DIPARTIMENTO
DI INFORMATICA
SAPIENZA
UNIVERSITÀ DI ROMA

Basi di Dati I

Author

Emanuele D'Agostino

GitHub

Rurik-D

AA 22/23

Indice

1 - Introduzione

INFORMAZIONI STRUTTURATE

Le informazioni in formato digitale possono essere memorizzate come:

- **dati strutturati** - gli oggetti rappresentati da brevi stringhe di simboli e numeri
- **dati non strutturati** - testo scritto in linguaggio naturale

INFORMATION SYSTEM (sistema informativo)

Un **Information System** è un **insieme di dati fisicamente organizzato in memorie secondarie** e gestito in modo tale da consentire la creazione e l'accesso ai dati.
Esso è un componente di un'organizzazione, dedito alla gestione (acquisizione, processamento, memorizzazione, comunicazione) delle informazioni.
Normalmente il **Sistema Informativo** opera in supporto ad altri componenti dell'organizzazione. La nozione di **Sistema Informativo** non coincide per forza con la sua "computerizzazione": un esempio di Information System può essere quello di un archivio.

DATABASE MANAGEMENT SYSTEM (DBMS)

Una **base di dati** (**database - DB**) è un **insieme di file mutualmente interconnessi** tra loro. Esso è inoltre una risorsa integrata condivisa tra varie componenti dell'Information System.
Un **file** è un contenitore di dati in formato digitale, tipicamente presente su un supporto digitale di memorizzazione. I dati, contenuti nei file, sono organizzati in strutture dati che facilitano la loro creazione e modifica, ottimizzando al contempo le risorse fisiche utilizzate.
Il ***DBMS*** (**Sistema di Gestione di Basi di Dati**) è uno **strumento software** che consente la **gestione di grandi masse di dati** residenti su memorie secondarie.

COMPONENTI DI UN INFORMATION SYSTEM

- **Database (DB)**
- **DBMS**
- **Application Software** che gestisce la comunicazione tra le componenti
- **Computer Hardware** dove vengono memorizzati fisicamente i dati
- **Personale** dedito allo sviluppo, alla gestione e alla fruizione dell'IS

L'integrazione e la condivisione tra le varie componenti consente di ridurre la ridondanza (dati parzialmente o totalmente replicati) e di conseguenza l'inconsistenza.

INFORMAZIONI CONDIVISE

La **condivisione di informazioni di un database non si interrompe mai**. Ciò consente il controllo della privacy dei dati e la regolazione degli accessi a tali dati.
La condivisione dei dati da parte di un database implica il bisogno di gestire accessi simultanei agli stessi dati: **controllo di concorrenza** (**control of concurrency**).

MODELLO DEI DATI

Strutture utilizzate per organizzare i dati di interesse e le loro relazioni.
2 tipi di modelli principiali:

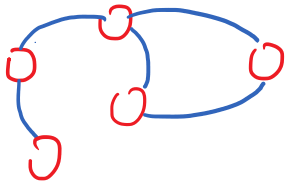
- **modelli logici** - **indipendenti dalle strutture fisiche** ma disponibili nei DBMS. (es. reticolari (network), gerarchici, relazionali, ad oggetti)
- **modelli concettuali** - **indipendenti dalle modalità di realizzazione**; hanno lo scopo di rappresentare le entità del mondo reale e le loro relazioni.

  Non esiste ancora però un modello universalmente riconosciuto.

MODELLO RETICOLARE (network)

Dati rappresentati come una collezione di **record** di tipo **omogeneo**, connessi tramite relazioni binarie (**link** implementati come **puntatori**).
Il modello è rappresentato come una struttura a grafo dove:

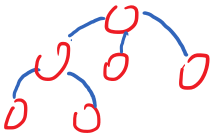
- **nodi** = **record**
- **archi** = **link**



Modello reticolare più famoso: **CODASYL**.

MODELLO GERARCHICO (albero)

Specifico tipo di modello reticolare: reticolo composto da una **collezione gerarchica di alberi**, dove ogni nodo ha al più un genitore.



MODELLO RELAZIONALE

Dati e relazioni sono rappresentati come **valori**.
Non ci sono riferimenti espliciti, cioè puntatori come nei modelli reticolare e gerarchico. Viene usata una rappresentazione di livello più alto "=>".

- **oggetto** = **record**
- **campi** = **informazioni di interesse**

Esempio:

CODICE	COGNOME	NOME	RUOLO	ASSUNZIONE
COD1	Rossi	Mario	Analista	1995
COD2	Verdi	Sara	Psicologa	2001

In questo caso ogni riga è un record di tipo "persona", mentre ogni campo contiene un informazione di interesse associata ai valori "codice, cognome, nome, ruolo, assunzione".

MODELLO A OGGETTI

Modello basato su oggetti e classi. Gli **attributi** descrivono le caratteristiche (lo **stato**) di un oggetto mentre i **metodi** ne descrivono i **comportamenti**. L'oggetto incapsula sia lo stato che i comportamenti.

I 3 LIVELLI DI ASTRAZIONE DI UN DB

- **Schema esterno** : descrizione di una porzione della base di dati in un modello logico attraverso **viste parziali o derivate**, che possono prevedere organizzazioni dei dati diverse rispetto a quelle utilizzate nello schema logico e che **riflettono esigenze e privilegi** di accesso di particolari tipologie di utenti.

- **Schema logico** : **descrizione dell'intera base di dati** nel modello logico **principale** del DBSM, ad esempio la struttura delle tabelle.

- **Schema fisico** : rappresentazione dello schema logico per mezzo di **strutture fisiche di memorizzazione**, cioè i file.

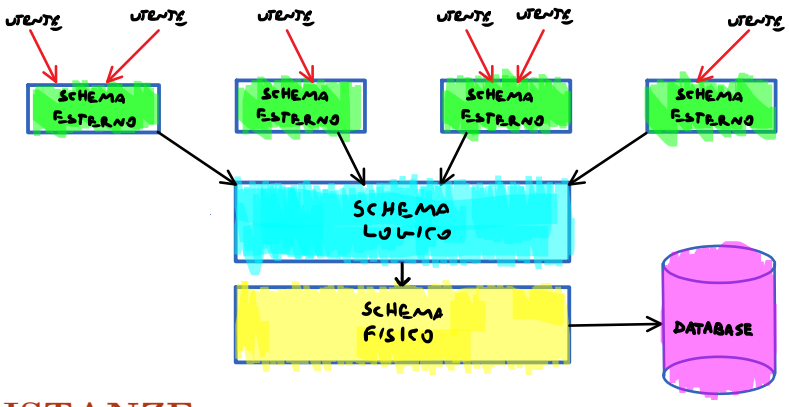
Gli accessi alla base di dati avvengono esclusivamente attraverso lo schema esterno, che in alcuni casi può coincidere con lo schema logico.

Il livello logico e quello esterno sono indipendenti da quello fisico **(indipendenza fisica):**

- Una relazione è usata allo stesso modo, qualunque sia la sua implementazione fisica (organizzazione e allocazione fisica dei file).
- La realizzazione fisica può cambiare senza che debbano cambiare anche i programmi.

Il livello esterno è indipendente da quello logico **(indipendenza logica):**

- Aggiunte o modifiche alle viste non richiedono modifiche a livello logico.
- Modifiche allo schema logico che lasciano inalterato lo schema esterno sono trasparenti.



SCHEMI E ISTANZE

In ogni base di dati esiste:

- lo **schema**, sostanzialmente **invariante nel tempo**, che ne descrive la struttura: nel modello relazionale sono le intestazioni delle tabelle (liste di attributi e loro tipi).
- l'**istanza**, i **valori correnti che possono cambiare** molto rapidamente: nel modello relazionale è il corpo di ciascuna tabella.

LINGUAGGI PER LE BASI DI DATI

LINGUAGGI:

- **DDL** (Data Definition Language) : per la definizione di schemi (esterni, logici, fisici) e altre operazioni generali.
- **DML** (Data Manipulation Language) : per l'interrogazione e l'aggiornamento di (istanze di) basi di dati.
- **SQL** (Structured Query Language) : linguaggio standardizzato per database basati sul modello relazionale RDBMS.
  In SQL i due tipi di funzionalità sono integrati in un unico linguaggio di comandi.

OPERAZIONI:

- **Transazione** - Sequenza di operazioni che costituiscono un'unica operazione logica.

  Es.
  - "Trasferisci €1000 dal cc c1 al cc c2"
    - cerca c1
    - modifica saldo in saldo-1000
    - cerca c2
    - modifica saldo in saldo+1000

  Una transazione deve essere **eseguita completamente** (**commited**) o non deve essere **eseguita affatto** (**rolled back**).

- **Ripristino -** Per ripristinare un valore corretto della base di dati:
  - **transaction log** - contiene i dettagli delle operazioni (valori precedenti e seguenti la modifica)
  - **dump** - copia periodica della base di dati

2 - Il Modello Relazionale

Il modello relazionale è un modello basato sulla nozione matematica di **relazione**. Una relazione può essere implementata come una **tabella** in cui ogni riga è una **tupla** della relazione, differente da ogni altra, ed ogni colonna corrisponde ad un **attributo** (valori omogenei, cioè provenienti dallo stesso dominio).

Definizioni

- **Dominio** - Insieme possibilmente infinito di valori (es. insieme dei numeri interi).
  Siano $D_1, D_2, \dots, D_k$ domini non necessariamente distinti.
  il prodotto cartesiano di tali domini denotato da

  $$D_1 x D_2 x \dots x D_k$$

  è l'insieme $\{(V_1, V_2, \dots, V_k) : V_1 \in D_1,\ V_2 \in D_2, \dots,\ V_k \in D_k\}$

  Generalmente si usano domini predefiniti comuni ai linguaggi di programmazione, ad esempio *String*, *Integer*, *Real*.

- **Relazione matematica** - Un qualsiasi sottoinsieme del prodotto cartesiano tra uno o più domini.

- **Relazione di grado k** - Una relazione che è sottoinsieme del prodotto cartesiano di k domini.

- **Tupla** - Una tupla t su R è una funzione che associa ad ogni attributo A_i in R un valore $t[A_i]$ nel corrispondente dominio $dom(A_i)$.

  Il numero di tuple di una relazione ne definisce la **cardinalità**.

  Ogni tupla di una relazione di grado k, contiene k valori ordinati (l'i-esimo valore deriva dall'i-esimo dominio), non c'è però ordinamento tra le tuple. Le tuple di una relazione sono tutte distinte, almeno per un valore.

- **Attributo** - Definito da un nome univoco A, che ne descrive il ruolo, e dal dominio dell'attributo A, indicato come $dom(A)$.
  Sia R un insieme di attributi, una tupla su R è una funzione definita su R che associa ad ogni attributo A in R un elemento di $dom(A)$.

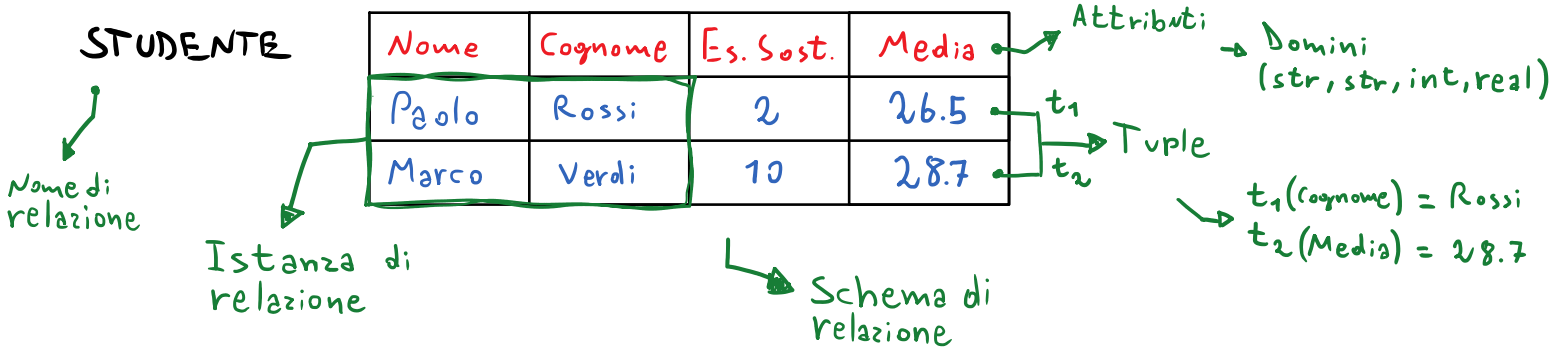
  Se t è una tupla su R ed A è un attributo di R, allora indichiamo con $\boldsymbol{t(A)}$ il valore assunto dalla funzione (tupla) t in corrispondenza dell'attributo A.

