qualora venissero inglobati nell'entità che partecipa con cardinalità (0,1):

Impiegato(Codice,Cognome,Stipendio)

Dipartimento(Nome,Telefono,Sede, Direttore,InizioDirezione)

Qualora tutte le entità partecipano alla associazione con cardinalità (0,1), sono possibili due soluzioni:

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

oppure

Direttore(Codice,Cognome,Stipendio)

Dipartimento(Nome,Telefono, Sede, Direttore, InizioDirezione)

Esiste in teoria l'opzione di fondere tutto in un'unica tabella; tale opzione viene però scartata perché se in sede di progettazione concettuale era stato deciso di avere due entità legate da una relazione, diventa inopportuno tornare indietro in sede di progettazione logica.

Vincoli del modello

- Vincolo di dominio: i valori di ciascun attributo sono valori **atomici** appartenenti al corrispondente dominio.
- Vincolo di chiave: per ogni relazione *R* esiste un sottoinsieme di attributi *sk* tale che t1[sk]≠t2[sk].
  - Il sottoinsieme *sk* dicesi **superchiave**.
  - Dicesi **chiave** un sottoinsieme di attributi che goda della ulteriore proprietà di essere minimo, cioè privo di attributi ridondanti.

    Es.: studente(nome, cognome, matricola, data_nascita)

    Il sottoinsieme *nome, cognome, matricola* è superchiave, come pure *cognome, matricola.*

    Matricola è chiave (non ridondante)
  - In genere possono esserci più sottoinsiemi di attributi che verificano le due proprietà: **chiavi candidate**
  - La chiave designata è detta **chiave primaria**.

Vincoli del modello (2)

- Integrità di entità: nessuna chiave primaria può assumere valore null.
- Integrità referenziale: una tupla in una relazione R1 che si riferisca ad un'altra relazione R2 deve riferirsi ad una tupla esistente in R2.
- Il precedente vincolo è normalmente espresso mediante il concetto di chiave esterna (foreign key).
- Un sottoinsieme di attributi FK di una relazione R1 è una chiave esterna se:
  1. Gli attributi di FK hanno lo stesso dominio degli attributi della chiave primaria PK di un'altra relazione R2
  2. Un valore di FK in una tupla t1 di R1 o è presente identicamente come valore di PK di qualche tupla t2 in R2 o è nullo. Nel primo caso si avrà t1[FK]=t2[PK] e si dice che t1 referenzia t2.

Normalizzazione dello schema

- La presenza di dati ridondanti comporta:
  - spreco di memoria
  - probabile introduzione di inconsistenze e/o perdite di informazioni a causa di operazioni di manipolazione della base di dati
- Le ridondanze possono essere eliminate mediante scomposizione degli schemi di relazione
- Quando una relazione non è normalizzata presenta anomalie di *inserimento*, *cancellazione* e *modifica*
- Lo scopo della normalizzazione è di progettare dei "buoni" schemi di basi di dati

Info_Fornitore

Nome_Forn	Indir_Forn	Nome_Prod	Prezzo
Rossi	Via Marte 3	Divano	$ 1,000
Rossi	Via Marte 3	Scarpe	$ 2,000
Verdi	Via Venere 5	Perrier	$ 5
Bianchi	Via Nettuno 23	Tavolo	$ 1,500
Bianchi	Via Nettuno 23	Perrier	$ 4

Esempi di Anomalie

- L'indirizzo del fornitore è ripetuto per ogni prodotto fornito dal fornitore stesso (anomalia di aggiornamento)
  - Per modificare l'indirizzo di un fornitore è necessario essere sicuri di modificarlo in tutte le tuple in cui appare il fornitore altrimenti si hanno delle inconsistenze
- La chiave della relazione è formata dagli attributi "nome fornitore" e "nome prodotto" (anomalia di inserimento)
  - Non possiamo registrare le informazioni di un fornitore se questo non fornisce almeno un prodotto
- Se cancellassimo tutti i prodotti di un fornitore, perderemmo traccia del suo indirizzo (anomalia di cancellazione)
- I problemi visti non si verificano se la relazione Info_Fornitore viene "spezzata" in due relazioni
  - Fornitore(Nome_Forn,Indir_Forn)
  - Prodotto(Nome_Prod,Nome_Forn,prezzo)

Progetto di Laboratorio – II parte

- Partendo dal modello E-R precedentemente definito si definiscano le relazioni (tabelle) risultanti avendo cura di esplicitare i vincoli di integrità:
  - Chiave primaria
  - Chiave esterna

Progettazione Fisica ...

La progettazione fisica produce in uscita lo schema fisico della base di dati, costituito dalle effettive definizioni delle relazioni e soprattutto delle strutture fisiche utilizzate.

L'attività di progettazione fisica può essere molto complessa, perché oltre alle scelte relative alle strutture fisiche (organizzazione dei files e degli indici), può essere necessario definire molti parametri, come dimensioni iniziali dei file, possibilità di espansione e contiguità di allocazione.

La specifica dei parametri fisici di memorizzazione dei dati dipende dallo specifico sistema di gestione (DBMS) scelto.

