



Basi di Dati

BASE DI DATI

Ogni organizzazione è dotata di un SISTEMA INFORMATIVO che gestisce e organizza le informazioni necessarie all'organizzazione.

Per indicare la parziale AUTOMATIZZAZIONE del sistema, di solito viene utilizzato il termine **SISTEMA INFORMATICO**.

le informazioni vengono rappresentate per mezzo di **DATI**.
dato che consente di avere
conoscenza esatta dei fatti

elementi di informazione costituiti da
simboli che devono essere elaborati.



(i dati da soli non hanno alcun significato)

➡ **BASE DI DATI**: è una collezione di dati, utilizzati per rappresentare le informazioni per un sistema informatico.

DBMS: Un **DATA BASE MANAGEMENT SYSTEM** è un sistema software in grado di gestire collezioni di dati:
• GRANDI,
assicurando la loro:
• AFFIDABILITÀ,
• EFFICIENTE,
• PRIVATEZZA,
• EFFICACE.

• CONDIVISE,
• PERSISTENTI;

• **GRANDI**: possono avere grandissime dimensioni, molto più grandi della memoria centrale disponibile, infatti devono prevedere una gestione dei dati in memoria secondaria. I sistemi devono poter gestire i dati senza pose limite alle dimensioni (l'unica limite è la dimensione fisica del dispositivo)

• **CONDIVISE**: applicazioni e utenti devono poter accedere, secondo opportune modalità, a dati comuni. In questo modo si riduce la RIDONDANZA dei dati, si evitano ripetizioni e si riduce la possibilità di INCONSENSENZE (se esistono copie degli stessi dati è possibile che in qualche momento non siano uguali)

• **PERSISTENTI**: hanno un tempo di vita che non è limitato a quello delle singole esecuzioni dei programmi che le utilizzano.
(i dati gestiti da un programma in memoria centrale hanno una vita che inizia e termina con l'esecuzione del programma)

• **AFFIDABILITÀ**: capacità del sistema di conservare intatto il contenuto della base di dati (o di permetterne la ricostruzione) in caso di malfunzionamenti hardware e software.

• **PRIVATEZZA**: ciascun utente, opportunamente riconosciuto, viene abilitato a svolgere determinate azioni sui dati, attraverso meccanismi di AUTORIZZAZIONE.

• **EFFICIENTI**: capaci di svolgere le operazioni utilizzando un insieme di risorse (tempo e spazio) accettabile per gli utenti.
(dipende dall'implementazione del DBMS e i DBMS forniscono un ampio insieme di funzionalità richiedendo molte risorse e garantiscono l'efficienza solo se il sistema informatico su cui è installato è adeguato)

• **EFFICACI**: capaci di rendere produttive le attività dei loro utenti.

• **TRANSAZIONE**: sequenza ordinata di operazioni da considerare INVIOVISIBILE (atomico),

corretto anche in presenza di concorrenza e con effetti definitivi. Le transazioni concorrenti devono essere coerenti e i risultati sono permanenti.

Es (atomico). "Spostare i soldi dal conto bancario A a B: o il ritiro viene effettuato su A e trasferito su B oppure nessuna operazione viene eseguita."

Es (concorrente). "Se due agenzie diverse vogliono prenotare lo stesso (disponibile) posto su un treno, esso non deve essere prenotato due volte."

DBMS VS FILE SYSTEM

da gestione di collezioni di dati è possibile anche con sistemi più semplici, come il **FILE SYSTEM**, presente in tutti i sistemi operativi. I file gestiscono insiemi di dati "localmente" a una specifica procedura o applicazione. I DBMS estendono

le funzioni del file system, fornendo l'accesso condiviso agli stessi dati da parte di più utenti e applicazioni, garantendo inoltre altri servizi e utilizzando i file con un'organizzazione dei dati più sofisticata.

RAPPRESENTAZIONE DEI DATI

I dati vengono rappresentati a livelli diversi, permettendo così l'INDIPENDENZA dei dati dalla loro rappresentazione fisica.

Ogni software (che non utilizza DBMS) contiene una diversa rappresentazione dei dati, causando così incoerenza nella loro rappresentazione. Mentre in DBMS una porzione del database contiene una descrizione centralizzata dei dati ed è utilizzata tra tutte le applicazioni software.

→ Ogni programma può interpretare, ad alto livello, i dati in modo diverso, indipendentemente da come sono rappresentati a basso livello: **MODELLO DEI DATI**

insieme di costrutti utilizzati per organizzare i dati di interesse e descrivere la dinamica.

→ **MODELLO RELAZIONALE**: permette di definire tipi per mezzo dei costrutti RELAZIONE, che consente di organizzare i dati in insiemi di record a struttura fissa.

viene rappresentato per mezzo di una TABELLA

l'ordine è irrilevante { RIGHE: rappresentano specifici record

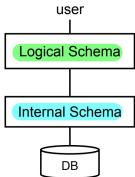
COLONNE: corrispondono ai campi del record

Nella base di dati troviamo:
• **SCHEMA**, parte invariante nel tempo, costituita dalle caratteristiche dei dati (l'intestazione tabella)
• **ISTANZA**, parte variabile nel tempo, costituita dai valori effettivi (corpo della tabella)

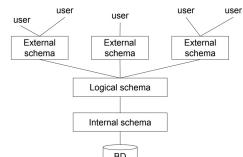
da notare che il modello può essere ulteriormente sviluppato tenendo presenti diversi LIVELLI DI ASTRATTO nella progettazione del database:

- **MODELLO CONCETUALE**: livello più alto di astrazione, rappresenta la struttura dei dati in modo indipendente dal DBMS utilizzato per implementarlo. Descrive cosa rappresentano i dati e COME SONO COLLEGATI tra loro, utilizzando un linguaggio comprensibile. Utilizzato nelle fasi preliminari di progettazione, il più diffuso è il modello **ENTITY-RELATIONSHIP (ER)**.
- **MODELLO LOGICO**: è il livello successivo di astrazione e rappresenta il database in modo più dettagliato, ma ancora indipendentemente dalle specifiche tecnologie di implementazione. Si basa sul modello CONCETUALE e definisce come i dati saranno strutturati nel database, senza tenere conto delle limitazioni fisiche o delle funzionalità specifiche del DBMS. (es. modello RELAZIONALE).

ARCHITETTURA DBMS



- **SCHEMA LOGICO**: descrizione dell'intera base di dati nel modello logico principale del DBMS. (struttura tabella)
- **SCHEMA INTERNO (FISICO)**: rappresenta lo schema logico a livello di archiviazione, utilizzando specifiche strutture dati (guzzi), come file, records e data pointers.



INDIPENDENZA DEI DATI

d'accesso ai dati avviene solo tramite il livello esterno

• **INDIPENDENZA FISICA**: il livello logico e quello esterno sono indipendenti da quello fisico: una relazione è utilizzata nello stesso modo qualunque sia la realizzazione fisica. (DBMS)

- **INDIPENDENZA LOGICA**: il livello esterno è indipendente da quello logico. Aggiunte o modifiche non richiedono modifiche a livello logico.

(tabelle di base)

LINGUAGGI PER BASI DI DATI

Un altro aspetto che rende efficaci le basi di dati è la sua disponibilità attraverso diversi linguaggi ed interfacce:

- **SQL**: (STRUCTURED QUERY LANGUAGE), linguaggio testuale interattivo;
- **DML**: (DATA MANIPULATION LANGUAGE), per l'interrogazione e l'aggiornamento di istanze nella base di dati;
- **DDL**: (DATA DEFINITION LANGUAGE), per la definizione di schemi (logici, esterni, fisici) e altre operazioni generali.

PERSONAGGI E INTERPRETI

Categorie di persone che interagiscono con una base di dati o DBMS:

- **DATABASE ADMINISTRATOR**: (DBA) l'amministratore della base di dati è:
 - responsabile della progettazione, controllo e amministrazione del DB;
 - media le esigenze espresse dagli utenti, garantendo un controllo CENTRALIZZATO sui dati;
 - garantisce sufficienti prestazioni, assicura l'affidabilità del sistema e gestisce le autorizzazioni di accesso ai dati
- **PROGETTISTI E PROGRAMMATORI**: definiscono e realizzano i programmi che accedono alla base di dati utilizzando DML, oppure altri strumenti di supporto alla generazione di interfacce.
- **UTENTI**: utilizzano la base di dati per le proprie attività, possono essere divisi in due categorie:
 - **UTENTI FINALI**: utilizzano TRANSAZIONI, cioè programmi che realizzano attività predefinite e di frequenza elevata (previste a priori)
 - **UTENTI CASUALI**: utilizzano linguaggi interattivi per l'accesso al DB ed eseguono operazioni non previste (es. interrogazioni e aggiornamenti)

VANTAGGI E SVANTAGGI DEI DBMS

PROS

- permettono di considerare i dati come una risorsa comune a disposizione di tutte le componenti dell'organizzazione
- Disponibilità di SERVIZI INTEGRATI (insieme di funzionalità e strumenti offerto dal DBMS)
- Controllo CENTRALIZZATO dei dati (sono raccolti, gestiti e mantenuti in un'unica posizione logica o fisica)
- La condivisione permette di ridurre RIBONDANZE e INCONSENSENZE.
- d'**INDIPENDENZA DEI DATI**: favorisce lo sviluppo di applicazioni più flessibili e facilmente modificabili

CONS

- Sono costosi e complessi.
- Formiscono, in forma INTEGRATA, una serie di servizi associati a un costo, scardinale quelli non necessari da quelli richiesti comporta una riduzione delle prestazioni

MODELLO logici

- Tre modelli logici TRADIZIONALI:
 - MODELLO GERARCHICO

• NETWORK



• RELAZIONALE → basato sul concetto matematico di RELAZIONE. Rappresentazione per mezzo di tabella.

Indipendenza dei dati

- Altri modelli logici recenti:
 - OBJECT ORIENTED

• XML

- RELAZIONE MATEMATICA = RELAZIONE + RELATIONSHIP

↓
Rappresenta una classe di fatti, nel modello ENTITY-RELATIONSHIP (ASSOCIAZIONE)

↓
secondo il modello relazionale dei dati

come nella teoria degli insiemi

- Una tabella rappresenta una relazione se:
 - ① le righe sono diverse fra loro
 - ② le intestazioni delle colonne sono diverse fra loro
 - ③ i valori di ogni colonna sono fra loro omogenei

RELAZIONI E TABELLE

Dati $m > 0$ insiemi D_1, D_2, \dots, D_m , non necessariamente distinti, il prodotto cartesiano indicato con $D_1 \times D_2 \times \dots \times D_m$ è costituito dall'insieme delle m -uple (v_1, v_2, \dots, v_m) t.c. $v_i \in D_i$ per $1 \leq i \leq m$. Una relazione matematica sui domini D_1, \dots, D_m è un sottoinsieme del prodotto cartesiano $D_1 \times D_2 \times \dots \times D_m$.

Queste relazioni possono essere utilizzate per rappresentare i dati di interesse per qualche applicazione.

Esempio: relazione che contiene i dati relativi ai risultati di un insieme di partite di calcio

$\text{Matches} \subseteq \text{String} \times \text{String} \times \text{int} \times \text{int}$

Barca	Bayern	3	1
Bayern	Real	2	0
Barca	Psg	0	2
Psg	Real	0	1

←
ciascuna m -upla contiene dati fra

loro collegati e stabilisce un legame fra loro. Se m -uple sono ORDINATE, è definito un ordinamento fra i domini

↳ L'ORDINAMENTO è una caratteristica insoddisfacente del concetto di relazione matematica, solitamente in informatica si tende a privilegiare metizioni NON-POSIZIONALI

STRUTTURA DATI NON POSIZIONALE

Ogni nome, unico, nella tabella (ATTRIBUTO) è associato ad un dominio. L'attributo fornisce così il "ruolo" del dominio.

Name	Age	Goals	Goals
Barca	Bayern	3	1
Bayern	Real	2	0
Barca	Psg	0	2
Psg	Real	0	1

TABELLE E RELAZIONI

Una tabella rappresenta una relazione: • ogni riga può assumere qualunque posizione

• ogni colonna può assumere qualunque posizione

- Una tabella rappresenta una relazione se:
- tutte le righe sono diverse
 - tutte le colonne sono diverse
 - i valori nelle colonne sono omogenei

I riferimenti tra dati memorizzati in diverse relazioni sono rappresentate attraverso valori dei domini che compaiono nelle tuple.

Vantaggi:

- Indipendenza dalle strutture fatiche che possono cambiare dinamicamente

- Si rappresenta solo ciò che è rilevante dal punto di vista dell'applicazione

- L'utente finale vede gli stessi dati dei programmati

- Portabilità dei dati

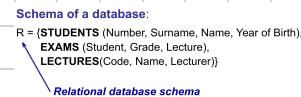
- Puntatori unidirezionali

DEFINIZIONI

- **SCHEMA DI RELAZIONE**: una relazione detta R con un insieme di attributi $A_1, \dots, A_m : R(A_1, \dots, A_m)$



- **SCHEMA DI BASE DI DATI**: insieme di schemi di relazione $R = \{R_1(X_1), \dots, R_k(X_k)\}$



- **N-UPLA** su un insieme di attributi X: funzione che associa a ciascun attributo $A \in X$ un valore del dominio di A.

- **$t[A]$** : dimostra le varie delle m-uple t sull'attributo A. es. $t[Home] = Barca$

- **ISTANZA DI RELAZIONE SU UNO SCHEMA $R(x)$** : insieme r di m-uple su X

- **ISTANZA DI BASE DI DATI** su uno schema $R = \{R_1(X_1), \dots, R_k(X_k)\}$: insieme di relazioni $r = \{r_1, \dots, r_m\}$ (con r_i relazione su R_i)

LEZIONE 02 - Relational Model

2.1.15 Informazione Incompleta e Valori Nulli

Modello relazionale impone un certo grado di rigidità.

informazioni rappresentate con

TUPLE di dati sfaccendati.

es. Persona (Cognome, Nome, Indirizzo, Telefono)

Non sempre tutti gli attributi sono disponibili in ogni tupla (es. manca il campo "Telefono"), non è utile utilizzare un valore del dominio per rappresentare l'assenza dell'informazione in quanto potrebbero causare delle ambiguità.

→ utilizziamo il **VALORE NULLO**, valore aggiuntivo rispetto a quelli del dominio.

Studenti

Matricola	Cognome	Nome	Data di nascita
276545	Rossi	Maria	null
null	Neri	Anna	23/04/2001
null	Verdi	Fabio	12/02/2001

Esami

Studente	Voto	Corso
276545	28	01
null	27	null
200768	24	null

Corsi

Codice	Titolo	Docente
01	Analisi	Giani
03	Chimica	null
null	Chimica	Belli

Tutte le valenze nulle non sono sempre utilizzabili, infatti se il muto sul numero di matricola o sul codice del corso può generare problemi, in quanto sono utilizzati per stabilire correlazioni fra tuple di relazioni diverse.

Tre casi per la mancanza di un valore: 1) UNKNOWN VALUE

2) INEXISTANT VALUE

3) UNINFORMATIVE VALUE

nei moderni DBMS non ci sono specifiche distinzioni

2.2 VINCOLI DI INTEGRITÀ

Per evitare situazioni in cui i dati rappresentati non sono corretti viene introdotto il **VINCOLO DI INTEGRITÀ**

proprietà che deve essere soddisfatta dalle istanze
ogni vincolo può essere visto come una funzione booleana (o predicato)

Possiamo classificare i vincoli in due categorie:

- **VINCOLO INTRARELACIONALE**: se il suo soddisfacimento è definito rispetto a singole relazioni della base di dati.
 - **VINCOLO DI TUPLA**: vincolo che può essere valutato su ciascuna tupla, indipendentemente dalle altre.
 - **VINCOLO SU VALORI**: (o vincolo di dominio) impone una restrizione sul dominio dell'attributo definito con riferimento a singoli valori (es. voto >= 30 in "esami")
- **VINCOLO INTERRELACIONALE**: vinkolo che coinvolge più relazioni (es. viene richiesto che m° di matricola compare nella relazione "esami" solo se compare nella relazione "studenti")

I DBMS non supportano tutti i tipi di vincoli:

- possiamo indicare i tipi di vincoli e il DBMS impedisce la loro violazione
- quando non sono supportati sono l'utente e il programmatore a configurare i vincoli al di fuori del DBMS.

Intra-relational over values				
EXAM	Student	Grade	Lauda	Lecture
Inter-relational	276545	32		01
	276545	30	yes	02
	787643	27	yes	03
	739430	24		04

Intra-relational over tuples				
STUDENT	Number	Surname	Name	
	276545	Rossi	Mario	
	787643	Neri	Piero	
	787643	Bianchi	Luca	

2.2.1 VINCOLO DI TUPLA

La sintassi è esprimibile utilizzando espressioni booleane (\wedge, \vee, \neg) con atomi che confrontano valori di attributo o espressioni aritmetiche (uguaglianza, disegualanza e ordinamento)

es. $(voto >= 18) \wedge (voto <= 30) \quad (\text{not}(Lode = "lode")) \text{ or } (voto = 30)$

2.2.2 CHIAVE

→ insieme di attributi per identificare univocamente le tuple di una relazione.

Distinguiamo:

- **SUPERCHIAVE**: un insieme K di attributi è superchiave su r se r non contiene due tuple distinte t₁ e t₂

con $t_1[K] = t_2[K]$

- **CHIAVE**: K è chiave di r se è una superchiave minimale di r, ovvero non esiste un'altra superchiave K' di r che sia contenuta in K come sottinsieme proprio.

Matricola	Cognome	Nome	Nascita	Corso
4328	Rossi	Luigi	29/04/2000	Ing. Informatica
6328	Rossi	Dario	29/04/2000	Ing. Informatica
4766	Rossi	Luca	01/05/2001	Ing. Civile
4856	Neri	Luca	01/05/2001	Ing. Meccanica
5536	Neri	Luca	05/03/1999	Ing. Meccanica

es.
l'insieme {Matricola} è superchiave, è anche superchiave minimale in quanto contiene un solo attributo, quindi {Matricola} è chiave.
{Cognome, Nome, Nascita} è superchiave, ma non è superchiave minimale, perché nessuno dei suoi sottointesi è superchiave. (possono esistere tuple con cognome e nascita uguali, ecc...)
{Matricola, Corso} è superchiave, ma non è una superchiave minimale, perché il sottointeso {Matricola} è una superchiave minimale
{Nome, Corso} non è superchiave perché possono comparire tuple fra loro uguali sia su nome che su corso.

de chiavi che ci "interessano" sono quelle corrispondenti ai vincoli di integrità \Rightarrow definendo uno SCHEMA, associamo ad es. SCHEMA: Studente (Matricola, Nome, Cognome, Nascita, Corso)

VINCOLO: 2 chiavi {Matricola}

{Cognome, Nome, Nascita}

N.B. Ciascuna relazione ha una chiave

2.2.3 CHIAVI E VALORI NULLI

In presenza di valori nulli non è sempre possibile identificare univocamente una tupla.

\hookrightarrow CHIAVE PRIMARIA: chiave in cui vietiamo la presenza di valori nulli.

\hookrightarrow gli attributi che la compagno vengono evidenziati con una SOTTOUNIVERSITÀ.

\hookrightarrow La maggior parte dei riferimenti tra relazioni viene realizzata con i suoi valori.

CHIAVE $\xrightarrow{\text{sempre}}$ SUPERCHIAVE (ma il contrario)

\hookrightarrow è chiave solo se è SUPERCHIAVE MINIMALE.

2.2.4 VINCOLI DI INTEGRITÀ REFERENZIALE

FOREIGN KEY (REFERENTIAL INTEGRITY CONSTRAINT) fra un insieme di attributi X di una relazione R₁ e un'altra relazione R₂ è soddisfatto se i valori su X di ciascuna tupla dell'istanza R₁ compaiono come valori della CHIAVE PRIMARIA dell'istanza R₂.

Nel caso in cui la chiave di R₂ è unica e composta di un solo attributo B e l'insieme X è a sua volta costituito di un solo attributo A, il vincolo tra R₁ e R₂ è soddisfatto se: \forall tupla t₁ \in R₁ per cui t₁[A] ≠ Null, \exists t₂ \in R₂ t.c. t₁[A] = t₂[B].

+ ciascuno degli attributi in X deve corrispondere a un preciso attributo della PK K di R₂,

X = A₁, A₂, ..., A_p e K = B₁, B₂, ..., B_p il vincolo di integrità è soddisfatto se

\forall tupla t₁ \in R₁ dove t₁[A_i] ≠ Null \exists t₂ \in R₂ t.c. t₁[A_i] = t₂[B_i] \forall i $1 \leq i \leq p$

Im caso siamo presenti più chiavi \rightarrow una deve essere sottointesa come PK \rightarrow i riferimenti vengono fatti verso la PK

Se una tupla viene eliminata \rightarrow VIOLAZIONE vincolo: - rigetto dell'op.

- eliminazione in cascata

- introduzione valori nulli

LEZIONE 03. ALGEBRA AND RELATIONAL CALCULUS

• DATABASE LANGUAGE: - op. sugli schemi: DDL, data definition language

- op. sui dati: DML, data manipulation language - query instructions, per estrarre i dati di interesse

- update instructions, per inserire nuovi dati o modificare quelli già presenti

- LINGUAGGI DI INTERROGAZIONE:
 - DICHIAIRATIVI: specificano le proprietà del risultato "che cosa"
 - PROCEDURALI: specificano le modalità di generazione del risultato "come"

es. algebra relazionale \rightarrow procedurale

calcolo relazionale \rightarrow dichiarativo (teoricamente, non implementato)

SQL (structured query language) \rightarrow parzialmente dichiarativo (implementato)

QBE (query by example) \rightarrow dichiarativo (implementato)

3.1 ALGEBRA RELAZIONALE

linguaggio PROCEDURALE basato su comandi di tipo algebrico. formato da OPERATORI:

1) INSIEMISTICI TRADIZIONALI: UNIONE, INTERSEZIONE, DIFFERENZA

RELAZIONE \rightarrow insieme di tuple omogenee definite sugli stessi attributi.

- l'**UNIONE** di due relazioni r_1 e r_2 definite sullo stesso insieme di attributi X è indicata come $r_1 \cup r_2$ ed è una relazione ancora su X contenente le tuple che appartengono a r_1 , oppure a r_2 , oppure a entrambe (no duplicati)
- l'**INTERSEZIONE** di $r_1(X)$ e $r_2(X)$ è indicata come con $r_1 \cap r_2$ ed è una relazione su X contenente le tuple che appartengono contemporaneamente a r_1 e r_2 .
- la **DIFERENZA** di $r_1(X)$ e $r_2(X)$ è indicata con $r_1 - r_2$ ed è una relazione su X contenente le tuple che appartengono a r_1 e non appartengono a r_2 .

2) OPERATORI SPECIFICI: RIDENOMINAZIONE, SELEZIONE, PROIEZIONE

- RIDENOMINAZIONE: aggiorna i nomi degli attributi, a seconda delle necessità, lasciando inalterato il contenuto delle relazioni.

def P relazione definita sull'insieme di attributi e sia Y un altro insieme di attributi con $\#X = \#Y$. Siamo

A_1, \dots, A_k e B_1, \dots, B_k rispettivamente un ordinamento per gli attributi in X e in Y . Allora:

$$P_{B_1, \dots, B_k \leftarrow A_1, \dots, A_k} (r)$$

es. P (Paterno)

b) SELEZIONE: produce un sottoinsieme delle tuple su tutti gli attributi. (\rightarrow genera decomposizioni orizzontali). È indicata

con σ e si pedisce inseriamo la combinazione di selezione.

formula proposizionale F su X ottenuta combinando:

- connettivi AND, OR, NOT (\wedge, \vee, \neg)

- combinazioni atomiche: $A \oplus B$ $A \otimes C$



La selezione $\sigma_F(r)$, dove r relazione e F formula proposizionale, produce una relazione sugli stessi attributi di r

che contiene le tuple di r su cui F è vera.

- PROIEZIONE: sul risultato troviamo tutte le tuple, ma su un sottoinsieme degli attributi (\rightarrow genera decomposizioni verticali)

def Dati una relazione $r(x)$ e un sottoinsieme Y di X , la proiezione di r su Y , indicata con $\pi_Y(r)$, è l'insieme di

tuple di r considerando solo i valori su Y :

Impiegati				
Cognome	Nome	Età	Stipendio	
Rossi	Mario	25	2000,00	
Neri	Luca	40	3000,00	
Verdi	Nico	36	4500,00	
Rossi	Marco	40	3900,00	

$\sigma_{\text{Eta}>30 \wedge \text{Stipendio}>4000,00}(Impiegati)$

Cognome	Nome	Età	Stipendio
Verdi	Nico	36	4500,00

$R_4(\cdot) = \{t[4] | t \in r\}$

Impiegati			
Cognome	Nome	Reparto	Capo
Rossi	Mario	Vendite	Gatti
Neri	Neri	Produzione	Gatti
Verdi	Mario	Personale	Lupi
Rossi	Marcos	Personale	Lupi

$\pi_{\text{Cognome}, \text{Nome}}(\text{Impiegati})$			
Cognome	Nome	Reparto	Capo
Rossi	Mario	Vendite	Gatti
Neri	Neri	Produzione	Gatti
Verdi	Mario	Personale	Lupi
Rossi	Marcos	Personale	Lupi

(chiave $\{\text{Cognome}, \text{Nome}\}$)

- se Y e' superchiave, allora r non contiene tuple uguali su Y , quindi ogni tupla da un contributo diverso alla proiezione

- se la proiezione ha tante tuple quante l'operando, allora ciascuna tupla di r contribuisce alla proiezione con valori diversi (r non contiene copie di tuple uguali su Y)

Impiegati			
Cognome	Nome	Reparto	Capo
Rossi	Mario	Vendite	Gatti
Neri	Luca	Vendite	Gatti
Verdi	Mario	Personale	Lupi
Rossi	Marcos	Personale	Lupi

$\pi_{\text{Reparto}, \text{Capo}}(\text{Impiegati})$			
Reparto	Capo	Reparto	Capo
Vendite	Gatti	Vendite	Gatti
Personale	Lupi	Personale	Lupi

 $(\{\text{Reparto}, \text{Capo}\} \text{ NO superchiave})$

3) OPERATORE DI JOIN: permette di correlare dati contenuti in relazioni diverse confrontando i valori contenuti in esse.

a) JOIN NATURALE, correla dati in relazioni diverse, sulla base di valori uguali in attributi con lo stesso nome. È indicato con Δ .

RISULTATO: una relazione sull'unione degli insiemi di attributi degli operandi, le tuple sono ottenute combinando le tuple con valori uguali sugli attributi comuni.

$r_1 \Delta r_2 = ? t \text{ su } X_1, X_2 | \exists t_1 \in r_1 \wedge t_2 \in r_2 \text{ con } t[X_1] = t_1 \wedge t[X_2] = t_2$

r_1	Impiegato	Reparto
Rossi	vendite	
Neri	produzione	
Bianchi	produzione	

r_2	Reparto	Capo
produzione	Mori	
vendite	Bruni	

$r_1 \Delta r_2$	Impiegato	Reparto	Capo
Rossi	vendite	Bruni	
Neri	produzione	Mori	
Bianchi	produzione	Mori	

Infrazioni						
Codice	Data	Agente	Articolo	Stato	Numeri	
143256	25/10/2021	567	44	I	AB 234 ZK	
987554	26/10/2021	456	34	I	AB 234 ZK	
987557	26/10/2021	456	34	I	CB 123 AA	
630876	15/10/2021	456	55	P	CB 123 AA	
539856	12/10/2021	567	44	P	CB 123 AA	

Auto	Stato	Numero	Proprietario	Indirizzo
I	CB 123 AA	Verdi Piero	Via Tigli ...	
I	DE 834 ZZ	Verdi Piero	Via Tigli ...	
I	AB 234 ZK	Bini Luca	Via Aceri ...	
F	CB 123 AA	Beau Marcel	Rue Louis ...	

Infrazioni Δ Auto						
Code	Data	Ag	Art	Stato	Numero	Proprietario
143256	25/10/2021	567	44	I	AB 234 ZK	Bini Luca
987554	26/10/2021	456	34	I	AB 234 ZK	Bini Luca
987557	26/10/2021	456	34	I	CB 123 AA	Verdi Piero
630876	15/10/2021	456	53	P	CB 123 AA	Beau Marcel
539856	12/10/2021	567	44	P	CB 123 AA	Beau Marcel

b) JOIN COMPLETO: ogni tupla contribuisce al risultato finale

E INCOMPLETO: alcune tuple degli operandi NON contribuiscono al risultato, perché c'è altra relazione che non contiene tuple con gli stessi valori sull'attributo comune

CASI LIMITI nessuna delle tuple è combinabile → risultato: RELAZIONE VUOTA

tutte le tuple sono combinabili → risultato: #tuple = $|r_1| \times |r_2|$

DIMENSIONI RISULTATO di un JOIN:

$r_1 \Delta r_2 \rightarrow$ contiene un numero di tuple tra $0 \leq |r_1| \times |r_2| \leq |r_1| + |r_2|$

(il valore della chiave da r_2 è unico, quindi le tuple di r_2 possono incontrare più tuple di r_1)

→ se contiene una chiave da r_2 : $o \in |r_1, MR_2| \subseteq |r_1|$

(le tuple di r_2 sono associate ad almeno una tupla di r_1)

→ se contiene una chiave da r_2 e UNIQ FOREIGN KEY: $|r_1, MR_2| = |r_1|$

(una tupla di r_2)

c) JOIN ESTERNO: il join tende a trascurare tuple senza una controparte, ma può essere pericoloso, il join esterno utilizza tutte le tuple ed eventualmente con valori nulli dove non trova la corrispondenza.

3 tipi:

• JOIN ESTERNO SINISTRO: estende solo le tuple del primo operando A ΔB

• JOIN ESTERNO DESTRO: estende solo le tuple del secondo operando A ΔB

• JOIN ESTERNO COMPLETO: estende tutte le tuple A ΔB

d) THETA-JOIN: dal momento che il PRODOTTO CARTESIANO ($\epsilon_{\text{condition}}(R_1 \times R_2)$) ha solitamente poca utilità, immette concerne tuple non necessariamente correlate, uniamo il THETA-JOIN:

$R_1 \bowtie_{\text{condizione}} R_2$ (prodotto cartesiano seguito da una selezione)

EQUI-JOIN: e' un THETA-JOIN in cui la condizione di selezione e' una congiungione di atomi di uguaglianza, permette di specificare su quale attributo unire senza utilizzare la ridenominazione.

N.B.: se usiamo il theta-join su due relazioni con attributi in comune, non me tener conto, prenderà solo quelli specificati nella condizione.

	UNIONE ($R_1 \cup R_2$)
INSIEMISTICI	INTERSEZIONE ($R_1 \cap R_2$)
	DIFERENZA ($R_1 - R_2$)
SPECIFICI	SELEZIONE ($\Theta_F(R)$) sulle tuple
	PROIEZIONE ($\pi_F(R)$) sugli attributi
	RIDENOMINAZIONE ($P_{X_1 \leftarrow X_2}(R)$)
JOIN	NATURALE ($R_1 \bowtie R_2$)
	THETA-JOIN ($R_1 \bowtie_F R_2$)

■ ESPRESSIONI DI ALGEBRA RELAZIONALE EQUIVALENTE

= se due espressioni E_1 e E_2 generano lo stesso risultato

$$1. \Theta_{C1 \text{ AND } C2}(R) = \Theta_{C1}(\Theta_{C2}(R))$$

$$2. x, y \in R : \pi_x(R) = \pi_x(\pi_{x \rightarrow y}(R))$$

$$3. \pi_{x \rightarrow y}(\Theta_x(R)) = \Theta_x(\pi_{x \rightarrow y}(R))$$

$$4. \text{CER}_2 : \Theta_C(R_1 \bowtie R_2) = R_1 \bowtie (\Theta_C(R_2)) \quad (\text{riduzione dimensione intermedia del risultato})$$

$$5. x_1 \in R_1, x_2 \in R_2, y_2 \in R_2 \text{ e } (x_1 \cap x_2) \subseteq y_2 \Rightarrow x_2 - y_2 \text{ non e' coinvolto nel JOIN} : \pi_{x_1, y_2}(R_1 \bowtie R_2) = R_1 \bowtie \pi_{y_2}(R_2)$$

$$6. \Theta_C(R_1 \bowtie R_2) = R_1 \bowtie R_2$$

$$7. \Theta_C(R_1 \cup R_2) = \Theta_C(R_1) \cup \Theta_C(R_2)$$

$$8. \Theta_C(R_1 - R_2) = \Theta_C(R_1) - \Theta_C(R_2)$$

$$9. \pi_x(R_1 \cup R_2) = \pi_x(R_1) \cup \pi_x(R_2) \quad \text{N.B. da proiezione non e' distributiva sulla differenza}$$

$$10. \Theta_{C1 \text{ OR } C2}(R) = \Theta_{C1}(R) \cup \Theta_{C2}(R)$$

$$11. \Theta_{C1 \text{ AND } C2}(R) = \Theta_{C1}(R) \cap \Theta_{C2}(R) = \Theta_{C1}(R) \bowtie \Theta_{C2}(R)$$

$$12. \Theta_{C1 \text{ AND } T_{C2}}(R) = \Theta_{C1}(R) - \Theta_{C2}(R)$$

$$13. R_1 \bowtie (R_2 \cup R_3) = (R_1 \bowtie R_2) \cup (R_1 \bowtie R_3)$$

3.1.9 ALGEBRA CON VALORI NULLI

Possiamo "rimediarci" alla presenza di valori nulli in 2 modi:

1. LOGICA A TRE VALORI: VERO

FALSO
UNKNOWN

} non e' necessario

A is null

2. traduciamo i null sintatticamente introducendo 2 condizioni atomiche

A is not null

3.1.10 VISTE O RELAZIONI DERIVATE

↳ relazioni definite per mezzo di funzioni (create da un'interrogazione)

↳ rappresentazioni diverse per gli stessi dati

Due tipi: • **VISTE MATERIALIZZATE**: relazioni derivate effettivamente memorizzate nel DB.