  Le prime lettere dell'alfabeto $(A, B, C\dots)$ denotano per convenzione singoli attributi, mentre le ultime $(X, Y\dots)$ denotano insiemi di attributi.
  Se X ed Y sono insiemi di attributi XY denota $X \cup Y$.

- **Schema di relazione** - L'insieme degli attributi di una relazione; non varia nel tempo.
  Se una relazione R ha come attributi $A, A, \dots, A$, lo schema è spesso indicato da

  $$R(A_1, A_2, \dots, A_k)$$

- **Istanza di relazione** con schema $R(X)$ - insieme di tutte le tuple su X, con X = *insieme di attributi*.
  L'istanza contiene i valori correnti, possono cambiare nel tempo, anche rapidamente.



- **Schema di base di dati** - insieme di schemi di relazione con nomi differenti.
- **Schema di base di dati relazionale** - insieme $\{R_1, R_2, \dots, R_n\}$ di schemi di relazione.
- **Base di dati relazionale** con schema $\{R_1, R_2, \dots, R_n\}$ - Insieme $\{r_1, r_2, \dots, r_n\}$ dove r_i è una istanza di relazione con schema R_i.
- Se Y è un sottoinsieme di attributi di uno schema X di una certa relazione, allora $t[Y]$ è il sottoinsieme di valori nella tupla t che corrispondono ad attributi contenuti in Y. Ciò è detto **restrizione di t**.
- **NULL** - valore polimorfo per indicare la mancanza di un valore.
  Non appartiene a nessun dominio ma può sostituire valori di qualsiasi dominio.

  Due valori NULL, anche se dello stesso dominio, sono considerati diversi.

VINCOLI

Ogni realtà che si voglia rappresentare tramite una base di dati, contiene dei vincoli, o condizioni, che vanno soddisfatti.
Quando si rappresenta una realtà di interesse deve essere possibile anche rappresentare tali vincoli. Un **vincolo** è quindi una **rappresentazione**, nello schema di una base di dati, **di una condizione valida nella realtà di interesse**.
Un'istanza di base di dati viene definita **legale** se soddisfa tutti i vincoli (cioè è una rappresentazione fedele della realtà di interesse).

Un DBMS permette di:
- **definire** insieme allo schema di base di dati i relativi vincoli
- **verificare** che un'istanza di base di dati sia legale
- **impedire** l'inserimento di tuple che violerebbero i vincoli

VINCOLI DI INTEGRITÀ

Vincolo di integrità: proprietà che deve essere soddisfatta da ogni istanza della base di dati.

2 tipi principali di vincoli:
- **Intrarelazionali** - Definiti sui valori di singoli attributi, o tra valori di attributi di una stessa tupla o tra tuple della stessa relazione.
- **Interrelazionali** - Definiti tra più relazioni.

ESEMPIO

ESAMI

Studente	Voto	Lode	Corso
276545	32		01
276545	30	si	02
787643	27	si ?	03
739430 ?	24		04

(voto ≥ 18) AND (voto ≤ 30)
lode = "si" if (voto = 30) AND (lode = "si")
Studente references Studenti.Matricola
vincolo interrelazionale
vincoli intrarelazionali

STUDENTI

Matricola	Cognome	Nome
276545	Rossi	Mario
787643	Neri	Piero
787643	Bianchi	Luca

Matricola unique

Vincoli **intrarelazionali**:

- **Vincoli di chiave primaria (primary key)** : unica e mai nulla
- **Vincoli di dominio**
  Es.
  $(Voto \geq 18)\ and\ (Voto \leq 30)$
- **Vincoli di unicità (unique)**
  Es.
  $Matricola\ unique$
- **Vincoli di esistenza di un valore per un certo attributo (not-null)**
- **Vincoli di tupla**
  Es.
  $Lode = \text{"si"}\ \ if\ (Voto = 30)\ and\ (Lode = \text{"si"})$

Vincoli **interrelazionali**:

- **Vincoli di integrità referenziale (foreign key)** : porzioni di informazione in relazioni diverse sono correlate attraverso
  valori di chiave.
  Es.
  $Studente\ references\ Studenti.Matricola$

CHIAVI

Una **chiave** di una relazione, è un attributo, o gruppo di attributi, che **determinano una particolare dipendenza funzionale**.

Un insieme X di attributi di una relazione R è una chiave di R se soddisfa le seguenti condizioni:

1. **Per ogni istanza** di R, non esistono 2 tuple distinte t_1 e t_2 che hanno gli stessi valori per tutti gli attributi in X, tali che $t_1[X] = t_2[X]$.
2. Nessun sottoinsieme proprio di X soddisfa la condizione 1.

Una relazione può avere più chiavi alternative.

Viene definita **chiave primaria** la chiave più usata/costituita dal minor numero di attributi. La chiave primaria **non ammette valori nulli**.

DIPENDENZE FUNZIONALI

Una **dipendenza funzionale** stabilisce un **particolare legame semantico tra due insiemi non-vuoti di attributi** X e Y appartenenti ad uno schema R.
Tale vincolo si denota come $\boldsymbol{X \rightarrow Y}$ e si legge $\boldsymbol{X}$ ***determina funzionalmente*** $\boldsymbol{Y}$.

$\boldsymbol{X}$ = **"determinante"**
$\boldsymbol{Y}$ = **"dipendente"**

Diremo che una relazione r con schema R soddisfa la dipendenza funzionale $X \rightarrow Y$ se:

1. La dipendenza funzionale $\boldsymbol{X \rightarrow Y}$ **è applicabile ad** $\boldsymbol{R}$, cioè X e Y sono sottoinsiemi di R.
2. Le **tuple in** $\boldsymbol{r}$ **che concordano su** $\boldsymbol{X}$ **concordano anche su** $\boldsymbol{Y}$, cioè se le tuple sono uguali su X allora devono essere uguali anche su Y.

3 - Algebra Relazionale

L'algebra relazionale è un linguaggio **procedurale** (descrizione della procedura da attuare per ottenere il risultato) che permette di esaminare le **query** (comando scritto dall'utente per ricavare informazioni), da effettuare nell'ambito di utilizzo di un **database**.
L'interrogazione di una o più basi di dati avviene tramite **un'espressione** composta da operatori **algebrici** e **istanze di relazioni**. Ciò avviene seguendo una precisa sequenza che definisce l'ordine delle operazioni e dei loro operandi.
Di seguito affronteremo le principali operazioni eseguibili.

PROIEZIONE π

Consente di effettuare un "taglio verticale" su una relazione, consente quindi di **selezionare solo alcune colonne (attributi)**.
Si denota con il simbolo π:

$$\pi_{A_1, A_2, \ldots, A_k}(r)$$

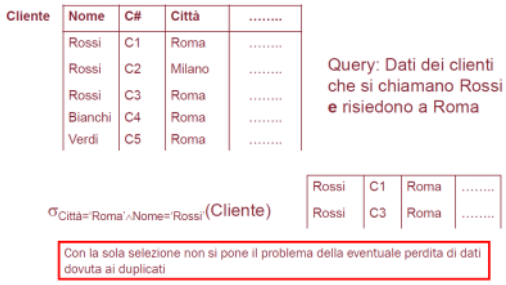
Seleziona le **colonne di r che corrispondono agli attributi $A_1, A_2, \ldots, A_k$**.

SELEZIONE σ

Consente di effettuare un "taglio orizzontale" su una relazione, consente quindi di **selezionare solo le righe (tuple) che soddisfano una data condizione**.
Si denota con il simbolo σ:

$$\sigma_C(r)$$

Seleziona le **tuple di r che soddisfano la condizione C**.



UNIONE $\cup$

Consente di costruire una relazione contenente tutte le tuple che appartengono ad almeno uno dei due operandi.
Si denota con il simbolo $\cup$.

L'operazione di unione si può applicare solo a operandi **union compatibili**, tali che:

- hanno lo stesso numero di attributi.
- gli attributi corrispondenti (nell'ordine) hanno stesso dominio.

Docenti

Nome	CodDoc	Dipartimento
Rossi	D1	Matematica
Rossi	D2	Lettere
Bianchi	D3	Matematica
Verdi	D4	Lingue

Amministrativi

Nome	CodAmm	Dipartimento
Esposito	A1	Lingue
Riccio	A2	Matematica
Pierro	A3	Lettere
Verdi	A4	Lingue
Bianchi	A5	Lingue

Personale

Nome	Cod	Dipartimento
Rossi	D1	Matematica
Rossi	D2	Lettere
Bianchi	D3	Matematica
Verdi	D4	Lingue
Esposito	A1	Lingue
Riccio	A2	Matematica
Pierro	A3	Lettere
Verdi	A4	Lingue
Bianchi	A5	Lingue

Personale = Docenti ∪ Amministrativi

In caso di incongruenze tra gli attributi utilizzare l'operazione di proiezione per rendere **union compatibili** le due relazioni.

NB!

Anche se non è richiesto che il nome degli attributi sia uguale, è comunque necessario che il senso degli attributi sia omogeneo.
Non avrebbe ad esempio senso unire due attributi "Nome" e "Dipartimento" insieme.

DIFFERENZA —

Si applica ad operandi **union compatibili**.
Consente di costruire una relazione con tutte le tuple che appartengono al primo operando e non al secondo.

NB!

La differenza non è commutativa come l'unione, conta l'ordine degli operandi!

Studenti

Nome	CodFiscale	Dipartimento
Rossi	C1	Matematica
Rossi	C2	Lettere
Bianchi	C3	Matematica
Verdi	C4	Lingue

Amministrativi

Nome	CodFiscale	Dipartimento
Esposito	C5	Lettere
Riccio	C6	Matematica
Pierro	C7	Lingue
Bianchi	C3	Matematica

Studenti – Amministrativi

Studenti	Nome	CodFiscale	Dipartimento
	Rossi	C1	Matematica
	Rossi	C2	Lettere
	Verdi	C4	Lingue

Amministrativi - Studenti

Amministrativi	Nome	CodFiscale	Dipartimento
	Esposito	C5	Lettere
	Riccio	C6	Matematica
	Pierro	C7	Lingue

INTERSEZIONE ∩

Si applica ad operandi **union compatibili**.
Consente di costruire una relazione contenente tutte e sole le tuple appartenenti ad entrambi gli operandi.

Studenti

Nome	CodFiscale	Dipartimento
Rossi	C1	Matematica
Rossi	C2	Lettere
Bianchi	C3	Matematica
Verdi	C4	Lingue

Amministrativi

Nome	CodFiscale	Dipartimento
Esposito	C5	Lettere
Riccio	C6	Matematica
Pierro	C7	Lingue
Bianchi	C3	Matematica

- L'operazione di intersezione è commutativa

•Studenti ∩ Amministrativi = studenti che sono **anche** amministrativi

Studenti Amm	Nome	CodFiscale	Dipartimento
	Bianchi	C3	Matematica

PRODOTTO CARTESIANO *X*

Consente di costruire una relazione contenente tutte le tuple che si ottengono concatenando una tupla del primo operando con una tupla del secondo.
Si denota con simbolo *X;*

$R_1 \; X \; R_2$

Si usa quando le informazioni che occorrono a rispondere ad una query si trovano in relazioni diverse

1

Cliente

Nome	C#	Città
Rossi	C1	Roma
Rossi	C2	Milano
Bianchi	C3	Roma
Verdi	C4	Roma

Ordine

O#	C#	A#	N-pezzi
O1	C1	A1	100
O2	C2	A2	200
O3	C3	A2	150
O4	C4	A3	200
O1	C1	A2	200
O1	C1	A3	100

Per poter distinguere gli attributi con lo stesso nome nello schema risultante possiamo usare l'operazione di ridenominazione (ρ) per utilizzare una copia della relazione Ordine in cui l'attributo C# diventa CC#

$OrdineR = \rho_{CC\# \leftarrow C\#}(Ordine)$

2

Nome	C#	Città	O#	CC#	A#	N-pezzi
Rossi	C1	Roma	O1	C1	A1	100
Rossi	C1	Roma	O2	C2	A2	200
Rossi	C1	Roma	O3	C3	A2	150
Rossi	C1	Roma	O4	C4	A3	200
Rossi	C1	Roma	O1	C1	A2	200
Rossi	C2	Milano	O1	C1	A1	100
...	...	...		...	...	...
Bianchi	C3	Roma	O3	C1	A1	100
...	...	...		...	...	...
Verdi	C4	Roma	O4		A3	200
...	...	...		...	...	...