La progettazione fisica è ricondotta principalmente all'attività di individuazione degli indici da definire su ciascuna relazione (a parte la specifica degli schemi delle relazioni in DDL)

Cosa sono gli indici?

L'indice è una struttura ausiliaria per l'accesso efficiente ai dati

E' conveniente quindi utilizzare degli indici su una colonna di una tabella quando prevediamo che si faranno molte query basate su quella colonna, o se quella colonna sarà coinvolta in molte operazioni di join.

In generale un indice è un elenco di parole chiave accompagnate dalla posizione in cui quella parola chiave appare in un file.
Si pensi all'indice analitico di un libro. Le parole chiave sono elencate in ordine alfabetico, e a fianco c'è l'elenco delle pagine in cui la parola chiave appare nel libro. La ricerca alfabetica è relativamente facile (o poco complessa) e una volta trovata la parola chiave nell'indice, è facile trovare le pagine di riferimento.

Il grande vantaggio degli indici sta quindi nella rapidità delle ricerche.
Tuttavia non conviene definire un indice su ogni colonna. Infatti un indice occupa spazio di memoria, per cui non è opportuno utilizzarli, a meno che la colonna considerata non sia coinvolta spesso in interrogazioni.

INDICE – Primario e secondario

Indice primario: è quello che viene normalmente definito di tipo **`unique`** sulla chiave primaria (**`primary key`**) di ciascuna tabella. Si osservi che chiave ed indice primario sono concetti differenti, il primo fa riferimento ad una proprietà astratta dello schema ed il secondo ad una proprietà della implementazione fisica della base di dati.

Indici secondari: sono quelli che possono essere ti tipo **`unique`** e **`multiple`**; nel secondo caso ad ogni valore di una chiave dell'indice possono corrispondere varie tuple.

Le operazioni più delicate in una base di dati relazionale sono quelle di **selezione** (che corrisponde all'accesso ad uno o più record sulla base dei valori di uno o più attributi) e di **join** (che richiede di combinare tuple di relazioni diverse sulla base dei valori di uno o più attributi di ognuna di tali relazioni). Ciascuna delle due operazioni può essere eseguita in modo molto più efficiente se sui campi interessati è definito un indice, che permette un accesso diretto.

Possono essere definiti ulteriori indici su altri campi su cui vengono effettuate operazioni di selezione oppure su cui è richiesto un ordinamento in uscita (perché un indice ordina logicamente i record di un file, rendendo nullo il costo di un ordinamento).

È buona norma, dopo l'aggiunta di un indice, verificare che le interrogazioni ne facciano uso (in genere, esiste un comando **`show plan`** che descrive la strategia di accesso scelta dal DBMS).

INDICE – SINTASSI

Il comando per la creazione di un indice è messo a disposizione da ciascun DBMS ma non fa parte dello standard SQL-2.

CREATE [unique] INDEX *NomeIndice* **on** *NomeTabella (Lista attributi)*

L'ordine in cui compaiono gli attributi nella lista è significativo: le chiavi dell'indice vengono ordinate prima in base ai valori del primo attributo della lista, poi a parità di valore del primo attributo si usano i valori del secondo attributo, e così in sequenza fino all'ultimo attributo.

L'uso della parola `unique` specifica che nella tabella non sono ammesse tuple che abbiano lo stesso valore relativamente agli attributi dell'indice (in termini relazionali si tratta di una "chiave candidata").

Il comando **DROP INDEX** *NomeIndice* elimina l'indice specificato.

In generale non è semplice stabilire quali indici creare su una base di dati per migliorare le prestazioni complessive delle applicazioni poiché se da un lato gli indici accelerano le operazioni di ricerca, dall'altro occupano memoria e rallentano le operazioni di aggiornamento.

Scelta degli indici

- Occorre analizzare il carico e individuare gli attributi coinvolti in operazioni di join e di selezioni
- Si predispongono indici sugli attributi di join, in modo da rendere possibili piani esecutivi vantaggiosi
- Gli attributi chiave sono spesso coinvolti in join e selezioni, e pertanto su di essi tipicamente vengono predisposti indici.
- Considerare anche gli aspetti negativi:
  - L'occupazione di un indice può essere dello stesso ordine, o addirittura superare quella della tabella
  - Gli aggiornamenti comportano operazioni costose sugli indici, oltre che sulla tabella

Progettazione fisica: sintesi

- Occorre stabilire:
  - Per ciascuna tabella come e dove viene memorizzata
  - Dimensioni iniziali del file, vincoli di allocazione etc
  - Su quali attributi definire indici e di che tipo
  - Parametri relativi alla gestione del buffer, del controllo della concorrenza e dell'affidabilità
- **N.B. *Il DBMS offre comunque accettabili valori di default***
- A seguito dell'analisi dei risultai/prestazioni ottenute è possibile adottare varie strategie (tuning):
- Aggiunta o ridefinizione di indici, per rendere possibili diversi piani esecutivi e/o metodi di join
- Ridefinizione dei parametri relativi al buffer, allo scopo di migliorare il rapporto tra I/O logico e I/O fisico;
- Eventuali ristrutturazioni dello schema logico;
- **N.B. Spesso la soluzione è di *forzare l'ottimizzatore***