PRO: subito disponibili per le interrogazioni

CONTRO: - ridondanza dei dati - eccessivamente supportate dal DBMS

- appesantiscono gli aggiornamenti

• **VISTE VIRTUALI**: relazioni definite per mezzo di interrogazioni, non memorizzate nel DB.

↳ le interrogazioni sulle viste vengono eseguite sostituendo alla vista la sua definizione (avendo, componendo le due interrogazioni)

PRO: supportate da tutti i DBMS.

- Ogni utente vede quindi: ciò che gli interessa

ciò che è autorizzato a visualizzare

Strumento di programmazione: possiamo semplificare la sintassi delle interrogazioni

↳ l'utilizzo di viste non influenza sull'efficienza delle interrogazioni.

AGGIORNARE LE VISTE := cambiare la tauta base in modo che la vista aggiornata riferisca l'aggiornamento.

3.2 CALCOLO RELAZIONALE

↳ famiglia di linguaggi di interrogazione, basati sui calcoli dei predicati del primo ordine, che hanno la caratteristica di essere DICHIASTRATIVI

↳ specificano la proprietà del risultato delle interrogazioni, anziché la procedura seguita

Due definizioni: • **CALCOLO RELAZIONALE SU DOMINI**

• **CALCOLO SU TUPLE CON DICHIARAZIONI DI RANGE**

3.2.1 CALCOLO RELAZIONALE SU DOMINI

Le espressioni hanno forma: $\{A_1 : x_1, \dots, A_k : x_k \mid f\}$, dove:

• A_1, \dots, A_k sono attributi distinti (possono non compiere nello schema del DB)

• x_1, \dots, x_k sono variabili (supponiamo distinte)

• f è una formula che utilizza operatori booleani e quantificatori.

La lista di copie $A_1 : x_1, \dots, A_k : x_k$ viene chiamata TARGET LIST, definisce la struttura del risultato.

ESEMPIO:

Impiegati (Matricola, Nome, Età, Stipendio)

Supervisione (Capo, Impiegato)

A.R.: Ø Stipendio > 40 (Impiegati)

C.R.D.: {Matricola: m, Nome: n, Età: e, Stipendio: s |

Impiegati (Matricola: m, Nome: n, Età: e, Stipendio: s)

$\wedge (s > 40)$] predicato x selezione le tuple

costituito dalla relazione su A_1, \dots, A_k che

contiene le tuple i cui valori, sostituiti a

x_1, \dots, x_k , rendono vera la formula rispetto a

un'istanza di DB a cui l'espressione viene applicata

- Return the chiefs' number and name having all employees earning more than 40

$$\begin{aligned} & \Pi_{Number, Name} (\text{EMPLOYEE} \bowtie_{\substack{\text{Number} = \text{Chief} \\ (\text{Chief} \in \text{SUPERVISOR})}} \text{Employee} = \text{Number} \sigma_{Wage > 40} (\text{EMPLOYEE}))) \\ & \{ \text{Number: } c, \text{Name: } n \mid \\ & \text{EMPLOYEE}(\text{Number: } c, \text{Name: } n, \text{Age: } a, \text{Wage: } w) \wedge \\ & \text{SUPERVISOR}(\text{Chief: } c, \text{Employee: } m) \wedge \\ & \neg \exists m' (\exists n' (\exists a' (\exists w' (\text{EMPLOYEE}(\text{Number: } m', \text{Name: } n', \text{Age: } a', \text{Wage: } w') \wedge \\ & w' \leq 40 \wedge \text{SUPERVISOR}(\text{Chief: } c, \text{Employee: } m'))))) \end{aligned}$$

- Return the employees earning more money than their boss; for both such employees and chiefs return the number, name and salary

$$\begin{aligned} & \Pi_{Number, Name, Wage, NumC, NameC, WageC} (\sigma_{Wage > WageC} (\rho_{NumC, NameC, WageC, AgeC \leftarrow \text{Number}, \text{Name}, \text{Wage}, \text{Age}} (\text{EMPLOYEE}))) \\ & \bowtie_{\substack{\text{NumC} = \text{Chief} \\ (\text{SUPERVISOR} \bowtie_{\text{Employee} = \text{Number}} \text{EMPLOYEE}))}} \\ & \{ \text{Number: } m, \text{Name: } n, \text{Wage: } w, \text{NumC: } c, \text{NameC: } nc, \text{WageC: } wc \mid \\ & \text{EMPLOYEE}(\text{Number: } m, \text{Name: } n, \text{Age: } a, \text{Wage: } w) \wedge \\ & \text{SUPERVISOR}(\text{Chief: } c, \text{Employee: } m) \wedge \\ & \text{EMPLOYEE}(\text{Number: } c, \text{Name: } nc, \text{Age: } ac, \text{Wage: } wc) \wedge w > sc \} \end{aligned}$$

RECALL - Leggi di DeMorgan

- $\neg(f \wedge g) = \neg f \vee \neg g$
- $\neg(f \vee g) = \neg f \wedge \neg g$

quantificatori:

- $\neg \forall x A = \exists x \neg A$
- $\forall x A = \neg \exists x \neg A$
- $\neg \exists x A = \forall x \neg A$
- $\exists x A = \neg \forall x \neg A$

+: $\neg A \vee B \rightarrow \text{if } A \text{ then } B$

3.2.2 PROS AND CONS del calcolo su domini

PRO: Dichiarativo

CONS: • "verboso", troppe variabili

• ammette espressioni che non fanno senso. es: $\{A: x \mid \neg R(A: x)\}, \{A: x, B: y \mid R(A: x)\}$

PLUS: DRC e RA sono equivalenti:

per ogni espressione dipendente dal dominio

im DRC, esiste un'espressione equivalente im RA,

e viceversa.

queste espressioni sono dipendenti dal dominio e le

dovremmo evitare + nel A.R. Non le possiamo usare perché

sono dipendenti dal dominio.

3.2.3 CALCOLO SU TUPLE CON DICHIARAZIONI DI RANGE

→ ha forma: $\{T \mid L \mid f\}$, dove:

- T è la TARGET LIST, con elementi del tipo $Y: x_1 \dots x_n$, con x variabile e Y e $x_1 \dots x_n$ sequenze di attributi, gli attributi in Y devono comporre tutto lo schema della relazione che costituisce il RANGE di X.
- L è la RANGE LIST, elenca le variabili libere della formula f con i relativi range. L è una lista di elementi del tipo $x(R)$, con x variabile e R nome di relazione.
- f è una formula con ATOMI, CONNETTIVI, QUANTIFICATORI.

⇒ viene ridotto il numero di variabili

• tutti i valori devono venire dati da,

- Return the chiefs' number and name having all employees earning more than 40

$$\begin{aligned} & \{ \text{Number: } c, \text{Name: } n \mid \\ & \text{EMPLOYEE}(\text{Number: } c, \text{Name: } n, \text{Age: } a, \text{Wage: } w) \wedge \\ & \text{SUPERVISOR}(\text{Chief: } c, \text{Employee: } m) \wedge \\ & \neg \exists m' (\exists n' (\exists a' (\exists w' (\text{EMPLOYEE}(\text{Number: } m', \text{Name: } n', \text{Age: } a', \text{Wage: } w') \wedge \\ & w' \leq 40))) \} \\ & \{ e.(\text{Number}, \text{Name}) \mid s(\text{SUPERVISOR}), e(\text{EMPLOYEE}) \mid \\ & s.\text{Chief} = e.\text{Number} \wedge \neg \exists e'(\text{EMPLOYEE})(\exists s'(\text{SUPERVISOR}) \\ & (s.\text{Chief} = s'.\text{Chief} \wedge s'.\text{Employee} = e'.\text{Number} \wedge e'.\text{Wage} \leq 40)) \} \end{aligned}$$

ES. Matricola, Nome, Età e Stipendio degli impiegati che guadagnano più di 40 mila euro:

$$\{ i * \mid i \mid (\text{Impiegati}) \mid i.\text{Stipendio} > 40 \}$$

restituisce tutti gli attributi

LIMITI:

Il calcolo su tuple con dichiarazioni di range non permette di esprimere le interrogazioni i cui risultati possono provenire indifferentemente da due o più relazioni (come l'operatore di unione).

→ Infatti, essendo SQL basato sul calcolo su tuple con dichiarazioni di range, prevede un costrutto esplicito di UNIONE, mentre l'INTERSEZIONE e la DIFFERENZA li possiamo esprimere in altri modi.

CALCOLO e ALGEBRA

- Sono praticamente EQUIVALENTI.
- Alcune query utili non possono essere espresse:
 - possiamo solo estrarre valori, non possiamo calcolare nuovi.
 - ES: - su ogni tupla: (conversioni, somme, differenze, ...)
 - su un set di tuple: (sommatoria, media, ...)
 - estensioni adottate però da SQL
 - query ricorsive, come la chiusura transitiva (TRANSITIVE CLOSURE)

CHIUSURA TRANSITIVA

La chiusura transitiva R^+ di una relazione binaria R su un set X è la più piccola relazione su X che contiene R ed è transitiva. Data una relazione R su $A \times A$, la chiusura transitiva è una relazione R^+ tale che

$$R^+ = \{ \langle x, y \rangle \mid \exists y_1, \dots, y_m \in A, m \geq 2, y_1 = x, y_m = y, \langle x, y_i \rangle, \langle y_i, y_{i+1} \rangle \in R, i=1, \dots, m-1 \}$$

Se X è un set di aeroporti $x \in X$ significa "c'è un volo diretto da aeroporto x a aeroporto y ", la chiusura transitiva $x \in R^+ y$ significa "è possibile volare da x a y in uno o più voli"

- We could use both joins with renaming in order to express such relations

- But:

Employee	Chief	Employee	Superior
Rossi	Lupi	Rossi	Lupi
Neri	Bruni	Neri	Bruni
Lupi	Falchi	Lupi	Falchi
Falchi	Leoni	Falchi	Leoni

Nell'algebra relazionale non possiamo esprimere la chiusura transitiva per ogni relazione binaria, ma si darebbe ricreare ogni volta un'espressione diversa.

128

3.3 DATALOG

L'idea su cui si basa DATALOG è quella di adattare alle basi di dati il linguaggio di programmazione logica PROLOG.

Si basa sul calcolo dei predicati del primo ordine, possiamo trovare due tipi di predicati:

- i predicati **ESTENSIONALI**, corrispondono alle relazioni nella base di dati.
- i predicati **INTENSIONALI**, sono specificati per mezzo di regole logiche. Questi predicati degli insieme viste (relazioni virtuali) sulla base di dati.

Le REGOLE DATALOG hanno la forma: **testa ← corpo**

• la TESTA è un predicato atomico

Sono imposte delle CONDIZIONI:

• il CORPO è una lista di condizioni atomiche.

- i predicati estensioniali possono comparire solo nel corpo delle regole
- se una variabile compare nella testa di una regola, allora deve comparire anche nel corpo della stessa regola.
- se una variabile compare in un atomo di confronto, allora deve comparire anche in un atomo nel corpo della stessa regola.

- Return the employees' number, name, age and salary being 30 years old

{ Number: m , Name: n , Age: a , Wage: w }
EMPLOYEE(Number: m , Name: n , Age: a , Wage: w) $\wedge a = 30$

DRC
DATALOG
132

- Return the chiefs' number and name having all employees earning more than 40

- We need negation

CHIEFSOFNORICHERS(Chief: c) ←
EMPLOYEE(Number: m , Name: n , Age: a , Wage: w),
 $w \leq 40$, SUPERVISOR(Chef: c , Employee: m)
CHIEFSOONLYRICHER(Number: c , Name: n) ←
EMPLOYEE(Number: c , Name: n , Age: a , Wage: w),
SUPERVISOR(Chef: c , Employee: m),
NOT CHIEFSOFNORICHERS(Chef: c)
? CHIEFSOONLYRICHER(Number: c , Name: n)¹³⁶

La definizione di query ricorsive diventa complicata nel caso della negazione:

- datalog mon ricorsivo senza negazione equivale al calcolo senza negazione e senza quantificatore universale.
- datalog mon ricorsivo con negazione equivale al calcolo e all'algebra.
- non possiamo comparare datalog ricorsivo senza negazione e calcolo.
- datalog ricorsivo con negazione è più espressivo del calcolo e dell'algebra.

che cosa significa?

SQL

DATA DEFINITION

- CREATE DATABASE db_name : ogni database creato contiene tabelle, viste, trigger, ...
- CREATE SCHEMA db_schema: uno schema in SQL è identificato da un nome e descrive gli elementi che gli appartengono, come tabelle, tipi, vincoli, viste, domini... può essere seguito dalla keyword AUTHORIZATION per indicare il nome dell'utente proprietario dello schema.

```
CREATE SCHEMA schema_name
AUTHORIZATION 'user_name'
```
- CREATE TABLE table_name : ogni tabella viene definita associandole un nome ed elencando gli attributi che ne compongono lo schema. Per ogni attributo si definiscono un nome, un dominio ed eventualmente un insieme di vincoli che devono essere rispettati dai valori dell'attributo.

BASIC DATA TYPES

- Domino character permette di rappresentare singoli caratteri o stringhe. La lunghezza può essere fissa o variabile.
- Tipi numerici, famiglia che contiene i domini che permettono di rappresentare valori esatti, interi o con una parte decimale.
- Istanti temporali : - date (year, month, day)
- time (hour, minute, second)
- interval, per rappresentare intervalli di tempo
- SQL-3 : - boolean, per rappresentare singoli valori booleani
- BLOB e CLOB, permettono di rappresentare oggetti di grandi dimensioni, costituiti da una sequenza arbitraria di valori binari (BLOB, binary large object) o di caratteri (CLOB, character large object). È possibile memorizzare questi valori ma non possono essere utilizzati come criterio di selezione nelle query.

CUSTOM DATA TYPES

Ogni tipo di dati standard può essere utilizzato per definire nuove relazioni, indicando vincoli e valori di default

Es. CREATE DOMAIN Grade
AS SMALLINT DEFAULT NULL
CHECK (value >= 18 AND value <= 30)

VINCOLI INTRARELAZIONALI

Proprietà che devono essere verificate da ogni istanza della base di dati.

- NOT NULL**: indica che il valore null non è ammesso come valore dell'attributo. Nel caso in cui all'attributo è associato un valore di default (diverso da null), allora è possibile fare l'inserimento anche senza fornire un valore per l'attributo.

Cognome varchar(20) not null

- UNIQUE**: si applica ad un attributo, o ad un insieme di attributi, e impone che i valori dell'attributo siano una (super)chiave.

Esezione per il valore nullo → si assume che i valori null siano tutti diversi tra loro.

Matricola character(6) unique

Nome varchar(20) not null,
Cognome varchar(20) not null,
unique (Cognome, Nome)

- PRIMARY KEY**, è possibile specificare la chiave primaria una sola volta per tabella, può essere definita su un singolo attributo o su un insieme di attributi. Gli attributi che ne fanno parte non possono assumere valore null.

Nome varchar(20),
Cognome varchar(20),
primary key (Cognome, Nome)

Es.

```
CREATE TABLE EMPLOYEE (
    Number CHARACTER(6) PRIMARY KEY,
    Name CHARACTER(20) NOT NULL,
    Surname CHARACTER(20) NOT NULL,
    Dept CHARACTER(15),
    Wage NUMERIC(9) DEFAULT 0,
    FOREIGN KEY(Dept) REFERENCES
        DEPARTMENT(Dept),
    UNIQUE (Surname, Name)
)
```

WARNING: Name CHARACTER(20) NOT NULL,
Surname CHARACTER(20) NOT NULL,
UNIQUE (Surname, Name)
!=
Name CHARACTER(20) NOT NULL UNIQUE,
Surname CHARACTER(20) NOT NULL UNIQUE,

VINCOLI INTERRELAZIONALI

Questo vincolo crea un legame tra i valori di un attributo della tabella su cui è definito (interna) e i valori di un attributo di un'altra tabella (esterna). L'unico requisito è che l'attributo cui si fa riferimento sia oggetto al vincolo **unique**.

Può essere definito in due modi:

1. Su un solo attributo: utilizziamo **REFERENCES**, dove si specifica la tabella esterna e l'attributo della tabella esterna al quale l'attributo in questione deve essere legato.
2. Su un insieme di attributi: utilizziamo **FOREIGN KEY**, dove vengono elencati gli attributi della tabella coinvolti, cui segue la definizione dei corrispondenti attributi della tabella esterna con **REFERENCES**.

```
CREATE TABLE OFFENCES (
    Code CHARACTER(6) PRIMARY KEY,
    Day DATE NOT NULL,
    Officer INTEGER NOT NULL REFERENCES OFFICER(Id),
    State CHARACTER(2),
    Number CHARACTER(6),
    FOREIGN KEY(State, Number) REFERENCES CAR(State, Number)
)
```

Dopo ogni vincolo referenziale, possiamo specificare l'azione da invocare se l'operazione è respinta:

- per le op. di modifica:

- **CASCADE**, il nuovo valore della t. esterna viene riportato sulle righe corrispondenti della t. interna.

- **SET NULL**, all'attributo riferente viene assegnato il valore nullo al posto di quello modificato nella t. esterna.

- **SET DEFAULT**, all'attributo riferente viene assegnato il valore di default al posto di quello modificato nella t. esterna.

- NO ACTION, l'azione di modifica non viene consentita

- violazioni prodotte dall'eliminazione:

- CASCADE: tutte le righe della t. intorno corrispondenti alla riga cancellata vengono cancellate

- SET NULL: dell'attributo riferente viene assegnato il valore nullo al posto del valore cancellato nella tabella esterna

- SET DEFAULT: dell'attributo riferente viene assegnato il valore di default al posto del valore cancellato nella tabella esterna

- NO ACTION: la cancellazione non viene consentita

MODIFICA DEGLI SCHEMI

- **ALTER** : permette di modificare i domini e schemi di tabella.

- **ALTER DOMAIN** : permette di modificare domini già definiti. Viene utilizzato con:

- SET DEFAULT
- DROP DEFAULT
- ADD CONSTRAINT
- DROP CONSTRAINT

- **ALTER TABLE** : permette di modificare tabella già definite. Viene utilizzato con:

- ALTER COLUMN
- ADD COLUMN
- DROP COLUMN
- DROP CONSTRAINT
- ADD CONSTRAINT

- **DROP DOMAIN** : rimuove un tipo di dati definito dall'utente

- **DROP TABLE** : rimuove un'intera istanza di tabella con il suo schema e i suoi dati

DEFINIZIONE INDICI

- **CREATE INDEX** utilizzato per creare un indice su una o più colonne di una tabella. Migliora la velocità delle operazioni di lettura,

- ma rallenta le op. di scrittura. (simile all'indice di un libro)

- CREATE INDEX idx_surname
ON OFFICER (Surname)

OPERAZIONI SUI DATI

- Query: **SELECT**

- Edit: **INSERT, DELETE, UPDATE**

- **SELECT <AttributeList> //target post**
FROM <TableList> //statement
[**WHERE <Condition>**] //statement

- Data una relazione R(A,B)
SELECT *
FROM R
-> (*) seleziona tutti gli attributi di R

- **SELECT ***
FROM PEOPLE
WHERE Name **LIKE 'J_m'**

Ritorna le persone che hanno il nome che inizia con la 'J' e che hanno come terza lettera 'm'
-> '_' sostituisce un singolo carattere
-> '%' sostituisce zero o più caratteri

- **SELECT DISTINCT <AttributeList>**
FROM <Table>
-> DISTINCT viene utilizzato per restituire solo valori unici in una query eliminando i duplicati

- **R1(A1, A2), R2(A3, A4)**
SELECT DISTINCT R1.A1, R2.A4
FROM R1, R2
WHERE R1.A2 = R2.A3
-> (FROM) prodotto cartesiano
-> (WHERE) selezione
-> (SELECT) proiezione

- **Alias & Ridenominazione**
SELECT X.A1 AS B1...
FROM R1 AS X, R2 AS Y,...
WHERE X.A2 = Y.A3 AND...

VALUTAZIONE DELLE QUERIES

SQL è un linguaggio dichiarativo, significa che indichiamo cosa vuoi ottenere, senza specificare come il DB debba eseguire il lavoro. I DBMS fanno piani di esecuzione per eseguire le query in modo efficiente:

- le selezioni vengono eseguite il prima possibile per ridurre il numero di righe da processare
(WHERE)
- dove possibile, le join vengono eseguite invece dei prodotti cartesiani, in quanto le join sono più efficienti perché combinano solo le righe che formano una corrispondenza.

Infatti non dobbiamo scrivere necessariamente query efficienti poiché è il DBMS ad ottimizzarle, pertanto è più importante che le query siano facili da capire.

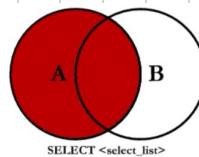
JOIN STATEMENT

- **INNER JOIN**: esplicito : `SELECT` , implicito: `SELECT`
`FROM R JOIN R2` `FROM R, R2`
`ON R.A = R2.B` `WHERE R.A, R2.B`

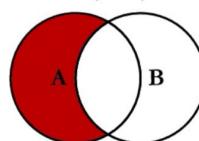
- **NATURAL JOIN**: si usa al posto del JOIN se due tabelle hanno colonne con lo stesso nome

- **OUTER JOIN**: per evitare la perdita di informazioni quando una tupla non trova il match

- a) **LEFT JOIN**: torna tutte le tuple della tabella a sinistra, quelle senza un match con la tabella a destra sono riempiti con valori NULL
- b) **RIGHT JOIN**: torna tutte le tuple della tabella a destra, quelle senza un match con la tabella a sinistra sono riempiti con valori NULL
- c) **FULL JOIN**: torna tutte le tuple sia della prima che della seconda tabella, dove non c'è un match troviamo valori NULL.



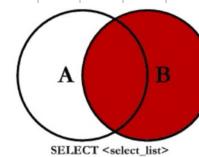
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



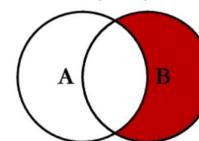
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```



Actually

© C.I. Moffatt 2008

ORDINAMENTO

→ ORDER BY AttrDiOrdinamento [asc | desc]

permette di specificare un ordinamento delle righe del risultato di una query

ASC → ascending order (default)

DESC → descending order

UNION, INTERSECTION, DIFFERENCE

• UNION : SELECT ...

UNION [ALL] → si definisce tutte le righe sono uniche, tranne quando viene utilizzato ALL (multi-set union)

SELECT ...

quando due tabeles hanno schemi diversi si assumono i nomi degli attributi del primo operando

• DIFFERENCE : SELECT ... → è possibile esprimere la differenza anche attraverso query annidate

FROM ... • INTERSECTION SELECT ...

EXCEPT

FROM ...

SELECT R.A

SELECT ...

INTERSECT

FROM R, R1

FROM ...

SELECT ...

WHERE R.A=R1.B

FROM ...

QUERY ANNIDATE

Possono essere formulate utilizzando i predicati ANY o ALL con gli operatori ($>$, $<$, $=$, ...)

Attribute op ANY / ALL (Expr)

• IN : la tupla della query esterna e' un match se il suo valore e' contenuto tra gli elementi ritornati da Expr

Attribute IN (Expr)

ANY, ALL, IN possono essere negati con IN

→ A IN (Expr) \equiv A = ANY (Expr)

A NOT IN (Expr) \equiv A \neq ALL (Expr)

■ VISIBILITÀ :

Non e' possibile fare riferimento a variabili dichiarate all'interno dei blocchi interni. Se il nome di una variabile e' omesso, prendiamo far dichiarazione più "vicina". Possiamo fare riferimento a variabili definite:

• mello scope della query in cui e' definita (blocchi esterni)

• mello scope di una query annidata, a qualsiasi livello (blocco interno) all'interno di esso.

La query interna viene eseguita una volta per ogni tupla dell'interno della query esterna, l'unico modo per evitare e' creare una vista, che va però a modificare lo schema del database.

EXISTENTIAL QUANTIFICATION

usually query annidata

EXISTS (Expr) Il predicato e' vero se Expr torna almeno una tupla

Advanced SQL

Nested Queries Positions

- *WHERE*: uso standard
- *FROM*: necessita di una nuova fonte di dati, l'alternativa è creare una vista che va però a modificare lo schema del database.
- *SELECT*: è equivalente ad un JOIN. Richiede necessariamente una tupla come risultato.

Provide the name and the income of Jim's children

```
SELECT Name, Income  
FROM PEOPLE P, (SELECT Child  
                  FROM FATHERHOOD  
                  WHERE Father='Jim') AS  
                  JIMCHILD  
WHERE Name = JIMCHILD.Child
```

Provide the total shipping charges for each customer in the customer table

```
SELECT CUSTOMER.Num,  
      (SELECT SUM(ShipCharge)  
       FROM ORDERS  
       WHERE CUSTOMER.Num=ORDERS.Num)  
      AS TotalShipCharge  
FROM CUSTOMER
```

Funzioni di aggregazione

Nella target list possiamo mettere espressioni che calcolano i valori da un insieme di tuple attraverso funzioni aggregate:

- COUNT
- MIN
- MAX

- AVG
- SUM

UPDATING OPERATIONS:

- INSERT
- DELETE
- UPDATE

Vincoli di integrità generici

- CHECK (Predicate): le condizioni ammesse sono le stesse della clausola WHERE . La condizione deve essere sempre verificata affinchè la base di dati sia corretta, in questo modo è possibile specificare tutti i vincoli intrarelazionali.
- ASSERTION : vincoli associati allo schema (no tabella / attributo). E' possibile esprimere tutti i vincoli che coinvolgono più tabelle o che richiedono che una tabella abbia cardinalità minima. Possiedono un nome con il quale possono essere eliminate esplicitamente dallo schema con drop .

```
create assertion <name>
check (predicate)
```

I vincoli possono essere:

- *immediati*: sono verificati subito dopo ogni modifica
- *differiti*: sono verificati al termine dell'esecuzione di una serie di operazioni

Viste

Sono tavole "virtuali" il cui contenuto dipende dalle altre tavole. Vengono definite con un nome e una lista di attributi al risultato dell'esecuzione

```
create view NomeVista [(ListaAttributi)] as SelectSQL
[with [local | cascaded] check option]
```

```
create view ADMINEMPLOYEES
  (Name, Surname, Salary) as
  select Name, Surname, Salary
  from EMPLOYEE
  where Dept = 'Administration' and
        Salary > 10
```

Alcuni sistemi considerano una vista aggiornabile solo se è definita su una sola tabella, altri richiedono anche che l'insieme di attributi della vista contenga almeno una chiave primaria della tabella base.

- La clausola check option può essere usata solo in questa categoria di viste, specifica che gli aggiornamenti sono ammessi solo sulle righe della vista e che dopo ogni modifica tutte le righe devono continuare ad appartenere alla vista. Permette di aggiornare la vista, solo se la tupla inserita appartiene alla vista (rispetta le condizioni);

- Nel caso in cui una vista è definita in termini di altre viste:
 - local : l'aggiornamento della tupla deve essere eseguito solo all'ultimo livello della vista;
 - cascade : l'aggiornamento della tupla deve essere propagato a tutti i livelli di definizione.

Querying a View

Le viste possono essere utilizzate per formulare interrogazioni che non sarebbero esprimibili altrimenti es.

Stipendi totali di ogni dipartimento:

```
create view BudgetStipendi(Dip,TotaleStipendi) as
  select Dipart, sum(Stipendio)
  from Impiegato
  group by Dipart
```

Dipartimento con lo stipendio massimo:

```
select Dip
from BudgetStipendi
where TotaleStipendi = (select max(TotaleStipendi)
                           from BudgetStipendi)
```

Query scorretta, la sintassi di SQL non permette l'utilizzo di funzioni di aggregazione nidificate:

```
select avg(count(distinct Office))
from EMPLOYEE
group by Dept
```

Possiamo utilizzare una vista per ottenere il numero medio di uffici per dipartimento:

```
create view DEPTOFFICES(NameDept,OffNum) as
  select Dept, count(distinct Office)
  from EMPLOYEE
  group by Dept;

select avg(OffNum)
from DEPTOFFICES
```

Query Ricorsive

Esempio: per ogni persona restituire il suo antenato:

FATHERHOOD(Father, Child)

```
with recursive ANCESTORS(Anccestor,Descendant) as
( select Father, Son
  from FATHERHOOD
 union all
  select Anccestor, Son
  from ANCESTORS, FATHERHOOD
  where Descendant = Father
 )
```

```
select *
from ANCESTORS
```

with definisce la vista ANCESTORS, costruita ricorsivamente utilizzando *FATHERHOOD*.

Funzioni Scalari

Funzioni a livello di tuple che forniscono un unico valore per tuple.

Temporali:

- `current_date()` restituisce la data attuale
- `extract(yearExpression)` estraе parte di una data da una determinata espressione

Manipolazione delle stringhe:

- `char_length` ritorna la lunghezza della stringa
- `lower` converte la stringa in minuscolo
- `upper` converte la stringa in maiuscolo
- `substring` restituisce una parte della stringa identificata da indici

Conversione di dominio:

- `cast` permette di convertire un valore in un dominio nella sua rappresentazione in un altro dominio (es. `Data as char(10)`)

Funzioni condizionali:

- `coalesce` prende una sequenza di espressioni e restituisce il primo valore non nullo. Può essere utilizzato anche per convertire i valori nulli in valori definiti dal programmatore.

es 1. Per ogni impiegato tornare un numero di cellulare valido o il suo numero di telefono:

```SQL

```
select number, coalesce(Mobile, PhoneHome)
from EMPLOYEE
```

es 2. per ogni dipartimento il valore NULL è sostituito da 'none':

```SQL

```
select Name, Surname,
       coalesce(Dept, 'None')
from EMPLOYEE
```

- `nullif` prende come argomento un'espressione e un valore costante; se l'espressione è pari al valore costante la funzione restituisce il valore nullo, altrimenti restituisce il valore dell'espressione.
- `case` permette di specificare strutture condizionali cui risultato dipende dalla valutazione del contenuto delle tabelle.

es: calcolare le tasse di circolazione dei veicoli immatricolati dopo il 1975, in base ad un tariffario secondo il tipo di veicolo

Veicolo (*Targa, Tipo, Anno, KWatt, Lunghezza, NAssi*)

```
select Targa,
       (case Tipo
            when 'Auto' then 2.58 * KWatt
            when 'Moto' then (22.00 + 1.00 *KWatt)
            else null
        end) as Tassa
```

```
from Veicolo  
where Anno > 1975
```

Database Security

SQL consente di definire per ogni utente specifico a quali risorse possono avere accesso, questo sistema si basa su un concetto di privilegio, esso è caratterizzato da:

1. la risorsa a cui si riferisce
2. l'utente che concede il privilegio
3. l'utente che riceve il privilegio
4. l'azione che viene permessa sulla risorsa
5. se il privilegio può essere trasmesso o meno ad altri utenti

Quando viene creata una risorsa il sistema concede automaticamente tutti i privilegi al creatore di essa, in più esiste un utente predefinito `_system` che possiede tutti i privilegi.