3

Nome	C#	Città	O#	CC#	A#	N-pezzi
Rossi	C1	Roma	O1	C1	A1	100
Rossi	C1	Roma	O1	C1	A2	200
Rossi	C1	Roma	O1	C1	A3	100
Rossi	C2	Milano	O2	C2	A2	200
Bianchi	C3	Roma	O3	C3	A2	150
Verdi	C4	Roma	O4	C4	A3	200

Query: Dati dei clienti e dei loro ordini

$\sigma_{C\#=CC\#}(Cliente \times OrdineR)$

4

Nome	C#	Città	O#	A#	N-pezzi
Rossi	C1	Roma	O1	A1	100
Rossi	C1	Roma	O1	A2	200
Rossi	C1	Roma	O1	A3	100
Rossi	C2	Milano	O2	A2	200
Bianchi	C3	Roma	O3	A2	150
Verdi	C4	Roma	O4	A3	200

Query: Dati dei clienti e dei loro ordini

$\pi_{Nome\ C\#\ Città\ O\#\ A\#\ N\text{-}pezzi}(\sigma_{C\#=CC\#}(Cliente \times OrdineR))$

Eliminiamo gli attributi duplicati (che di solito sono proprio quelli della condizione di selezione)

5

Nome	C#	Città	O#	A#	N-pezzi
Rossi	C1	Roma	O1	A2	200
Rossi	C2	Milano	O2	A2	200
Bianchi	C3	Roma	O3	A2	150
Verdi	C4	Roma	O4	A3	200

Query: Dati dei clienti e dei loro ordini che superano i 100 pezzi

$\pi_{Nome\ C\#\ Città\ O\#\ A\#\ N\text{-}pezzi}(\sigma_{C\#=CC\#\wedge\ N\text{-}pezzi>100}(Cliente \times OrdineR))$

JOIN NATURARLE ⋈

Consente di **selezionare le tuple del prodotto cartesiano** dei due operandi che soddisfano le seguenti condizioni

$$R_1.A_1 = R_2.A_1 \quad \wedge \quad R_1.A_2 = R_2.A_2 \quad \wedge \cdots \wedge \quad R_1.A_k = R_2.A_k$$

dove R_1 e R_2 sono delle relazioni e A_1, A_2, ... , A_k sono gli attributi comuni, **cioè con lo stesso nome**, delle relazioni, eliminando le ripetizioni degli attributi.

Denotiamo il join naturale come

$$R_1 \bowtie R_2 = \pi_{XY}\left(\sigma_C(R_1 \times R_2)\right)$$

dove:

- $C = R_1.A_1 = R_2.A_1 \wedge R_1.A_2 = R_2.A_2 \wedge \cdots \wedge R_1.A_k = R_2.A_k$
- $X = insieme\ degli\ attributi\ di\ R_1$
- $Y = insieme\ degli\ attributi\ di\ R_2\ che\ non\ sono\ attributi\ di\ R_1$

NB!

Nel join naturale gli **attributi della condizione**, che consente di unire solo le tuple giuste, **hanno lo stesso nome**. Vengono unite le tuple in cui questi attributi hanno lo stesso valore.

Cliente

Nome	C#	Città
Rossi	C1	Roma
Rossi	C2	Milano
Bianchi	C3	Roma
Verdi	C4	Roma

Ordine

O#	C#	A#	N-pezzi
O1	C1	A1	100
O2	C2	A2	200
O3	C3	A2	150
O4	C4	A3	200
O1	C1	A2	200
O1	C1	A3	100

Query: Dati dei clienti e dei loro ordini

$Cliente \bowtie Ordine$

Nome	C#	Città	O#	A#	N-pezzi
Rossi	C1	Roma	O1	A1	100
Rossi	C1	Roma	O1	A2	200
Rossi	C1	Roma	O1	A3	100
Rossi	C2	Milano	O2	A2	200
Bianchi	C3	Roma	O3	A2	150
Verdi	C4	Roma	O4	A3	200

Casi limite:

- Le relazioni contengono attributi con lo stesso nome ma non esistono tuple con lo stesso valore per tali attributi in entrambe le relazioni.

  Risultato => il join naturale è vuoto

- Le relazioni non contengono attributi con lo stesso nome, quindi la condizione C diventa vuota se non esistono attributi comuni, quindi non c'è condizione e si degenera nel prodotto cartesiano.

θ-JOIN

Consente di **selezionare le tuple del prodotto cartesiano** dei due operandi di che soddisfano una condizione del tipo

$$A\theta B$$

dove:

- ϑ è un **operatore di confronto** ($\theta \in \{<, =, >, \leq, \geq\}$)
- A è un attributo dello schema del primo operando
- B è un attributo dello schema del secondo operando
- $dom(A) = dom(B)$

$$R_1 \underset{A\theta B}{\bowtie} R_2 = \sigma_{A\theta B}(R_1 \text{ X } R_2)$$

NEGAZIONE ¬

Usata per identificare il complementare di un confronto (o un'operazione che ritorna un valore booleano).

Cliente

Nome	C#	Città
Rossi	C1	Roma
Rossi	C2	Milano
Bianchi	C3	Roma
Verdi	C4	Roma

Query: Dati dei clienti che si chiamano Rossi e NON risiedono a Roma

$\sigma_{\neg(\text{Città='Roma'})\wedge\text{Nome='Rossi'}}(\text{Cliente})$

Rossi	C2	Milano

ESERCIZI ALGEBRA RELAZIONALE

- Abbiamo una base di dati contente le seguenti tabelle relative agli spettacoli teatrali in Italia:
  ***TEATRO*(*ID_Teatro*, *Nome*, *Città*)** => Tabella dei teatri.
  ***OPERA*(*Titolo*, *Autore*)** => Tabella di tutte le opere teatrali.
  ***ESECUZIONE*(*ID_Teatro*, *Data*, *Titolo*)** => Tabella di tutte le opere che sono state eseguite almeno una volta.

  Esprimere in algebra relazionale le seguenti interrogazioni:

1. Definire una query contenente i nomi dei teatri a Roma dove nel 2020 sono state svolte opere di Pirandello.

   2 METODI:

   I. $TEATRI\_ROMA = \pi_{Nome}(\sigma_{Città = "Roma"}(TEATRO))$
   $OPERE\_PIRANDELLO = \pi_{Titolo}(\sigma_{Autore = "Pirandello"}(OPERA))$
   $SCENA2020 = \sigma_{01.01.2020 \le Data \le 31.12.2020}(ESECUZIONE)$

   II. $SCENA2020 = \pi_{\substack{ID_{Teatro},\\ Nome}}\left(\sigma_{\substack{01.01.2020 \le Data \le 31.12.2020 \wedge\\ Autore = Pirandello \wedge\\ Città = "Roma"}}\left(ESECUZIONE \bowtie OPERA \bowtie TEATRO\right)\right)$

2. Definire una query contenete i titoli delle che sono state messe in scena esclusivamente a Roma.

   $TEATRI\_NON\_ROMA = \pi_{ID\_Teatro}(\sigma_{Città \neq "Roma"}(TEATRO))$
   $SCENA\_NON\_ROMA = \pi_{Titolo}(TEATRI\_NON\_ROMA \bowtie ESECUZIONE)$
   $SCENA\_ESEGUITA = \pi_{Titolo}(ESECUZIONE)$
   $SOLO\_ROMA = SCENA\_ESEGUITA - SCENA\_NON\_ROMA$

- Dato il seguente schema di base di dati:
  ***PRODOTTI*(*Codice*, *Descrizione*, *Prezzo_unit*)**
  ***MAGAZZINI*(*Codice*, *Indirizzo*, *Citta`*)**
  ***SCORTE*(*CodiceP*, *CodiceM*, *NumPezzi*)** => contiene i dati relativi alla presenza (N-pezzi≥1) dei prodotti nei vari magazzini.

  Esprimere in algebra relazionale le seguenti interrogazioni:

1. Definire una query con i dati dei magazzini che hanno una scorta di almeno 10 frigoriferi.

$$\pi_{\substack{CodiceM,\\ Indirizzo,\\ Città}}\left(\sigma_{\substack{Descrizione = "Frigorifero" \wedge\\ NumPezzi \ge 10}}\left(\left(SCORTE \underset{\substack{CodiceP\\ =\\ Codice}}{\bowtie} PRODOTTI\right) \underset{\substack{CodiceM\\ =\\ Codice}}{\bowtie} MAGAZZINI\right)\right)$$

2. Definire una query con i dati dei prodotti per cui non ci sono scorte in alcun magazzino di Roma.

$$PRODOTTI - \left(\pi_{\substack{CodiceP,\\ Descrizione,\\ Prezzo\_unit}}\left(\sigma_{Città = "Roma"}\left(PRODOTTI \underset{\substack{CodiceP\\ =\\ Codice}}{\bowtie} SCORTE \underset{\substack{CodiceM\\ =\\ Codice}}{\bowtie} MAGAZZINI\right)\right)\right)$$

4 - Dipendenze Funzionali

Dati uno schema di relazione R e un insieme F di dipendenze funzionali su R, un'istanza di R è **legale** se soddisfa tutte le dipendenze di F.

CONSIDERIAMO: $F=\{A \rightarrow B, B \rightarrow C\}$

R

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a2	b2	c1	d3

Ogni istanza legale (cioè ogni istanza che soddisfa sia $A \rightarrow B$ che $B \rightarrow C$) soddisfa sempre anche la dipendenza funzionale $A \rightarrow C$. $A \rightarrow C$ **non** fa però parte dell'insieme delle dipendenze funzionali F. Possiamo comunque considerarla come se fosse in F?

Possiamo intanto affermare che:

> "Dato uno schema di relazione R e un'insieme F di dipendenze funzionali su R, **possono esistere** delle dipendenze funzionali che **non sono in F**, ma che sono comunque soddisfatte **da ogni istanza legale di R**".

CHIUSURA DI UN INSIEME DI DIPENDENZE FUNZIONALI: F^+

Dati uno schema di relazione R e un insieme F di dipendenze funzionali su R,

> la **chiusura di F** è l'insieme delle dipendenze funzionali che sono soddisfatte da ogni istanza legale di R (a prescindere che tali dipendenze siano o meno dichiarate in F).

Notazione: F^+

Banalmente si ha che $F \subseteq F^+$.

Ma come si calcola F^+? Guarda **"Assiomi di Armstrong"**.

PROPRIETA' DELLE DIPENDENZE FUNZIONALI

Dati uno schema di relazione R e un insieme F di dipendenze funzionali su R e due sottoinsieme non vuoti di R, X e Y, abbiamo che:

$$X \rightarrow Y \in F^+ \Leftrightarrow \forall A \in Y (X \rightarrow A \in F^+)$$

$X \rightarrow Y$ deve essere soddisfatta da **ogni** istanza legale di R.

- Se $t_1[X] = t_2[X]$ **allora deve essere** $t_1[Y] = t_2[Y]$.
- Se $A \in Y$ **e** $t_1[A] \neq t_2[A]$, **non può essere** $t_1[Y] = t_2[Y]$.
- Se $\forall A \in Y : t_1[A] = t_2[A]$, **avremo** $t_1[Y] = t_2[Y]$.