PostgreSQL

Introduzione

- PostgreSQL è un ORDBMS (Object Relational Database Management System)
  - Conservata l'organizzazione relazionale dei dati
  - Gestione dell'ereditarietà semplice e multipla
  - Nuovi tipi di dati
- E' un prodotto open-source mantenuto da un team mondiale di sviluppatori ed esperti
- Nasce nel 1986 come The Berkeley POSTGRES Project
- Nel 1994 Andrew Yu and Jolly Chen integrano un interprete SQL al progetto che viene rilasciato col nome Postgres95
- Postgres95 viene rilasciato come progetto open source
- Il motore di Postgres95 viene scritto interamente in ANSI C
- Dal 1996 viene adottato il nome PostgreSQL
- Allo stato attuale vengono supportate le versioni 8.4, 8.3, 8.2, 8.1 e 8.0

Caratteristiche principali

- Modello client-server: i dati sono gestiti in modo centralizzato (server) e messi a disposizione di più fruitori (client)
- Avanzato supporto per lo standard SQL2003: garantisce la portabilità di applicazioni
- Point in time recovery : capacita di recuperare la base dati sottoposta a backup specificando il momento esatto che si vuole ripristinare con estrema precisione (es. prima di eventuali corruzioni o cancellazioni accidentali dei dati).
- Query planner: le query sono analizzate per individuare il miglior percorso di accesso ai dati
- Tablespaces: è un raggruppamento logico interno al database cui corrisponde una locazione fisica (permettono di posizionare gli oggetti, come indici e tabelle, in aree specifiche del filesystem).
- Partizionamento: possibilità di separare i dati e organizzarli secondo le esigenze fisiche di storage.
- Linguaggi integrati: PL/Perl, PL/Java, PL/PGSQL, ...

Numeri limite

- PostgreSQL fa della gestione di grandi volumi di dati il suo punto di forza. Di seguito l'elenco dei numeri limite relativi alla versione 8.4.
  - Dimensione massima per un database illimitata
  - Dimensione massima per una tabella 32 TB
  - Dimensione massima per una riga 1.6 TB
  - Dimensione massima per un campo 1 GB
  - Numero massimo di righe per tabella illimitate
  - Numero massimo di colonne per tabella da 250 a 1600
  - Numero massimo di tabelle illimitate
  - Numero massimo di indici per tabella illimitati

Installazione

- Il demone PostgreSQL gestisce un cluster di database, ovvero un insieme di più database possono risiedere sullo stesso host e possono essere gestiti dallo stesso processo.
- Appena installato il cluster mette a disposizione un database vuoto, usato come template per la creazione di altri database: template1.
- Tutte le caratteristiche di template1 possono essere modificate dall'utente, e saranno riflesse in ogni nuovo database creato.
- Il database template0 rappresenta una copia di sicurezza di template1.

Il file postgresql.conf

- Il file postgresql.conf contiene parametri fondamentali per la configurazione del server. Alcuni interessanti sono:
  - listen_address = 'localhost' {*, ip address}
  - specifica per quali interfacce il server deve accettare connessioni
  - log_statement = 'none' {none, all, ddl, mod}
  - consente di abilitare il logging dei comandi SQL eseguiti dal server, utile per il debugging o il monitoring dell'attività del server
  - shared_buffers = 24MB {almeno 16k a connessione}
  - indica la memoria disponibile per PostgreSQL in cui conservare le pagine dati
  - work_mem = 1MB
  - è la memoria usata per il sorting (clausole ORDER BY)

postgresql.conf: ottimizzazione

 # Definiscono quali metodi l'ottimizzatore può usare per l'accesso ai dati
 # (ad esempio si potrebbe voler impedire di usare il seqscan...)

- **enable_bitmapscan = on**
- **enable_hashagg = on**
- **enable_hashjoin = on**
- **enable_indexscan = on**
- **enable_mergejoin = on**
- **enable_nestloop = on**
- **enable_seqscan = on**
- **enable_sort = on**
- **enable_tidscan = on**

 # Costi (relativi) di accesso alle operazioni. Sostanzialmente sono
 # espressi in una scala arbitraria, si noti che il costo della pagina
 # è superiore a quello di un'operazione di CPU (es. sorting)