I privilegi disponibili sono:

- `insert` : permette di inserire un nuovo oggetto
- `update` : permette di aggiornare il valore di un oggetto
- `delete` : permette di rimuovere oggetti
- `select` : permette di leggere la risorsa e utilizzarla in un interrogazione
- `references` : permette di definire vincoli d'integrità referenziale
- `usage` : permette che venga usata la risorsa

I privilegi sull'utilizzo di `drop` e `alter` non possono essere concessi.

I privilegi vengono concessi o revocati tramite le istruzioni `grant` e `revoke`.

Granting Privileges

Sintassi: `grant Privilegi on Risorsa to Utenti [with grant option]`

Permette di concedere i *Privilegi* sulla *Risorsa* agli *Utenti*.

`grant option` permette all'utente di propagare i suoi privilegi ad altri utenti.

Revoking Privileges

Sintassi: `revoke Privilegi on Risorsa from Utenti [restrict | cascade]`

I privilegi possono essere rimossi solo da chi li aveva concessi.

- `restrict` (default), il comando non viene eseguito se la revoca dei privilegi all'utente comporta altre revocate di privilegi.
- `cascade` forza l'esecuzione di `revoke`, tutti i privilegi propagati vengono revocati e tutti gli elementi costruiti sulla base di quei privilegi vengono rimossi.

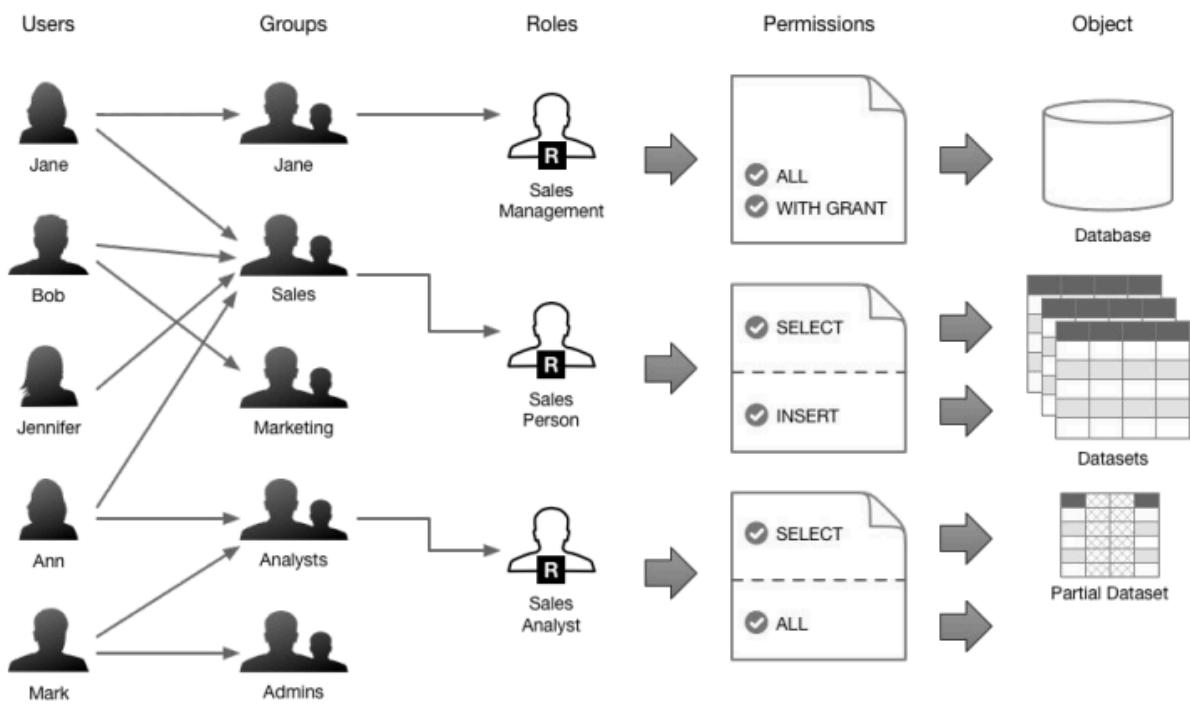
Role-Based Access Control

RBAC è un modello di controllo dell'accesso basato sui *ruoli*. Un ruolo:

- viene creato con `create role NomeRuolo ;`
- si comporta come un contenitore di privilegi, che gli vengono attribuiti con il comando `grant`, lo stesso comando viene utilizzato anche per concedere agli utenti la possibilità di ricoprire un certo ruolo.

Per utilizzare i privilegi l'utente deve invocare il comando `set role NomeRuolo`. In ogni momento l'utente dispone dei privilegi che gli sono stati attribuiti direttamente e dei privilegi associati al ruolo.

che è stato esplicitamente attivato.



Transazioni

Una **transazione** è una sequenza di operazioni che vengono eseguite come un'unità logica ed ha come obiettivo quello di garantire l'integrità dei dati anche in presenza di errori o guasti.

Rispettano quattro proprietà fondamentali (*ACID*):

- **Atomicità (Atomicity)**: una transazione è un'unità *indivisibile* di esecuzione → o vengono resi visibili tutti gli effetti di una transazione, oppure la transazione non deve avere alcun effetto sulla base di dati;
- **Consistenza (Consistency preservation)**: l'esecuzione della transazione non deve violare i vincoli di integrità definiti sulla base di dati. Quando il sistema rileva una violazione interviene per annullare la transazione o per correggere la violazione del vincolo.
 - vincolo di integrità di tipo *immediato*: la verifica può essere fatta nel corso della transazione.
 - vincolo di integrità di tipo *differito*: la verifica deve essere effettuata alla conclusione della transazione. In caso di violazione gli effetti della transazione vengono annullati.
- **Isolamento (Isolation)**: l'esecuzione di una transazione deve essere indipendente dalla contemporanea esecuzione di altre transazioni;
- **Persistenza (Durability)**: l'effetto di una transazione che ha eseguito il commit correttamente deve essere persistente e non deve essere perso

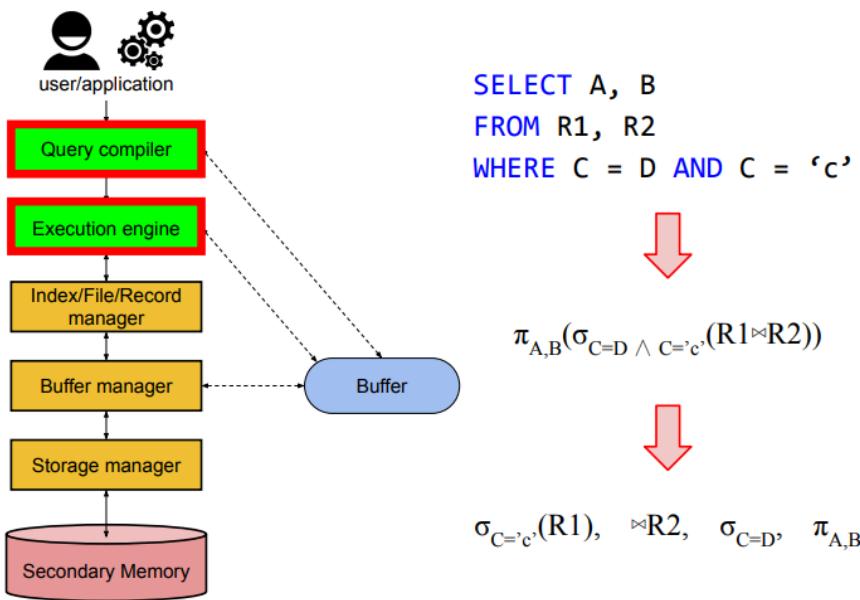
DBMS Architecture

Un database management system (DBMS) è un software per creare, gestire e manipolare grandi quantità di dati in modo efficiente e persistente. L'architettura di un DBMS è composta da:

- **Query Processor:**

è il componente che interpreta ed esegue la query SQL. Si compone di:

- *parsing*: verifica la sintassi della query e costruisce un albero sintattico
- *preprocessing*: verifica la semantica (es. se le tabelle e colonne esistono) e traduce la query in un albero di operazioni algebriche
- *ottimizzazione*: trova la sequenza più efficiente di operazioni per eseguire la query (es. l'ordine delle tabelle di un join)
- **Execution Engine**: esegue il piano ottimizzato interagendo con altri componenti, come la memoria e i moduli di gestione dei file.



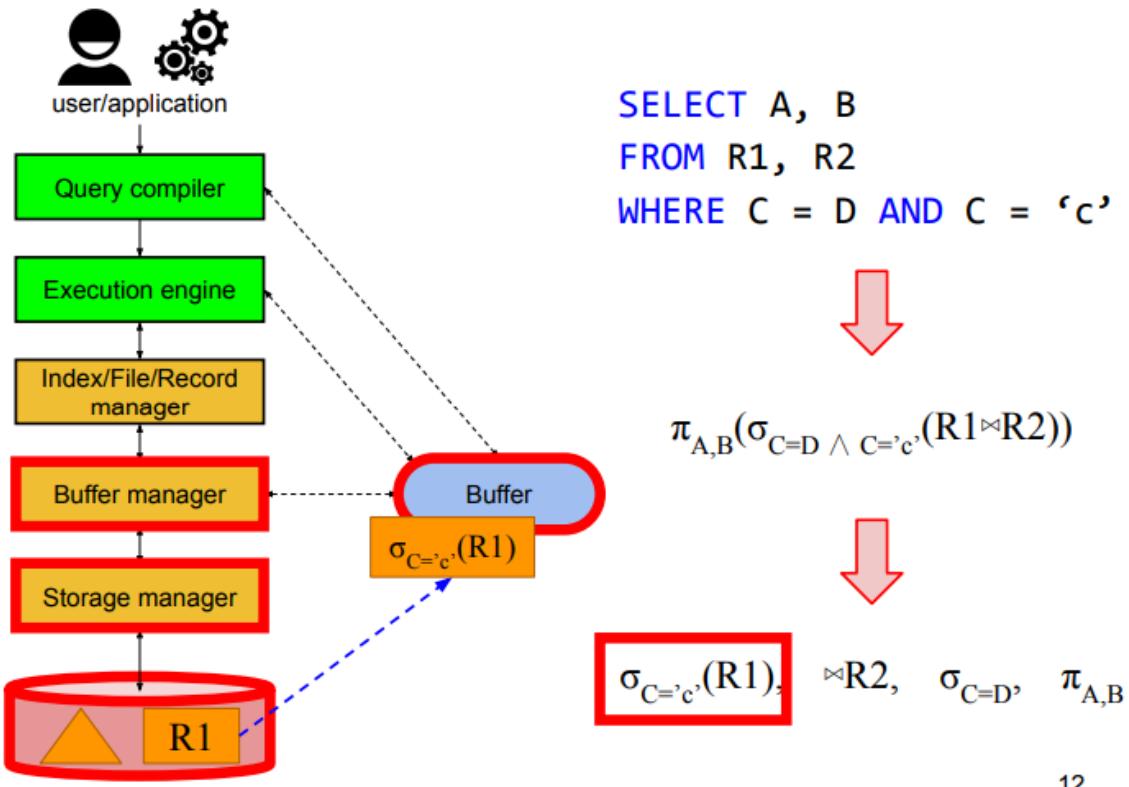
8

- **Resource Manager:**

Si occupa di gestire i file a livello fisico. E' responsabile della persistenza dei dati e dell'efficienza dell'accesso. Include:

- *Index/File/Record Manager*: conosce la struttura dei dati e utilizza indici per velocizzare le ricerche;
- *Buffer Manager*: carica i blocchi di dati dalla memoria secondaria alla memoria principale;

- **Storage Manager:** gestisce i file su disco, localizzandoli e leggendo i blocchi richiesti.



12

- **Transaction Manager:**

si occupa delle transazioni, unità logiche di lavoro nel database spesso costituite da più query SQL. Esempio: transazione di trasferimento bancario:

```
START TRANSACTION;
UPDATE BANKACCOUNT SET Balance = Balance - 500 WHERE AccountNumber = 42177;
UPDATE BANKACCOUNT SET Balance = Balance + 500 WHERE AccountNumber = 12202;
COMMIT;
```

Concorrenza e Deadlock

In un sistema di database, più transazioni possono essere eseguite contemporaneamente, ma potremmo incontrare problematiche come:

- interferenza tra transazioni (es. una transazione modifica un dato mentre un'altra lo legge).
- inconsistenza: quando l'accesso simultaneo ai dati causa risultati non coerenti.

Il **Concurrency-Control Manager** utilizza tecniche per garantire che le transazioni parallele non interferiscano negativamente tra loro:

1. Locking (blocco):

- un lock è un meccanismo che impedisce ad altre transazioni di accedere a un dato finché non viene rilasciato
- due tipi principali:
 - *shared lock* (lettura): più transazioni possono leggere lo stesso dato
 - *exclusive lock* (scrittura): solo una transazione può modificare il dato

2. *schedulazione*: l'ordine di esecuzione delle transazioni viene determinato per garantire che i risultati siano equivalenti a una loro esecuzione sequenziale (serializzabilità).

Un **deadlock** si verifica quando due o più transazioni sono bloccate in attesa di risorse detenute dall'altra. Esempio:

- Transazione T1 blocca il record X e vuole accedere al record Y.
- Transazione T2 blocca il record Y e vuole accedere al record X. Nessuna delle due può procedere. Il DBMS utilizza diverse strategie per risolvere questo tipo di conflitti:

1. **timeout**: una transazione viene terminata automaticamente se resta bloccata troppo a lungo.

2. **detection e rollback**

- il sistema rileva i deadlock costruendo un grafo di attesa.
- identificato il deadlock, una transazione viene abortita (rollback) per liberare le risorse.

Logging and Recovery

Logging: Il *Log Manager* registra ogni modifica effettuata nel database, creando un registro che può essere utilizzato per recuperare il sistema in caso di guasto.

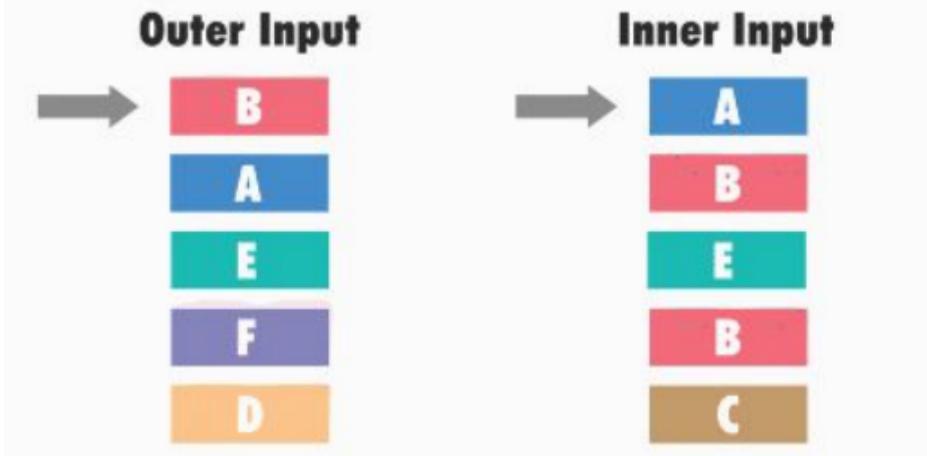
Recovery: Quando il sistema subisce un guasto, il *Recovery Manager* utilizza il log per riportare il database a uno stato consistente.

Tecniche di JOIN

Il join è un'operazione computazionalmente costosa, quindi il DBMS usa diverse tecniche per ottimizzarla.

Nested-loop JOIN

- più semplice, ma meno efficiente.
 - per ogni riga della tabella R (outer), vengono scansionate tutte le righe della tabella S (inner)
- PRO: non richiede precondizioni. CONTRO: estremamente lento per tabelle grandi.
(es. R ha righe 1000 righe e S ne ha 500, vengono effettuate $1000 * 500 = 500000$ confronti)



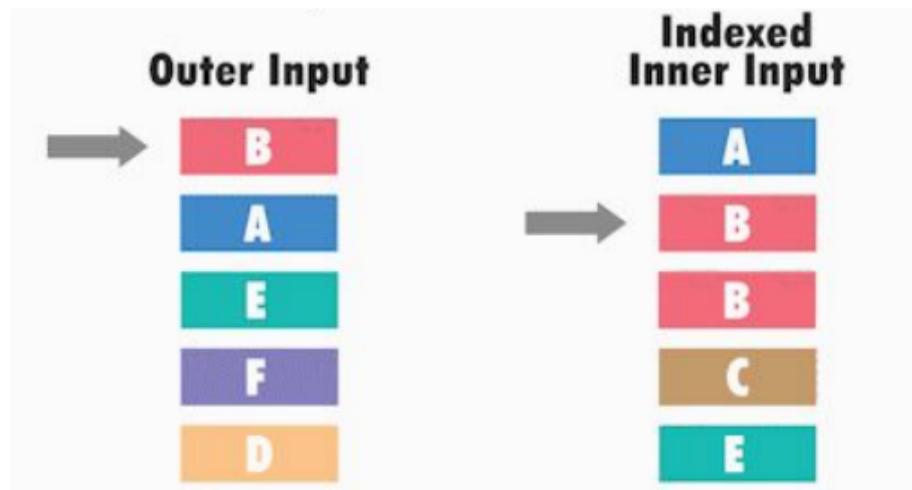
Single-loop JOIN

- usa un indice (o struttura hash) su una delle due tabelle.
- per ogni riga di una tabella (es. S), l'indice viene utilizzato per cercare direttamente le righe corrispondenti nell'altra tabella (es. R).

Se esiste un indice sull'attributo di join di R, l'accesso a ogni riga è diretto, riducendo il numero totale di confronti.

PRO: più efficiente rispetto al nested-loop se gli indici esistono.

CONTRO: richiede indici preesistenti.

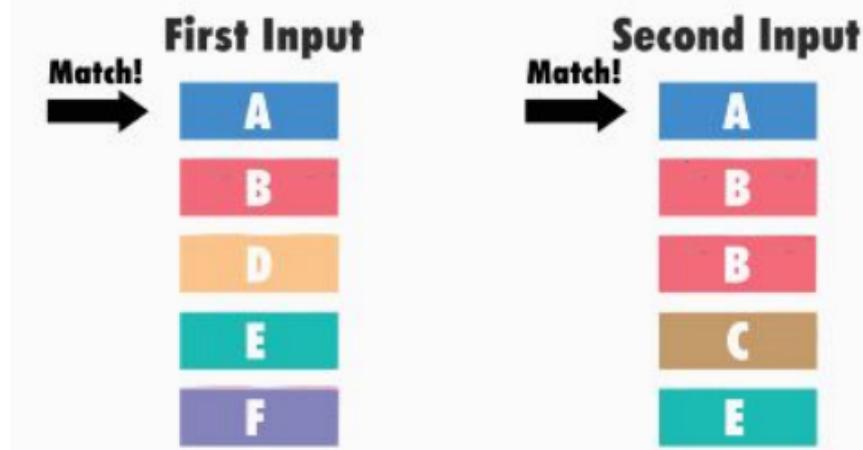


Sort-merge JOIN

- ordina entrambe le tabelle sugli attributi di JOIN.
- scansiona le tabelle ordinate una sola volta, abbinando le righe con valori uguali.

PRO: ottimo per tabelle grandi, specialmente quando devono essere ordinate.

CONTRO: richiede una fase di ordinamento iniziale se le tabelle non sono già ordinate.



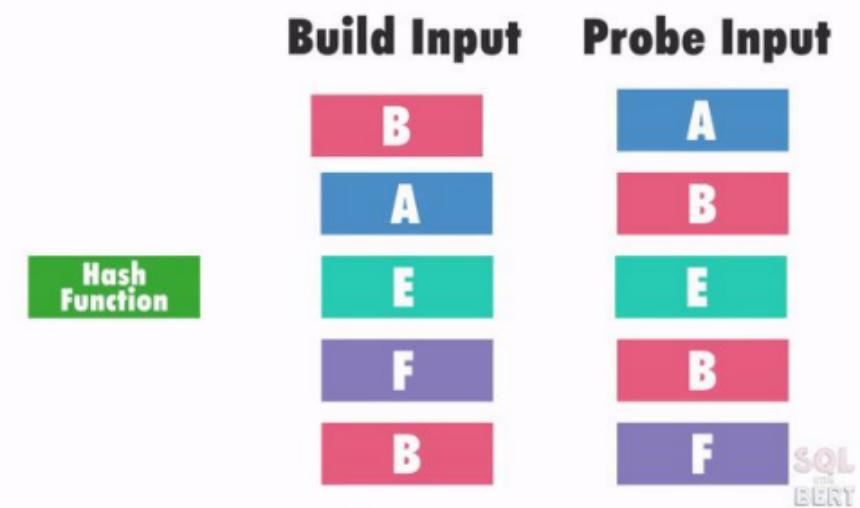
Hash-based JOIN

- usa una funzione di hashing per suddividere entrambe le tabelle in *bucket*.
- le righe nei bucket corrispondenti vengono abbinate.

Esempio: con una funzione hash che divide una tabella in 10 bucket, ogni bucket viene processato separatamente, riducendo il numero di confronti.

PRO: efficiente per tabelle grandi

CONTRO: richiede memoria sufficiente per gestire i bucket.



Transactions

Un DBMS è un ambiente multiutente in cui vari programmi interagiscono con i dati. Include componenti quali:

- *transaction manager*
 - *concurrency control*
 - *logging and recovery*
- e altri moduli per gestire memoria, esecuzione di query e sicurezza dei dati.

Un programma è una sequenza di:

read(X): legge un elemento del database chiamato X;

write(X): scrive un valore nell'elemento del database.

Una **transazione** rappresenta l'esecuzione del programma di un utente come unità indivisibile (insieme di operazioni di lettura e scrittura).

Problemi principali:

- esecuzione concorrente: più transazioni possono essere eseguite simultaneamente motivazioni:
 - gli accessi al disco sono lenti; la CPU non dovrebbe rimanere inattiva.
 - Per migliorare le prestazioni, più transazioni possono essere eseguite in modo concorrente, mantenendo la CPU occupata.
 - **Problema:** l'utente non deve percepire l'interleaving delle transazioni, cioè l'esecuzione simultanea di più operazioni.
- **Recupero da crash:** il database deve rimanere consistente anche in caso di errori o guasti.
 - **Problema:** il DBMS deve garantire che altre transazioni non siano influenzate dall'errore e deve lasciare il database in uno stato consistente.

Proprietà ACID

Per garantire l'esecuzione sicura e simultanea delle operazioni, ogni transazione deve rispettare le proprietà ACID:

Atomicità

- Una transazione deve essere completata integralmente o non avere alcun effetto sul database.
- **Gestione dell'abort:**
 - Cause di abort:
 1. Anomalie interne all'esecuzione (bloccate dal DBMS).
 2. Condizioni di eccezione rilevate dalla transazione (autosospensione).
 3. Crash di sistema (es. errori hardware, software o di rete).
 - In caso di abort, il DBMS:
 - Utilizza il file di log per annullare la transazione (rollback) e ripristinare lo stato consistente precedente.

Consistenza

- La transazione inizia e termina rispettando tutti i vincoli di integrità del database.

Isolamento

- Ogni transazione deve essere eseguita come se fosse l'unica attiva, indipendentemente dall'interleaving con altre transazioni.

Durabilità

- Una volta che una transazione è confermata (commit), le modifiche apportate diventano permanenti, anche in caso di crash.

Una transazione:

- parte sempre in uno stato consistente dove tutti i vincoli referenziali sono soddisfatti;
- potrebbe passare attraverso uno stato intermedio di inconsistenza;
- termina sempre in uno stato consistente dove tutti i vincoli referenziali sono soddisfatti.

Schedulazione delle Transazioni

- **Definizione:** Una schedule rappresenta la sequenza cronologica di operazioni (read, write, commit, abort) eseguite da più transazioni.

Esempio

Dati:

- **T1:** read1(A) → write1(A) → read1(C) → write1(C)
- **T2:** read2(B) → write2(B)

Schedule:

```
read1(A) → write1(A) → read2(B) → write2(B) → read1(C) → write1(C)
```

Tipi di schedule:

1. **Completa:** Include commit o abort per ogni transazione.
2. **Seriale:** Tutte le operazioni di una transazione vengono eseguite consecutivamente (una transazione attiva alla volta).
3. **Serializzabile:** Non è necessariamente seriale, ma produce lo stesso risultato di una schedule seriale.

Anomalie nell'Esecuzione Interleaved

Quando più transazioni sono eseguite simultaneamente, possono verificarsi anomalie che lasciano il database in uno stato inconsistente.

Tipi di conflitti:

1. Write-Read Conflict (Dirty Read):

- **T1** aggiorna A ma non esegue il commit.
- **T2** legge A e utilizza un dato potenzialmente errato.

- Se **T1** fa rollback, il dato letto da **T2** è invalido.
- Given the following two transactions T_1 and T_2 and the schedule S:

$T_1: A := A + 100, B := B - 100$

$T_2: A := A * 1.06, B := B * 1.06$

- **Please note:** T_1 writes a value in A that could make the DB inconsistent, and this value is read by T_2
- **Problem:** A is written by T_1 and read by T_2 before T_1 commits

| T_1 | T_2 |
|----------|----------|
| read(A) | |
| write(A) | |
| | read(A) |
| | write(A) |
| | read(B) |
| | write(B) |
| | commit |
| read(B) | |
| write(B) | |
| commit | |

9

2. Read-Write Conflict (Unrepeatable Read):

- **T1** legge $A = 500$.
- **T2** aggiorna A a 600 e conferma.
- **T1** legge di nuovo A e trova 600, ma le letture di **T1** non sono coerenti.
- Given the following two transactions T_1 and T_2 and the schedule S:

$T_1: A := A + 1$

$T_2: A := A - 1$

- **Please note:** if T_1 repeats the reading of A, it will get a different value
- **Problem:** A is written by T_2 before T_1 commits

| T_1 | T_2 |
|----------|----------|
| read(A) | |
| | read(A) |
| | write(A) |
| | commit |
| write(A) | |
| commit | |

3. Write-Write Conflict (Lost Update):

- **T1** aggiorna A a 600.

- **T2** sovrascrive **A** a 450, perdendo l'aggiornamento di **T1**.
 - Given the following two transactions T_1 and T_2 and the schedule S:

T_1 : $A := 1000, B := 1000$

T_2 : $A := 2000, B := 2000$

■ **Please note:** the values of A and B are equal at the end of each of the two transactions

■ **Problem:** A and B have different values at the end of the schedule

| T_1 | T_2 |
|----------|---------------|
| write(A) | |
| | write(A) |
| | write(B) |
| | commit |
| | write(B) |
| | commit |

4. Phantom Anomalies:

- Una transazione legge un insieme di dati (es. una tabella) e successivamente rileva che l'insieme è cambiato (es. inserimento/cancellazione di righe da un'altra transazione).

Aborted Transactions

In linea di principio, tutte le azioni di una transazione aborted devono essere cancellate. Tuttavia non è sempre possibile. Esempio:

| T_1 | T_2 |
|--------------|---------------|
| read(A) | |
| write(A) | |
| | read(A) |
| | write(A) |
| | read(B) |
| | write(B) |
| | commit |
| read(B) | |
| write(B) | |
| abort | |

problema: T2 legge il valore di A che non avrebbe dovuto leggere, a causa di un write-read conflict

soluzione non soddisfacente: T2 dovrebbe anche essere abortito, ma violerebbe la proprietà di durabilità delle transazioni.

Transactions Parameters

- **Access Mode** imposta l'autorizzazione per modificare le tabelle utilizzate nella transazione:
 - **read only**: permette soltanto la lettura del DB, il tentativo di modificare il DB causa un errore
 - **read write**: consente le operazioni di lettura e scrittura
- **Statement Mode**: specifica le azioni da eseguire al termine di una transazione

- **Isolation Level:** specifica come gestire le transazioni che modificano il DB:
 - **READ UNCOMMITTED:** la transazione richiede blocchi per scrivere oggetti ma non blocchi per la lettura. Consente *dirty read*.
 - **READ COMMITTED:** la transazione richiede blocchi per la scrittura e blocchi condivisi per la lettura. Previene *dirty read*.
 - **REPEATABLE READS:** blocca letture e scritture fino al commit.
 - **SERIALIZABLE:** blocca transazioni che potrebbero causare conflitti.

| Level | Dirty read
(Write-Read) | Unrepeatable read
(Read-Write) | Lost update
(Write-Write) | Phantom |
|-------------------------|----------------------------|-----------------------------------|------------------------------|-------------|
| READ UNCOMMITTED | may occur | may occur | may occur | may occur |
| READ COMMITTED | don't occur | may occur | may occur | may occur |
| REPEATABLE READS | don't occur | don't occur | don't occur | may occur |
| SERIALIZABLE | don't occur | don't occur | don't occur | don't occur |

Concurrency Control Approaches

L'interleaving è necessario e preferibile per migliorare le performance, d'altronde non tutte le schedule sono possibili, alcune azioni devono essere riavvolte (roll back).

Diversi modi di controllare la concorrenza:

- **RESTRICTIVE:** (conflict-serializability) ogni transazione esegue due fasi:
 1. *acquisizione*: richiede blocchi
 2. *rilascio*: libera i blocchi dopo il commit

Esempio:

T1 acquisisce un blocco esclusivo su A e lo modifica.
 T2 deve aspettare il rilascio del blocco di A.

Strict Two-phase Locking (Strict 2PL):

- se una transazione vuole leggere/scrivere un oggetto, deve richiedere un accesso esclusivo.
- dopo aver rilasciato un lock, la transazione non può chiederne altre
- quando viene eseguito il commit la transazione rilascia tutti gli accessi esclusivi
- Strict 2PL garantisce la serializzabilità, in particolare:
 - se le transazioni accedono a oggetti diversi, possono essere interleaved
 - altrimenti, se almeno due transazioni vogliono l'accesso allo stesso oggetto e almeno una delle due vuole modificarlo, le due transazioni dovranno essere eseguite in serie
il Lock Manager tiene traccia, per ogni oggetto del DB, dell'accesso in una *lock table*.
- **OPTIMISTIC:** esegue tutte le transazioni concorrenti e verifica la presenza di conflitti prima del commit
- **TIMESTAMPING:** assegna timestamps alle transazioni e compara tutti i valori per determinare l'ordine delle operazioni.

Deadlock Prevention

Una concorrenza locking-based potrebbe causare *deadlocks*. Solitamente i DBMS assegnano una priorità alle transazioni in base al tempo di inizio.

Se T1 richiede un lock posseduto da T2, il lock manager può:

- **wait-die**: se T1 è la transazione antecedente, T1 aspetta. Altrimenti T1 viene terminata e ripresa successivamente, con un random delay ma con lo stesso timestamp
- **wound-wait**: se T1 è la transazione antecedente, T2 viene terminata e ripresa successivamente con random delay, ma stesso timestamp. Altrimenti, T1 aspetta.

Se i deadlock sono rari, il DBMS lascia che si verifichino e li risolve invece di adottare le policy per evitarli. Due approcci più comuni:

- il lock manager mantiene una struttura chiamata **waits-for graph**, che utilizza per identificare il cicli deadlock. Il grafo è periodicamente analizzato e i cicli deadlock sono risolti abortendo alcune transazioni.
- se una transazione aspetta per un periodo più lungo del timeout assegnato, il lock manager assume che la transazione è deadlocked e la abortisce.

Optimistic Concurrency Control

Il protocollo basato sul locking adotta un approccio pessimistico per prevenire i conflitti.

L'approccio ottimistico assume che le transazioni non vadano in conflitto (o che lo facciano raramente).

Validation è un approccio ottimistico dove le transazioni possono accedere ai dati senza lock, e al momento opportuno, controlla che le transazioni si siano comportate in modo seriale.

Le transazioni sono eseguite in tre fasi:

- **read**: la transazione è eseguita leggendo il dato dal DBMS e scrivendo in un area privata
- **validation**: prima del commit, il DBMS controlla che non ci siano stati conflitti. In tal caso, la transazione viene abortita e restartata automaticamente.
- **write**: se la fase di validation viene conclusa con successo, il dato scritto nell'area privata viene copiato nel DBMS.

Timestamping concurrency control

Timestamping è un altro approccio ottimistico che assegna per ogni transazione il timestamp TS del suo start time. Per ogni operazione a1 eseguita da T1:

- se l'operazione a1 è in conflitto con l'operazione a2 eseguita da T2 e $TS1 < TS2$, allora a1 deve essere eseguita prima di a2.
- se un'operazione eseguita da T viola questo ordine, la transazione T è abortita e ripresa con un TS maggiore.