R

A	B	C	D
a1	b1	c1	d1
a2	b2	c1	d2
a1	b1	c1	d3

$A \rightarrow BC \in F^+$

$\Downarrow\Uparrow$

$A \rightarrow B \in F^+$

$A \rightarrow C \in F^+$

CHIAVI pt.2

Dati uno schema di relazione R e un insieme F di dipendenze funzionali su R, un **sottoinsieme K** di uno schema di reazione R è una **chiave** di R se:

1) $K \to R \in F^+$ (K determina funzionalmente ogni attributo di R)

2) $\nexists\, K' \subseteq K \boxtimes K' \to R \in F^+$ (altrimenti sarebbe K' la chiave)

Possono però esistere più chiavi in una relazione R.
In SQL una di esse verrà scelta come **chiave primaria** (non può assumere valore nullo).

ESEMPIO

$$Studente = Matr, \boldsymbol{CF}, Cognome, Nome, Data$$

ASSIOMI DI ARMSTRONG: F^A

Denotiamo con F^A l'insieme delle dipendenze funzionali definito nel modo seguente

Se $f \in F \Rightarrow f \in F^A$	$F \subseteq F^A$
Se $Y \subseteq X \subseteq R \Rightarrow X \to Y \in F^A$ $Nome \subseteq (Nome, Cognome) \Rightarrow (Nome, Cognome) \Rightarrow Nome$	**riflessività**
Se $X \to Y \in F^A \Rightarrow XZ \to YZ \in F^A \; \forall\, Z \subseteq R$ $CF \to Cognome \Rightarrow (CF, Indirizzo) \to (Cognome, Indirizzo)$	**aumento**
Se $X \to Y \in F^A$ e $Y \to Z \in F^A \Rightarrow X \to Z \in F^A$ $Matricola \to CF \;\; e \;\; CF \to Cognome \Rightarrow Matricola \to Cognome$	**transitività**

$F^+ = F^A$: la chiusura transitiva di un insieme di dipendenze funzionali F, può essere ottenuta a partire da F applicando ricorsivamente gli assiomi di riflessività, aumento e transitività, conosciuti come **"assiomi di Armstrong"**.

Introduciamo 3 regole, conseguenza degli assiomi, che consentono di derivare da dipendenze funzionali in F^A altre dipendenze funzionali in F^A

Se $X \to Y \in F^A \;\; e \;\; X \to YZ \in F^A \Rightarrow X \to YZ \in F^A$ $CF \to Nome \;\; e \;\; CF \to Cognome \Rightarrow CF \to (Nome, Cognome)$	**unione**
Se $X \to Y \in F^A \;\; e \;\; Z \subseteq Y \Rightarrow X \to Z \in F^A$ $CF \to (Nome, Cognome) \Rightarrow CF \to Nome \;\; e \;\; CF \to Cognome$	**decomposizione**
Se $X \to Y \in F^A \;\; e \;\; WY \to Z \in F^A \Rightarrow WX \to Z \in F^A$ $Matricola \to CF \;\; e \;\; (Indirizzo, CF) \to Cognome \Rightarrow$ $(Indirizzo, Matricola) \to Cognome$	**pseudotransitività**

CHIUSURA DI UN INSIEME DI ATTRIBUTI: X_F^+

Siano:

- R uno schema di relazione
- F un insieme di dipendenze funzionali su R
- X un sottoinsieme R
- A un attributo i insieme di attributi di R

Definiamo la **"chiusura di X rispetto a F"**, denotata con X_F^+ (o semplicemente X^+ se non sorgono ambiguità) come segue

$$X_F^+ = \{A : X \to A \in F^A\} \; .$$

Fanno quindi parte della chiusura di un insieme di attributi X, tutti quelli che sono determinati funzionalmente da X eventualmente applicando gli assiomi di Armstrong.
Difatti $X \subseteq X_F^+$ (riflessività).

Lemma

Siano R uno schema di relazione ed F un insieme di dipendenze funzionali su R. Si ha che:

$$X \rightarrow Y \in \boldsymbol{F^A} \quad \boldsymbol{se\ e\ solo\ se}\ \boldsymbol{Y \subseteq X^+}$$

Dimostrazione

Sia $Y = A_1, A_2, \dots, A_n$

- Poiché $\boldsymbol{Y \subseteq X^+}$, per ogni $i = 1, \dots, n$ si ha che $X \rightarrow A_i \in F^A$.
  Quindi, per la regola dell'**unione**, $\boldsymbol{X \rightarrow Y \in F^A}$.
- Poiché $\boldsymbol{X \rightarrow A_i \in F^A}$, per la regola **della decomposizione** si ha che, $X \rightarrow A_i \in F^A$, cioè $A_i \in X^+$, per ogni $i = 1, \dots, n$ e quindi $\boldsymbol{Y \subseteq X^+}$.

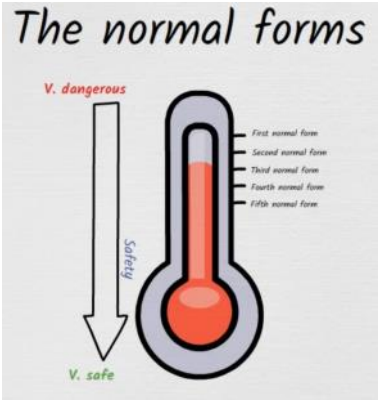
NORMALIZZAZIONE DI UNA BASE DI DATI

La **normalizzazione** è una tecnica di progettazione dei database, mediante la quale si elimina la ridondanza dei dati, al fine di evitare anomalie in seguito ad operazioni di **inserimento**, **cancellazione** o **aggiornamento**.

Questa procedura rende quindi la base di dati più facile da consultare, più facile da estendere e più solida contro errori e anomalie.

La **normalizzazione** viene eseguita in varie fasi (5 in totale): al termine di ciascuna fase il database si trova in uno degli stati di normalizzazione, o come si dice di solito, è in una delle **"forme normali"**.

La normalizzazione va utilizzata come **tecnica di verifica** dei risultati della progettazione di una base di dati, non costituisce quindi una metodologia di progettazione.



PROBLEMI DI PROGETTAZIONE DI UNA BASE DI DATI

Ridondanza - informazione o gruppi di informazioni ripetute più volte (spreco di spazio).

Anomalia di aggiornamento - aggiornamento di un informazione che avviene più volte del necessario.

Anomalia di inserimento - impossibilità di inserire una tupla finché non contiene un valore per ogni attributo (a meno che non si usino valori nulli, rischiando però di aumentare lo spreco di spazio)

Anomalia di cancellazione - l'eliminazione di alcuni dati rischia di eliminarne completamente altri.

PRIMA FORMA NORMALE : 1NF

REGOLE

a. Vietato usare l'ordine delle righe per trasmettere informazioni (ad esempio ordinare dei nomi di persone in base all'altezza della persona a cui si riferisce, utilizzare piuttosto un attributo "altezza").

b. Ogni colonna, quindi ogni attributo, fa riferimento ad un singolo tipo di dato, non possono esserci tipi di dato diversi nella stessa colonna, più specificatamente ogni campo deve contenere esclusivamente una singola informazione.

c. Ogni tabella deve avere una **chiave primaria**.

d. Non possono essere presenti più tuple identiche (contenenti gli stessi valori).

Student Name	Course Titles
Kevin Drumm	Computer Science, Mathematics, Physics
Murvin Drake	Physics, Chemistry
John Jones, 1234	Music
Sally-Jane Jones	Biology, Economics
David , Married	Mathematics, Physics
Murvin Drake	Physics, Chemistry

ID	Student Name	Marital Status	Course Title
1	Kevin Drumm	Single	Computer Science
1	Kevin Drumm	Single	Mathematics
1	Kevin Drumm	Single	Physics
2	Murvin Drake	Single	Physics
2	Murvin Drake	Single	Chemistry
3	John Jones	Single	Music
4	Sally-Jane Jones	Single	Biology
4	Sally-Jane Jones	Single	Economics
5	David	Married	Mathematics
5	David	Married	Physics
6	Murvin Drake	Single	Physics
6	Murvin Drake	Single	Chemistry

Nell'esempio sopracitato vengono messe a confronto due tabelle, una non normalizzata e una invece in **1NF**. Nella tabella di sinistra **non è presente una chiave** che distingue univocamente le tuple, ci sono ripetizioni di tuple, ci sono tipi di dati diversi mischiati, ci sono informazioni diverse nello stesso campo.
A destra invece una possibile versione in 1NF della tabella di sinistra mostra come i dati siano molto più **comprensibili** e facilmente **accessibili**.

La prima forma normale lascia però la porta aperta ad anomalie di inserimento, cancellazione e aggiornamento. Vedremo tra poco come la **seconda forma normale** risolva in gran parte questi problemi.

SECONDA FORMA NORMALE : 2NF

Prima di introdurre la **seconda forma normale**, facciamo prima un altro esempio di 1NF per mostrarne le criticità. (Il requisito principale della 2NF è che la tabella sia in 1NF)

Player_Inventory

Player_ID	Item_Type	Item_Quantity	Player_Rating
jdog21	amulets	2	Intermediate
jdog21	rings	4	Intermediate
gila19	copper coins	18	Beginner
trev73	shields	3	Advanced
trev73	arrows	5	Advanced
trev73	copper coins	30	Advanced
trev73	rings	7	Advanced

Se in questa tabella, l'utente *gila19* perdesse tutte le sue monete di rame, tutti i suoi dati verrebbero rimossi dalla tabella **(anomalia di cancellazione).**

E se invece modificando il *Player_Rating* di un giocatore, accidentalmente la macchina, o una persona, dimenticasse di modificare uno o più campi in cui compare il valore da modificare? **(anomalia di aggiornamento)**.

Nel caso in cui invece volessimo aggiungere un nuovo giocatore, ad esempio *tina42* , i suoi dati non comparirebbero fin quando non venga aggiunto qualcosa al suo inventario **(anomalia di inserimento)**.

Player_Inventory

Player_ID	Item_Type	Item_Quantity	Player_Rating
jdog21	amulets	2	Advanced
jdog21	rings	4	Intermediate
trev73	shields	3	Advanced
trev73	arrows	5	Advanced
trev73	copper coins	30	Advanced
trev73	rings	7	Advanced

** deletion anomaly **
** update anomaly **
** insertion anomaly **
tina42
(Beginner)

La 2NF si riferisce a come le colonne **non appartenenti** ad una chiave, siano in relazione con la **chiave primaria**. In particolare la 2NF richiede che:

> **"Ogni attributo non chiave, deve dipendere dall'intera chiave primaria"**.

Dobbiamo quindi chiederci "le singole colonne non appartenenti alla chiave primaria, dipendono dall'intera chiave primaria, o solo in parte?".

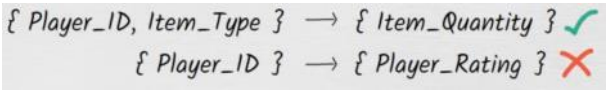
Riprendiamo l'esempio sopra citato:

Player_ID	Item_Type	Item_Quantity	Player_Rating
jdog21	amulets	2	Intermediate
jdog21	rings	4	Intermediate
gila19	copper coins	18	Beginner
trev73	shields	3	Advanced
trev73	arrows	5	Advanced
trev73	copper coins	30	Advanced
trev73	rings	7	Advanced

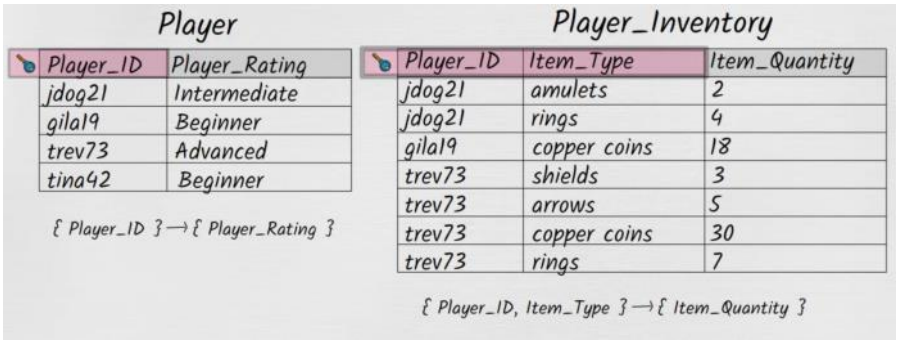
Notiamo quindi *che Player_ID* e *Item_ Type* costituiscono la **chiave primaria**, mentre *Item_ Quantity* e *Player_ Rating* sono **attributi non-chiave**.