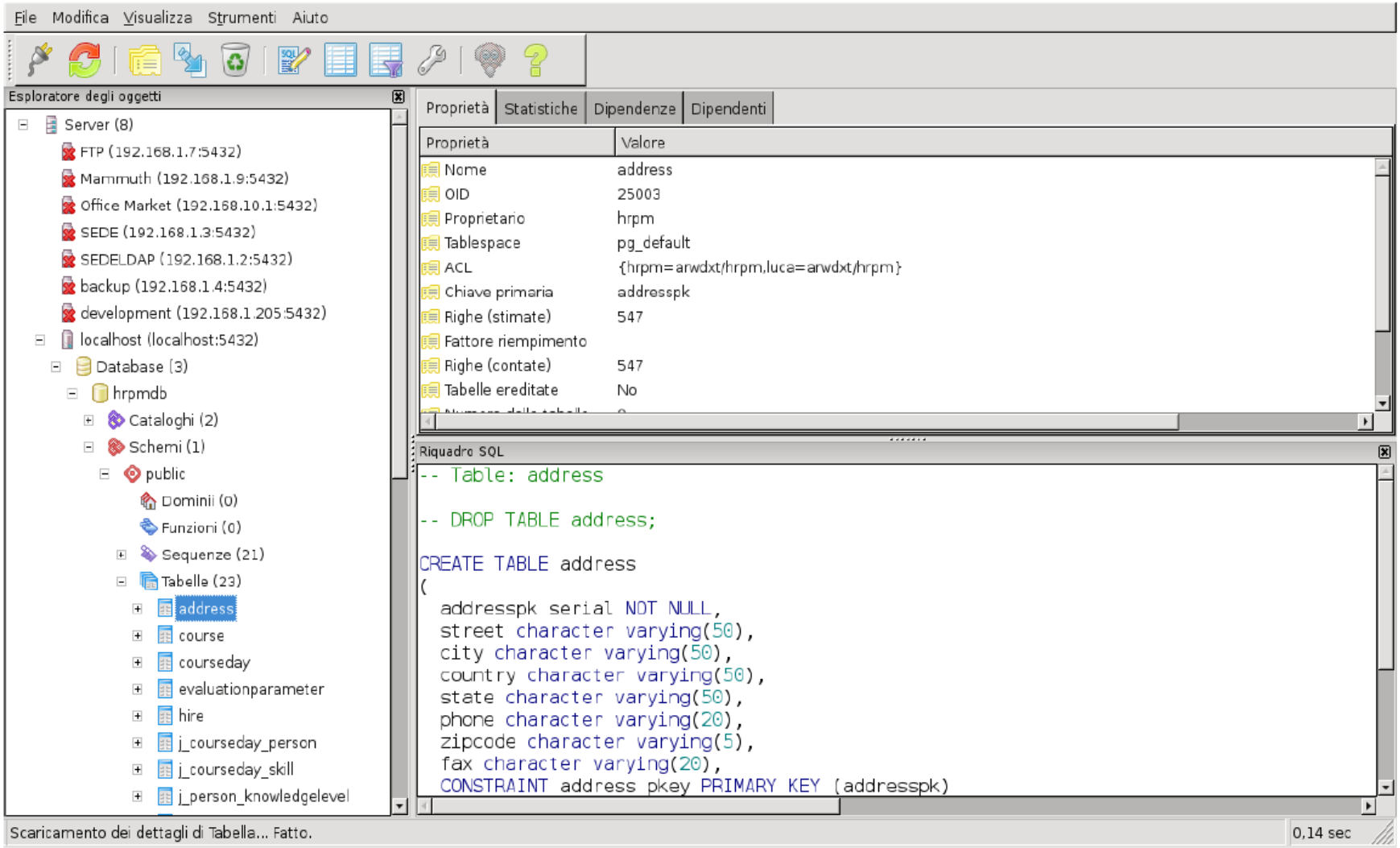
- **seq_page_cost = 1.0 # measured on an arbitrary scale**
- **random_page_cost = 4.0 # same scale as above**
- **cpu_tuple_cost = 0.01 # same scale as above**
- **cpu_index_tuple_cost = 0.005 # same scale as above**

Il file pg_hba.conf

- Il file pg_hba.conf (host base access) contiene le regole per l'accesso al server PostgreSQL da parte dei client della rete. Occorre specificare il database, la maschera di rete dei client (o l'indirizzo ip) e il metodo di accesso (trust, md5,...):

# TYPE	*DATABASE*	*USER*	*CIDRADDRESS*	*METHOD*
local	**all**	**all**		**trust**
host	**all**	**all**	**127.0.0.1/32**	**md5**
host	**linuxdb**	**linux**	**192.168.1.0/24**	**md5**
host	**all**	**all**	**::1/128**	**md5**

Connessione al database: PgAdmin III



Catalogo di sistema

- Il catalogo di sistema e' costituito da un insieme di tabelle comuni a tutti database del cluster e dal dizionario dati
- Il dizionario dati si occupa del parsing dei comandi SQL (parser stage) e viene adoperato dal planner per la risoluzione del piano di esecuzione
- L'owner delle tabelle e' l'utente postgres che e' anche il super user del cluster database
- Gli utenti non privilegiati possono visualizzarne i dati attraverso delle viste di sistema
- Le viste di sistema vengono utilizzate per ottenere informazioni sul funzionamento del cluster.