Crash Recovery

Il **Logging and recovery manager** del DBMS deve assicurare:

- **atomicità**: operazioni eseguite da transazioni non-committed sono rolled back.
- **persistenza**: operazioni eseguite da transazioni committed devono persistere ad un crash di sistema

ARIES Recovery Algorithm

Advanced Recovery and Integral Extraction System è un algoritmo di ripristino eseguito dal Logging and Recovery Manager sugli arresti anomali di sistema. Tre fasi:

- **fase di analisi:** identifica le pagine sporche del buffer pool (cioè, le modifiche non ancora scritte sul disco) e le operazioni attive nel momento del crash
- **fase di redo:** partendo da un dato checkpoint nel log file, ripete tutte le operazioni e riporta il DB nello stato in cui si trovava al momento del crash
- **fase di undo:** cancella le operazioni delle transazioni che erano attive al momento del crash, ma che non erano committed, in ordine inverso.

Principi:

- **write-ahead logging:** qualunque cambiamento ad un oggetto del DB deve prima essere registrato nel log file. Successivamente, il log file deve essere riportato sulla memoria secondaria. Infine, le pagine modificate possono essere aggiornate.
- **repeating history during redo:** Al riavvio, dopo un arresto, il sistema viene riportato allo stato in cui si trovava prima dell'anomalia. Le operazioni delle transazioni ancora attive durante il crash vengono cancellate.
- **logging changes during redo:** i cambiamenti fatti al DB durante l'annullamento delle transazioni sono registrati per assicurare che quell'azione non venga ripetuta nel caso di un altro arresto anomalo.

Log File

Il log file tiene traccia di tutte le azioni eseguite dal DBMS. È fisicamente organizzato in registrazioni memorizzate in uno storage stabile, che dovrebbe resistere ad incidenti / guasti dell'hardware. I log records sono ordinati sequenzialmente con un id unico, *Log Sequence Number* (LSN).

La parte più recente del log file, detta **log tail**, è conservata in **log buffers** e salvata periodicamente in storage stabili. Il log file e i dati vengono scritti su disco con gli stessi meccanismi.

Logging Data Structures

La **dirty page table** tiene i record di tutti gli ID delle pagine che sono state modificate e non ancora scritte in uno storage stabile, e della LSN dell'ultimo log entry che l'ha causato.

Log File Record

<LSN, Transaction ID, Page ID, Redo, Undo, Previous LSN>

- i campi Transaction ID e Page ID identificano la transazione e la pagina
- i campi Redo e Undo tengono le informazioni sulle modifiche che il log record salva e sul come annullarle
- il campo Previous LSN è una reference al precedente log record creato per la stessa transazione. Permette il roll back di transazioni abortite.

Creazione di Log File Record

Un record è scritto nel Log File per ognuno dei seguenti eventi:

- **page update:** un record deve essere scritto in memoria stabile prima di modificare effettivamente i dati della pagina. Mantiene sia il vecchio che il nuovo valore della pagina per rendere possibili le

operazioni di undo e redo.

- **commit**: un record traccia che una transazione è stata completata con successo e il log tail viene scritto nello stable storage.
- **abort**: un record traccia una transazione abortita, e la transazione undo viene ripresa
- **end**: dopo un commit/abort, sono necessarie alcune operazioni di finalizzazione al fine della quale viene scritta una registrazione finale
- **undoing operation**: durante un recovery o durante l'undoing delle operazioni, viene un scritto un tipo speciale di file record, il Compensation Log Record (CLR). Un record CLR non viene mai ripristinato e traccia che un'operazione è stata già annullata

Checkpoint

Un checkpoint è uno snapshot dello stato del DB. I checkpoint riducono il tempo di ripristini. Invece di dover eseguire l'intero log file, è sufficiente eseguire all'indietro fino ad un checkpoint. ARIES crea checkpoints in tre step:

- **begin-checkpoint**: il record viene scritto nel log file
- **end-checkpoint**: il record, contenente *dirty page table* e *transaction table*, viene scritto nel log file
- alla fine dell'end-checkpoint il record viene scritto nello stable storage, un record, **Master Record**, contenente LSN del *begin-checkpoint* viene scritto in una parte conosciuta dello stable storage
Questo tipo di checkpoint viene detto **fuzzy checkpoint** e non è costoso in termini di performance.
Non interrompe le normali operazioni del DBMS e non richiede la scrittura delle pagine del buffer pool.

ARIES Crash Recovery

- **analysis phase**:
 - determina la posizione del log file da dove comincia la fase di redo, ovvero l'inizio dell'ultimo checkpoint
 - determina quale pagine del buffer pool contiene il dato modificato che non era ancora stato scritto al momento dell'anomalia.
 - identifica le transazioni che erano in corso al momento dell'anomalia
- **redo phase**: dalla *dirty page table*, ARIES identifica il minimo LSN di una dirty page. Da qui, ripete le azioni fino all'anomalia, nel caso in cui non erano già state rese persistenti
- **undo phase**: i cambiamenti di una transazione uncommitted devono essere annullate in modo da ripristinare il DB in uno stato consistente.

Database Design

La progettazione è una delle attività che fanno parte del processo di sviluppo de sistemi informativi. Deve essere vista in un contesto più generale: il ciclo di vita dei sistemi informativi, comprende:

- **studio di fattibilità:** definisce i costi delle varie alternative possibili e le priorità di realizzazione delle componenti del sistema
- **raccolta e analisi dei requisiti:** individua e studia le proprietà e le funzionalità che il sistema informativo dovrà avere. E' necessaria l'interazione con l'utente per poter produrre una descrizione completa, generalmente informale, dei dati.
- **progettazione:**
 - *progettazione dei dati:* individua la struttura e l'organizzazione che i dati dovranno avere
 - *progettazione delle applicazioni:* definisce le caratteristiche dei programmi applicativi sono complementari e possono procedere in parallelo o in cascata. Le descrizioni dei dati e delle applicazioni diventano formali e fanno riferimento a modelli specifici.
- **implementazione:** realizzazione del sistema informativo. La base di dati viene costruita e popolata e viene prodotto il codice dei programmi;
- **funzionamento:** il sistema informativo diventa operativo.

Il processo non è quasi mai strettamente sequenziale, spesso durante l'esecuzione di una delle attività citate, bisogna rivedere decisioni prese nell'attività precedente.

Le basi di dati costituiscono solo una delle componenti di un sistema informativo, che comprende anche i programmi applicativi, le interfacce con l'utente e altri programmi di servizio. I dati però hanno un ruolo centrale.

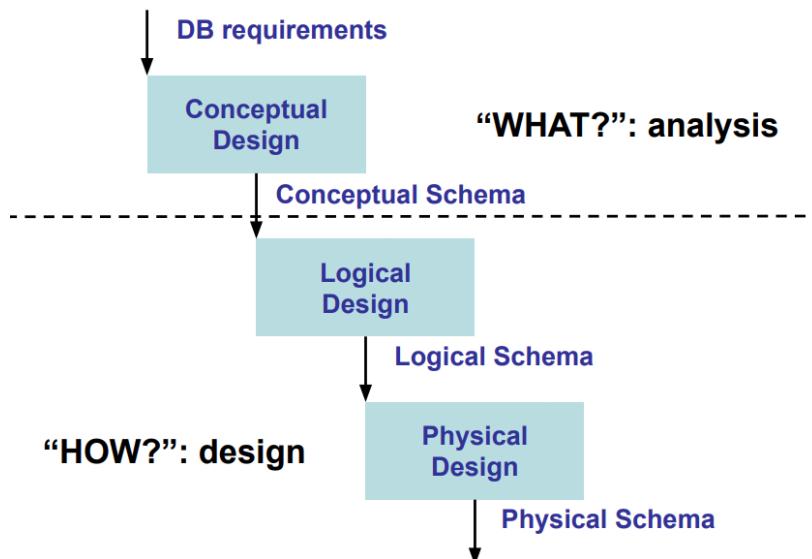
Metodologie di progettazione

Una metodologia di progettazione consiste in:

- una *decomposizione* dell'attività di progetto in passi indipendenti tra loro
- delle *strategie* da seguire e alcuni criteri per la scelta in caso di alternative
- *modelli di riferimento* per descrivere i dati di ingresso e uscita delle varie fasi inoltre deve garantire:
- la *generalità* rispetto alle applicazioni e ai sistemi
- la *qualità del prodotto* in termini di correttezza, completezza ed efficienza
- la *facilità d'uso* delle strategie e dei modelli di riferimento

Tale metodologia è articolata in tre fasi principali: separare in maniera netta le decisioni relative a "cosa" rappresentare nella base di dati (prima fase), da quelle relative a "come" farlo (seconda e

terza fase).



- **progettazione concettuale:**

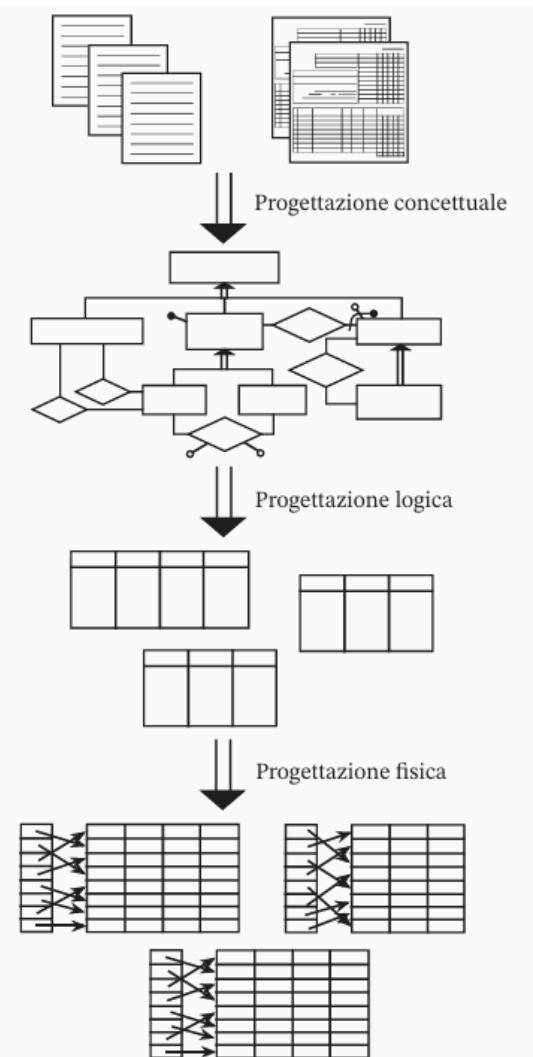
- rappresenta le specifiche informali della realtà di interesse con una descrizione formale e completa.
- Il prodotto di questa fase è lo *schema concettuale* che fa riferimento a un *modello concettuale*.
- Consente di descrivere l'organizzazione dei dati a un alto livello di astrazione, senza tenere conto degli aspetti implementativi, cercando di rappresentare il contenuto informativo della base di dati.

- **progettazione logica:**

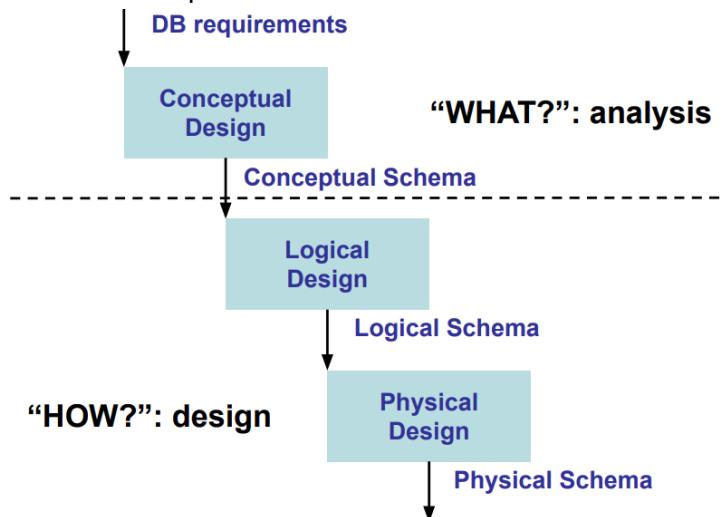
- traduzione dello schema concettuale in un modello di rappresentazione dei dati adottato dal sistema di gestione di base di dati a disposizione.
- Il prodotto di questa fase è lo *schema logico* che fa riferimento a un *modello logico*.
- Consente di scrivere i dati secondo una rappresentazione ancora indipendente da dettagli fisici, ma comunque concreta.
- Le scelte progettuali si basano su criteri di ottimizzazione delle operazioni da effettuare sui dati. (nel caso del modello relazionale dei dati, la tecnica comunemente utilizzata è quella della *normalizzazione*).

- **progettazione fisica:**

- lo schema logico viene completato con la specifica dei parametri fisici di memorizzazione dei dati (organizzazione dei file o degli indici)
- Il prodotto di questa fase è lo *schema fisico* e fa riferimento a un *modello fisico*



Distinguiamo le *specifiche sui dati*, che riguardano il contenuto della base di dati, da le *specifiche sulle operazioni*, ovvero l'uso che utenti e applicazioni fanno della base di dati. Nella progettazione concettuale si fa uso delle specifiche sui dati mentre le specifiche sulle operazioni servono a verificare che lo schema contenga le informazioni necessarie per eseguire tutte le operazioni previste. Nello schema logico, lo schema concettuale riassume le specifiche sui dati, mentre le specifiche sulle operazioni si utilizzano per ottenere uno schema logico efficiente. Bisogna conoscere il modello logico adottato ma non è necessario conoscere il DBMS che verrà usato. Nella progettazione fisica si fa uso dello schema logico e delle specifiche sulle operazioni per ottimizzare le prestazioni del sistema.

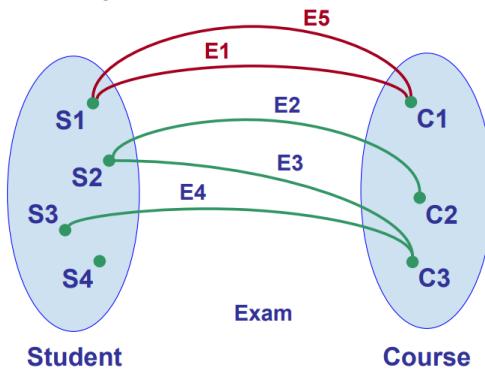


Modello Entità-Relazione

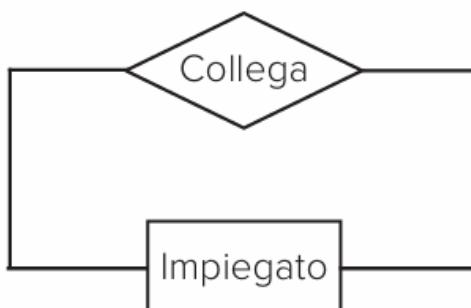
E' un modello concettuale di dati che fornisce dei *costrutti* per descrivere la realtà di interesse in maniera facile da comprendere e che prescinde dai criteri di organizzazione dei dati nei calcolatori.

Costrutti Principali:

- **Entità:** rappresentano classi di oggetti che hanno proprietà comuni ed esistenza "autonoma" (cose, persone, posti). Ogni entità ha un nome unico e significativo.
- **Relazioni:**
 - rappresentano legami logici (associazioni) tra due o più entità.
 - L'insieme delle occorrenze di una relazione del modello E-R è una relazione matematica tra le occorrenze delle entità coinvolte, ovvero un sottoinsieme del loro prodotto cartesiano, questo significa che non ci possono essere ennuple ripetute.



- è possibile avere relazioni *ricorsive*, ovvero tra un'entità e se stessa.



- è possibile avere relazioni n-arie, ovvero che coinvolgono più di due entità
- talvolta può essere utile promuovere le relazioni a entità, ad esempio la relazione *esame* tra *Studente* e *Corso* non cattura la situazione in cui uno studente ha dato più volte lo stesso esame.

Attributi:

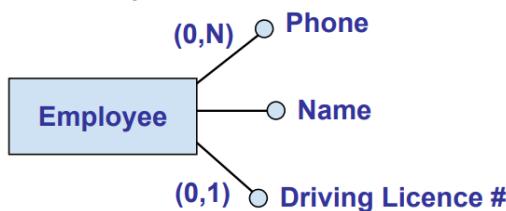
- descrivono le proprietà elementari di entità o relazioni che sono di interesse ai fini dell'applicazione;
- un attributo associa a ciascuna occorrenza di entità (o di relazione) un valore appartenente a un insieme, detto *dominio*, che contiene i valori ammissibili per l'attributo (es. l'attributo cognome può avere come dominio l'insieme delle stringhe di 20 caratteri); i domini non vengono riportati nello schema, ma sono generalmente descritti nella documentazione associata;
- è possibile raggruppare attributi di una medesima entità o relazione che presentano affinità nel loro significato ottenendo così un attributo composto



- **Cardinalità delle relazioni:**
 - sono specificate per ciascuna partecipazione di entità a una relazione e descrivono il numero minimo e massimo di occorrenze di relazione a cui una occorrenza dell'entità può partecipare: quante volte in una relazione tra entità, un'occorrenza di una di queste entità può essere legata a occorrenze delle altre entità coinvolte;
 - è possibile assegnare un qualunque intero non negativo a una cardinalità, con l'unico vincolo che la cardinalità minima deve essere minore o uguale della cardinalità massima:
 - per la cardinalità minima:
 - **0**: la partecipazione dell'entità relativa è *opzionale*;
 - **1**: la partecipazione è *obbligatoria*.
 - per la cardinalità massima:
 - **1**: la partecipazione dell'entità relativa può essere vista come una funzione che associa a una occorrenza dell'entità una solo occorrenza dell'altra entità;
 - **N**: c'è una associazione con un numero arbitrario di occorrenze dell'altra entità.
 - *classificazione relazioni binarie in base alla cardinalità*:
 - **relazioni uno a uno**, relazioni aventi cardinalità massima pari a 1 per entrambe le entità coinvolte, corrispondenza uno a uno tra le occorrenze;
 - **relazioni uno a molti**: relazioni aventi un'entità con cardinalità massima pari a 1 e l'altra con cardinalità massima pari a N;
 - **relazioni molti a molti**: relazioni aventi cardinalità massima N per entrambe le entità coinvolte.

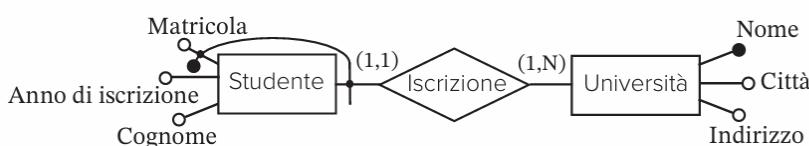
- **Cardinalità degli attributi:**

- possono essere specificate per gli attributi di entità o relazione e descrivono il numero minimo e massimo di valori dell'attributo associati a ogni occorrenza di entità o relazione.
- un attributo con cardinalità:
 - minima uguale a 0 è *opzionale*;
 - minima uguale a 1 è *obbligatorio*;
 - massima uguale a N è *multivalore*.



- **Identificatori delle entità**

- vengono specificati per ciascuna entità e descrivono i concetti (attributi e/o entità) che permettono di identificare in modo univoco le occorrenze delle entità.
- due casi:
 - *identificatore interno*: uno o più attributi di un'entità sono sufficienti a individuare un identificatore;
 - *identificatore esterno*: gli attributi dell'entità non sono sufficienti e l'identificatore è ottenuto utilizzando altre entità.



- considerazioni generali:
 - un identificatore può coinvolgere uno o più attributi, ognuno dei quali deve avere cardinalità (1,1);
 - un'identificazione esterna può coinvolgere una o più entità, ognuna delle quali deve essere membro di una relazione alla quale l'entità da identificare partecipa con cardinalità (1,1);
 - un'identificazione esterna può coinvolgere un'entità che è a sua volta identificata esternamente, purché non vengano generati cicli di identificazioni esterne;
 - ogni entità deve avere almeno un identificatore (interno o esterno), ma ne può avere in generale più di uno; nel caso di più identificatori, le restrizioni indicate possono essere rilassate in quanto gli attributi e le entità coinvolte in alcune identificazioni, tranne una, possono essere opzionali (cardinalità minima uguale a 0).
- **Generalizzazioni:**
 - rappresentano legami logici tra un'entità E, detta *genitore*, e una o più entità E₁,..., E_n, dette entità *figlie*, di cui E è più generale, nel senso che le comprende come caso particolare.
 - E è *generalizzazione* di E₁,..., E_n e le entità E₁,..., E_n sono *specializzazioni* dell'entità E.
 - Tra le entità coinvolte in una generalizzazione valgono le seguenti proprietà:
 - ogni occorrenza di un'entità figlia è anche un'occorrenza dell'entità genitore;
 - ogni proprietà dell'entità genitore (attributi, identificatori, relazioni e altre generalizzazioni) è anche proprietà delle entità figlie → *ereditarietà*;
 - possono essere classificate sulla base di due proprietà tra loro ortogonali:
 - *generalizzazione totale*, se ogni occorrenza dell'entità genitore è un'occorrenza di almeno una delle entità figlie, altrimenti è *parziale*;
 - *generalizzazione esclusiva*, ogni occorrenza dell'entità genitore è al più un'occorrenza di una delle entità figlie, altrimenti è *sovraposta*.
 - es.

Persona ⇒ *Uomo* e *Donna*: totale ed esclusiva;
Professionista ⇒ *Ingegnere* e *Dottore*: parziale ed esclusiva, si presume che ciascun professionista abbia una sola professione principale e che ci siano altre professioni;
Persone ⇒ *Studente* e *Lavoratore*: parziale e sovrapposta, esistono studenti che sono anche lavoratori.

 - una stessa entità può essere coinvolta in più generalizzazioni diverse.
 - possono esserci generalizzazioni su più livelli: *gerarchia* di generalizzazioni.
 - una generalizzazione può avere una sola entità figlia: *sottoinsieme*.

Documentazione di schemi E-R

Uno schema E-R non è quasi mai sufficiente da solo, in quanto:

- compaiono solo i nomi dei vari concetti in esso presenti e può non essere sufficiente per comprenderne il significato;
- nel caso di schemi particolarmente complessi potremmo non riuscire a rappresentare in maniera comprensibile ed esaustiva i concetti;
- non si possono rappresentare *vincoli di integrità sui dati*.

Diventa indispensabile corredare ogni schema E-R con una documentazione di supporto.

Regole Aziendali

Le regole aziendali, o *business rules*, sono uno strumento utilizzato per la descrizione delle proprietà di un'applicazione che non si riescono a rappresentare con modelli concettuali. Una regola aziendale può essere:

- *descrizione di un concetto* rilevante per l'applicazione, ovvero la definizione precisa di un'entità, di un'attributo o di una relazione dello schema;
- *vincolo di integrità* sui dati dell'applicazione;
- una *derivazione*, un concetto che può essere ottenuto, attraverso un'inferenza o un calcolo aritmetico, da altri concetti dello schema.

Le regole che descrivono vincoli di integrità possono essere espresse sotto forma di asserzioni, affermazioni che devono essere sempre verificate nella base di dati:

< concetto > deve/non deve < espressione su concetti >

es. (RV) un impiegato *non deve* avere uno stipendio maggiore del direttore del dipartimento al quale afferisce.

Le regole aziendali che esprimono derivazioni possono essere espresse specificando le operazioni:

< concetto > si ottiene < operazione su concetti >

es (RD) il numero degli impiegati di un dipartimento *si ottiene* contando gli impiegati che vi afferiscono.

Tecniche di documentazione

Le regole aziendali di tipo descrittivo possono essere rappresentate facendo uso di un *dizionario dei dati* in cui troviamo due tabelle:

1. descrive le entità dello schema con il nome, una definizione informale in linguaggio naturale, l'elenco di tutti gli attributi e i possibili identificatori;

| Entity | Description | Attributes | Identifier |
|------------|----------------------|---------------------|--------------|
| Employee | Employee in a Dept. | Code, Name, Surname | Code |
| Project | Projects of a Dept. | Name, Budget | Name |
| Department | Structure of a Dept. | Name, Phone | Name, Office |
| Office | Office's location | City, Address | City |

2. descrive le relazioni con il nome, una loro descrizione informale, l'elenco degli attributi e l'elenco delle entità coinvolte insieme alla loro cardinalità.

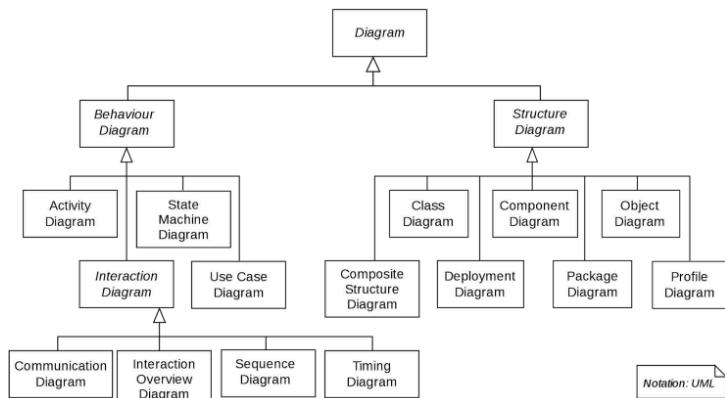
| Relationships | Description | Components | Attributes |
|---------------|----------------------------|-------------------|------------|
| Management | Management of a Dept. | Employee, Dept. | |
| Affiliation | Affiliation to a Dept. | Employee, Dept. | Date |
| Attendance | Attendance for a project | Employee, Project | |
| Composition | Composition of Departments | Dept., Office | |

Possiamo anche avere una tabella per rappresentare i vincoli:

| Integrity Constraints on Data |
|---|
| 1. A department director must belong to that department |
| 2. An employee must not have an income greater than the director of the department which he is affiliated |
| 3. A department placed in Rome must be directed by an employee with at least 10 years of service |
| 4. An employee that isn't affiliated to any department must not attend to any project |

Modellazione dei dati in UML

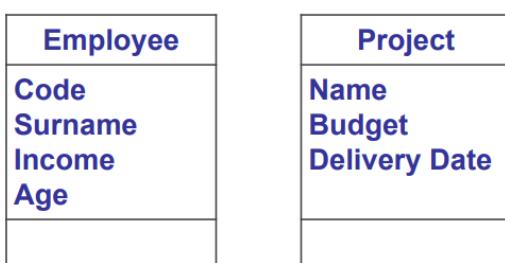
UML (Unified Modeling Language) è un linguaggio grafico per la modellazione di applicazioni software basate sulla programmazione orientata agli oggetti. Utilizza i *diagrammi delle classi* che descrivono le classi di oggetti di interesse per l'applicazione e le relazioni che intercorrono tra di esse. Inoltre in base al principio di *incapsulamento* della oop, prevede una stretta correlazione tra dati e operazioni; infatti è possibile rappresentare oltre agli aspetti strutturali dell'applicazione, ovvero i dati sui quali opera, anche quelli "comportamentali", ovvero le procedure associate ai dati.



Rappresentazione dei dati con i diagrammi delle classi

- **Classi:**

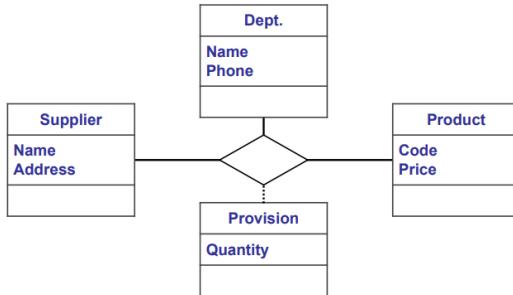
- componenti principali dei diagrammi delle classi;
- corrispondono alle entità del modello E-R;
- rappresentato come un rettangolo contenente:
 - in alto: il nome della classe;
 - all'interno: gli attributi associati alla classe;
 - in basso è possibile specificare i relativi *metodi*, ovvero le operazioni ammissibili su oggetti della classe secondo il principio di incapsulamento.



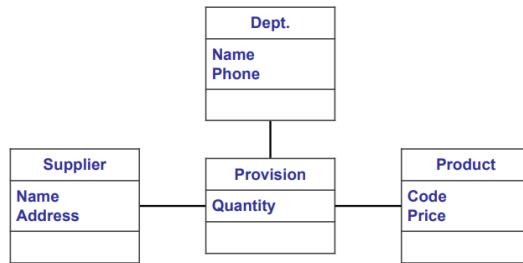
- **Associazioni:**

- corrispondono alle relazioni del modello E-R;

- le associazioni binarie si rappresentano con semplici linee che congiungono le classi coinvolte, il nome è posto sulla linea, ma non è obbligatorio perché in UML possono esistere associazioni senza nome;
- è possibile associare ruoli alle classi coinvolte in un'associazione;
- non è possibile assegnare attributi alle associazioni, per farlo si da uso delle *classi di associazioni*;
- se l'associazione è n-aria, viene rappresentata da un rombo e da linee che congiungono il rombo con le classi che partecipano all'associazione;

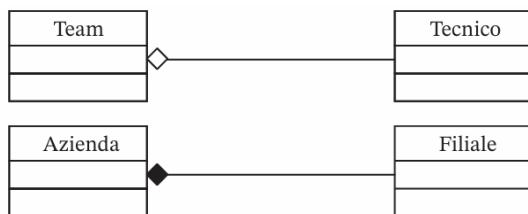


le associazioni n-arie vengono usate di rado e viene suggerito di reificare, ovvero trasformarle in una classe legata alle classi originarie con associazioni binarie;



- *aggregazioni* di concetti:

- sono associazioni che definiscono una relazione tra un concetto composito e uno o più concetti che ne costituiscono una sua parte;
- si indicano con una linea avente un rombo attaccato alla classe che rappresenta il concetto "aggregante"; dall'altro capo della linea c'è una classe che costituisce una sua "parte";
- il rombo si lascia in bianco se un oggetto della classe "parte" può esistere senza dover appartenere a un oggetto della classe "aggregante", altrimenti viene annerito e l'aggregazione viene chiamata *composizione*.



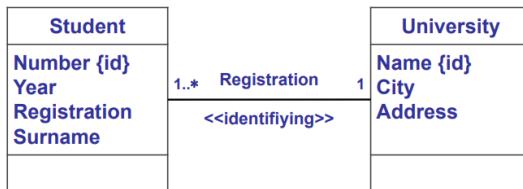
- **Molteplicità:**

- è possibile indicare le cardinalità di partecipazione come coppia di valori che specificano la cardinalità minima e massima di partecipazione di un oggetto della classe dell'associazione;
- la cardinalità minima viene separata dalla massima da due punti (0..1);
- quando si specifica solo *si intende 0..*, ovvero (0, N);
- quando si specifica solo 1 si denota la coppia di cardinalità 1..1, considerata quella di default per le classi.

- **Identificatori:** non esiste una notazione per esprimere identificatori di classi, secondo il paradigma oop ogni oggetto è dotato implicitamente di un identificatore che ne consente l'identificazione

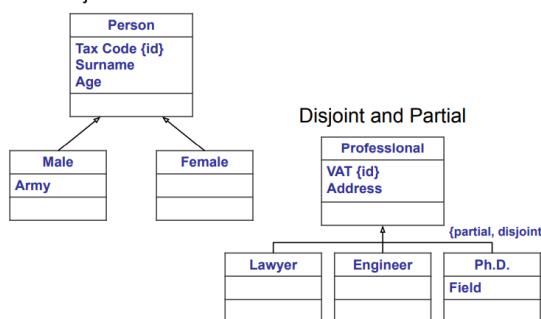
univoca. D'altronde, gli identificatori sono indispensabili nel modello relazionale e perciò introduciamo il *vincolo utente*, ovvero la possibilità di definire vincoli di integrità su associazioni e su attributi specificandoli tra parentesi graffe vicino all'elemento oggetto del vincolo.

- gli attributi che compongono un identificatore sono indicati con il vincolo utente {id};
- è possibile specificare un solo identificatore per classe;
- le identificazioni esterne, dato che sintatticamente non è possibile usare un vincolo, si ricorre a uno stereotipo: indicato da un nome racchiuso tra i simboli <>.



- **Generalizzazioni:** esiste in UML la possibilità di definire generalizzazioni, eventuali proprietà possono essere rappresentate con vincoli; in particolare possiamo indicare se la generalizzazione è totale oppure parziale e se è esclusiva oppure sovrapposta.

Disjoint and Total



Conceptual Design

L'analisi dei requisiti non è standardizzabile ma ci sono alcune regole pratiche nella fase di sviluppo della base di dati.

- *raccolta dei requisiti*: completa individuazione dei problemi che l'applicazione deve risolvere e le caratteristiche che dovrà avere;
- *analisi dei requisiti*: organizzazione delle specifiche dei requisiti.
Principali fonti di informazione:
 - gli *utenti dell'applicazione*, mediante le interviste oppure attraverso una documentazione scritta;
 - *documentazione esistente*, moduli, regolamenti interni, procedure aziendali e normative. E' richiesta un'attività di raccolta e selezione che viene assistita dagli utenti a carico del progettista;
 - *realizzazioni preesistenti*, applicazioni che si devono rimpiazzare o che devono interagire con il sistema da realizzare.