Analizziamo le dipendenze funzionali degli attributi non-chiave:

- *Item_ Quantity* si riferisce alla quantità di un oggetto appartenente ad un giocatore. Esso dipende quindi da entrambi gli attributi della chiave primaria.
- *Player_ Rating* si riferisce al grado di esperienza di un giocatore, non è collegato in nessun modo al tipo di oggetti trasportati dal giocatore. Esso dipende quindi solo da *Player_ID* , e non dall'intera chiave primaria.



In sintesi la 2NF richiede che ogni tabella riporti i dati di un singolo concetto. Per risolvere i problemi sopracitati, basta semplicemente separare i dati del giocatore da i dati del suo inventario.



Player_ID	Player_Rating
jdog21	Intermediate
gila19	Beginner
trev73	Advanced
tina42	Beginner

Player_ID	Item_Type	Item_Quantity
jdog21	amulets	2
jdog21	rings	4
gila19	copper coins	18
trev73	shields	3
trev73	arrows	5
trev73	copper coins	30
trev73	rings	7

In tal modo **ogni attributo non chiave dipenderà dall'intera chiave primaria** e non da una parte di essa.

TERZA FORMA NORMALE : 3NF

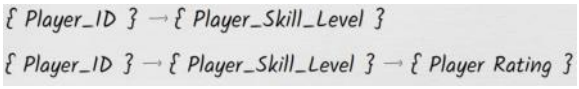
Per introdurre la **terza forma normale**, riprendiamo l'esempio precedente supponendo di voler estendere la tabella *Player* : vogliamo aggiungere un nuovo attributo *Player_ Skill_ Level* che specifica il livello effettivo del giocatore.

Player_ID	Player_Rating	Player_Skill_Level
jdog21	Intermediate	4
gila19	Beginner	3
trev73	Advanced	8
tina42	Beginner	1

Questo nuovo attributo, *Player_Skill_Level*, definisce funzionalmente *Player_Rating*, seguendo la logica per cui *"1 è il livello minimo, 9 è il livello massimo, ogni 3 livelli si passa di grado"* .

Sorge un nuovo problema: poniamo il caso in cui il giocatore *gila19* passa al livello 4, e di conseguenza passa di grado. Cosa accadrebbe se accidentalmente si aggiornasse il *Player_Skill_Level* dimenticando però di aggiornare il *Player_Rating* ? **(errore di inconsistenza)**

Il problema sta nel fatto che *Player_Skill_Level* dipende da *Player_ID* , mentre *Player_Rating* dipende da
Player_Skill_Level che dipende da *Player_ID* .



Quest'ultimo tipo di dipendenza viene chiamato **"dipendenza transitiva"**, ed è proprio questo tipo di dipendenza che la **3NF** vieta, cioè la dipendenza di un attributo non-chiave da un altro attributo non-chiave.
Il problema si risolve semplicemente eliminando *Player_Rating* dalla tabella e creando una tabella a parte con *Player_Skill_Level* come chiave, che definisca l'avanzamento di grado

Riassumendo quindi, la **terza forma normale** richiede che:

> **"Ogni attributo nella tabella deve dipendere dalla chiave, dall'intera chiave, e da nient'altro oltre alla chiave"**.

DEFINIZIONI FORMALI

1) Dati uno schema di relazione R e un insieme di dipendenze funzionali F su R, diciamo che:
   - Un attributo A di R è **primo se appartiene ad una chiave di *R***
   - Un sottoinsieme X di R è una **superchiave se contiene una chiave di *R***
2) Siano R uno schema di relazione e F un insieme di dipendenze funzionali su R:
   - $X \rightarrow A \in F^+$ è una **dipendenza parziale** su R se A non è primo ed X è contenuto propriamente in una chiave di R
   - $X \rightarrow A \in F^+$ è una **dipendenza transitiva** su R se A non è primo e per ogni chiave K di R si ha che X non è contenuto propriamente in K e $K - X \neq \emptyset$
3) Siano R uno schema di relazione e F un insieme di dipendenze funzionali su R. R è in 3NF se per ogni dipendenza funzionale $X \rightarrow A \in F^+ : A \notin X$ si ha che:
   - ***A* è <u>primo</u>**
     oppure (inclusivo)
   - ***X* è una <u>superchiave</u>**

DECOMPOSIZIONI pt.2

Per **decomposizione** intendiamo quel processo che porta alla scomposizione di uno schema non in 3NF, in sottoschemi in 3NF interconnessi tra di loro.

Quando si decompone uno schema di relazione R si cui è definito uno schema di dipendenze funzionali F, oltre ad ottenere schemi in 3NF occorre:

1. **Preservare** le dipendenze funzionali
2. **Poter ricostruire tramite join** tutta e sola l'informazione originaria

Uno schema che non è in 3NF può essere **decomposto in più modi** in un insieme di schemi in 3NF. Ad esempio lo schema $\boldsymbol{R = ABC}$ con l'insieme di dipendenze funzionali $\boldsymbol{F = \{A \to B, B \to C\}}$ non è in 3NF per la presenza in $\boldsymbol{F^+}$ della dipendenza **transitiva** $\boldsymbol{B \to C}$, dato che la chiave è evidentemente A.

$\boldsymbol{R}$ può essere decomposto in:

- $R1 = AB$ con $\{A \to B\}$
- $R2 = BC$ con $\{B \to C\}$

oppure

- $R1 = AB$ con $\{A \to B\}$
- $R2 = AC$ con $\{A \to C\}$

In entrambi i casi gli schemi ottenuti sono in 3NF, tuttavia la seconda soluzione non è soddisfacente (in quanto non vengono preservate le dipendenze funzionali).

Le dipendenze funzionali che si vogliono preservare sono tutte quelle che sono **soddisfatte da ogni istanza legale di $\boldsymbol{R}$**, cioè le dipendenze funzionali in $\boldsymbol{F^+}$.

Sarebbe necessario quindi calcolare $\boldsymbol{F^+}$**,** ma l'applicazione ricorsiva dei **teoremi di Armstrong** richiede tempo esponenziale in $|R|$. Per i nostri scopi è sufficiente avere un metodo per stabilire se una data dipendenza funzionale $X \to Y$ appartiene a F^+.

Ciò può essere fatto calcolando X^+ e verificando se $Y \subseteq X^+$.
Ricordiamo infatti il lemma " $X \to Y \in F^A$ se e solo se $Y \subseteq X^+$ " e l'uguaglianza $F^A = F^+$.

CALCOLO DI X^+

Per il calcolo della chiusura dell'insieme di attributi X, denotata con X^+, usiamo il seguente algoritmo:

ALGORITMO : CALCOLO CHIUSURA DI UN'INSIEME DI ATTRIBUTI

Input schema di relazione $\boldsymbol{R}$
insieme $\boldsymbol{F}$ di dipendenze funzionali su R
sottoinsieme $\boldsymbol{X}$ di R

Output chiusura di X rispetto ad F (restituita nella variabile Z)

Inizio
Z = X
S = { A : Y → V ∈ F ∧ A ∈ V ∧ Y ⊆ X }

While S ⊄ Z :
Z = Z ∪ S
S = { A : Y → V ∈ F ∧ A ∈ V ∧ Y ⊆ Z }

return Z
Fine

Per comprendere l'algoritmo appena descritto procediamo subito con un esempio.

ESEMPIO

Dato lo schema di relazione $R = (A, B, C, D, E, H)$ e il seguente insieme di dipendenze funzionali su R ☒

$$F = \{ AB \rightarrow CD, EH \rightarrow D, D \rightarrow H \}$$

calcolare le chiusure degli insiemi A, D e AB .

Inizio
Z = A
S = { L : Y → V ∈ F ∧ L ∈ V ∧ Y ⊆ A } S = ∅ A da solo non determina alcun altro attributo

while S ⊄ Z : ∅ ⊂ A la condizione è falsa, non entriamo nel while
Z = Z ∪ S
S = { A : Y → V ∈ F ∧ A ∈ V ∧ Y ⊆ Z }

return Z
Fine

$$A^+ = A$$

Inizio
Z = D
S = { L : Y → V ∈ F ∧ L ∈ V ∧ Y ⊆ D } S = H per la dipendenza $D \rightarrow H$

while S ⊄ Z : H ⊄ D
Z = Z ∪ S Z = DH
S = { A : Y → V ∈ F ∧ A ∈ V ∧ Y ⊆ Z } S = H per la dipendenza $D \rightarrow H$

2° **while** : H ⊂ DH

return Z
Fine

$$D^+ = DH$$

Inizio
Z = AB
S = { L : Y → V ∈ F ∧ L ∈ V ∧ Y ⊆ AB } S = {C, D} per la dipendenza $AB \rightarrow CD$

while S ⊄ Z : CD ⊄ AB
Z = Z ∪ S Z = ABCD
S = { A : Y → V ∈ F ∧ A ∈ V ∧ Y ⊆ Z } S = {C, D, H} per le dipendenze $AB \rightarrow CD$ e $D \rightarrow H$

2° **while** : CDH ⊄ ABCD
Z = Z ∪ S Z = ABCDH
S = { A : Y → V ∈ F ∧ A ∈ V ∧ Y ⊆ Z } S = {C, D, H} per le dipendenze $AB \rightarrow CD$ e $D \rightarrow H$

3° **while** : CDH ⊂ ABCDH

return Z
Fine

$$AB^+ = ABCDH$$

L'algoritmo si ferma quando il nuovo insieme S che otteniamo è (già) contenuto nell'insieme Z, cioè quando non possiamo aggiungere nuovi attributi alla chiusura transitiva di X.

CHIAVI pt.3

Per determinare le chiavi di uno schema R, su cui è definito un insieme di dipendenze funzionali F, utilizziamo il calcolo della chiusura di un insieme di attributi.

ESEMPIO

Dati:

$\boldsymbol{R = (A,\ B,\ C,\ D,\ E,\ H)}$
$\boldsymbol{F = \{AB \to CD,\ C \to E,\ AB \to E,\ ABC \to D\}}$

Calcolare la chiusura dell'insieme **ABH**

Inizio
Z = ABH
S = { L : Y → V ∈ F ∧ L ∈ V ∧ Y ⊆ ABH } $S = \{C,\ D,\ E\}$ per le dipendenze $AB \to CD$ e $AB \to E$

while S ⊄ Z : CDE ⊄ ABH
Z = Z ∪ S Z = ABCDEH
S = { A : Y → V ∈ F ∧ A ∈ V ∧ Y ⊆ Z } $S = \{C,\ D,\ E\}$

2° **while** S ⊄ Z: CDE ⊂ ABCDEH

return Z
Fine

$$\boldsymbol{ABH^{+} = ABCDEH}$$

Abbiamo quindi che $\boldsymbol{ABH^{+} = ABCDEH = R}$, ABH è quindi una chiave di R?

Ricordiamo le condizioni:

Dati uno schema di relazione R e un insieme F di dipendenze funzionali su R, un **sottoinsieme K** di uno schema di reazione R è una **chiave** di R se:

1) $\boldsymbol{K \to R \in F^{+}}$
2) $\boldsymbol{\nexists\ K' \subseteq K \boxtimes K' \to R \in F^{+}}$

Dobbiamo quindi **sempre** verificare che **nessun** sottoinsieme di K determini funzionalmente R.
Riapplichiamo quindi l'algoritmo di calcolo sui sottoinsiemi di ABK.

Considerazione:

$\boldsymbol{H}$ non è determinato da altri attributi, deve perciò essere necessariamente parte di **qualunque** chiave.

I sottoinsiemi da verificare sono quindi 2 $\Rightarrow$ $\boldsymbol{AH}$ e $\boldsymbol{BH}$.