Le viste di sistema

- **pg_user** vista di sistema che opera sulla tabella pg_shadow fornisce informazioni sugli utenti
- **pg_tables** tabella di sistema, fornisce informazioni sulle tabelle del cluster
- **pg_settings** tabella di sistema, fornisce informazioni sui parametri runtime dell'istanza
- **pg_database** tabella di sistema, fornisce informazioni sui database presenti nel cluster
- **pg_statistic** tabella di sistema, contiene le statistiche degli oggetti del cluster. Viene aggiornata durante le operazioni di analyze degli oggetti del cluster → L'indice valuta questi valori, il loro istogramma e decide se vale la pena usare l'indice per la clausola di selezione indicata.

MVCC e Vacuum

Approccio Multiversion Concurrency Control (MVCC)- evita l'uso di lock espliciti su gli oggetti della query e garantisce prestazioni ottimali in ambienti multiutente → ogni transazione vede sempre una fotografia dei dati come erano al momento del lancio della query indipendentemente dallo stato dei dati su disco.

Come garantire la consistenza in lettura dei dati?

Ogni volta che viene operato un update in fondo alla tabella viene inserita una riga, mentre la vecchia viene marcata come “dead” e resta a disposizione di eventuali transazioni, avviate prima della sua cancellazione, in modo da garantire la consistenza in lettura

Comando Vacuum - elimina le dead tuples da tutto il database o da una singola tabella e, se lanciato con la clausola *analyze*, aggiorna le statistiche di utilizzo degli oggetti ad uso dell'ottimizzatore.

L'esecuzione di vacuum è un'operazione pesante, perché il sistema deve acquisire un lock esclusivo sull'oggetto

Può essere abilitato di default nel file postgresql.conf: **autovacuum = on**

I dati geometrici

- I tipi geometrici possono rappresentare e gestire oggetti spaziali bidimensionali.
- I dati geometrici permettono di memorizzare posizioni o elementi geometrici all'interno del database come qualsiasi altro tipo di dato.

```
CREATE TABLE italian.nairo_terrains
(
      i_id_terrain serial NOT NULL ,
      p_terrain_coord point NOT NULL ,
      b_flag_activate boolean NOT NULL DEFAULT false ,
      i_terrain_slots smallint NOT NULL DEFAULT 0,
      -- terrain slots available for users
      CONSTRAINT pk_i_id_terrain PRIMARY KEY ( i_id_terrain )
)
WITH ( OIDS = FALSE );
```

Array: creazione e inserimento

PostgreSQL permette la creazione di colonne definite come array multidimensionali a lunghezza variabile.
Il dato array non permette la presenza di campi null all'interno dell'array ma solo
dell'array nella sua interezza.
La creazione di un dato array è banalmente semplice, basta posporre a qualsiasi definizione di dato due parentesi quadre.

```
CREATE TABLE test_array (a_intero integer[]);
```

Crea una tabella con un campo intero definito come array multidimensionale.

```
INSERT INTO test_array VALUES ('{3}');
```

Inserisce un array monodimensionale contenente un solo elemento dal valore pari a 3.

```
INSERT INTO test_array VALUES ('{3,4}');
```

Inserisce un array monodimensionale contenente due elementi dal valore 3 e 4

Array: selezione

L'estrazione dei dati array può avvenire sull'intero array oppure sul singolo elemento.

```
SELECT * FROM test_array ;
-----------------
{3}
{3 ,4}
{{3 ,4} ,{34 ,22}}
(3 rows )

SELECT a_intero [1] FROM test_array ;
----------
3
3
(3 rows )
```

Indici

- PostgreSQL consente l'utilizzo di diversi tipi di indice. Il costo di accesso tramite indice viene effettuato mediante la stima del numero di tuple restituite e dell'operatore applicato, assieme alle statistiche sulle colonne coinvolte
- **B-tree**: sono il default, vengono usati per query di uguaglianza e range con operatori <, =, >. Su indice multi-colonna solo l'uguaglianza limita le pagine di indice, gli altri operatori vengono rivalutati al volo.
- **Hash**: utili solo per uguaglianze strette
- **GIST**: Generalized Index Search Tree, fornisce una interfaccia per l'integrazione di indici user-defined (es. indici R-Tree, per similarità, ecc.). L'utente deve fornire l'implementazione di 7 funzioni C (consistent, union, picksplit, compress, decompress, same, penalty)
- **GiN**: indice inverso per la ricerca full-text (su colonne tsvector). E' più veloce che un generico GiST.

Ereditarietà

- **Le tabelle figlie ereditano**
  - Vincolo NOT NULL
  - Vincoli d'integrità generici
  - Valori di default degli attributi
- **Le tabelle figlie NON ereditano**
  - indici
  - Vincoli d'integrità referenziale
  - Permessi
- Gli aggiornamenti si propagano dalla tabella padre alle tabelle figlie
- Non è possibile rinominare direttamente gli attributi delle tabelle figlie
- È possibile abilitare o disabilitare l'ereditarietà sulle tabelle figlie

Partizionamento

- Il constraint exclusion permette di partizionare i dati su disco dando la possibilità all'ottimizzatore di percorrere solo le tabelle ereditate che rispettano le constraint verificate dalla condizione di WHERE.
- Impostando a ON il parametro constraint_exclusion l'ottimizzatore è in grado di escludere a priori le tabelle ereditate non "incluse" nella where condition.

```
CREATE TABLE fatture (
id_fattura int not null ,
data_fattura date not null ,
importo int ,
iva int
);
```