Alcune regole generali:

- *scegliere il corretto livello di astrazione*: evitare termini troppo generici o troppo specifici;
- *standardizzare la struttura delle frasi*: è preferibile utilizzare sempre lo stesso stile sintattico;
- *evitare frasi contorte*: le definizioni devono essere semplici e chiare;
- *individuare sinonimi/omonimi e unificare i termini*: l'uso di sinonimi e omonimi può generare ambiguità e perciò si tende ad utilizzare un termine unico per i sinonimi, mentre nel caso di omonimi si utilizzano termini diversi;
- *rendere esplicito il riferimento tra termini*: l'assenza di un contesto di riferimento può rendere alcuni concetti ambigui, allora occorre esplicitare il riferimento tra termini;
- *costruire un glossario dei termini*: utile per la comprensione e la precisazione dei termini usati, deve contenere per ogni termine una breve descrizione, possibili sinonimi e altri termini nel glossario con i quali esiste un legame logico.

Design Pattern

Ci sono alcune buone pratiche per una corretta rappresentazione concettuale dei dati, come dei *criteri generali di rappresentazione* o *design pattern*, ovvero soluzioni progettuali a problemi comuni.

Criteri generali di rappresentazione

Premessa, non esiste una rappresentazione univoca di un insieme di specifiche, ma è utile avere delle indicazioni sulle scelte più opportune:

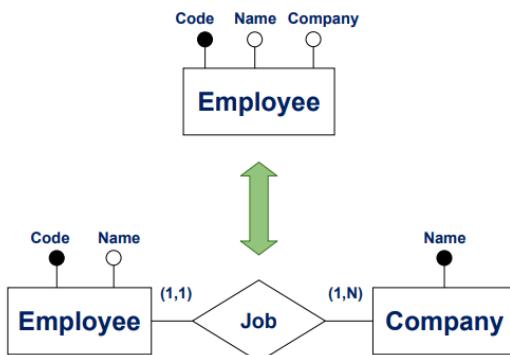
- se un concetto ha proprietà significative e/o descrive classi di oggetti con esistenza autonoma, è opportuno rappresentarlo con un'entità;
- se un concetto ha una struttura semplice e non possiede proprietà rilevanti associate è opportuno rappresentarlo con un attributo di un altro concetto a cui si riferisce;
- se sono state individuate due (o più) entità e nei requisiti compare un concetto che le associa, questo concetto può essere rappresentato da una relazione;

- se uno o più concetti risultano essere casi particolari di un altro, è opportuno rappresentarli facendo uso di una generalizzazione.
- Questi criteri hanno validità generale, cioè sono indipendenti dalla strategia di progettazione scelta.

Pattern di Progetto

1. Reificazione degli attributi:

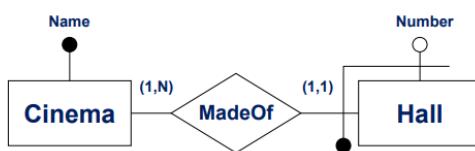
Viene rappresentato un *impiegato* con codice, nome e azienda, ma non stiamo rappresentando anche il concetto di *azienda*; per poterlo rappresentare esplicitamente reifichiamo l'attributo facendolo diventare un'entità.



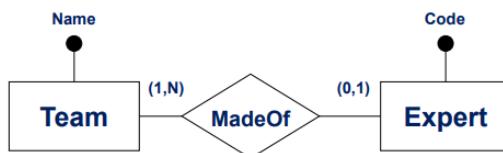
2. Part-Of

Si vuole rappresentare il fatto che un'entità è parte di un'altra entità, tipicamente queste relazioni sono uno a molti e si rappresentano in due forme:

- l'esistenza di un'occorrenza dell'entità "parte" dipende dall'esistenza di un'occorrenza dell'entità che la contiene:



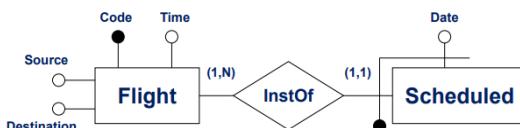
- l'entità contenuta nell'altra ha esistenza autonoma:



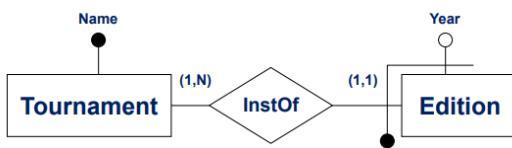
3. Instance-Of:

Talvolta sono necessarie due entità distinte, una che coinvolgono la rappresentazione astratta e un'altra che memorizza le informazioni necessarie per i nostri requisiti.

- un'entità che descrive il concetto astratto di volo e un'altra entità che rappresenta il volo reale:

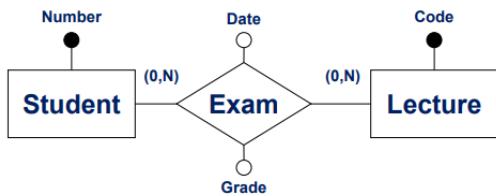


- viene rappresentato il concetto di torneo sportivo e una sua edizione:

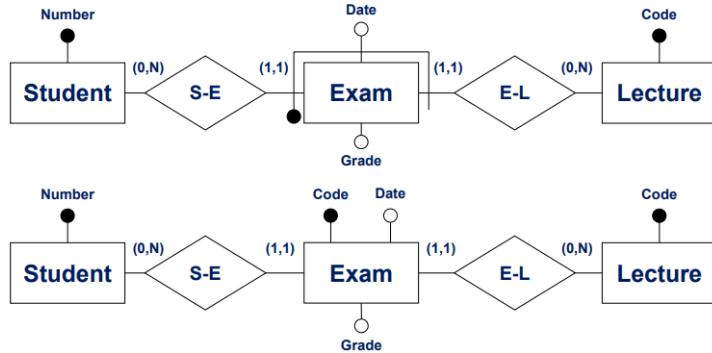


4. Reification: Binary Relations:

si utilizza una relazione, tipicamente molti a molti, per descrivere un concetto che lega altri due concetti, ad esempio: rappresentiamo un'esame come relazione tra *studente* e *corso*:

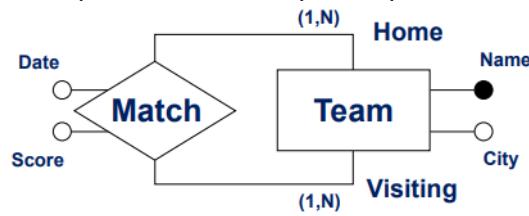


Questa soluzione è valida solo se ogni studente può sostenere una sola volta un certo esame (per def: una occorrenza della relazione *esame* è un insieme di coppie *studente-corso*, senza duplicati). La soluzione corretta si ottiene reificando la relazione *esame* rappresentandola come entità:

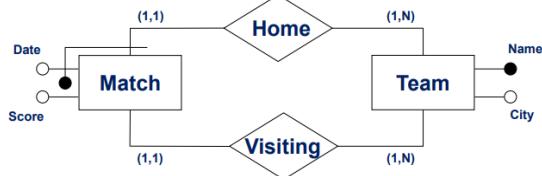


5. Reification: Recursive Relation:

Esempio: il concetto di *partita* può essere visto come una relazione ricorsiva sull'entità squadre,



ma se in un torneo due squadre si incontrano più volte è necessario reificare la relazione binaria:

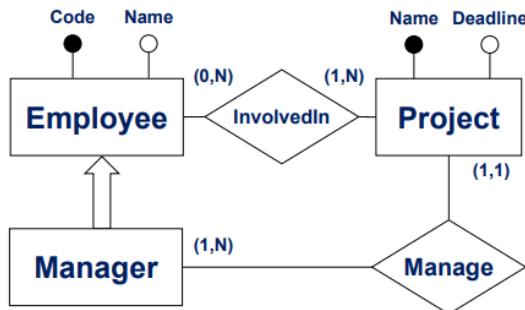


L'identificazione dell'entità *partita* coinvolge solo la data e la squadra che gioca in casa perché si assume che una squadra non possa giocare due partite nello stesso giorno.

6. Generalizzazioni:

- *specific case*:

Le generalizzazioni sono usate per definire casi specifici, nell'esempio *manager*, dell'entità *data*, *employee*; in questo caso non tutte gli impiegati gestiscono un progetto:

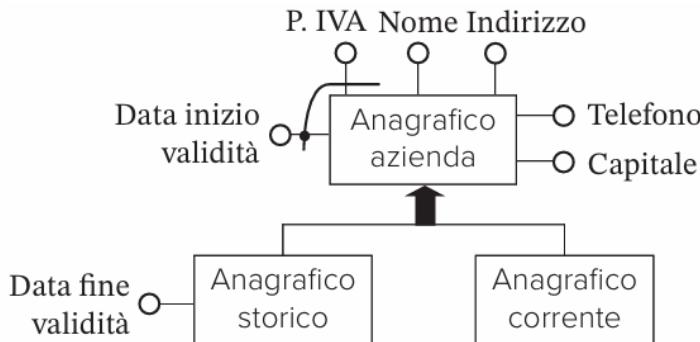


- *storicizzazione di un'entità*:

esempio, vogliamo memorizzare le informazioni correnti di un'azienda, tenendo traccia dei dati che sono variati.

soluzione: utilizzare due entità con gli stessi attributi, una rappresenta il concetto di interesse con le informazioni aggiornate e l'altra lo storico. Le proprietà di queste entità vengono messe

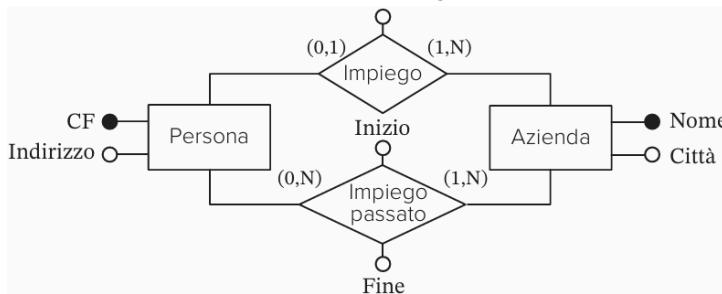
a fattore comune mediante una generalizzazione la cui entità genitore rappresenta tutte le informazioni anagrafiche delle aziende, sia quelle correnti sia quelle passate; in più vengono introdotti degli attributi per definire l'intervallo di validità dei dati (data inizio e data fine). L'identificazione si ottiene aggiungendo alla chiave originaria (p. iva) la data di inizio, istante che diventerà anche la data di fine validità delle informazioni che vengono soppiantate.



- **storicizzazione di una relazione:**

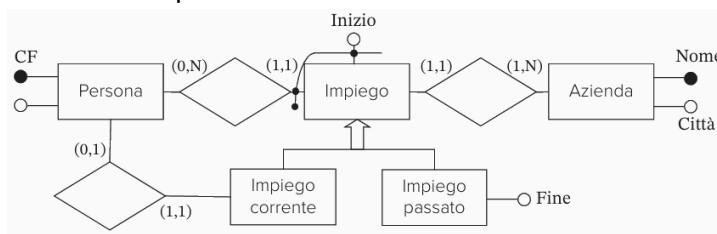
esempio, vogliamo rappresentare gli impieghi presenti e passati di una persona.

soluzione uno: rappresentare separatamente i dati correnti e i dati storici e introdurre opportuni attributi per specificare gli intervalli di validità.



Ma, osservando le cardinalità delle partecipazioni dell'entità *persona* alle due relazioni, vediamo che non possiamo rappresentare il fatto che una persona possa aver lavorato, in periodi diversi, per la stessa azienda.

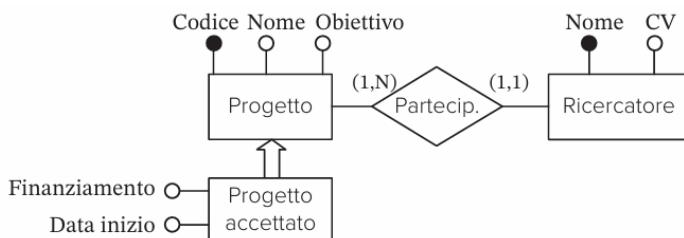
soluzione due: reificazione delle relazioni, inoltre è necessario l'inserimento di un vincolo esterno allo schema che impone che tutte le occorrenze della relazione tra *persona* e *impiego corrente* compaiano anche tra le occorrenze della relazione tra *persona* e *impiego*.



- **estendere un concetto:**

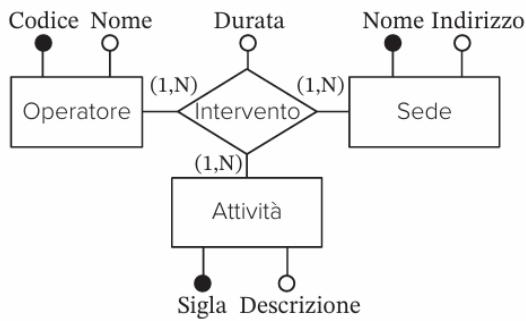
vogliamo rappresentare il fatto che un certo concetto subisce un'evoluzione nel tempo che può essere diversa per le diverse occorrenze del concetto.

esempio: si vuole rappresentare progetti che vengono proposti con l'obiettivo di ottenere un finanziamento. Solo alcuni vengono accettati e per questi vengono aggiunte ulteriori informazioni: data di inizio ufficiale e finanziamento.



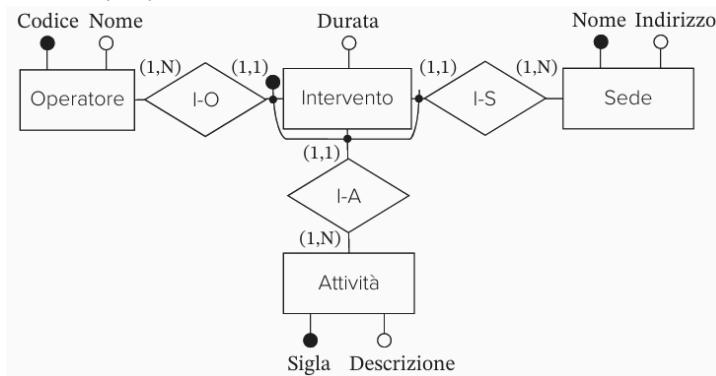
- **Relazioni ternarie:**

esempio 1: si vuole modellare il caso in cui un operatore può effettuare operazioni che consistono in attività diverse svolte in sedi diverse, in ogni sede possono operare operatori diversi svolgendo attività diverse e le attività possono essere svolte da operatori diversi e in sedi diverse.

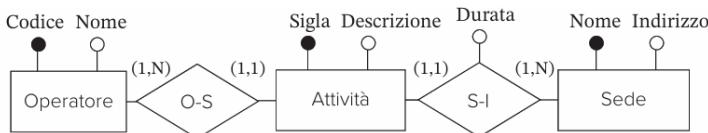


Questo schema può essere reificato per modellare altre realtà:

- Ogni lavoro è definito da un *Operatore* (I-O) che lavora in una certa *Sede* (I-S) per una certa *Attività* (I-A).



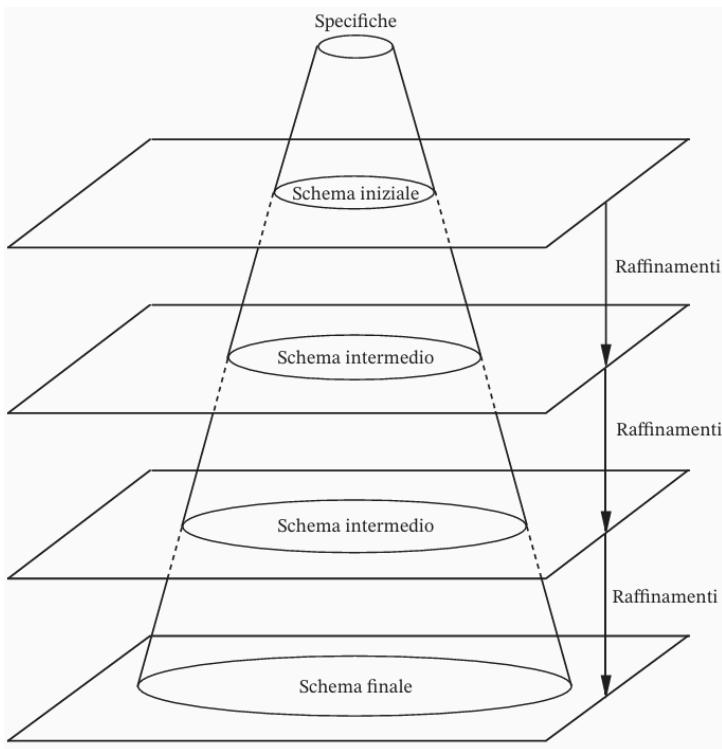
- Se un'attività può essere eseguita da un solo operatore in un solo ufficio, allora la reificazione può essere semplificata.



Strategie di Progetto

Strategia top-down

Si parte da uno schema iniziale che descrive tutte le specifiche con pochi concetti molto astratti, questo schema viene poi via via raffinato mediante trasformazioni che aumentano il dettaglio dei vari concetti.



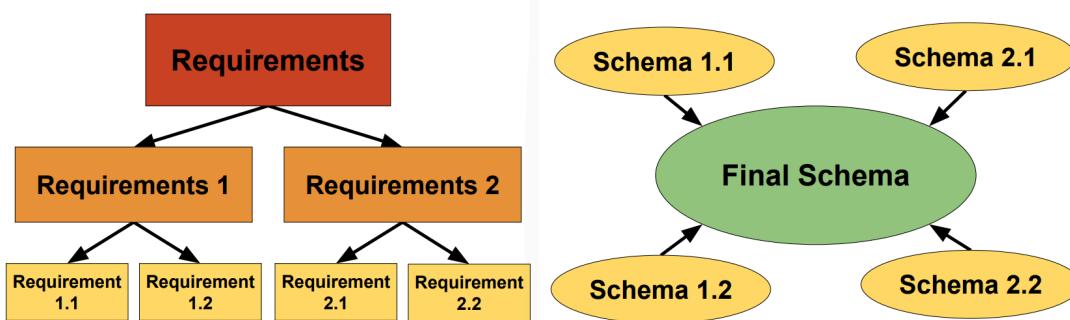
Ognuno di questi piani descrive le stesse informazioni a un diverso livello di dettaglio. Con questa strategia tutti gli aspetti presenti nello schema finale sono presenti, in linea di principio a ogni livello di raffinamento. Il vantaggio sta nel poter descrivere inizialmente tutte le specifiche dei dati trascurandone i dettagli, per poi entrare nel merito di un concetto alla volta. Questo però è possibile solo quando si possiede una visione globale di tutte le componenti del sistema.

Strategia bottom-up

Le specifiche iniziali sono suddivise in componenti via via sempre più piccole. Le varie componenti vengono rappresentate da semplici schemi concettuali, questi vengono poi fusi fino a giungere allo schema concettuale finale. Questo schema si ottiene attraverso alcune trasformazioni elementari, denominate *primitive di trasformazione bottom-up*.

Vantaggio: si adatta a una decomposizione del problema in componenti più semplici, affrontabili in parallelo da progettisti diversi.

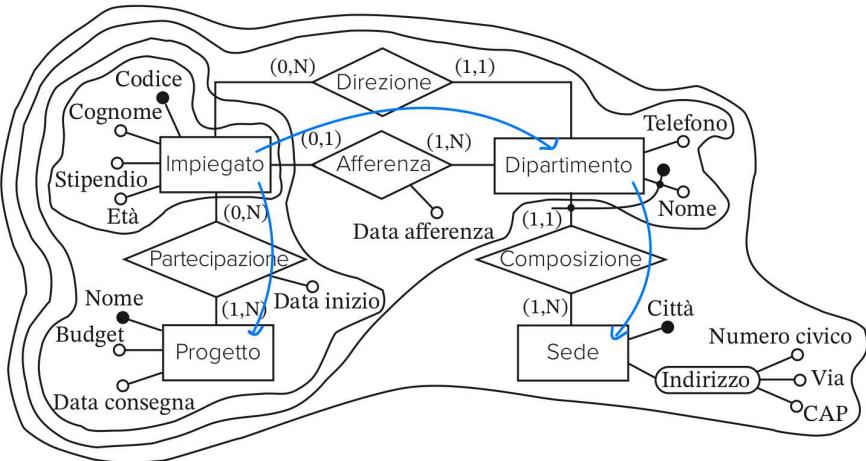
Svantaggio: richiede operazioni di integrazione di schemi concettuali diversi, che in caso di problemi complessi, presentano grosse difficoltà.



Strategia inside-out

Può essere vista come un caso particolare della strategia bottom-up. Si individuano inizialmente solo alcuni concetti importanti e poi si procede, a partire da questi. Si rappresentano prima i concetti in

relazione con i concetti iniziali, per poi muoversi verso quelli più lontani.



Rule of Thumb

Usare sempre un approccio misto:

- prima, si crea una bozza utilizzando le entità più rilevanti;
- successivamente, si decomponge lo schema;
- infine, raffinare (top-down), integrare (bottom-up) ed espandere (inside-out).

Metodologia Generale

1. Analisi dei requisiti:

- costruire un glossario dei termini;
- analizzare i requisiti ed eliminare le ambiguità presenti;
- raggruppare i requisiti in insiemi omogenei.

2. Passo Base:

- individuare i concetti più rilevanti e rappresentarli in uno schema scheletro.

3. Passo di decomposizione (da effettuare se necessario):

- effettuare una decomposizione dei requisiti con riferimento ai concetti presenti nello schema scheletro.

4. Passo iterativo, da ripetere per tutti i sottoschemi (se presenti) finché ogni specifica è stata rappresentata:

- raffinare i concetti presenti sulla base delle loro specifiche;
- aggiungere nuovi concetti allo schema per descrivere specifiche non ancora descritte.

5. Passo di integrazione (da effettuare solo se è stato eseguito il passo 3):

- integrare i vari sottoschemi in uno schema generale facendo riferimento allo schema scheletro.

6. Analisi di qualità:

controllare la qualità dello schema e modificarlo.

Qualità di uno schema concettuale

- **Correttezza** → quando utilizza propriamente i costrutti messi a disposizione.
- **Completezza** → quando rappresenta tutti i dati di interesse e quando tutte le operazioni possono essere eseguite a partire dai concetti descritti nello schema.

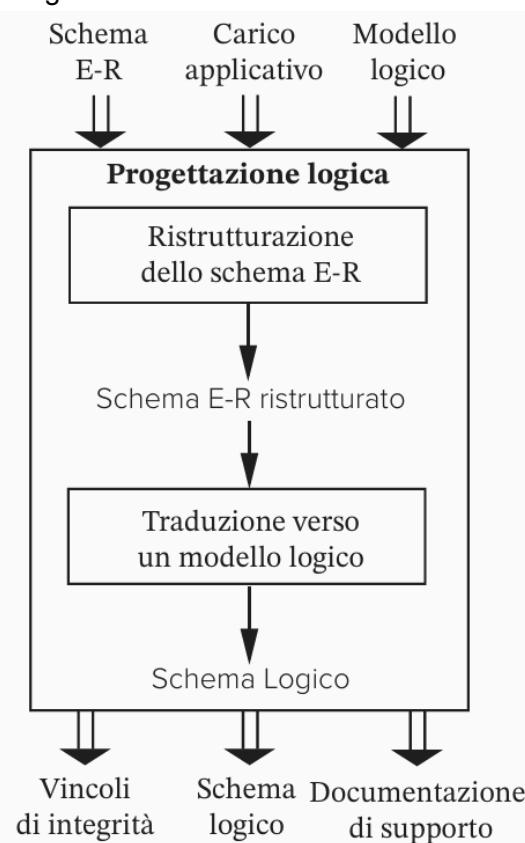
- **Leggibilità** → quando rappresenta i requisiti in maniera naturale e facilmente comprensibile (es. mediante una scelta opportuna dei nomi da dare ai concetti).
- **Minimalità** → quando tutte le specifiche sui dati sono rappresentate una volta sola nello schema (es fonte di ridondanza: presenza dei cicli dovuta alla presenza di relazioni e/o generalizzazioni).

Logical Design

Le attività principali della progettazione logica sono:

- *ristrutturazione dello schema Entità-Relazione*: si basa su criteri di ottimizzazione dello schema e di semplificazione della fase successiva;
- *traduzione verso il modello logico*: fa riferimento a uno specifico modello logico e può includere un'ulteriore ottimizzazione che si basa sulle caratteristiche del modello logico stesso.

I dati di ingresso sono lo schema concettuale e il carico applicativo previsto, ovvero la dimensione dei dati e le caratteristiche delle operazioni. Il risultato è uno schema E-R ristrutturato. Questo schema e il modello logico scelto sono i dati di ingresso della seconda fase che produce lo schema logico; in questa fase si fanno verifiche della qualità dello schema ed eventuali ulteriori ottimizzazioni. I prodotti finali della progettazione logica sono lo schema logico finale, i vincoli di integrità definiti su di esso e la relativa documentazione.



Analisi delle prestazioni

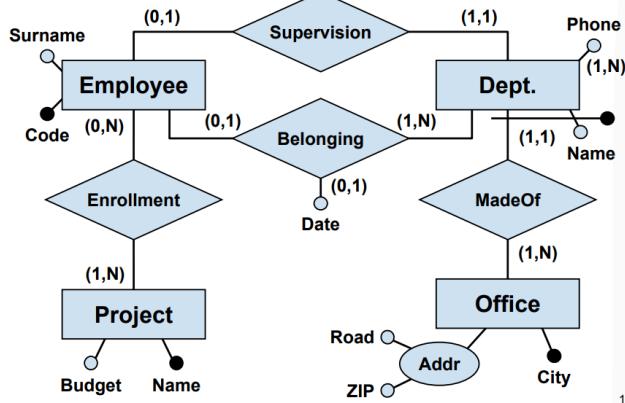
Uno schema E-R può essere modificato per ottimizzare alcuni *indici di prestazione*, chiamati così perché non sono valutabili in maniera precisa in sede di progettazione logica in quanto sono dipendenti anche da parametri fisici, ecc..

- *costo di un'operazione*: valutato in termini di numero di occorrenze di entità e associazioni che mediamente vanno visitate per rispondere a un'operazione sulla base di dati;
- *occupazione di memoria*: valutato in termini dello spazio di memoria necessario per memorizzare i dati descritti dallo schema.

Per lo studio di questi parametri serve conoscere:

- *volume dei dati*:

- numero di occorrenze di ogni entità e associazione dello schema;
- dimensioni di ciascun attributo (di entità o associazione).
- *caratteristiche delle operazioni:*
 - tipo dell'operazione;
 - frequenza (numero medio di esecuzioni in un certo intervallo di tempo);
 - dati coinvolti.



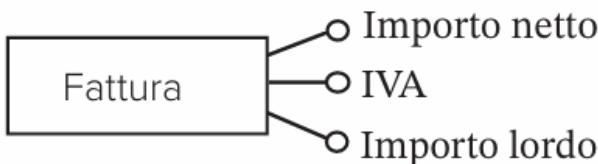
| Name | Type | Size |
|-------------|------|-------|
| Office | E | 10 |
| Dept. | E | 80 |
| Employee | E | 2'000 |
| Project | E | 500 |
| MadeOf | R | 80 |
| Belonging | R | 1'900 |
| Supervision | R | 80 |
| Enrollment | R | 6'000 |

Ristrutturazione di schemi E-R

Suddivisa in:

- **analisi delle ridondanze:**

- si decide se eliminare o mantenere eventuali ridondanze (:= presenza di un dato che può essere derivato da altri dati);
- casi più frequenti:
 - attributi derivabili da altri attributi della stessa entità (o associazione):



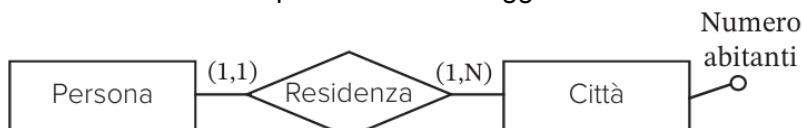
uno degli attributi è deducibile dagli altri attraverso un'operazione di somma o differenza.

- attributi derivabili da attributi di altre entità (o associazioni), di solito attraverso funzioni aggregative:



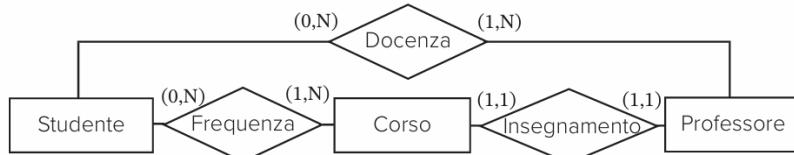
importo totale è derivabile attraverso l'associazione *composizione* dall'attributo *prezzo*, sommando i prezzi dei prodotti di un acquisto.

- attributi derivabili da operazioni di conteggio di occorrenze:



numero abitanti può essere derivato contando le occorrenze dell'associazione *Residenza* a cui tale città partecipa.

- associazioni derivabili dalla composizione di altre associazioni in presenza di cicli:

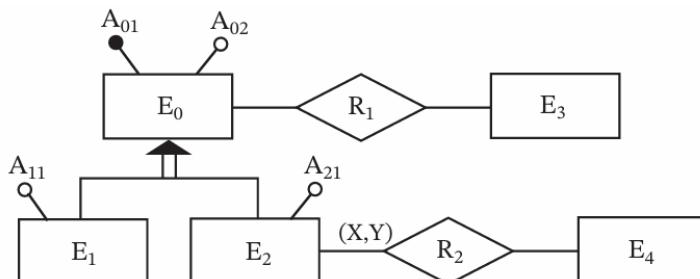


l'associazione *Docenza* può essere derivata dalle associazioni *Frequenza* e *Insegnamento*.

- La presenza di un dato derivato presenta:

- vantaggio: riduzione degli accessi necessari per calcolare il dato derivato;
- svantaggio: maggiore occupazione di memoria e la necessità di effettuare operazioni aggiuntive per mantenere il dato derivato aggiornato.

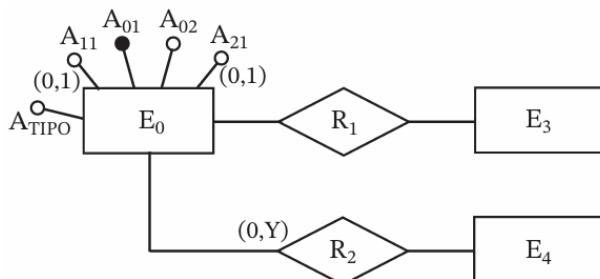
- eliminazione delle generalizzazioni:**



- i sistemi per la gestione delle basi di dati non consentono di rappresentare direttamente le generalizzazioni, perciò è necessario rappresentarle mediante entità o associazioni, per farlo esistono tre alternative:

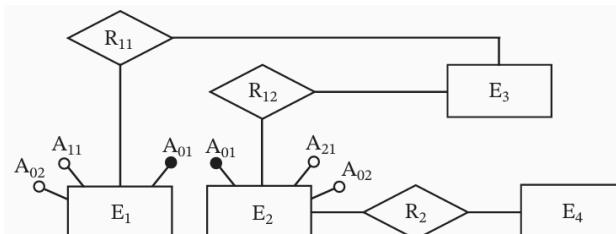
- accorpamento delle figlie della generalizzazione nel genitore:**

le entità E_1 ed E_2 vengono eliminate e le loro proprietà vengono aggiunte all'entità genitore E_0 . Inoltre viene aggiunto un attributo che serve a distinguere il "tipo" di un'occorrenza di E_0 , ovvero se apparteneva a E_1 o a E_2 nel caso di generalizzazione totale, o a nessuna di esse nel caso di generalizzazione parziale.



- accorpamento del genitore delle generalizzazioni nelle figlie:**

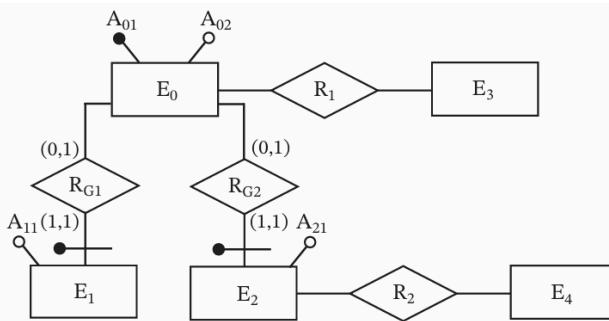
l'entità genitore E_0 viene eliminata, i suoi attributi, il suo identificatore e le relazioni a cui partecipava vengono ereditate dalle entità figlie.