$\boldsymbol{R = (A,\ B,\ C,\ D,\ E,\ H)}$
$\boldsymbol{F = \{AB \to CD,\ C \to E,\ AB \to E,\ ABC \to D\}}$

Inizio
Z = AH
S = { L : Y → V ∈ F ∧ L ∈ V ∧ Y ⊆ AH } $S = \emptyset$

while S ⊄ Z : $\emptyset \subset$ AH

return Z
Fine

$$\boldsymbol{AH^{+} = AH \subset R}$$

Inizio
Z = BH
S = { L : Y → V ∈ F ∧ L ∈ V ∧ Y ⊆ BH } $S = \emptyset$

while S ⊄ Z : $\emptyset \subset$ BH

return Z
Fine

$$\boldsymbol{BH^{+} = BH \subset R}$$

Siccome è inutile provare con sottoinsiemi di cardinalità ancora minore, abbiamo verificato che:

1. $ABH \to R \in F^+$
2. $\nexists\, K' \subseteq ABH \;:\; K' \to R \in F^+$

Quindi ABH è una chiave di R.

ESERCIZI

a. Determinare se esistono altre chiavi per

$$\boldsymbol{R = (A,\ B,\ C,\ D,\ E,\ H)}$$
$$\boldsymbol{F = \{AB \to CD,\ C \to E,\ AB \to E,\ ABC \to D\}}$$

a. Dati

$$\boldsymbol{R = (A,\ B,\ C,\ D,\ E,\ G,\ H)}$$
$$\boldsymbol{F = \{AB \to D,\ G \to A,\ G \to B,\ H \to E,\ H \to G,\ D \to H\}}$$

determinarne le 4 chiavi.

CHIAVI e 3NF

Una volta individuate le chiavi in uno schema di relazione, possiamo determinare se lo schema è in 3NF.

ESEMPIO

Dati:

$$\boldsymbol{R = (A,\ B,\ C,\ D,\ E,\ G,\ H)}$$
$$\boldsymbol{F = \{AB \to CD,\ EH \to D,\ D \to H\}}$$

Determinare la chiave di R e, sapendo che la chiave è unica, verificare che R non è in 3NF.

Analizzando F notiamo che A, B, E, G non vengono mai determinati da altri attributi.
Proviamo ad applicare l'algoritmo su $ABEG$.

Inizio

Z = ABEG
S = { L : Y → V ∈ F ∧ L ∈ V ∧ Y ⊆ ABEG } $S = \{C,\ D\}$ per la dipendenza $AB \to CD$

while S ⊄ Z : CD ⊄ ABEG
  Z = Z ∪ S Z = ABCDEG
  S = { A : Y → V ∈ F ∧ A ∈ V ∧ Y ⊆ Z } $S = \{C,\ D,\ H\}$ per $AB \to CD$ e $D \to H$

2° while S ⊄ Z: CDH ⊄ ABCDEG
  Z = Z ∪ S Z = ABCDEGH
  S = { A : Y → V ∈ F ∧ A ∈ V ∧ Y ⊆ Z } $S = \{C,\ D,\ H\}$

3° while S ⊄ Z: CDH ⊂ ABCDEGH

return Z

Fine

$ABEG$ è quindi chiave di R. È evidente che non esistano sottoinsiemi di $ABEG$ che determinino R.

Per verificare che lo schema non sia in 3NF, basta osservare la sola presenza della dipendenza parziale $AB \to CD$, decomponibile in $AB \to C$ e $AB \to D$, in entrambi i casi l'attributo di destra non è primo e AB è propriamente contenuto in $ABEG$.

Notiamo inoltre che $EH \to D$ e $D \to H$ sono dipendenze transitive:

- Per entrambe l'attributo di destra non è primo e l'insieme di destra non è superchiave.
- Data l'unica chiave dello schema abbiamo che $ABEG - EH = ABG \neq \emptyset$ e $ABEG - D = ABEG \neq \emptyset$

DECOMPOSIZIONI pt.3

Approfondiamo ora il discorso aperto precedentemente sulle decomposizioni.

ESEMPIO

Consideriamo lo schema $R = (\ Matricola,\ Comune,\ Provincia\ )$ con l'insieme di dipendenze funzionali $F = \{Matricola \rightarrow Comune,\ Comune \rightarrow Provincia\}$.
Lo schema non è in 3NF per la presenza in F^+ della dipendenza transitiva $Matricola \rightarrow Provincia$, dato che la chiave è evidentemente $Matricola$.

R può essere decomposto in:

a. $R1 = (\ Matricola, Comune\ )$ con $\{\ Matricola \rightarrow Comune\ \}$
   $R2 = (\ Comune, Provincia\ )$ con $\{\ Comune \rightarrow Provincia\}$

   oppure

b. $R1 = (\ Matricola, Comune\ )$ con $\{\ Matricola \rightarrow Comune\ \}$
   $R2 = (\ Matricola, Provincia\ )$ con $\{\ Matricola \rightarrow Provincia\ \}$

Entrambi gli schemi sono in 3NF, tuttavia la seconda soluzione non è soddisfacente (non vengono mantenute le dipendenze funzionali).

Consideriamo le istanze legali:

R1

Matricola	Comune
501	Tivoli
502	Tivoli

R2

Matricola	Provincia
501	Roma
502	Rieti

L'istanza dello schema originario R può essere ricostruita tramite un join naturale, con il seguente risultato.

R

Matricola	Comune	Provincia
501	Tivoli	Roma
502	Tivoli	Rieti

È evidente come un errore di inserimento non venga rilevato in $R2$, per questo è fondamentale mantere le dipendenze funzionali (nel caso $R2 = (\ Comune, Provincia\ )$ con $\{\ Comune \rightarrow Provincia\}$, non sarebbe potuto accadere).

Definiamo quindi rigorosamente le 3 condizioni per valutare se una decomposizione sia valida:

1. Ogni sottoschema deve essere in **3NF**
2. La decomposizione deve **preservare** tutte le **dipendenze funzionali** in F^+
3. La decomposizione deve permettere di ricostruire una istanza legale decomposta senza perdita di informazione (**join senza perdite**)

DEFINIZIONE : DECOMPOSIZIONE

> **Sia R uno schema di relazione, una decomposizione di R è una famiglia di sottoinsiemi $\rho = \{\ R_1, R_2, \dots, R_k\ \}$ di R che ricopre R: $\cup_{i=1}^{k} R_i = R$ (i sottoinsiemi possono avere intersezione non vuota).**

In sintesi R è un insieme di attributi, una decomposizione di R è una famiglia di insiemi di attributi, la cui unione è uguale ad R.

DEFINIZIONE : EQUIVALENZA TRA INSIEMI DI DIPENDENZE FUNZIONALI

> **Siano F e G due insiemi di dipendenze funzionali. F e G sono equivalenti $F \equiv G$ se $F^+ = G^+$ (F e G non contengono per forza le stesse dipendenze, ma le loro chiusure si).**

LEMMA : CHIUSURE

Siano F e G due insiemi di dipendenze funzionali. Se $F \subseteq G^+ \Rightarrow F^+ \subseteq G^+$.

Sia $f \in F^+ - F$, siccome f è derivabile da F mediante gli assiomi di armstrong, e ogni dipendenza funzionale in F è derivabile da G mediante gli assiomi di Armstrong, f è derivabile da G mediante gli assiomi di Armstrong.
Abbiamo quindi che

$$G \overset{A}{\rightarrow} F \overset{A}{\rightarrow} F^+$$

Con $\overset{A}{\rightarrow}$ denotiamo la derivazione tramite assiomi di Armstrong.

DEFINIZIONE : DECOMPOSIZIONE CHE PRESERVA F

Dati uno schema di relazione R, un insieme di dipendenze funzionali F su R, una decomposizione $\rho = \{ R_1, R_2, \dots, R_k \}$ di R: diciamo che ρ *preserva* F se $F \equiv \bigcup_{i=1}^{k} \pi_{R_i}(F)$ dove $\pi_{R_i}(F) = \{ X \rightarrow Y : X \rightarrow Y \in F^+ \wedge XY \subseteq R_i \}$.

Supponiamo di avere già una decomposizione e voler verificare che essa preservi le dipendenze funzionali. Bisogna verificare l'equivalenza tra F e $G = \bigcup_{i=1}^{k} \pi_{R_i}(F)$ tramite la doppia inclusione delle loro chiusure $F^+ \subseteq G^+$ e $F^+ \supseteq G^+$.
Per come G è stato definito **in questo caso**, sarà sicuramente $F^+ \supseteq G$ (ogni proiezione di F, che viene inclusa per definizione in G, è un sottoinsieme di F^+), per il **lemma delle chiusure** abbiamo che $F^+ \supseteq G^+$.

È quindi sufficiente verificare che $F \subseteq G^+$ (che implica $F^+ \subseteq G^+$).

ALGORITMO : CONTENIMENTO DI F IN G^+

Input due insiemi di dipendenze funzionali F e G

Output valore booleano

Inizio
  for every $X \rightarrow Y \in F$:
    calcola X_G^+
    if $Y \not\subset X_G^+$:
      return False
  return True
Fine

Basta verificare che **anche una sola dipendenza** non appartenga alla chiusura di G per poter affermare che l'equivalenza non sussista.

Abbiamo solo un problema adesso: come calcoliamo X_G^+?
Se volessimo utilizzare l'algoritmo per il calcolo della chiusura di un insieme di attributi dovremmo prima calcolare G, ma per la definizione di G in questo caso ciò richiederebbe il calcolo di F^+, che richiede tempo esponenziale.

Abbiamo quindi necessità di un altro algoritmo che ci permetta di calcolare X_G^+ a partire da F.

ALGORITMO : CALCOLO DI X_G^+ A PARTIRE DA F

Input schema R
insieme dipendenze funzionali su R F
decomposizione ρ
sottoinsieme di R X

Output chiusura di X rispetto a $G = \bigcup_{i=1}^{k} \pi_{R_i}(F)$ Z

Inizio

$Z = X$

$S = \emptyset$

for $i = 1$ **to** k :

$S = S \cup (X \cap R_i)_F^+ \cap R_i$

while $S \not\subset Z$:

$Z = Z \cup S$

for $i = 1$ **to** k:

$S = S \cup (Z \cap R_i)_F^+ \cap R_i$

return Z

Fine

Prima di dare una descrizione dell'algoritmo vediamo alcuni esempi per coglierne il funzionamento.

ESEMPI

1) Dati

$\boldsymbol{R} = \{ \boldsymbol{A},\ \boldsymbol{B},\ \boldsymbol{C},\ \boldsymbol{D} \}$

$\boldsymbol{F} = \{ \boldsymbol{AB} \to \boldsymbol{C},\ \boldsymbol{D} \to \boldsymbol{C},\ \ \boldsymbol{D} \to \boldsymbol{B},\ \boldsymbol{C} \to \boldsymbol{B},\ \boldsymbol{D} \to \boldsymbol{A} \}$

$\boldsymbol{\rho} = \{ \boldsymbol{ABC}, \boldsymbol{ABD} \}$

Dire se la decomposizione preserva le dipendenze in F (**verificare che ogni dipendeza funzionale in F si trovi in** $\boldsymbol{G^+}$ $(\boldsymbol{F \subseteq G^+})$).

Notiamo che è inutile controllare che vengano preservate le dipendenze per cui l'**unione di parte destra e sinistra** (**determinante** e **dipendete**) sia **contenuta interamente in un sottoschema** di R. Per definizione $\boldsymbol{\pi_{R_i}(F) = \{ X \to Y : X \to Y \in F^+ \wedge XY \subseteq R_i \}}$ quelle dipendenze fanno già parte di G^+.

Non ha senso, per esempio, controllare se $AB \to C$ viene preservata con la decomposizione ρ , in quanto $AB \cup C$ è contenuto interamente in uno dei sottoschemi ottenuti dalla decomposizione (ABC) $AB \to C$ è quindi palesemente preservato in G^+.

L'unica dipendenza che ha senso controllare in questo caso è quindi $D \to C$, in quanto è l'unica che fa riferimento ad attributi di sottoschemi diversi (e quindi non è scontato che sia preservata dalla decomposizione).