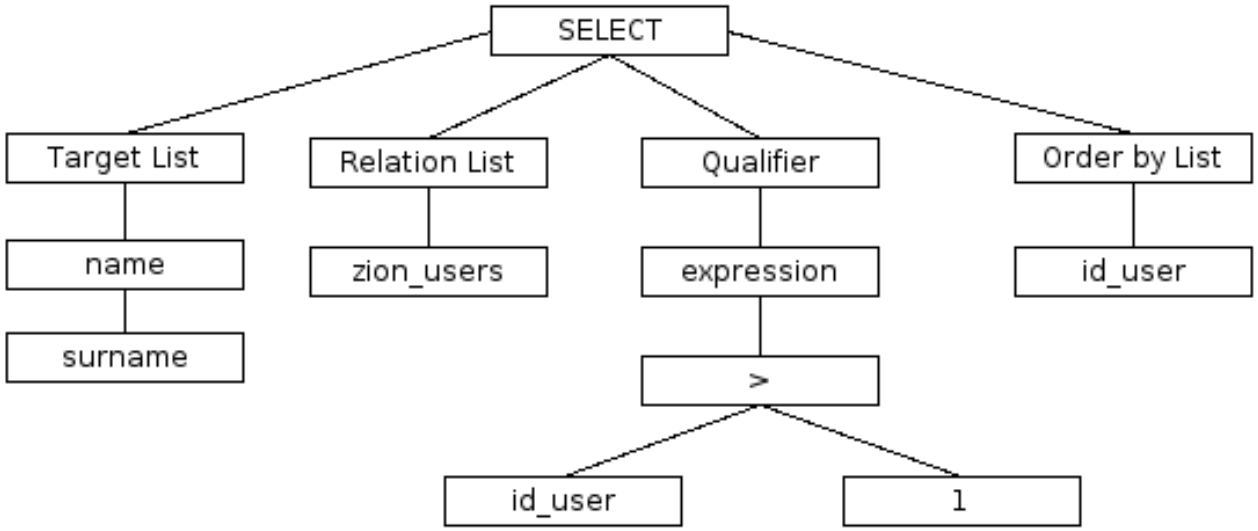
```
CREATE TABLE fatture_2006
(
CHECK (
data_fattura >= DATE '2006 -01 -01 '
AND data_fattura < DATE '2006 -12 -31 ')
)
INHERITS ( fatture );
```

```
CREATE INDEX fatture_2006idx ON fatture_2006 ( data_fattura );
```

Esecuzione della query(1)

- I step – il parser verifica la sintassi della query e la riscrive in quello che viene chiamato *parse tree*.
- II step - se il parse tree non contiene errori, viene rimandato di nuovo al parser che esegue l'interpretazione e la trasformazione semantica producendo al termine il *query tree*.

SELECT name, surname FROM zion_users WHERE id_user>1 ORDER BY id_user;



Esecuzione della query(2)

- Una volta definito il query tree, il **query planner** ne esegue una scansione alla ricerca di tutti i possibili piani d'esecuzione per risolvere la query
- Il planner assegna alle varie operazioni un costo che viene misurato in unità I/O più una percentuale maggiorativa di questo costo legata al tempo di processore richiesto → stime derivate dai parametri del le postgresql.conf
- Il piano la cui stima di costo è il minore di tutti viene adoperato per la risoluzione della query → Poiché l'intera operazione è una stima di costi capita spesso che il piano d'esecuzione scelto non corrisponda alla migliore risoluzione
- Il piano viene inviato all'**executor** che lo percorre per intero e al termine ritorna gli eventuali dati recuperati al backend

Explain

- Explain è un comando SQL che consente di analizzare le scelte dell'ottimizzatore per l'esecuzione di una query.
- Explain si basa sulle statistiche raccolte durante il funzionamento del database; le statistiche possono essere aggiornate anche eseguendo explain analyze.
- La lettura del piano d'esecuzione è un'operazione fondamentale per eseguire un tuning ottimale. Un piano d'esecuzione si legge a partire dalla riga più indentata a destra e più in basso.
- Le singole voci di un piano di esecuzione hanno la seguente struttura:

  Operatore (cost = COST_STARTUP .. TOTAL_COST rows = NUM_ROWS width = ROW_WIDT)

Explain: operatori

- **Seq_scan**
- Non è mai completamente disattivabile poiché è l'unico capace di accedere sempre e comunque ai file di tabella.
- Come parametro di input viene accettata una tabella e, in caso di where condition, l'operatore restituisce le sole righe che verificano le condizioni di filtro.
- Lavora in stream
- Il costo di esecuzione di un seq_scan è inferiore ad un full index scan
- Svantaggio: Portare in memoria tutte le pagine della tabella letta