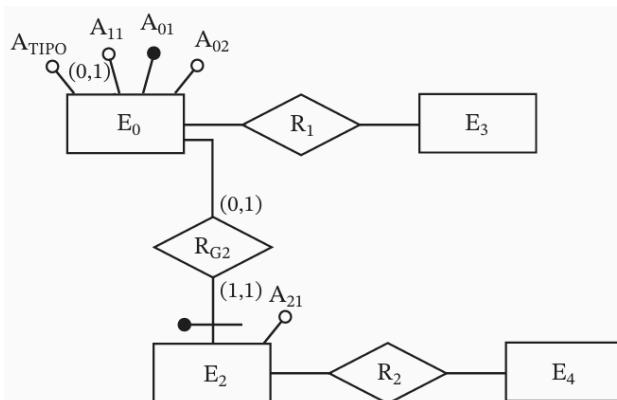
- sostituzione della generalizzazione con associazioni:**

la generalizzazione si trasforma in due associazioni uno a uno che legano l'entità genitore con le entità figlie. Non ci sono trasferimenti di attributi o associazioni e le entità figlie sono identificate esternamente dall'entità genitore. Vengono aggiunti dei vincoli:

ogni occorrenza di E_0 non può partecipare contemporaneamente a R_{G1} e R_{G2} ; se la generalizzazione è totale ogni occorrenza di E_0 deve partecipare obbligatoriamente a una delle due relazioni.

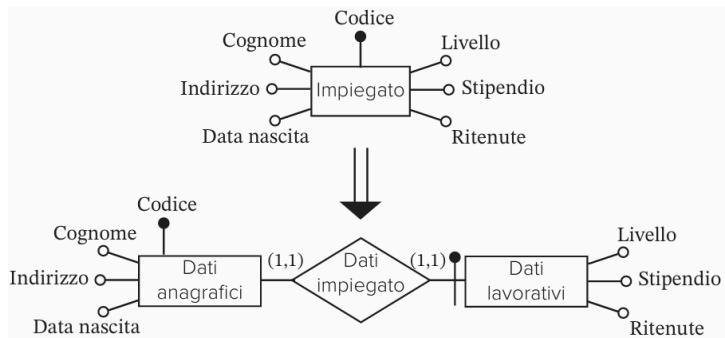


- la scelta tra le varie soluzioni possibili deve essere fatta considerando vantaggi e svantaggi relativamente all'occupazione di memoria e al costo delle operazioni:
 - scelta 1:**
 - conveniente quando le operazioni non fanno distinzione tra le occorrenze e tra gli attributi di E_0 , E_1 e E_2 ;
 - spreco di memoria per l'eventuale presenza di valori nulli;
 - ci assicura un numero minore di accessi rispetto alle altre soluzioni;
 - scelta 2:**
 - è possibile solo se la generalizzazione è totale, altrimenti le occorrenze dell'entità genitore che non sono occorrenze delle entità figlie non sarebbero rappresentate;
 - E' conveniente quando ci sono operazioni che si riferiscono solo a una delle due entità figlie;
 - risparmio di memoria rispetto alla scelta (1), perché gli attributi non assumono mai valori nulli;
 - riduzione degli accessi rispetto alla scelta (3) perché non si deve visitare l'entità genitore per accedere agli attributi dei figli.
 - scelta 3:**
 - è conveniente quando la generalizzazione non è totale e ci sono operazioni che si riferiscono solo a occorrenze di una delle entità figlie o dell'entità genitore;
 - risparmio di memoria rispetto alla scelta (1), per lo stesso motivo di prima;
 - incremento degli accessi per mantenere la consistenza delle occorrenze rispetto ai vincoli introdotti.
- le alternative viste non sono le uniche ammesse, talvolta è possibile utilizzare una combinazione delle tre soluzioni.

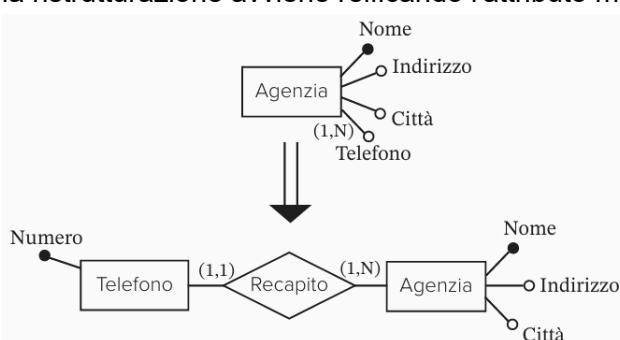


- partizionamento/accorpamento di entità e associazioni:**

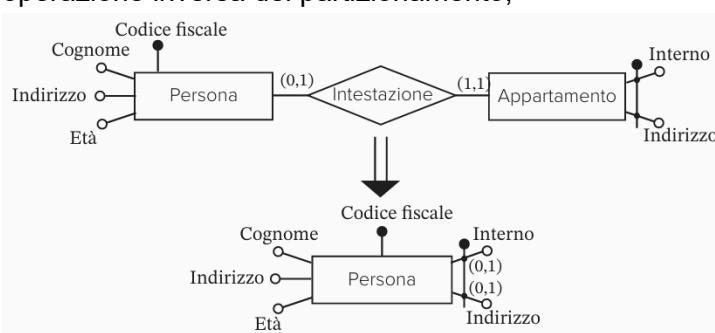
- entità e associazioni possono essere partizionati o accorpati in base a questo principio: gli accessi si riducono
 - separando attributi di uno stesso concetto che vengono acceduti da operazioni diverse;
 - raggruppando attributi di concetti diversi che vengono acceduti dalle stesse operazioni;
- *partizionamenti di entità*:
 - *decomposizione verticale*: si suddivide il concetto operando sui suoi attributi;
 - generano entità con pochi attributi che possono essere tradotte in strutture logiche sulle quali con un solo accesso è possibile recuperare molti dati.



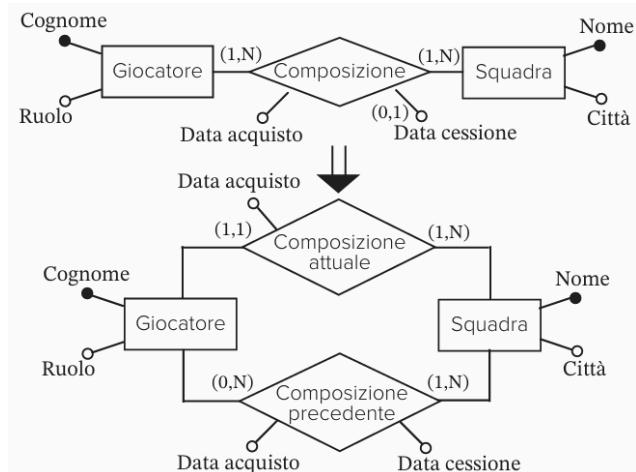
- *decomposizione orizzontale*: la suddivisione avviene sulle occorrenze dell'entità, può convenire decomporre l'entità in due entità distinte, corrisponde all'introduzione di una generalizzazione a livello logico.
 - effetto collaterale: dover duplicare le associazioni a cui l'entità originaria partecipava;
- *eliminazione di attributi multivalore*:
 - come le generalizzazioni, non sono rappresentabili nel modello relazionale;
 - la ristrutturazione avviene reificando l'attributo multivalore:



- *accorpamento di entità*:
 - operazione inversa del partizionamento;
- effetto collaterale: possibile presenza di valori nulli;
- generalmente si effettua su associazioni di tipo uno a uno, raramente su associazioni uno a molti, mai su associazioni molti a molti (in quest'ultimo caso generano ridondanze).



- il concetto di partizionamento e accorpamento di entità si può applicare anche sulle associazioni:
 - in alcuni casi può essere utile decomporre un'associazione tra due entità in due (o più) associazioni tra le stesse entità per separare le occorrenze:



- è possibile accoppare due (o più) associazioni tra le medesime entità (che si riferiscono a due aspetti dello stesso concetto) in un'unica associazione.
- scelta degli identificatori principali:**
 - essenziale nelle traduzioni verso il modello relazionale:
 - usate per stabilire legami tra dati in relazioni diverse;
 - i sistemi di gestione utilizzano la chiave primaria per la costruzione automatica di *indici*.
 - criteri di decisione:
 - gli attributi con valori nulli non possono essere identificatori principali;
 - un identificatore composto da uno o da pochi attributi è da preferire a quelli costituiti da molti attributi, in quanto:
 - garantisce che gli indici siano di dimensioni ridotte;
 - risparmio di memoria nella realizzazione dei legami logici tra le relazioni;
 - facilita le operazioni di join.
 - un identificatore interno con pochi attributi è preferibile rispetto ad uno esterno, infatti gli identificatori esterni vengono tradotti in chiaviche includono gli identificatori delle entità coinvolte nell'identificazione esterna;
 - un identificatore che viene utilizzato da molte operazioni per accedere alle occorrenze di un'entità è da preferire rispetto agli altri.
 - se nessuno degli identificatori candidati soddisfa i criteri viene introdotto un nuovo attributo *codice* generato appositamente per identificare le occorrenze delle entità.

Traduzione verso il modello relazionale

Questa seconda fase corrisponde a una traduzione tra modelli di dati diversi: a partire dallo schema E-R ristrutturato (senza generalizzazioni e attributi multivaleure, con un solo identificatore) si costruisce uno schema logico **equivalente**, in grado di rappresentare le stesse informazioni.

- entità e associazioni molti a molti**

- **Relazioni binarie:**



la sua traduzione nel modello relazionale prevede:

- per ogni entità:
 - una relazione con lo stesso nome;
 - avente come attributi gli stessi dell'entità;
 - per chiave il suo identificatore.
- per ogni associazione:
 - una relazione con lo stesso nome;
 - avente per attributi gli stessi dell'associazione;
 - gli identificatori delle entità coinvolte che formano la chiave della relazione.

schema relazionale corrispondente:

Impiegato(Matricola, Cognome, Stipendio)

Progetto(Codice, Nome, Budget)

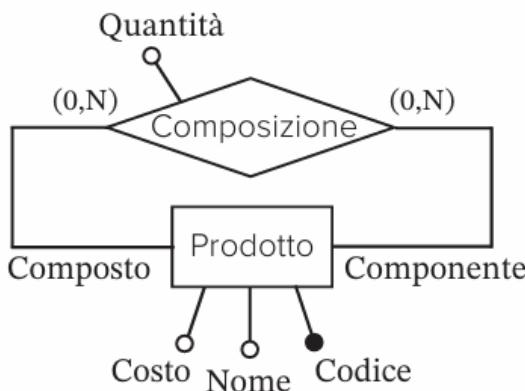
Partecipazione(Matricola, Codice, DataInizio)

Per rendere più comprensibile il significato dello schema è conveniente effettuare alcune ridenominazioni: **Partecipazione**(Impiegato, Progetto, DataInizio)

C'è un vincolo di integrità referenziale tra:

- Matricola in **Partecipazione** e la chiave di **Impiegato**;
- Codice in **Partecipazione** e la chiave di **Progetto**;

- **Relazioni ricorsive:**



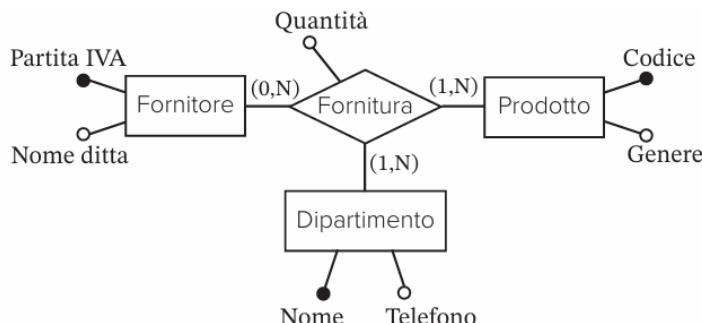
schema relazionale corrispondente:

Prodotto(Codice, Nome, Costo)

Composizione(Composto, Componente, Quantità)

gli attributi Composto e Componente contengono codici di prodotti: il primo ha il secondo come componente.

- **Associazioni con più di due entità:**



si traducono in maniera analoga alle associazioni binarie:

Fornitore(PartitaVA, NomeDitta)

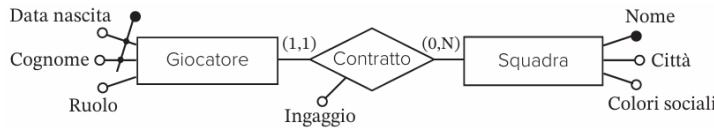
Prodotto(Codice, Genere)

Dipartimento(Nome, Telefono)

Fornitura(Fornitore, Prodotto, Dipartimento, Quantità)

- **associazioni uno a molti**

- **associazione binaria**



schema relazionale corrispondente:

Giocatore(Cognome, DataNascita, Ruolo)

Squadra(Nome, Città, ColoriSociali)

Contratto(Giocatore, DataNascitaGiocatore, NomeSquadra, Ingaggio)

In **Contratto** la chiave è costituita solo dall'identificatore di **Giocatore** perché la sua cardinalità implica che ogni giocatore ha un contratto con una sola squadra. Dal momento che **Giocatore** e **Contratto** hanno la stessa chiave è possibile fonderle in un'unica relazione:

Giocatore(Cognome, DataNascita, Ruolo, NomeSquadra, Ingaggio)

Squadra(Nome, Città, ColoriSociali)

Con questa soluzione:

- abbiamo meno relazioni;
 - è possibile avere valori nulli sugli attributi NomeSquadra e Ingaggio.
- Vincolo di integrità referenziale tra:
- NomeSquadra di **Giocatore** e Nome di **Squadra**.

- **associazioni ternarie**

L'**entità** che partecipa all'associazione ternaria con cardinalità massima uguale a 1, viene tradotta in una relazione che contiene anche gli identificatori delle altre entità coinvolte nell'associazione. Se l'entità **Prodotto** nell'esempio di associazione ternaria precedente avesse cardinalità (1,1), quindi per ogni prodotto esiste un solo fornitore e un solo dipartimento al quale viene fornito, allora:

Fornitore(PartitaVA, NomeDitta)

Dipartimento(Nome, Telefono)

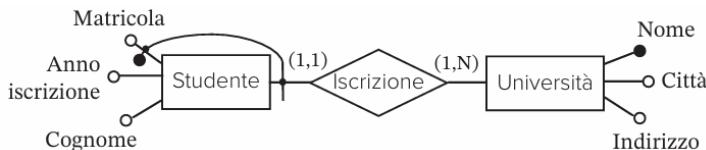
Prodotto(Codice, Genere, Fornitore, Dipartimento, Quantità)

Vincoli di integrità referenziale:

- tra l'attributo Fornitore della relazione **Prodotto** e l'attributo PartitaVA di **Fornitore**;
- tra l'attributo Dipartimento della relazione **Prodotto** e l'attributo Nome della relazione **Dipartimento**.

- **entità con identificatore esterno**:

danno luogo a relazioni con chiavi che includono gli identificatori delle entità "identificanti".



schema relazionale:

Studente(Matricola, NomeUniversità, Cognome, AnnoIscrizione)

Università(Nome, Città, Indirizzo)

vincolo di integrità referenziale:

- tra l'attributo NomeUniversità della relazione **Studente** e l'attributo Nome di **Università**
 Rappresentando l'identificatore esterno si rappresenta direttamente anche l'associazione,
 infatti le entità identificate esternamente partecipano all'associazione sempre con una
 cardinalità minima e massima pari a 1. Inoltre, questo tipo di traduzione è valido
 indipendentemente dalla cardinalità con cui l'altra entità partecipa all'associazione.

- associazioni uno a uno**

diverse possibilità di traduzione:

- partecipazioni obbligatorie per entrambe le entità*



due possibilità:

1. **Direttore(Codice, Cognome, Stipendio, DipartimentoDiretto, InizioDirezione)**
Dipartimento(Nome, Telefono, Sede)
 2. **Direttore(Codice, Cognome, Stipendio)**
Dipartimento(Nome, Telefono, Sede, Direttore, InizioDirezione)
- E' possibile rappresentare l'associazione in una qualunque delle relazioni che rappresentano le due entità.

- partecipazione opzionale per una sola entità*



una sola soluzione è preferibile rispetto alle altre:

Impiegato(Codice, Cognome, Stipendio)

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

in quanto in questo modo non è possibile avere valori nulli.

- partecipazione opzionale*

Impiegato(Codice, Cognome, Stipendio)

Dipartimento(Nome, Telefono, Sede)

Direzioni(Direttore, Dipartimento, DataInizioDirezione)

- non presenta mai valori nulli sugli attributi dell'associazione;
 - abbiamo una relazione in più \Rightarrow rende più complessa la base di dati.
- Questa soluzione è da prendere in considerazione solo se il numero di occorrenze dell'associazione è molto basso rispetto alle occorrenze delle entità che partecipano all'associazione.

Normalization

La "forma normale" è una proprietà di un database relazionale che ne garantisce la qualità. Quando una relazione non è in forma normale:

- presenta ridondanze;
- può avere comportamenti indesiderati durante gli aggiornamenti;

La **normalizzazione** deve essere usata come una tecnica di verifica per testare il risultato del design della base di dati, non è una metodo per il design del database.

Esempio:

| Impiegato | Stipendio | Progetto | Bilancio | Funzione |
|-----------|-----------|----------|----------|-------------|
| Rossi | 20 000 | Marte | 2000 | Tecnico |
| Verdi | 35 000 | Giove | 15 000 | Progettista |
| Verdi | 35 000 | Venere | 15 000 | Progettista |
| Neri | 55 000 | Venere | 15 000 | Direttore |
| Neri | 55 000 | Giove | 15 000 | Consulente |
| Neri | 55 000 | Marte | 2000 | Consulente |
| Mori | 48 000 | Marte | 2000 | Direttore |
| Mori | 48 000 | Venere | 15 000 | Progettista |
| Bianchi | 48 000 | Venere | 15 000 | Progettista |
| Bianchi | 48 000 | Giove | 15 000 | Direttore |

Le tuple di questa relazione rispettano queste proprietà:

- lo stipendio di ogni impiegato è unico ed è funzione del solo impiegato, indipendentemente dai progetti cui partecipa;
- il bilancio di ciascun progetto è unico e dipende dal solo progetto, indipendentemente dagli impiegati che partecipano.

Anomalie:

- **Ridondanza:** il valore dello stipendio di ciascun impiegato è ripetuto in tutte le tuple;
- **Anomalia di aggiornamento:** se lo stipendio di un impiegato varia, è necessario andarne a modificare il valore in tutte le tuple corrispondenti, comporta la necessità di più modifiche contemporaneamente;
- **Anomalia di cancellazione:** se un impiegato interrompe la partecipazione a tutti i progetti senza lasciare l'azienda, tutte le tuple corrispondenti vengono eliminate e non è possibile conservare traccia del suo nome e del suo stipendio;
- **Anomalia di inserimento:** se si hanno informazioni su un nuovo impiegato, non è possibile inserire finché non viene assegnato ad un progetto.

Queste anomalie sono causate dal fatto che è stata utilizzata una sola relazione per rappresentare relazioni che riuniscono concetti fra loro disomogenei.

Dipendenze Funzionali

Sono un particolare vincolo di integrità che descrive legami di tipo funzionale tra gli attributi di una relazione. Formalizzato:

Def

Data una relazione r su uno schema $R(X)$ e due sottoinsiemi non vuoti Y e Z di X , esiste su r una dipendenza funzionale tra Y e Z se, per ogni coppia di tuple t_1 e t_2 di r aventi gli stessi valori sugli

attributi Y , risulta che t_1 e t_2 hanno gli stessi valori anche sugli attributi di Z .

Una dipendenza funzionale tra Y e Z viene generalmente indicata con $Y \rightarrow Z$.

Viene associata ad uno schema: una relazione su quello schema verrà considerata corretta se soddisfa tale dipendenza funzionale.

Nel nostro esempio: $\text{Impiegato} \rightarrow \text{Stipendio}$

$\text{Progetto} \rightarrow \text{Bilancio}$

$\text{Impiegato Progetto} \rightarrow \text{Funzione}$

Osservazioni:

se l'insieme Z è composto dagli attributi A_1, A_2, \dots, A_k , allora una relazione soddisfa $Y \rightarrow Z \Leftrightarrow$ soddisfa tutte le k dipendenze $Y \rightarrow A_1, Y \rightarrow A_2, \dots, Y \rightarrow A_k$.

Diremo che una dipendenza funzionale $Y \rightarrow A$ è **non banale** se A non compare tra gli attributi di Y .

$\text{Impiegato Progetto} \rightarrow \text{Funzione}$ è una dipendenza **banale**.

Osservazione sulle dipendenze funzionali e il legame con la chiave:

Se prendiamo una chiave K di una relazione r , si verifica facilmente che esiste una dipendenza funzionale tra K e ogni altro attributo dello schema di r ; infatti, per definizione stessa di vincolo di chiave, non possono esistere due tuple con gli stessi valori su K .

Nel nostro esempio:

- le prime due dipendenze funzionali non sono chiavi e causano anomalie;
- la terza dipendenza funzionale ($\text{Impiegato Progetto}$) è una chiave e non causa anomalie;

Forma normale di Boyce-Codd

Def

Una relazione r è in *forma normale di Boyce-Codd* se per ogni dipendenza funzionale (non banale) $X \rightarrow A$ definita su di essa, X contiene una chiave K di r , cioè X è superchiave per r .

Anomalie e ridondanze non si presentano per relazioni in forma normale di Boyce-Codd, perché i concetti indipendenti sono separati, uno per relazione.

Decomposizione in BCNF

Data una relazione che non soddisfa la BCNF è possibile, in molti casi, sostituirla con due o più relazioni normalizzate attraverso un processo detto di **normalizzazione**: se una relazione rappresenta più concetti indipendenti, allora va decomposta in relazioni più, piccole, una per ogni concetto.

| Impiegato | Stipendio |
|-----------|-----------|
| Rossi | 20 000 |
| Verdi | 35 000 |
| Neri | 55 000 |
| Mori | 48 000 |
| Bianchi | 48 000 |

| Progetto | Bilancio |
|----------|----------|
| Marte | 2000 |
| Giove | 15 000 |
| Venere | 15 000 |

| Impiegato | Progetto | Funzione |
|-----------|----------|-------------|
| Rossi | Marte | Tecnico |
| Verdi | Giove | Progettista |
| Verdi | Venere | Progettista |
| Neri | Venere | Direttore |
| Neri | Giove | Consulente |
| Neri | Marte | Consulente |
| Mori | Marte | Direttore |
| Mori | Venere | Progettista |
| Bianchi | Venere | Progettista |
| Bianchi | Giove | Direttore |

Nell'esempio vengono costruite delle nuove relazioni in modo che a ciascuna dipendenza corrisponda una diversa relazione la cui chiave è proprio il primo membro della dipendenza stessa.

In molti casi, la decomposizione può essere effettuata producendo tante relazioni quante sono le dipendenze funzionali definite; in generale, le dipendenze possono avere una struttura complessa: può non essere necessario, o possibile, basare la decomposizione su tutte le dipendenze e può essere difficile individuare quelle su cui si deve basare la decomposizione.

| Employee | Project | Office |
|----------|---------|--------|
| Jones | Mars | Rome |
| Smith | Jupiter | Milan |
| Smith | Venus | Milan |
| White | Saturn | Milan |
| White | Venus | Milan |

Employee → Office

| Employee | Office |
|----------|--------|
| Jones | Rome |
| Smith | Milan |
| White | Milan |

Project → Office

| Project | Office |
|---------|--------|
| Mars | Rome |
| Jupiter | Milan |
| Venus | Milan |
| Saturn | Milan |



| Employee | Office |
|----------|--------|
| Jones | Rome |
| Smith | Milan |
| White | Milan |

| Office | Project |
|--------|---------|
| Rome | Mars |
| Milan | Jupiter |
| Milan | Venus |
| Milan | Saturn |

| Employee | Office | Project |
|----------|--------|---------|
| Jones | Rome | Mars |
| Smith | Milan | Jupiter |
| Smith | Milan | Venus |
| Smith | Milan | Saturn |
| White | Milan | Jupiter |
| White | Milan | Saturn |
| White | Milan | Venus |

| Employee | Office | Project |
|----------|--------|---------|
| Jones | Rome | Mars |
| Smith | Milan | Jupiter |
| Smith | Milan | Venus |
| White | Milan | Saturn |
| White | Milan | Venus |

DIFFERENT FROM THE ORIGINAL RELATION!

Decomposizione senza perdita

Def

Caso Generale: data una relazione r su un insieme di attributi X , se X_1 e X_2 sono due sottoinsiemi di X la cui unione sia pari a X stesso, allora il join delle due relazioni ottenute per proiezione da r su X_1 e X_2 , rispettivamente, è una relazione che contiene tutte le tuple di r , più eventualmente altre che possiamo chiamare "spurie". Diciamo che r si *decompon*se senza perdita su X_1 e X_2 se il join delle due proiezioni è uguale a r stessa (cioè non contiene tuple spurie).

Condizione per la decomposizione senza perdita:

Sia r una relazione su X e siano X_1 e X_2 sottoinsiemi di X tali che $X_1 \cup X_2 = X$;

inoltre, sia $X_0 = X_1 \cap X_2$;

allora: r si decomponse senza perdita su X_1 e X_2 se soddisfa la dipendenza funzionale $X_0 \rightarrow X_1$, oppure la dipendenza funzionale $X_0 \rightarrow X_2$.

In altre parole, r si decomponse senza perdita su due relazioni se l'insieme degli attributi comuni alle due relazioni è chiave per almeno una delle relazioni composte.

Conservazione delle dipendenze

Per garantire che tutte le dipendenze funzionali dello schema originale siano ancora rispettate nella decomposizione è necessario che: ogni dipendenza funzionale dello schema originale coinvolga attributi che compaiono tutti in almeno uno degli schemi della decomposizione; questo assicura che i vincoli e le relazioni tra i dati originali rimangano validi anche dopo la decomposizione.

Esempio:

- schema originale: $\mathbf{R}(A, B, C)$
- dipendenza funzionale: $A \rightarrow B$, ovvero il valore dell'attributo A determina univocamente il valore dell'attributo B
- decomposizione:
 1. $\mathbf{R}_1(A, B)$
 2. $\mathbf{R}_2(A, C)$
- verifica:
 - dopo la decomposizione lo schema $\mathbf{R}_1(A, B)$ conserva la dipendenza $A \rightarrow B$ perché entrambi gli attributi A e B sono inclusi in \mathbf{R}_1 .
⇒ la decomposizione conserva le dipendenze funzionali dello schema originale.

Qualità delle decomposizioni

- La **decomposizione senza perdita** garantisce che le informazioni nella relazione originaria siano ricostruibili con precisione;
- la **conservazione delle dipendenze** garantisce che le relazioni decomposte hanno la stessa capacità della relazione originaria di rappresentare i vincoli di integrità e quindi di rilevare aggiornamenti illeciti: a ogni aggiornamento lecito sulla relazione originaria corrisponde un aggiornamento lecito sulle relazioni decomposte.

Terza forma normale

- **limitazioni BCNF:**

| Chief | Project | Office |
|---------|---------|--------|
| Smith | Mars | Rome |
| Johnson | Jupiter | Milan |
| Johnson | Mars | Milan |
| White | Saturn | Milan |
| White | Venus | Milan |

Project Office → Chief

Chief → Office

la relazione non è in BCNF perché il primo membro della dipendenza **Chief → Office** non è superchiave. Inoltre non è possibile decomporre bene questa relazione in quanto la dipendenza **Project Office → Chief** coinvolge tutti gli attributi.

⇒ Talvolta la forma normale di Boyce-Codd non è raggiungibile.

Terza Forma Normale:

Diciamo che una relazione r è in *terza forma normale* se, per ogni dipendenza funzionale (non banale) $X \rightarrow A$ definita su di essa, almeno una delle seguenti condizioni è verificata:

- X contiene una chiave K di r ;
- A appartiene ad almeno una chiave di r .

- la BCNF è più forte della 3NF (la 3NF ammette relazioni con anomalie);
- 3NF può essere sempre raggiunta;
- Se una relazione ha una sola chiave, è in BCNF se e solo se è in 3NF.

Infatti, la dipendenza **Project Office → Chief** ha come primo membro una chiave della relazione, mentre **Chief → Office**, pur non contenendo una chiave al primo membro, ha un unico attributo a secondo membro che fa parte della chiave **Project → Office**.

Decomposizione in terza forma normale

Una relazione che non soddisfa la terza forma normale si decompone in relazioni ottenute per proiezione sugli attributi corrispondenti alle dipendenze funzionali, con l'accortezza di mantenere sempre una relazione che contiene una chiave della relazione originaria.

1. identificare le dipendenze funzionali nello schema
2. decomporre lo schema originale in più schemi, in modo che ogni schema soddisfi i requisiti della 3NF;
3. assicurarsi che la decomposizione conservi le dipendenze e che sia possibile ricostruire lo schema originale usando i sottoschemi (proprietà di lossless join).

Esempio:

$R(A, B, C)$

dipendenze funzionali sono:

4. $A \rightarrow B$

5. $B \rightarrow C$

dove A è la chiave primaria;

- la dipendenza $B \rightarrow C$ crea una dipendenza transitiva:
- $A \rightarrow B$ e $B \rightarrow C \Rightarrow A \rightarrow C$, ma C dipende indirettamente da A tramite B
dunque questo schema non è 3NF:
- creiamo due schemi:
 - $R_1(A, B)$: per rappresentare $A \rightarrow B$
 - $R_2(B, C)$: per rappresentare $B \rightarrow C$
- verifica:
 - schema $R_1(A, B)$: È in 3NF, perché B dipende direttamente dalla chiave primaria A;
 - Schema $R_2(B, C)$: È in 3NF, perché C dipende direttamente dalla chiave primaria B.
Unendo R_1 e R_2 , possiamo ricostruire lo schema originale senza perdita di dati.

Teoria delle dipendenze

Data una relazione e un insieme di dipendenze funzionali su di essa, generare una decomposizione di tale relazione che contenga solo le relazioni in forma normale che soddisfino le proprietà della decomposizione di cui abbiamo già parlato:

- decomposizione senza perdita;
- preservazione delle dipendenze.

Def

Diciamo che un insieme di dipendenze funzionali F implica un'altra dipendenza f se ogni relazione che soddisfa tutte le dipendenze in F soddisfa anche f .

Esempio: $A \rightarrow B$, $B \rightarrow C \Rightarrow A \rightarrow C$

Chiusura di un insieme di attributi

Siano dati uno schema di relazione $R(U)$ e un insieme di dipendenze funzionali F definite sugli attributi in U . Sia X un insieme di attributi contenuti in U (cioè $X \subseteq U$); la *chiusura* di X rispetto a F , indicata con X_F^+ , è l'insieme degli attributi che dipendono funzionalmente da X (esplicitamente o implicitamente):

$$X_F^+ = \{A \mid A \in U \text{ e } F \text{ implica } X \rightarrow A\}$$

Se vogliamo vedere se $X \rightarrow A$ è implicata da F , basta vedere se A appartiene a X_F^+ , a patto di saper calcolare X_F^+ .

Esiste un algoritmo per il calcolo di X_F^+ :

Input: un insieme X di attributi e un insieme F di dipendenze.

Output: un insieme X_P di attributi.

1. Inizializziamo X_P con l'insieme di input X .
2. Esaminiamo le dipendenze in F ; se esiste una dipendenza $Y \rightarrow A$ con $Y \subseteq X_P$ e $A \notin X_P$ allora aggiungiamo A a X_P .
3. Ripetiamo il passo 2 fino al momento in cui non vi sono ulteriori attributi che possono essere aggiunti a X_P .

Il concetto di chiusura X_F^+ è utile anche per formalizzare il legame fra il concetto di dipendenza funzionale e quello di chiave:

Def

un insieme di attributi K è chiave per uno schema di relazione $R(U)$ su cui è definito un insieme di dipendenze funzionali F se F implica $K \rightarrow U$. Di conseguenza, l'algoritmo mostrato può essere utilizzato per verificare se un insieme è chiave.

Un insieme F è:

- *non ridondante* se non esiste dipendenza $f \in F$ tale che $F - \{f\}$ implica f ;
- *ridotto* se è non ridondante e non esiste un insieme F' equivalente a F ottenuto eliminando attributi dai primi membri di una o più dipendenze di F .