Inizio

$Z = D$

$S = \emptyset$

for $i = 1$ **to** k :

$S = S \cup (D \cap R_i)_F^+ \cap R_i$

1^a iterazione: $R_i = ABC$

$S = \emptyset \cup (D \cap ABC)_F^+ \cap ABC = \emptyset \cup (\emptyset)_F^+ \cap ABC = \emptyset \cup \emptyset = \emptyset$

2^a iterazione: $R_i = ABD$

$S = \emptyset \cup (D \cap ABD)_F^+ \cap ABD = \emptyset \cup (D)_F^+ \cap ABD =$
$= \emptyset \cup ABCD \cap ABD = ABD$

while $S \not\subset Z$:

$Z = Z \cup S$

for $i = 1$ **to** k:

$S = S \cup (Z \cap R_i)_F^+ \cap R_i$

$ABD \not\subset D$

$Z = D \cup ABD = ABD$

1^a iterazione: $R_i = ABC$

$S = ABD \cup (ABD \cap ABC)_F^+ \cap ABC = ABD \cup (AB)_F^+ \cap ABC =$
$= ABD \cup ABC \cap ABC = ABCD$ (∩ ha precedenza su ∪)

2^a iterazione: $R_i = ABD$

$S = ABCD \cup (ABD \cap ABD)_F^+ \cap ABD =$
$= ABCD \cup (ABD)_F^+ \cap ABD = ABCD \cup ABCD \cup ABD = ABCD$

2° while $S \not\subset Z$:

$Z = Z \cup S$

for $i = 1$ **to** k:

$S = S \cup (Z \cap R_i)_F^+ \cap R_i$

$ABCD \not\subset ABD$

$Z = ABD \cup ABCD = ABCD$

1^a iterazione: $R_i = ABC$

$S = ABCD \cup (ABCD \cap ABC)_F^+ \cap ABC = ABCD$

2^a iterazione: $R_i = ABD$

$S = ABCD \cup (ABCD \cap ABD)_F^+ \cap ABD = ABCD$

3° while $S \not\subset Z$:

$ABCD \subset ABCD$

return Z

Fine

Terminato l'algoritmo va controllato il valore di $Z = (D)_G^+ = ABCD$ tramite l'algoritmo di contenimento di F in G^+

siccome $C \in (D)_G^+$ tutte le dipendenze sono preservate

2) Dati

$\boldsymbol{R} = \{ \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}, \boldsymbol{E} \}$
$\boldsymbol{F} = \{ \boldsymbol{AB} \to \boldsymbol{E}, \boldsymbol{B} \to \boldsymbol{CE}, \boldsymbol{ED} \to \boldsymbol{C} \}$
$\boldsymbol{\rho} = \{ \boldsymbol{ABE}, \boldsymbol{CDE} \}$

Dire se la decomposizione preserva le dipendenze in F (**verificare che ogni dipendeza funzionale in F si trovi in G^+ ($F \subseteq G^+$)**).

In questo caso l'unica dipendenza che ha senso controllare è $\boldsymbol{B \to CE}$.

Inizio

Z = B
S = ∅
for i = 1 **to** k:
  $S = S \cup (D \cap R_i)_F^+ \cap R_i$

1^a iterazione: $R_i = ABE$
$S = \emptyset \cup (B \cap ABE)_F^+ \cap ABE = (B)_F^+ \cap ABE = BCE \cap ABE = BE$

2^a iterazione: $R_i = CDE$
$S = BE \cup (B \cap CDE)_F^+ \cap ABD = BE \cup (\emptyset)_F^+ \cap ABD = BE$

while $S \not\subset Z$:
  $Z = Z \cup S$
  for i = 1 **to** k:
    $S = S \cup (Z \cap R_i)_F^+ \cap R_i$

$BE \not\subset B$
$Z = B \cup BE = BE$
1^a iterazione: $R_i = ABE$
$S = BE \cup (BE \cap ABE)_F^+ \cap ABE = BE \cup (BE)_F^+ \cap ABE =$
$= BE \cup BCE \cap ABE = BE$

2^a iterazione: $R_i = CDE$
$S = BE \cup (BE \cap CDE)_F^+ \cap CDE = BE \cup (E)_F^+ \cap CDE =$
$= BE \cup E \cap CDE = BE$

2° while $S \not\subset Z$:

$BE \subset BE$

return Z

Fine

Controlliamo Z

$Z = (B)_G^+ = BE$
$E \in (B)_G^+$ ma $C \notin (B)_G^+$

Quindi $B \to CE$ non è preservata, di consequenza, poiché nella chiusura manca uno degli attributi che dovrebbero essere determinati funzionalmente da B, la decomposizione non preserva le dipendenze funzionali.

Adesso abbiamo tutti gli strumenti per determinare se un sottoschema sia in 3NF e se tramite la decomposizione vengano preservate tutte le dipendenze funzionali.
Ci resta solo da analizzare come poter verificare che, data una decomposizione, abbia un join senza perdite.
(vedi pag. 22)

DECOMPOSIZIONI CON JOIN SENZA PERDITE

Se si decompone uno schema di relazione R, si vuole che la decomposizione $\{R_1, R_2, \dots, R_k\}$ ottenuta sia tale che ogni istanza legale r di R sia ricostruibile mediante un join naturale ($\bowtie$) da un'istanza legale $\{r_1, r_2, \dots, r_k\}$ dello schema decomposto $\{R_1, R_2, \dots, R_k\}$.

Poiché per ricostruire una tupla t di r è necessario che $t[R_i] \in r_i \ \forall \ i = 1, 2, \dots, k$, si deve avere

$$\mathbf{r_i} = \boldsymbol{\pi_{R_i}(r)} \quad \forall \quad \boldsymbol{i = 1, 2, \dots, k}$$

DEFINIZIONE

Sia R uno schema di relazione. Una decomposizione $\rho = \{R_1, R_2, \dots, R_k\}$ di R ha un join senza perdita se per ogni istanza legale di r si ha $\mathbf{r} = \boldsymbol{\pi_{R_1}(r) \bowtie \pi_{R_2}(r) \bowtie \dots \bowtie \pi_{R_k}(r)}$.
Unendo quindi tutte le istanze delle proiezioni deve tornarci esattamente l'istanza iniziale.

Prima di procedere introduciamo un nuovo operatore, il **join delle proiezioni**: $\boldsymbol{m_\rho}$, indicato come

$$\boldsymbol{m_\rho(\mathbf{r}) = \pi_{R_1}(r) \bowtie \pi_{R_2}(r) \bowtie \dots \bowtie \pi_{R_k}(r)} .$$

Per ogni istanza legale r di R si ha:

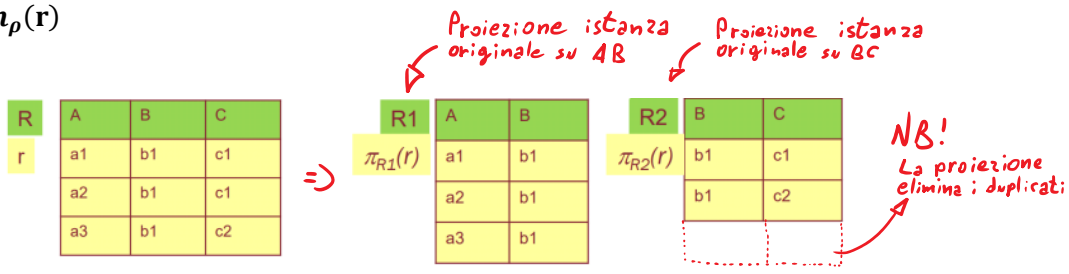
a) $\boldsymbol{r \subseteq m_\rho(\mathbf{r})}$

b) $\boldsymbol{\pi_{R_i}\left(m_\rho(\mathbf{r})\right) = \pi_{R_i}(r)}$

c) $\boldsymbol{m_\rho\left(m_\rho(\mathbf{r})\right) = m_\rho(\mathbf{r})}$

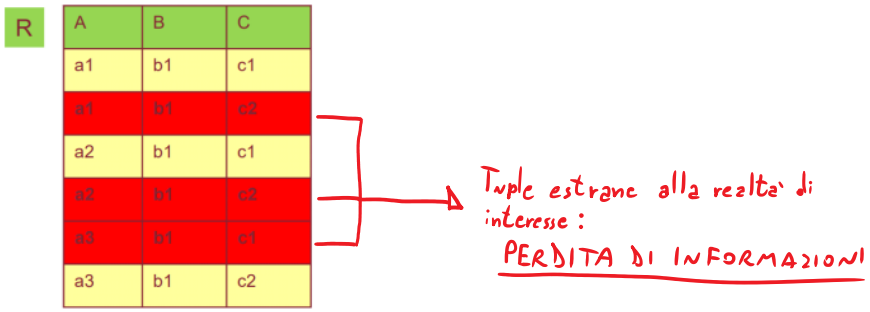
ESEMPIO

Consideriamo
$R = ABC$
$F = \{A \to B, C \to B\}$
$\varrho = \{AB, BC\}$



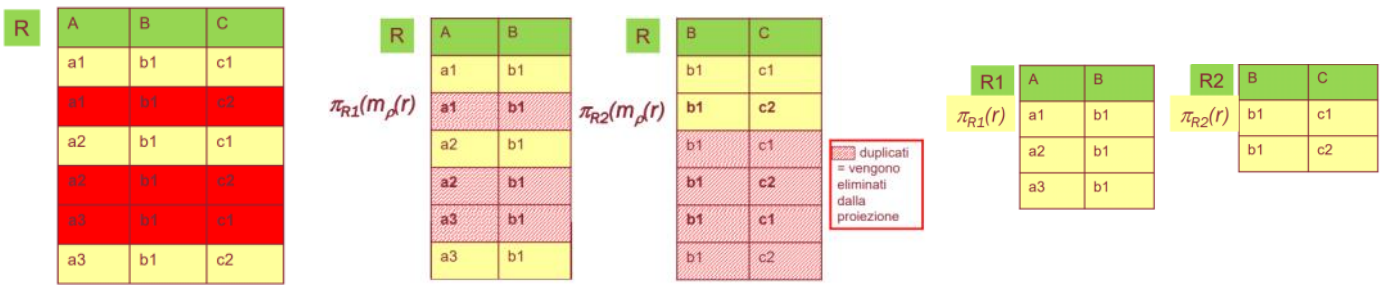
Se però applicassimo il join naturale alla decomposizione effettuata otterremmo il seguente risultato

$$\boldsymbol{m_\rho(\mathbf{r}) = \pi_{R_1}(r) \bowtie \pi_{R_2}(r)}$$



a) $r \subseteq m_\rho(r)$

Se invece applicassimo la proiezione sul join delle proiezioni ($\pi_{R_i}\left(m_\rho(\mathbf{r})\right)$) otterremmo nuovamente le proiezioni.

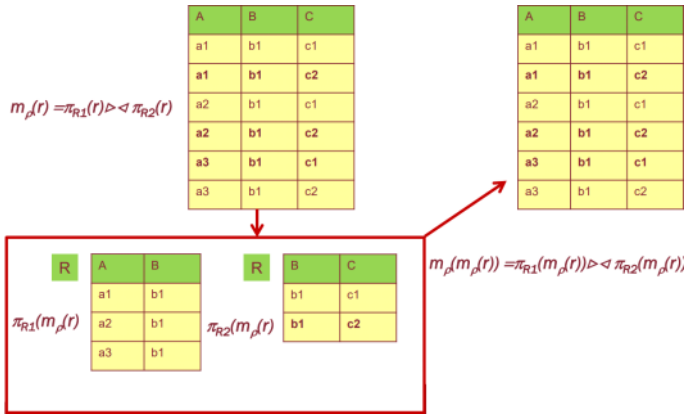


b) $\pi_{R_i}(m_\rho(r)) = \pi_{R_i}(r)$

Se invece applicassimo un join di proiezioni su un join proiezioni, otterremo il join di proiezioni iniziale.