Explain: operatori

- **Index_scan**
- accetta come parametro di input un indice definito su di una o più colonne e restituisce le sole voci di indice specificate dalla clausola di where.
- lavora per range → definito un valore max e min nella where condition l'indice viene attraversato solo per quei valori.
- i dati restituiti sono già ordinati
- a differenza di altri database commerciali, l'indice b-tree di PostgreSQL non contiene i dati indicizzati ma soltanto i riferimenti alle pagine di tabella corrispondenti → ogni lettura di indice corrisponda di fatto a due seek su disco

Explain: operatori

- **Bitmap_scan**
- permette di elaborare indici bitmap senza averli su disco evitando al database l'onere imposto dalle scadenti prestazioni in update di questa tipologia di indici.
- L'operatore è suddiviso in due sotto operatori, i bitmap_index_scan e il bitmap_heap_scan.
- Il primo sub-operatore esegue una lettura sequenziale delle pagine dell'indice btree senza percorrerlo
- il secondo sub-operatore, utilizzando le informazioni recuperate dal bitmap_index_scan, recupera con un accesso casuale le pagine dove sono presenti i dati che soddisfano la condizione di filtro
- questo operatore verrà quindi sempre scelto quando l'indice da scandire è più piccolo della tabella, cosa che garantisce un costo inferiore rispetto al seq_scan

Explain: operatori

- **Sort**
- Questo operatore si occupa di ordinare i dati accettando come input un set di dati e restituendoli ordinati.
- Il suo modo di operare richiede che l'intero set dati gli venga passato per intero → L'operatore sort impegna l'area di memoria work_mem denita nel le postgresql.conf
- Qualora la richiesta di memoria eccedesse la quantità specificata per work_mem l'ordinamento prosegue su disco diventando più oneroso per il sistema.
- La comparsa di questo operatore nel piano di esecuzione può avvenire per due motivi
  - Nella select è specificato un ORDER BY
  - Ci sono operatori che richiedono dei dati ordinati prima di poter essere applicati al set

Explain: Esempi

Piano d'esecuzione stimato

```
explain select * from art_article where i_id_art =62;
                                QUERY PLAN
-------------------------------------------------------------------------
Index Scan using pk_id_art on art_article ( cost =0.00..8.27 rows =1 width =801)
Index Cond : ( i_id_art = 62)
(2 rows )
```

Piano d'esecuzione reale

```
explain analyze select * from art_article where i_id_art =62;
                                QUERY PLAN
-----------------------------------------------------------------------
Index Scan using pk_id_art on art_article ( cost =0.00..8.27 rows =1 width =801)
    (
actual time =0.030..0.034 rows =1 loops =1)
Index Cond : ( i_id_art = 62)
Total runtime : 0.106 ms
(3 rows )
```

Progetto di Laboratorio – III parte

- Si crei il database `archeofoss2010` e le tabelle ottenute al passo precedente utilizzando le maschere di PgAdmin III.
- Se opportuno si ristrutturi il modello logico utilizzando le proprietà di PostgreSQL
- Si creino per ciascuna tabella e se necessari:
  - Chiave primaria
  - Chiave esterna (specificando la politica di reazione al vincolo d'integrità)
  - Vincolo di univocità
  - Vincolo di obbligatorietà
  - Check (condizione)

- Qualche interrogazione …

- Estrarre tutte le unità stratigrafiche
- Estrarre le sole unità stratigrafiche memorizzate come tali
- Quante unità stratigrafiche hanno uno stato di conservazione “buono”?
- Estrarre i dati dei reperti con quelli del rivestimento associato

Introduzione al linguaggio procedurale

- PL/pgSQL permette di incapsulare istruzioni SQL all'interno del database senza dover mandare ogni volta la query dal backend al server.
- Usando PL/pgSQL si ottengono i seguenti vantaggi:
  - Eliminazione di scambio dati inutile tra client e server
  - I dati prodotti durante le fasi intermedie dell'esecuzione non devono essere trasferiti tra client e server per l'elaborazione
  - I piani di esecuzione vengono generati solo all'inizio della sessione
- pl/pgsql e' un linguaggio strutturato a blocchi. Un blocco e' composto dalle seguenti sezioni:

```
DECLARE
dichiarazioni di
variabili
BEGIN
istruzioni
END ;
```

STORED PROCEDURES(1)

A partire da SQL-2 è possibile definire procedure dette stored procedures per il fatto che vengono memorizzate all'interno della base di dati come parti dello schema.
Le procedure permettono di associare un nome ad un'istruzione SQL, con la possibilità di specificare dei parametri da utilizzare per lo scambio di informazioni con la procedura. I vantaggi sono: minor quantità di dati trasferita tra client e server, una più facile manutenibiltà, favorisce il riutilizzo della logica dell'applicazione da parte di utenti diversi e la possibilità di ottenere in diversi casi un sensibile incremento delle prestazione. In tal modo però la logica di business risulta in gran parte inglobata nella base di dati.

```
CREATE TABLE admins
(
id_admin serial NOT NULL,
user_id character varying(30) NOT NULL,
passw character varying(100) NOT NULL,
flag_act boolean NOT NULL,
name character varying(50) NOT NULL,
surname character varying(50) NOT NULL,
email character varying(50) NOT NULL,
flag_su boolean DEFAULT false NOT NULL
);
```

```
CREATE OR REPLACE FUNCTION
auth_admins(character varying, character
varying) RETURNS smallint AS
$BODY$
DECLARE
     v_username alias FOR $1;
     v_password alias FOR $2;
     b_status int2;
BEGIN
     SELECT INTO b_status count(*)
     FROM admins
     WHERE user_id = v_username
     AND passw = md5(v_password);
RETURN b_status;
END;
$BODY$
LANGUAGE plpgsql;
```