Esempi:

- $F_1 = \{A \rightarrow B, AB \rightarrow C, A \rightarrow C\}$

- $F_2 = \{A \rightarrow B, AB \rightarrow C\}$

- $F_3 = \{A \rightarrow B, A \rightarrow C\}$

$\Rightarrow F_1$ è ridondante, perché $\{A \rightarrow B, AB \rightarrow C\}$ implica $A \rightarrow C$; F_1 è equivalente a F_2 ;

$\Rightarrow F_2$ è non ridondante ma non è ridotto, perché B può essere eliminato dal primo membro della seconda dipendenza: F_2 è equivalente a F_3 ;

$\Rightarrow F_3$ è ridotto.

Due insiemi di dipendenze funzionali F_1 e F_2 sono equivalenti se F_1 implica ogni dipendenza in F_2 e viceversa.

Se due insiemi sono equivalenti diciamo che sono uno copertura del altro.

Calcolare la copertura minima

Per trovare una copertura minima non ridondante esaminiamo ripetutamente le dipendenze dell'insieme dato, eliminando quelle implicate da altre, si procede in tre passi:

1. sostituiamo l'insieme dato con quello equivalente che ha tutti i secondi membri costituiti da singoli attributi;
2. eliminiamo le dipendenze ridondanti;
3. per ogni dipendenza verifichiamo se esistono attributi eliminabili dal primo membro: se F è l'insieme corrente, per ogni dipendenza $Y \rightarrow A \in F$, verifichiamo se esiste $Y \subseteq X$ tale che F è equivalente a $F - \{X \rightarrow A\} \cup \{Y \rightarrow A\}$.

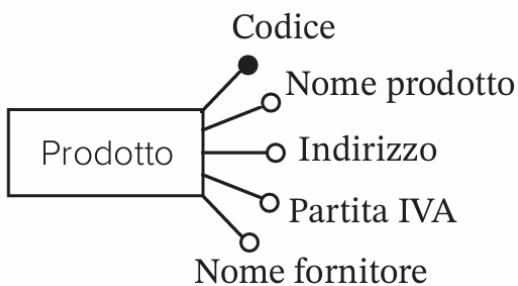
Algoritmo di sintesi per la terza forma normale

Def

Uno schema di relazione $R(U)$ con l'insieme di dipendenze F è in *terza forma normale* se, per ogni dipendenza funzionale (non banale) $X \rightarrow A \in F$, almeno una delle seguenti condizioni è verificata:

- X contiene una chiave K di r : cioè $X_F^+ = U$;
- A è contenuto in almeno una chiave di r : esiste un insieme di attributi $K \subseteq U$ tale che $K_F^+ = U$ e $(K - A)_F^+ \subset U$.

Verifiche di normalizzazione sulle entità

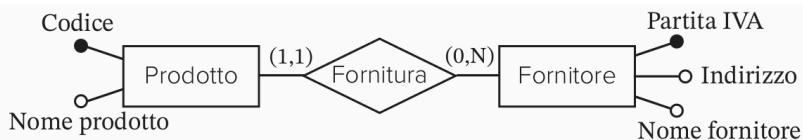


- Partita IVA → NomeFornitore, Indirizzo

Tutti gli attributi dipendono funzionalmente da Codice, ovvero l'identificatore di Prodotto.

⇒ l'entità viola la terza forma normale perché la dipendenza *Partita IVA* → *NomeFornitore*, *Indirizzo* ha un primo membro che non contiene l'identificatore e un secondo membro composto da attributi che non fanno parte della chiave.

Decomposizione:



Indexes & B+trees

Indici

L'indicizzazione è una tecnica di ottimizzazione utilizzata per velocizzare le interrogazioni, gli indici sono una struttura dati che contiene informazioni complementari che supportano un accesso efficiente ai dati. La chiave di ricerca è definita utilizzando alcuni attributi; le chiavi di ricerca non sono chiavi primarie, infatti la stessa chiave può contenere più valori.

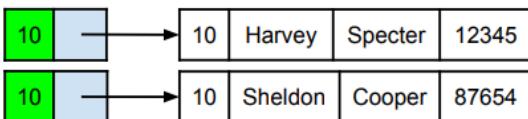
Un indice è una coppia *<key, label>* e supporta il recupero di tutte le etichette con un dato valore K in modo efficiente.

Le *etichette* (labels) possono essere:

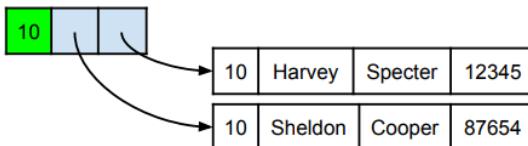
1. il dato stesso;

| | | | |
|----|---------|---------|-------|
| 10 | Harvey | Specter | 12345 |
| 10 | Sheldon | Cooper | 87654 |

2. l'identificatore del dato (RID) con il valore K della chiave;



3. una lista di identificatori dello stesso valore con chiave K;



La rappresentazione delle etichette è indipendente dal metodo di ricerca.

Osservazioni:

- in una base di dati, è possibile avere al massimo un solo indice sui dati utilizzando la prima rappresentazione;
- utilizzando la prima rappresentazione, la dimensione dell'indice è la stessa dei dati;
- la stessa chiave di ricerca può contenere più valori;
- la terza rappresentazione è la soluzione più compatta, ma le etichette hanno dimensioni variabili.

In SQL:

```
//per creare indici
CREATE [UNIQUE] INDEX IndexName ON Table(AttributeList)

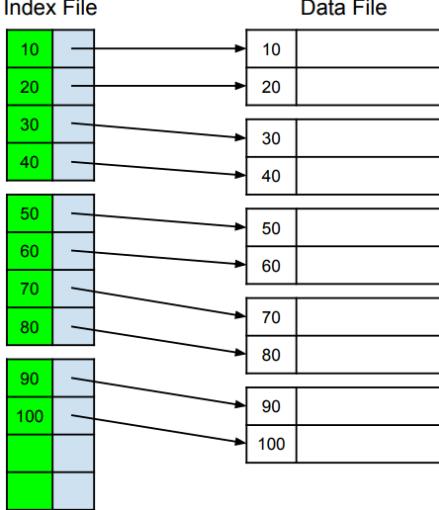
//per eliminare un indice
DROP INDEX IndexName
```

Classificazione

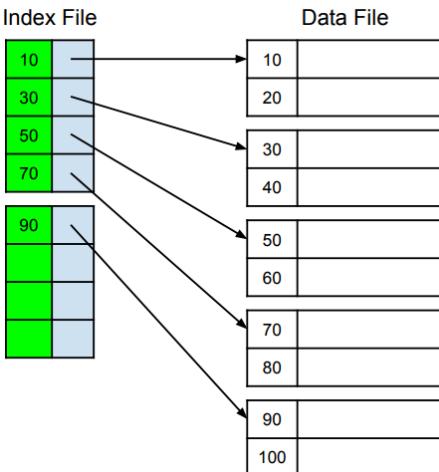
- **indice primario:**
 - è un indice basato su un insieme di attributi che include la chiave primaria;
 - i dati sono ordinati in base a questi attributi;
 - se l'indice non è basato sulla chiave primaria, allora è un **indice secondario**.
- **indice denso:**
 - ogni valore chiave di ricerca nel file dei dati ha almeno una voce corrispondente nell'indice;
 - se non tutti i valori chiave di ricerca sono rappresentati, allora l'indice è **sparso**.
- **indice clusterizzato:**
 - l'ordine dei record nel file dati corrisponde (o è simile) all'ordine delle etichette (chiavi) nell'indice;
 - se l'ordine dei record non corrisponde, allora l'indice è **non clusterizzato**.

Esempi:

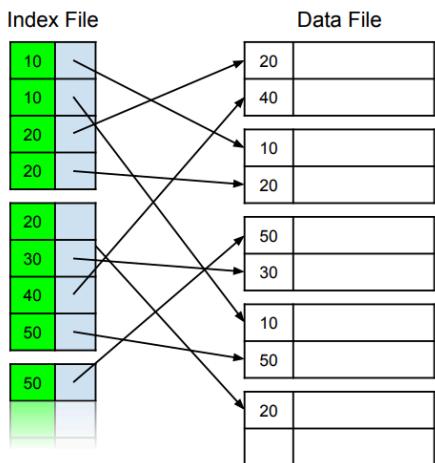
- indice clusterizzato e denso:



- indice sparso e clusterizzato:



- indice secondario, denso e non clusterizzato



gli indici non clusterizzati danno meno efficienza nell'accesso ai dati (ad esempio tre record con lo stesso valore sono memorizzati in tre blocchi diversi).

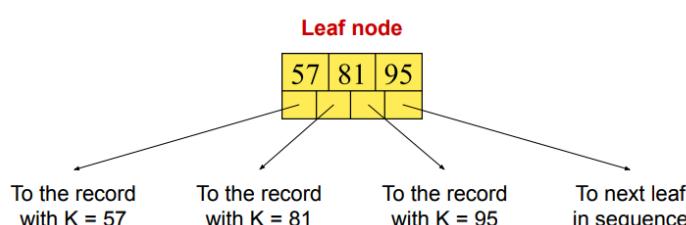
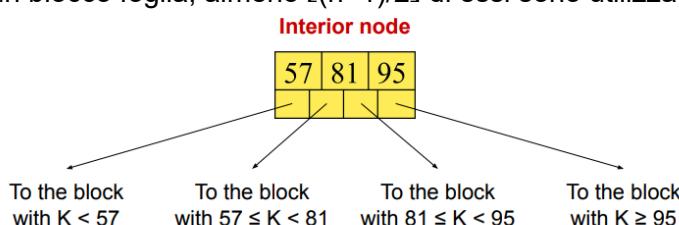
B+trees

Le strutture ad albero dinamiche di tipo B+trees (un tipo speciale di B-alberi), sono le più frequentemente usate nei DBMS relazionali per la realizzazione degli indici.

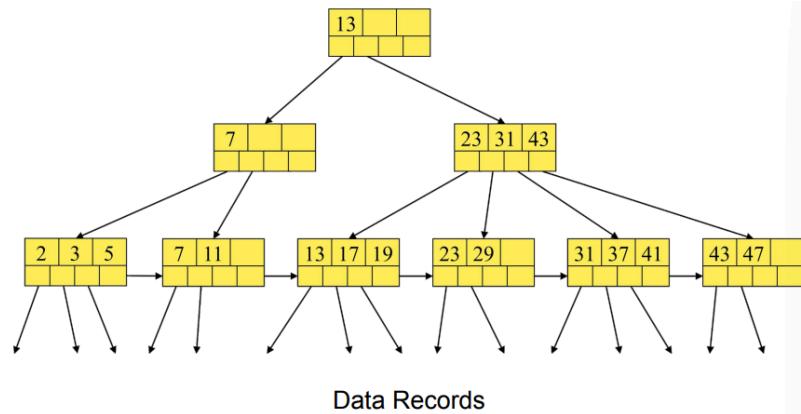
- Ogni albero è caratterizzato da un nodo radice, vari nodi intermedi e vari nodi foglia;
- ogni nodo ha un numero di discendenti che dipende dall'ampiezza della pagina;
- gli alberi sono **bilanciati**, ovvero la lunghezza di un cammino che collega il nodo radice a un qualunque nodo foglia è costante; in questo modo il tempo di accesso alle informazioni contenute nell'albero è lo stesso per tutte le foglie ed è pari alla profondità dell'albero.

Regole:

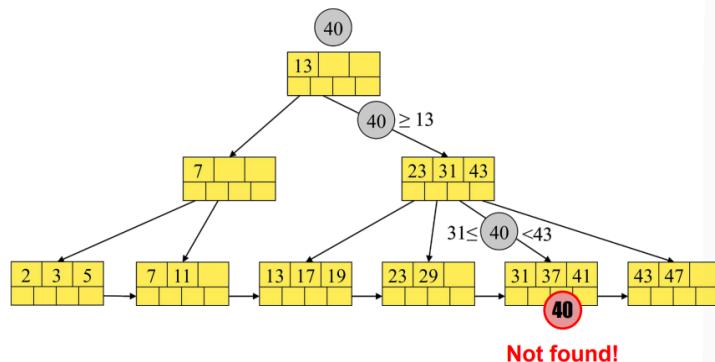
- le chiavi nei nodi foglia sono copie delle chiavi del data file. Queste chiavi sono distribuite tra le foglie in modo ordinato, da sinistra a destra.
- alla radice, ci sono almeno due puntatori utilizzati (con almeno due record di dati nel file). Tutti i puntatori puntano ai blocchi del livello sottostante;
- in presenza di n chiavi, bisogna avere n+1 puntatori;
- In un nodo interno, tutti i puntatori utilizzati puntano a blocchi al livello immediatamente inferiore e almeno $\lceil (n+1)/2 \rceil$ devono essere utilizzati;
- in una foglia, l'ultimo puntatore punta al blocco foglia successivo a destra. Tra gli altri puntatori in un blocco foglia, almeno $\lceil (n+1)/2 \rceil$ di essi sono utilizzati e puntano a un record di dati.



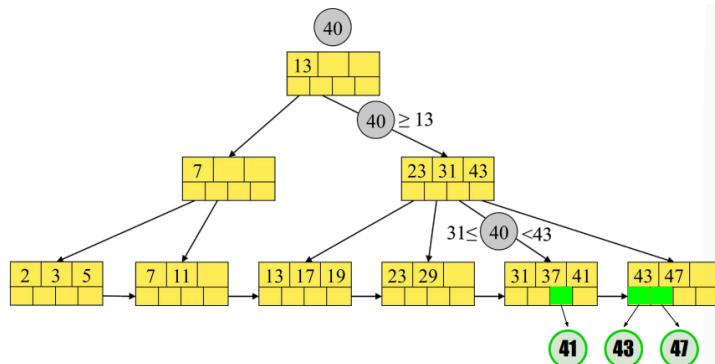
Esempio di B+tree:



Esempio di **equality search**:



Esempio di **range search**:



Inserimento:

L'inserimento è, in linea di principio, ricorsivo:

1. trovare il nodo corretto:

- per inserire un valore, si parte dalla radice e si scende lungo l'albero seguendo le chiavi fino a trovare il nodo foglia che dovrebbe contenerlo.

2. inserire il valore nel nodo foglia:

- se il nodo foglia ha spazio disponibile (cioè non ha già $n-1$ chiavi), semplicemente aggiungi la nuova chiave in ordine crescente.

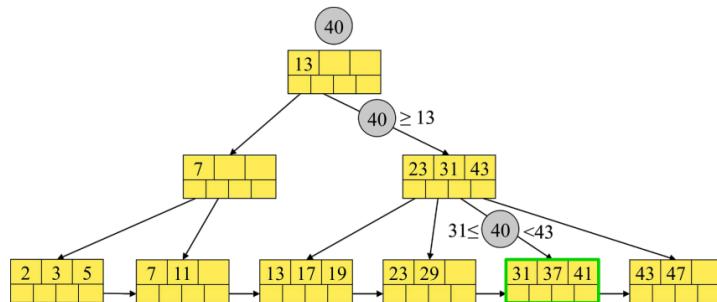
3. gestione dell'overflow se il nodo è pieno:

- se il nodo foglia è pieno, viene diviso in due nodi (split):
 - le chiavi vengono divise in due gruppi di dimensioni approssimativamente uguali;
 - la chiave centrale viene promossa al nodo genitore (il nodo superiore)
- se il genitore non ha spazio per la chiave promossa, si ripete il processo di divisione verso l'alto (propagazione dello split).

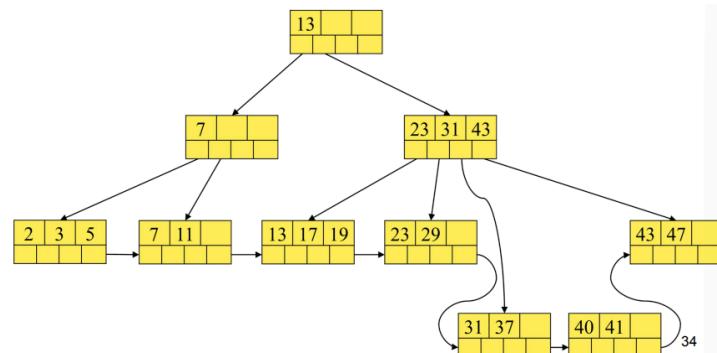
4. aggiornare la struttura dell'albero:

- se anche la radice deve essere divisa (overflow nella radice), si crea una nuova radice con due figli. L'altezza dell'albero aumenta di 1.

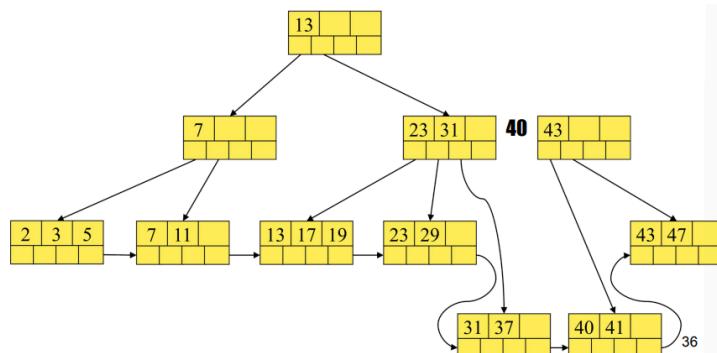
Esempio, inseriamo 40:

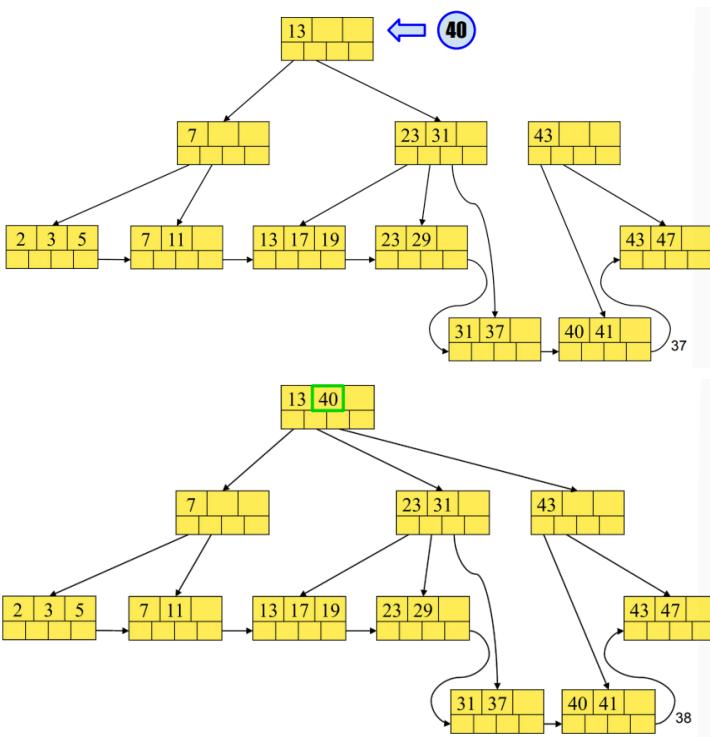


non c'è spazio nella foglia quindi, dobbiamo dividere:



lo split di una foglia al livello inferiore, corrisponde all'inserimento di una nuova coppia chiave-puntatore al livello superiore:





Eliminazione:

Dobbiamo garantire che l'albero rimanga bilanciato e rispetti tutte le sue proprietà.

1. trovare il nodo contenente la chiave da eliminare:

- partendo dalla radice, scendere lungo l'albero seguendo le chiavi per trovare il nodo foglia, non è necessario toccare le chiavi nei nodi intermedi immediatamente, in quanto i nodi intermedi contengono solo "puntatori" (chiavi-guida).

2. eliminare la chiave:

- se la foglia, dopo l'eliminazione della chiave, contiene ancora il numero minimo di chiavi, non dobbiamo fare più nulla;

3. problema di sotto-riempimento:

ci sono due modi per risolvere questo problema:

1. prestito da un fratello:

- se un nodo adiacente (fratello) ha più del numero minimo di chiavi, gli "prendiamo in prestito" una chiave.

- aggiorniamo anche la chiave-guida nel nodo genitore per riflettere i cambiamenti.

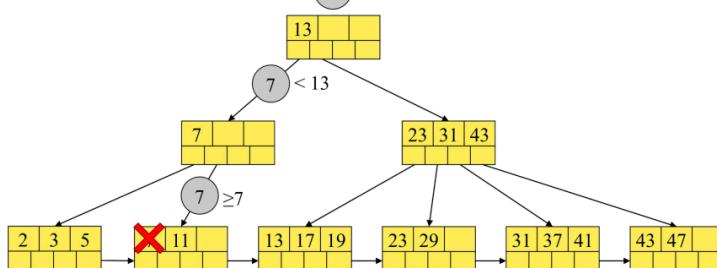
2. fusione con un fratello:

- se nessun fratello ha chiavi in eccesso, fondiamo il nodo sotto-riempito con un fratello adiacente.

- la chiave-guida nel genitore viene abbassata e inclusa nel nodo fuso.

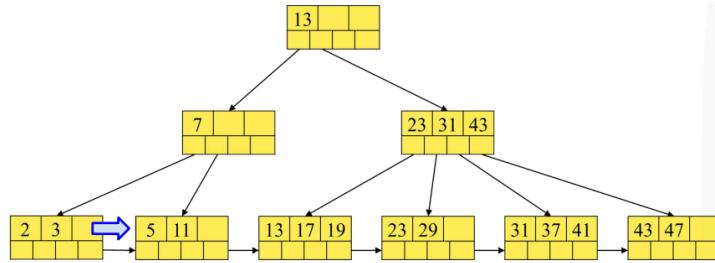
- se il genitore rimane sotto-riempito, si applica ricorsivamente lo stesso processo al genitore.

Esempio: cerchiamo la chiave con valore 7 e la eliminiamo:

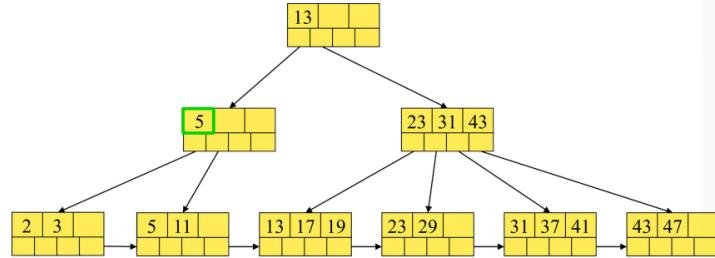


in questo modo la seconda foglia ha solo una chiave, mentre abbiamo bisogno di almeno due

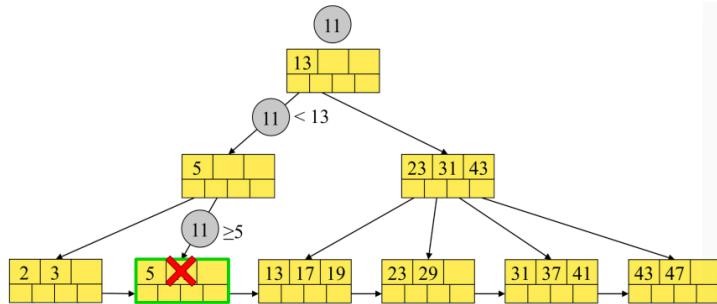
chiavi in ogni foglia; allora il nodo di sinistra "presta" una chiave al nodo di destra:



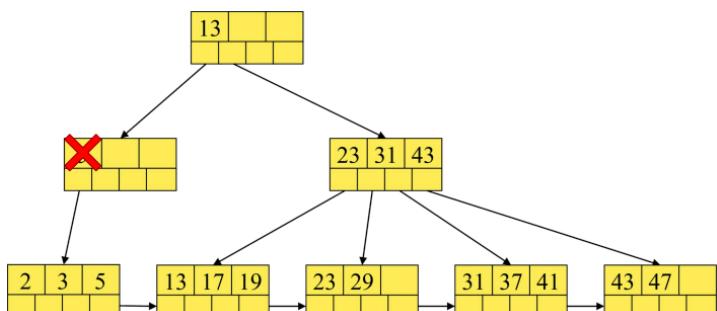
e aggiorniamo la "chiave-guida":



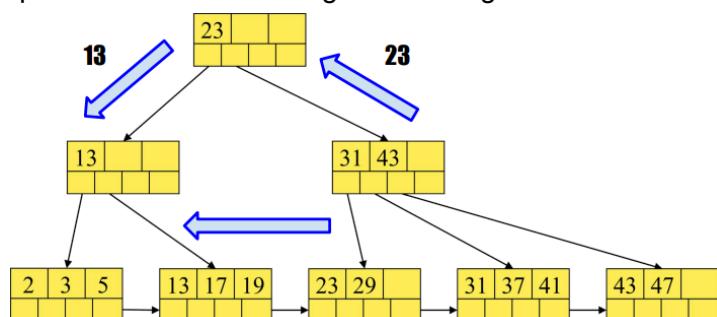
Secondo esempio: vogliamo eliminare la chiave con valore 11



Non possiamo prendere in prestito dalla prima foglia e non c'è nessun fratello a destra da cui prendere in prestito, perciò abbiamo bisogno di fondere la seconda foglia con un fratello (la prima):



I puntatori e le chiavi nel genitore vengono sistemate in modo da riflettere la situazione nei figli:



Search complexity:

$$\log_{(n/2)} N \leq L \leq \log_{(n/2)} N + 1$$

Let's assume that:

- Block size: 4096 B.
- Key size: 4 B.
- Pointer size: 8 B.
- There is no header information kept on the blocks.

The value of n is:

- A. 340
- B. 4096
- C. 48
- D. 8

Hint: we want to find the largest integer value of n such that

$$4n + 8(n + 1) < 4096$$

Hash table and Inverted Indexes

Indici basati su hashing

Gli indici basati su hashing utilizzano una funzione hash per mappare chiavi di ricerca e posizioni specifiche nei bucket di memoria, ottimizzando le operazioni di ricerca e aggiornamento:

- *vantaggio*: perfetti per ricerche di uguaglianza
- *svantaggio*: non supportano in modo efficiente le ricerche su intervalli
Si suddividono in due categorie principali:

1. **Static Hashing**: la dimensione del bucket è fissa, il che lo rende inadatto a dati dinamici;
2. **Dynamic Hashing**: tecniche come l'hashing estendibile e lineare permettono di gestire l'aumento dei dati senza degradare le prestazioni.

Hashing Statico

Definizione e funzione hash

- utilizza N bucket, ognuno identificato da un numero compreso tra 0 e $N-1$;
- una funzione hash $H(K)$ mappa ogni chiave di ricerca K a un bucket specifico. Ad esempio, $H(K) = i$, con i compreso tra 0 e $N-1$;
- la funzione hash può basarsi su un prefisso o un suffisso dei bit del valore della chiave. Ad esempio, la funzione H , restituisce i bit più significativi (o meno significativi) del valore binario della chiave.

Inserimento

- un record con chiave K viene inserito nel bucket $H(K)$;
- se un bucket è pieno, viene utilizzata una **catena di overflow**, che collega più blocchi per accogliere i record aggiuntivi.

Ricerca

- per trovare un record con chiave K , la funzione hash calcola il bucket $H(K)$;
- il sistema cerca nei blocchi del bucket (inclusi quelli di overflow, se presenti).

Cancellazione

- quando un record viene eliminato, si possono rimuovere blocchi di overflow, ma questo richiede attenzione per garantire che i dati rimanenti siano correttamente gestiti

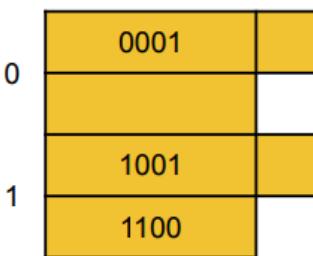
Esempio:

Supponiamo di avere $N = 2$ bucket, $i = 1$ bit per la funzione hash H_1 . I bucket sono numerati 0 e 1, e ogni bucket contiene un blocco.

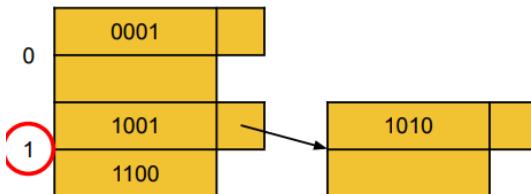
Esempi di chiavi e bucket:

- chiave $K = 0001 \rightarrow H_1(0001) = 0$ (va nel bucket 0);

- chiave $K = 1100 \rightarrow H_1(1100) = 1$ (va nel bucket 1);
- chiave $K = 1001 \rightarrow H_1(1001) = 1$ (va nel bucket 1);

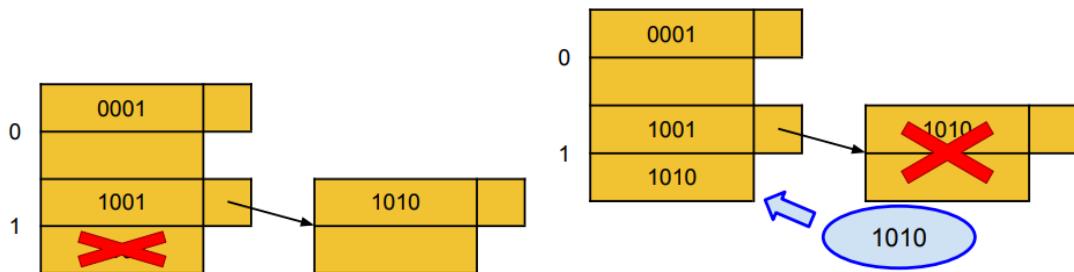


Supponiamo di voler aggiungere una nuova chiave K che genera però un overflow nel bucket, ad esempio $K = 1010$, aggiungiamo allora un blocco di overflow.



Supponiamo invece di voler rimuovere la chiave $K = 1100$, questo ci consente:

- in presenza di blocchi di overflow, di eliminarli per liberare spazio;
- di rimuovere un blocco se dopo l'eliminazione questo rimane vuoto.



Efficienza dell'hashing statico:

Dipende da:

1. Numero di bucket:

- se i bucket sono sufficienti a contenere i dati senza generare overflow, la ricerca richiede un solo accesso al disco;
- catene di overflow lunghe degradano le prestazioni, richiedendo accessi multipli al disco.

2. Distribuzione delle chiavi:

- una funzione hash ben progettata distribuisce uniformemente le chiavi nei bucket;
- una distribuzione sbilanciata (ad esempio, molte chiavi che generano lo stesso valore hash) causa lunghe catene di overflow.

Limiti dell'hashing statico

1. **Dimensione fissa:** non è possibile modificare il numero di bucket, questo rende il metodo inadatto per dataset in crescita o che cambiano frequentemente.
2. **Gestione inefficiente dello spazio:** se la dimensione dei dati è molto inferiore al numero di bucket, alcuni rimangono vuoti, sprecando spazio.
3. **Mancanza di supporto per ricerche su intervalli:** l'hashing statico è progettato per ricerche di uguaglianza (es. "trova il record con chiave K "), ma non supporta efficientemente ricerche su

intervalli (es. "trova tutti i record con chiavi tra $K1$ e $K2$ ").

Hashing Estendibile

E' una tecnica di hashing dinamico progettata per superare i limiti dell'hashing statico, come la dimensione fissa dei bucket e la gestione inefficiente di dataset dinamici.

Principi base

L'hashing estendibile utilizza una **directory di puntatori** ai bucket che può crescere dinamicamente. Ogni bucket è identificato da un **prefisso di bit** delle chiavi memorizzare:

- **profondità globale** (GD): numero di bit usati per indirizzare i bucket nella directory;
- **profondità locale** (LD): numero di bit usati per distinguere i record all'interno di un bucket.

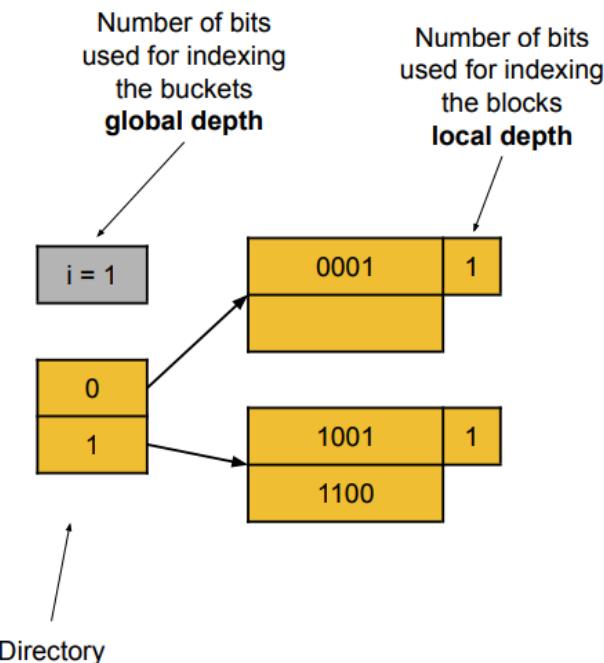
Struttura

1. Directory:

- contiene 2^{GD} puntatori ai bucket;
- la dimensione della directory aumenta raddoppiando quando GD viene incrementato.