STORED PROCEDURES(2)

- Tutte le keyword e gli identificatori sono case insensitive.
- I commenti su singola linea iniziano con la sequenza --.
- I commenti multiriga vengono fatti alla stessa maniera del linguaggio C /* COMMENTO MULTIRIGA */.
- **Override delle funzioni** - PostgreSQL permette di avere più funzioni aventi lo stesso nome ma con numero parametri o tipo dato di questi differenti
- **Cancellazione** - Per la cancellazione di una funzione si adopera il comando DROP FUNCTION seguito dalla funzione con segnatura completa
- **Esecuzione di una funzione** - SELECT auth_admins('foo', 'bar');
- **Errore di cast** - SELECT auth_admins(0,'bar') ;

Costrutti: decisioni

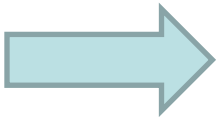
- Similmente ad altri linguaggi di programmazione pl/pgsql permette di prendere decisioni con il costrutto IF … THEN.
- Nella forma più estesa possibile IF si presenta così:

```
IF number = 0 THEN
      result := ’zero’;
ELSIF number > 0 THEN
      result := ’positive’;
ELSIF number < 0 THEN
      result := ’negative’;
ELSE
      result := ’NULL’;
END IF;
```

Costrutti: iterazioni(1)

- FOR seguito dalla variabile sulla quale verrà eseguito il ciclo. Successivamente ci sono i limiti minore e maggiore che la variabile dovrà assumere seguito da LOOP---END LOOP; che conterrà le istruzioni da eseguire nel ciclo FOR.
- La parola opzionale REVERSE specifica che la variabile dovrà decrescere nell'esecuzione del FOR.
- La parola opzionale BY determina invece l'ammontare dell'incremento assunto dalla variabile ad ogni iterazione

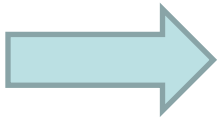
```
--iterazione da 1 a 5
FOR i IN 1..5 LOOP
    RAISE NOTICE 'i è %', i;
END LOOP;
```



```
NOTICE: i è 1
NOTICE: i è 2
NOTICE: i è 3
NOTICE: i è 4
NOTICE: i è 5
```

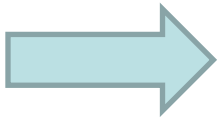
Costrutti: iterazioni(2)

```
--iterazione da 1 a 5
FOR i IN REVERSE 5..1 LOOP
    -- some computations here
    RAISE NOTICE 'i is %', i;
END LOOP;
```



```
NOTICE: i è 5
NOTICE: i è 4
NOTICE: i è 3
NOTICE: i è 2
NOTICE: i è 1
```

```
--iterazione da 1 a 10
FOR i IN REVERSE 10..1 BY 2 LOOP
    -- some computations here
    RAISE NOTICE 'i is %', i;
END LOOP;
```



```
NOTICE: i è 10
NOTICE: i è 8
NOTICE: i è 6
NOTICE: i è 4
NOTICE: i è 2
```

Ciclo FOR con una SELECT

- L'iterazione su di una select avviene in maniera simile al FOR classico ma con le seguenti varianti:
  - la variabile su cui iterare deve essere dichiarata nel blocco DECLARE come RECORD.
  - non viene specificato un range di numeri ma la select su cui si vuole iterare.

```
CREATE OR REPLACE FUNCTION test_select() RETURNS void AS
$BODY$
DECLARE
    r_results RECORD;
BEGIN
FOR r_results IN SELECT * FROM admins LOOP
    RAISE NOTICE 'Il nome utente è %', r_results.user_id;
END LOOP;
END;
$BODY$
LANGUAGE plpgsql;
```

Gestire gli errori

- comandi di RAISE NOTICE o RAISE EXCEPTION che inviando messaggi al logger di PostgreSQL permettono di capire l'evoluzione dell'esecuzione della funzione
- **costrutto exception -** viene posizionato in fondo ad un blocco in modo da catturare le eccezioni che si possono verificare durante l'esecuzione del blocco. La sua struttura e' la seguente:

```
EXCEPTION
WHEN condition [ OR condition ... ] THEN
    handler_statements
[ WHEN condition [ OR condition ... ] THEN
    handler_statements
```

- La condition puo' assumere il valore OTHERS come confronto jolly per catturare tutti gli errori oppure uno qualsiasi dei messaggi di errore di PostgreSQL