2. Bucket:

- ogni bucket è associato a un prefisso di lunghezza LD;
- un bucket può essere condiviso da più voci della directory se $LD < GD$.



Operazioni:

Inserimento:

- la funzione hash calcola un indice basato su GD bit della chiave;
- se il bucket corrispondente ha spazio, il record viene inserito;
- se il bucket è pieno:
 1. Si controlla se $LD < GD$:
 - il bucket viene **diviso**. I record vengono ridistribuiti tra due nuovi bucket in base al bit successivo della chiave;

- LD del bucket viene incrementato.

2. Se $LD = GD$:

- la directory viene raddoppiata;
- GD viene incrementato, e i puntatori nella directory vengono aggiornati per riflettere la nuova configurazione.

Ricerca:

- la funzione hash calcola l'indice del bucket nella directory utilizzando GD bit;
- si accede direttamente al bucket corretto tramite il puntatore della directory.

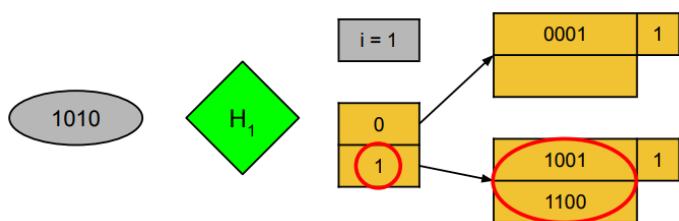
Cancellazione:

- il record viene eliminato dal bucket corrispondente;
- se un bucket diventa vuoto, può essere combinato con un altro bucket (se condividono lo stesso prefisso e LD è maggiore del minimo necessario).

Esempio:

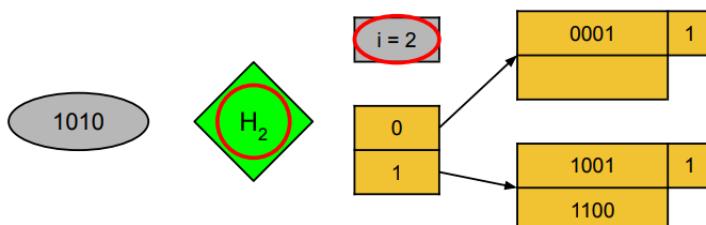
Chiamiamo GD $\rightarrow i$ e LD $\rightarrow j$.

Supponiamo di avere una directory iniziale con $GD = 1$ (1 bit) e due bucket (0 e 1). Ogni bucket può contenere al massimo due chiavi. Ora vogliamo inserire la chiave $K_2 = 1010$:

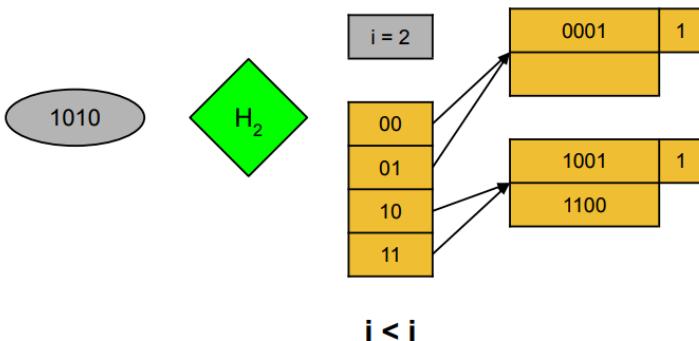


There is no room!

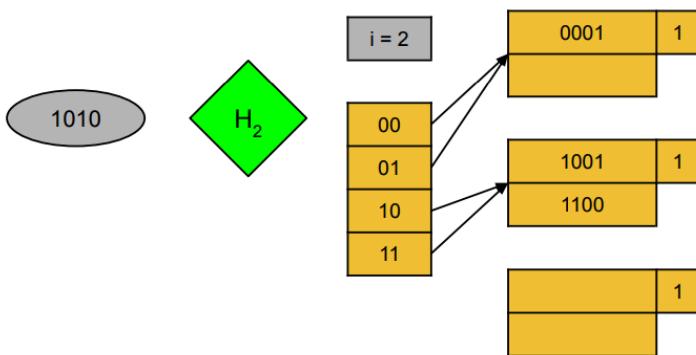
Controlliamo i valori i e j , dal momento che $i = j$ dobbiamo incrementare i :



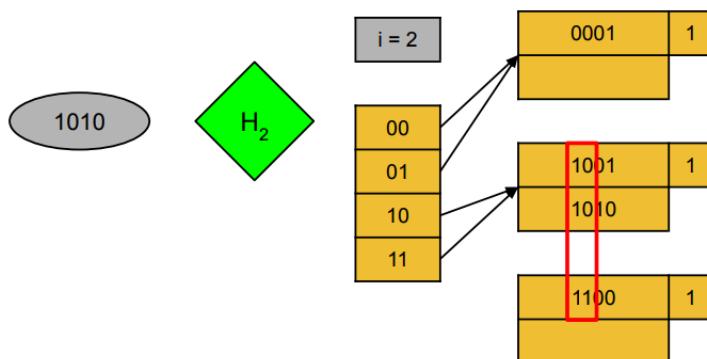
Adesso $i = 2$ e la directory raddoppia; il bucket 1 viene diviso in due nuovi bucket (10 e 11) :



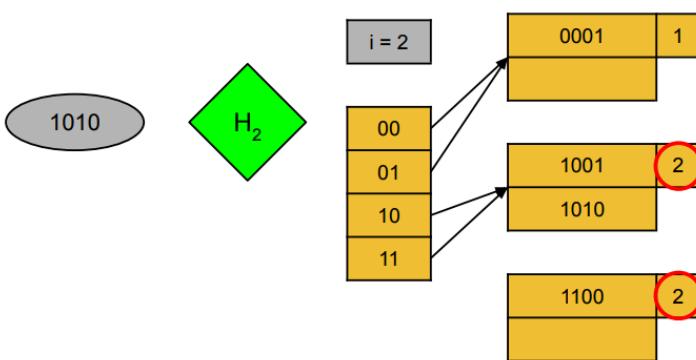
il blocco B viene ulteriormente diviso:



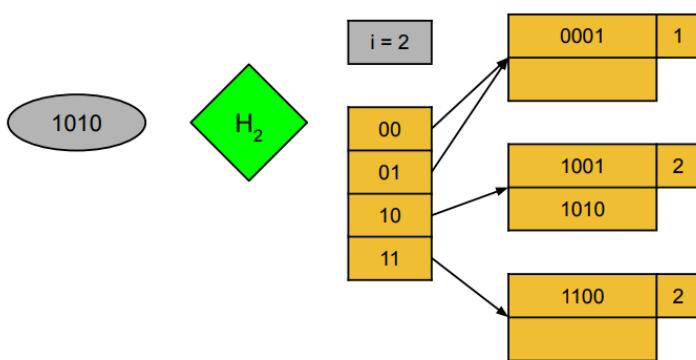
i record vengono ridistribuiti utilizzando $j+1$ bit:



il valore di j viene incrementato di 1:



infine, la directory viene aggiornata con il puntatore al nuovo blocco:



Gestione delle profondità:

1. Profondità globale (GD):

- indica quanti bit vengono usati dalla directory per reindirizzare i bucket;
- se un bucket pieno ha $LD = GD$, la directory deve essere raddoppiata.

2. Profondità locale (LD):

- determina quanti bit del prefisso sono condivisi dalle chiavi in un bucket;
- se un bucket viene diviso, LD viene incrementato.

Vantaggi e Svantaggi:

Vantaggi:

- **espansione dinamica**: la directory cresce solo quando necessario;
- **efficienza**: la ricerca rimane veloce anche con dataset in crescita;
- **riduzione degli overflow**: i bucket pieni vengono divisi anziché usare blocchi di overflow.

Svantaggi:

- la dimensione della directory può diventare molto grande se i dati sono distribuiti in modo non uniforme;
- aumento della complessità rispetto all'hashing statico.

Hashing lineare

E' una tecnica di hashing dinamico progettata per gestire dataset in crescita in modo flessibile, evitando gli inconvenienti di dimensioni fisse e riducendo la necessità di raddoppiare bruscamente la memoria, come nell'hashing estendibile.

Concetti fondamentali:

Struttura base:

- inizia con un numero n di bucket, organizzati in ordine sequenziale;
- i dati vengono distribuiti tra i bucket utilizzando una funzione hash $h(k)$;
- a differenza dell'hashing statico, la struttura può crescere gradualmente.

Hashing e split:

1. Funzione hash:

- inizialmente, si usa una funzione hash $h_0(k) = k \bmod 2^i$, dove i è il livello corrente di suddivisione;
- man mano che i bucket si riempiono, alcuni vengono divisi e si utilizza una funzione hash $h_1(k) = k \bmod 2^{i+1}$ per ridistribuire i dati.

2. Gestione dei bucket:

- l'hashing lineare mantiene un puntatore P che identifica quale bucket deve essere diviso successivamente;
- quando un bucket è pieno, il puntatore P avanza e il bucket viene suddiviso.

Operazioni:

Inserimento

1. conta dei record e dei bucket:

- si contano il numero di record r e il numero di bucket n attualmente utilizzati nella tabella hash.

2. controllo della condizione di split:

- se il rapporto r/n (numero medio di record per bucket) supera la soglia di 1.7, si aggiunge un nuovo bucket, $(n + 1)$ -esimo bucket.

3. divisione dei bucket:

- si utilizza una funzione hash H_i , che serve a distribuire i record ne bucket;

- tutti i bucket fino a 2^{i-1} -esimo vengono suddivisi secondo l'ordine in cui sono stati creati, indipendentemente da quale bucket abbia causato la divisione.

4. cambio della funzione hash:

- se n supera 2^i , ovvero il numero massimo gestibile dall'attuale funzione hash H_i , allora si passa alla funzione hash successiva H_{i+1} , e il processo di split ricomincia dal primo bucket.

5. inserimento normale:

- se la funzione hash $H_i(K)$ restituisce m , dove $m < n$:
 - il record con chiave K viene inserito nel bucket m ;
 - se il bucket m è pieno, si crea un overflow block (struttura temporanea per gestire l'eccesso).

6. inserimento durante lo split:

- se $H_i(K)$ restituisce m , ma $m \leq n$ (il bucket m è stato "spostato" a causa dello split):
 - si calcola un nuovo bucket $m' = (m - 2^{i-1})$ per posizionare il record;
 - anche qui, se il bucket è pieno, si crea un overflow block.

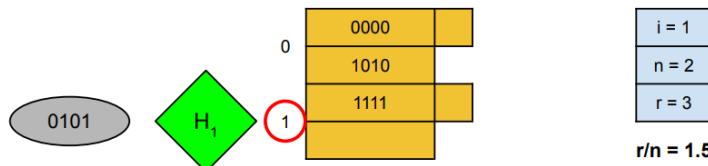
7. incremento e gestione dello split:

- se il rapporto $r/n > 1.7$, si procede con lo split:
 - si controlla se $n = 2^i$. In tal caso, si aumenta l'indice i della funzione hash.
 - si esegue lo split:
 1. si calcola il nuovo bucket $n_2 = a_1a_2\dots a_i$, dove $a_1 = 1$;
 2. si trasferiscono i record dal bucket m al nuovo bucket n , seguendo un controllo sui bit della funzione hash;
 3. si aggiunge il nuovo bucket n .
 - si incrementa n di 1.

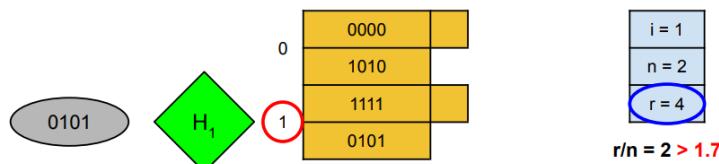
esempio:

vogliamo inserire la chiave 0101, N è il numero di bucket ($2^{i-1} < N \leq 2^i$):

- se $h_1(K) = m < n$, la chiave di ricerca viene inserita nel bucket m ;
 - se $h_1(K) = m \geq n$, la chiave di ricerca viene inserita nel bucket $m - 2^{i-1}$.
- $m = h_2(0101) = 1_2 = 1_{10} \Rightarrow m < n$

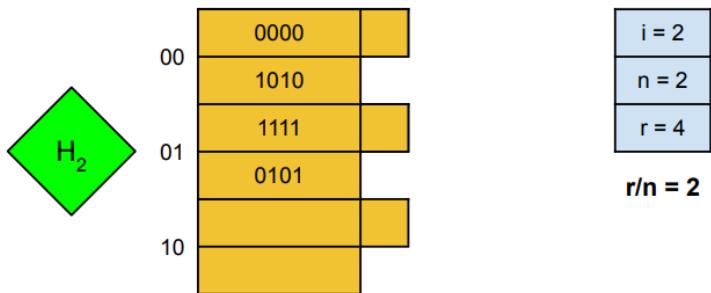


inseriamo la chiave nel bucket corretto e aggiorniamo r :



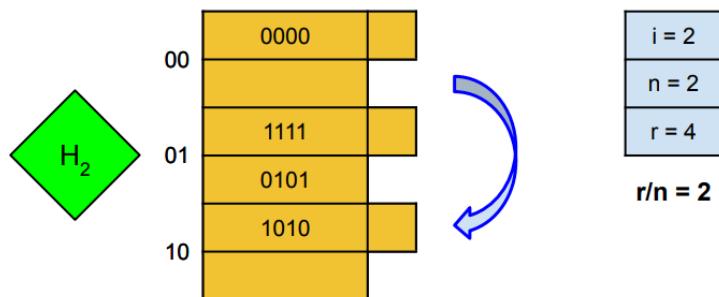
adesso però il rapporto $r/n > 1.7$ e $n = 2^i$, allora incrementiamo l'indice i :

- $n_2 = a_1a_2\dots a_i$ con $a_1 = 1 \Rightarrow n_2 = 10$
 - il primo bit in n viene "pulito" e memorizzato in m
- $a_1a_2\dots a_i \rightarrow 0a_2\dots a_i \Rightarrow m_2 = 00$
- aggiungiamo il bucket $n_2 = 10$:

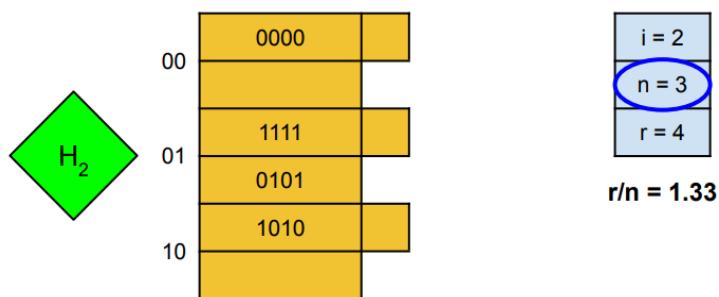


adesso spostiamo i record del bucket $m_2 = 0a_2a_3\dots a_i$ che hanno l'i-esimo bit più a destra uguale a 1:

- $n = 2_{10} = 10_2 (\equiv 1a_2a_3\dots a_i) \rightarrow 10_2$ identifica il nuovo bucket
- spostiamo i record da $00_2 (\equiv 0a_2a_3\dots a_i)$ a 10_2

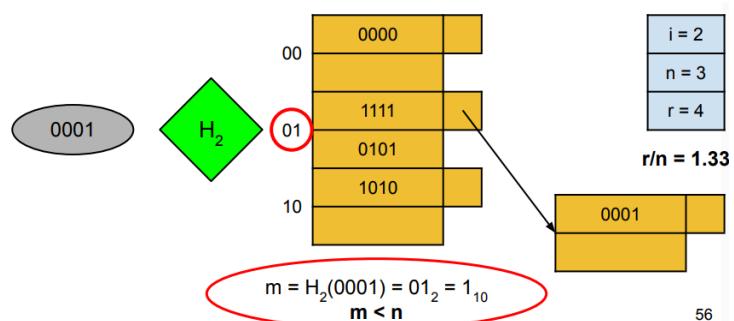


incrementiamo n:

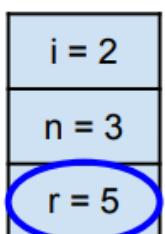


supponiamo adesso di voler inserire la chiave K = 0001

$$m = H_2(0001) = 01_2 = 1_{10} \Rightarrow m < n$$



dal momento che il bucket 01 è pieno viene creato un blocco di overflow,

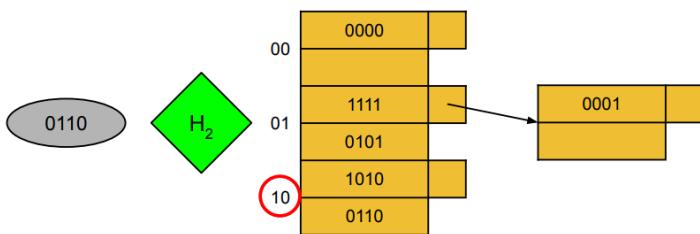


$$\mathbf{r/n = 1.66 < 1.7}$$

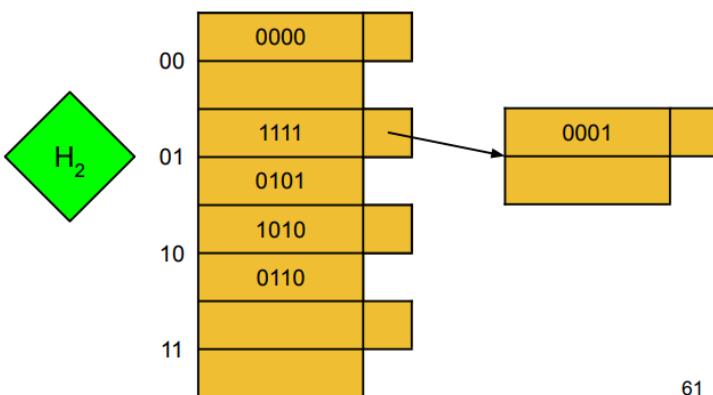
inoltre viene incrementato r:

ora supponiamo di voler aggiungere la chiave $K = 0110$

$$m = H_2(0110) = 10_2 = 2_{10} \Rightarrow m < n$$



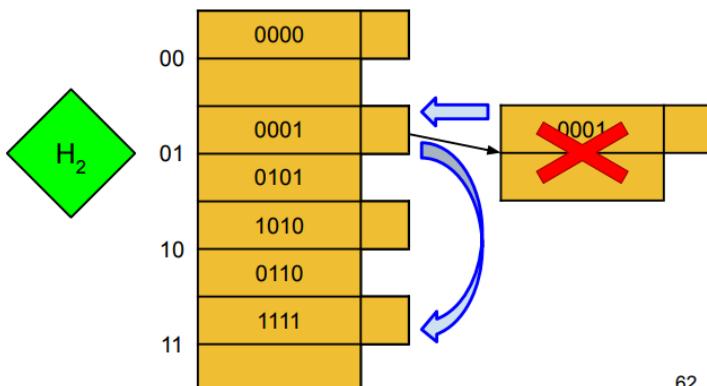
adesso però il rapporto $r/n = 2 > 1.7$ e $n \neq 2^i$, perciò non serve incrementare i ma basta aggiungere il bucket $n_2 = 11$:



61

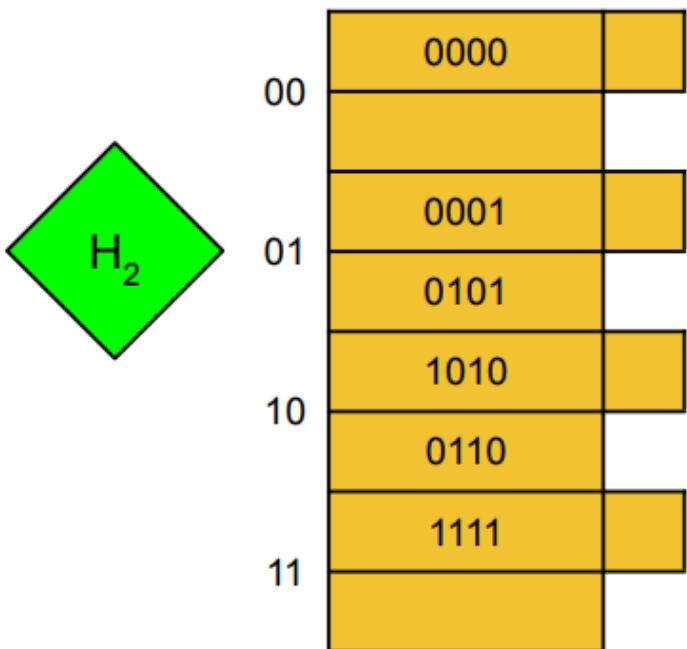
spostiamo nell'n-esimo bucket tutti i record dal bucket $0a_2a_3\dots a_i$ che hanno l'i-esimo bit più a destra uguale a 1:

- $n = 3_{10} = 11_2 (\equiv 1a_2a_3\dots a_i)$
- spostiamo da $01_2 (\equiv 0a_2a_3\dots a_i)$ a 11_2 .



62

situazione finale:



il rapporto $r/n = 1.5 < 1.7$.

Ricerca

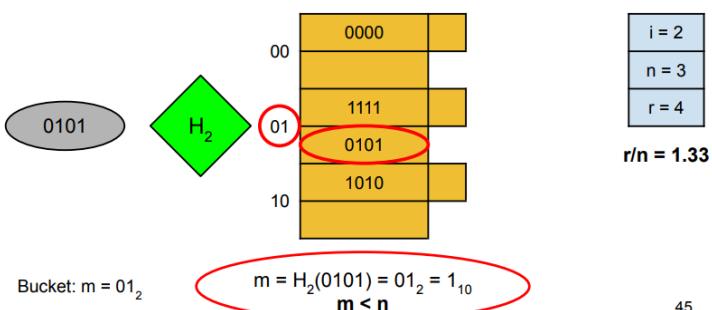
1. Calcola $h_0(k)$ e verifica se il record si trova nel bucket corrispondente;
2. in caso di overflow, cerca anche nel bucket creato durante la suddivisione.

Esempio:

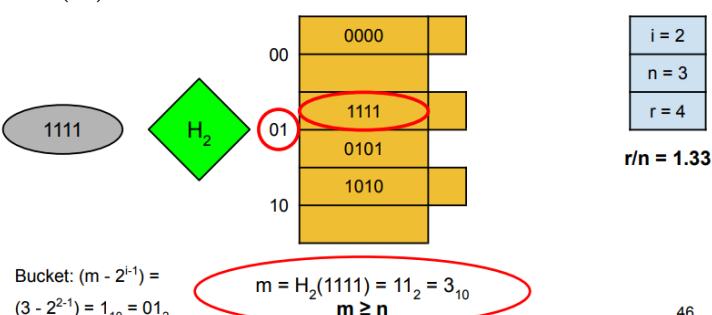
cerchiamo la chiave $K=0101$;

N numero di buckets (dove $2^{i-1} < N \leq 2^i$).

- se $h_i(K) = m < n$, la chiave di ricerca è nel bucket m :



- se $h_i(K) = m \geq n$, la chiave di ricerca è nel bucket $m - 2^{i-1}$:



Cancellazione

- elimina il record dal bucket;
- se un bucket diventa sottoutilizzato, non è previsto un accorpamento immediato, per mantenere l'efficienza.

Processo di suddivisione (Split)

Il processo di suddivisione dei bucket è il fulcro dell'hashing lineare.

Indici invertiti

Information Retrieval

L'information retrieval è il processo di ricerca e identificazione di documenti o informazioni rilevanti in un insieme di dati non strutturati.

applicazioni comuni:

- motori di ricerca (es. google);
- sistemi di ricerca per biblioteche digitali;
- sistemi di gestione documentale.

La ricerca di documenti basata su parole chiave è un problema complesso perché i documenti sono *non strutturati*, a differenza dei dati strutturati in tabelle.

Per i documenti non strutturati possiamo utilizzare tecniche come gli *indici invertiti*:

- mappano parole chiave ai documenti in cui appaiono;
- consentono di eseguire ricerche rapide su grandi raccolte di testo.

Un **indice invertito** è una struttura dati utilizzata per cercare rapidamente documenti non strutturati, come testo libero, basandosi su query che contengono parole chiave o frasi:

Def

Mappano ogni parola chiave (o termine) ai documenti in cui questa appare.

Invece di creare un indice per ciascun attributo o parola, viene costruito un indice invertito che rappresenta tutti i documenti in cui un termine specifico è presente.

Gli indici invertiti sono progettati per supportare due tipi principali di query:

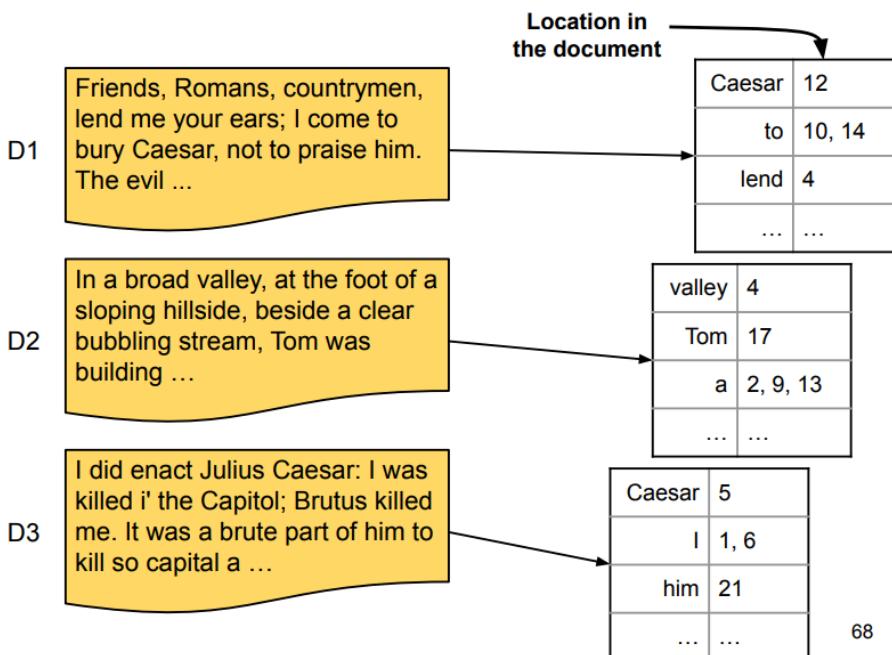
1. Query con set di parole chiave:

- *obiettivo*: recuperare tutti i documenti che contengono un dato insieme di parole chiave K_1, K_2, \dots, K_n .
- ad esempio: se cerchiamo i documenti che contengono "data" e "index":
 - l'indice invertito ci fornisce due liste di documenti una associata a "data" e una associata a "index";
 - le liste vengono intersecate per trovare i documenti che contengono entrambe le parole.

2. Query con sequenza di parole:

- *obiettivo*: recuperare tutti i documenti che contengono una sequenza precisa di parole chiave K_1, K_2, \dots, K_n .
- ad esempio: se cerchiamo "hash table performance":
 - l'indice invertito ci permette di verificare la posizione delle parole nei documenti;
 - restituisce solo i documenti in cui la sequenza delle parole è esattamente quella specificata.

Costruzione di un indice invertito:



We create an inverted index, consisting of a dictionary with pairs (document, position).

Document Terms:

| | |
|--------|-------------------------|
| a | (D2,2), (D2,9), (D2,13) |
| Brutus | (D3,12) |
| ... | ... |
| Caesar | (D1,12), (D3,5) |
| clear | (D2,14) |
| him | (D3,21) |
| I | (D3,1), (D3,6) |
| ... | ... |
| valley | (D2, 4) |

Inverted Index:

| | |
|--------|--------|
| Caesar | 12 |
| to | 10, 14 |
| lend | 1 |
| ... | ... |

| | |
|--------|----------|
| valley | 4 |
| Tom | 17 |
| a | 2, 9, 13 |
| ... | ... |

| | |
|--------|------|
| Caesar | 5 |
| I | 1, 6 |
| him | 21 |
| ... | ... |

69

Per migliorare l'efficienza e l'accuratezza delle ricerche attraverso l'uso degli indici invertiti abbiamo a disposizione tre tecniche che mirano a:

- velocizzare il recupero delle informazioni: ottimizzando la struttura dell'indice;
- migliorare la precisione dei risultati: eliminando elementi ridondanti o irrilevanti.

Token Normalization:

Def

La normalizzazione dei token consiste nel trasformare le parole in una forma standard, eliminando differenze superficiali tra le sequenze di caratteri.

Esempio:

- la parola "Windows" (con la maiuscola) viene trasformata in "windows" (w minuscola).

- questo permette di considerare equivalenti parole che differiscono solo per maiuscole/minuscole o altri aspetti formattivi.

Vantaggi:

- **uniformità:** riduce le variazioni superficiali, rendendo le ricerche più precise;
- **efficienza:** migliora il recupero di documenti perché non è necessario gestire forme diverse della stessa parola.

Stemming:



Lo stemming è il processo di riduzione delle parole alla loro radice o "stem", rimuovendo prefissi e suffissi.

Esempio:

- le parole "fishing", "fished" e "fisher" vengono ridotte alla radice comune "fish";
- i sostantivi plurali come "cars" possono essere trasformati nella loro forma singolare "car".

Vantaggi:

- **riduzione delle varianti:** consente di trattare parole con significati simili come equivalenti, migliorando il recupero di documenti rilevanti.
- **compressione:** riduce il numero di termini nell'indice, rendendolo più compatto.

Stop Words:



Le stop words sono le parole più comuni (come "the", "and", "of"...) che spesso non aggiungono valore semantico alla ricerca e vengono escluse dall'indice.

Motivazione: queste parole compaiono in quasi tutti i documenti, quindi non aiutano a distinguere documenti rilevanti da non rilevanti.

Esempio:

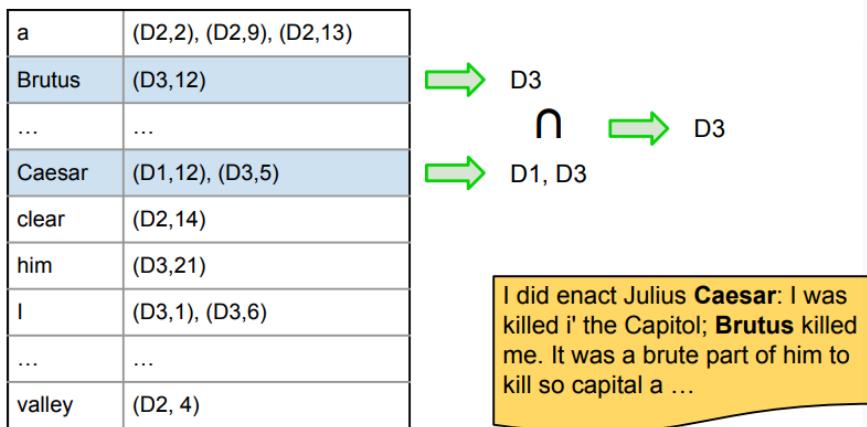
In una query come "find the best car", la parola "the" viene ignorata.

Vantaggi:

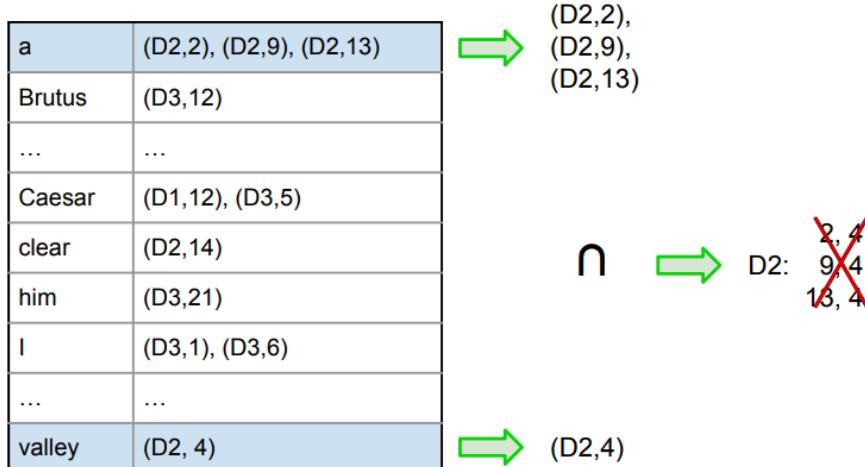
- **riduzione della dimensione dell'indice:** eliminando le stop words, l'indice diventa più piccolo e più veloce da interrogare.
- **precisione:** le query producono risultati più mirati eliminando parole poco significative.

Esempi:

Return all the documents that contain both the words “Brutus” and “Caesar”.



Return all the documents that contain the sequence “a valley”.



Return all the documents that contain the sequence “a clear”.