Per ***b***) abbiamo che $\boldsymbol{\pi_{R_i}\left(m_\rho(\mathrm{r})\right) = \pi_{R_i}(r)}$, pertanto, applicando la definizione di $\boldsymbol{m_\rho}$ avremo

$$\boldsymbol{m_\rho\left(m_\rho(\mathrm{r})\right) = \pi_{R_1}\left(m_\rho(\mathrm{r})\right) \bowtie \pi_{R_2}\left(m_\rho(\mathrm{r})\right) \bowtie \ldots \bowtie \pi_{R_k}\left(m_\rho(\mathrm{r})\right) =}$$
$$\boldsymbol{= \pi_{R_1}(r) \bowtie \pi_{R_2}(r) \bowtie \ldots \bowtie \pi_{R_k}(r) = m_\rho(\mathrm{r})}$$



c) $m_\rho(m_\rho(r)) = m_\rho(r)$

Proseguiamo adesso con l'algoritmo di verifica dell'esistenza del join senza perdite per una data decomposizione.

ALGORITMO : VERIFICA PRESENZA JOIN SENZA PERDITA IN UNA DECOMPOSIZIONE

Input	schema insieme dipendenze funzionali su R decomposizione	$\boldsymbol{R}$ $\boldsymbol{F}$ $\boldsymbol{\rho}$
Output	valore booleano	

Inizio

Costruisci una tabella r nel modo seguente:

- r ha |R| colonne e $|\rho|$ righe
- all'incrocio tra l' i-esima riga e alla j-esima colonna metti:
  - a_j se l'attributo $A_j \in R_i$
  - b_{ij} altrimenti

a_j e b_{ij} sono dei valori speciali, come delle label

RICORDA!
- R_i È UN SOTTO INSIEME DI R
  Ad esempio se R = ABCD
  R_i potrebbe essere AB
- A_j È UN SINGOLO ATTRIBUTO DI R_i
  Nell'esempio sopra
  $A_1 = A$, $A_2 = B$

for every $X \to Y \in F$:
  if $\exists\, t_1, t_2 \in r : t_1[X] = t_2[X]\ e\ t_1[Y] \neq t_2[Y]$:
    for every A_j **in** Y:
      if $t_1[A_j] = a_j$:
        $t_2[A_j] = t_1[A_j]$
      else
        $t_1[A_j] = t_2[A_j]$

until r ha una riga con tutte a **or** r non è cambiato

if r ha una riga con tutte a :
  return True (*ρ* ha un join senza perdita)
else :
  return False (*ρ* non ha un join senza perdita)

Fine

Per chiarire ogni dubbio procediamo subito con degli esempi.

ESEMPI

1) Dati:

$\boldsymbol{R = \{A,\ B,\ C,\ D,\ E\}}$
$\boldsymbol{F = \{C \to D,\ AB \to E,\ D \to B\}}$
$\boldsymbol{\rho = \{AC,\ ADE,\ CDE,\ AD,\ B\}}$

Costruiamo innanzitutto la tabella

$R_i \backslash A_j$	A	B	C	D	E
AC	a_1	b_{12}	a_3	b_{14}	b_{15}
ADE	a_1	b_{22}	b_{23}	a_4	a_5
CDE	b_{31}	b_{32}	a_3	a_4	a_5
AD	a_1	b_{42}	b_{43}	a_4	b_{45}
B	b_{51}	a_2	b_{53}	b_{54}	b_{55}

Le a_j vanno dove c'è corrispondenza tra un attributo in R_i e il corrispondente attributo A_j

Ad esempio nella tupla di AC, le a sono presenti sotto $A_1 = A$ e $A_3 = C$ posizioniamo a_j per ricordarci della corrispondenza

Per maggior chiarezza, all'interno della tabella il simbolo "→" verrà usato per indicare le modifiche ai valori.

	A	B	C	D	E
AC	a_1	b_{12}	a_3	$b_{14} \rightarrow a_4$	b_{15}
ADE	a_1	$b_{22} \rightarrow b_{12}$	b_{23}	a_4	a_5
CDE	b_{31}	$b_{32} \rightarrow b_{12}$	a_3	a_4	a_5
AD	a_1	$b_{42} \rightarrow b_{12}$	b_{43}	a_4	b_{45}
B	b_{51}	a_2	b_{53}	b_{54}	b_{55}

$C \rightarrow D$ La prima e la terza riga coincidono sull'attributo C = a_3, quindi cambiamo b_{14} in a_4 in modo che la dipendenza funzionale sia soddisfatta (se le righe hanno valori uguali in C, devono avere valori uguali in D).

$AB \rightarrow E$ Non viene utilizzata in questo passo: la dipendenza funzionale è gia' soddisfatta, in quanto non ci sono (ancora) tuple uguali su AB e diverse su E, quindi non devono essere effettuati cambiamenti.

$D \rightarrow B$ Nelle prime quattro righe D= a_4, quindi cambiamo b_{22} in b_{12}, b_{32} in b_{12}, b_{42} in b_{12} (potevano scegliere una diversa sostituzione delle b, purché le rendesse tutte uguali).

	A	B	C	D	E
AC	a_1	b_{12}	a_3	a_4	$b_{15} \rightarrow a_5$
ADE	a_1	b_{12}	b_{23}	a_4	a_5
CDE	b_{31}	b_{12}	a_3	a_4	a_5
AD	a_1	b_{12}	b_{43}	a_4	$b_{45} \rightarrow a_5$
B	b_{51}	a_2	b_{53}	b_{54}	b_{55}

$C \rightarrow D$ Non viene utilizzata in questo passo: la dipendenza funzionale è gia' soddisfatta, in quanto non ci sono tuple uguali su C e diverse su D.

$AB \rightarrow E$ La prima, la seconda e la quarta riga coincidono sugli attributi AB = a_1,b_{12} quindi cambiamo b_{15} in a_5 e b_{45} in a_5 in modo che la dipendenza funzionale sia soddisfatta (se le righe hanno valori uguali in AB, devono avere valori uguali in E).

$D \rightarrow B$ Non viene utilizzata in questo passo: la dipendenza funzionale è gia' soddisfatta, in quanto non ci sono tuple uguali su D e diverse su B.

	A	B	C	D	E
AC	a_1	b_{12}	a_3	a_4	$b_{15} \rightarrow a_5$
ADE	a_1	b_{12}	b_{23}	a_4	a_5
CDE	b_{31}	b_{12}	a_3	a_4	a_5
AD	a_1	b_{12}	b_{43}	a_4	$b_{45} \rightarrow a_5$
B	b_{51}	a_2	b_{53}	b_{54}	b_{55}

$C \to D$ La dipendenza è già soddisfatta – niente da modificare.
$AB \to E$ La dipendenza è già soddisfatta – niente da modificare.
$D \to B$ La dipendenza è già soddisfatta – niente da modificare.

In questa iterazione non vengono mai apportate modifiche alla tabella (in quanto è in una forma legale), l'algoritmo quindi termina.
Ora occorre verificare la presenza di una tupla con tutte *a* .

Poiché non c'è una riga con tutte *a*, il join non è senza perdita.

2) Dati:

$\boldsymbol{R} = \{ \boldsymbol{A},\ \boldsymbol{B},\ \boldsymbol{C},\ \boldsymbol{D},\ \boldsymbol{E},\ \boldsymbol{H},\ \boldsymbol{I} \}$
$\boldsymbol{F} = \{ \boldsymbol{A} \to \boldsymbol{B},\ \boldsymbol{B} \to \boldsymbol{AE},\ \ \boldsymbol{DI} \to \boldsymbol{B},\ \boldsymbol{D} \to \boldsymbol{HI},\ \ \boldsymbol{HI} \to \boldsymbol{C},\ \ \boldsymbol{C} \to \boldsymbol{A} \}$
$\boldsymbol{\rho} = \{ \boldsymbol{ACD},\ \boldsymbol{BDEH},\ \boldsymbol{CHI} \}$

	A	B	C	D	E	H	I
ACD	a_1	b_{12}	a_3	a_4	b_{15}	b_{16}	b_{17}
BDEH	b_{21}	a_2	b_{23}	a_4	a_5	a_6	b_{27}
CHI	b_{31}	b_{32}	a_3	b_{34}	b_{35}	a_6	a_7

	A	B	C	D	E	H	I
ACD	a_1	b_{12}	a_3	a_4	b_{15}	$b_{16} \to a_6$	b_{17}
BDEH	$b_{21} \to a_1$	a_2	$b_{23} \to a_3$	a_4	a_5	a_6	$b_{27} \to b_{17}$
CHI	$b_{31} \to a_1$	b_{32}	a_3	b_{34}	b_{35}	a_6	a_7

$A \to B$ Non viene utilizzata in questa iterazione.
$B \to AE$ Non viene utilizzata in questa iterazione.
$DI \to B$ Ci sono due tuple uguali su D ma non su I - non si applica in questa iterazione

$D \to HI$ La prima e la seconda riga coincidono sull'attributo D = a_4, quindi cambiamo H e I ma separatamente $b_{16} \to a_6$ mentre $b_{27} \to b_{17}$

$HI \to C$ Ora abbiamo due tuple uguali su HI (la prima e la seconda entrambe con valori a_6, b_{17} quindi modifichiamo i valori della C nelle stesse tuple – b23 → a3 .

$C \to A$ Le tuple sono tutte uguali su C, quindi le facciamo diventare uguali su A, e poiché abbiamo la prima con valore a, diventano tutte a (b21 → a1, b31 → a1) .

	A	B	C	D	E	H	I
ACD	a_1	$b_{12} \to a_2$	a_3	a_4	$b_{15} \to a_5$	a_6	b_{17}
BDEH	a_1	a_2	a_3	a_4	a_5	a_6	b_{17}
CHI	a_1	$b_{32} \to a_2$	a_3	b_{34}	$b_{35} \to a_5$	a_6	a_7

$A \to B$ Le tuple sono tutte uguali su A, quindi le facciamo diventare uguali su B, e poiché abbiamo la seconda con valore a, diventano tutte a ($b_{12} \to a_2$, $b_{32} \to a_2$).

$B \to AE$ Ora le tuple sono tutte uguali su B, quindi devono diventare uguali anche su AE; su A sono già uguali, la seconda tupla ha una a sull'attributo E, quindi diventano tutte a ($b_{15} \to a_5$, $b_{35} \to a_5$).

$DI \to B$ La prima e la seconda tupla sono uguali su DI a_4, b_{17} , quindi devono diventare uguali su B ma lo sono già.

$D \to HI$ La dipendenza è già soddisfatta – niente da modificare.

$HI \to C$ La dipendenza è già soddisfatta – niente da modificare.

$C \to A$ La dipendenza è già soddisfatta – niente da modificare.

	A	B	C	D	E	H	I
ACD	a_1	a_2	a_3	a_4	a_5	a_6	b_{17}
BDEH	a_1	a_2	a_3	a_4	a_5	a_6	b_{17}
CHI	a_1	a_2	a_3	b_{34}	a_5	a_6	a_7

$A \to B$ La dipendenza è già soddisfatta – niente da modificare.
$B \to AE$ La dipendenza è già soddisfatta – niente da modificare.
$DI \to B$ La dipendenza è già soddisfatta – niente da modificare.
$D \to HI$ La dipendenza è già soddisfatta – niente da modificare.
$HI \to C$ La dipendenza è già soddisfatta – niente da modificare.
$C \to A$ La dipendenza è già soddisfatta – niente da modificare.

In questa iterazione non vengono mai apportate modifiche alla tabella (in quanto è in una forma legale), l'algoritmo quindi termina.
Ora occorre verificare la presenza di una tupla con tutte *a* .

Poiché non c'è una riga con tutte *a*, il join non è senza perdita.

Per altri esempi consultare questo link.

COPERTURA MINIMALE

Fin ora abbiamo analizzato come **verificare** che una decomposizione data soddisfi tutte le condizioni per essere una **buona decomposizione**, in particolare verificando:

- se la decomposizione preserva le dipendenze funzionali
- se è possibile ricostruire ogni istanza dello schema originale, tramite un join naturale di istanze della decomposizione, senza perdite di informazione

Adesso ci occuperemo di **come sviluppare una decomposizione** che soddisfi le condizioni (non più di verificare la correttezza di una decomposizione già fatta).

Partiamo dal presupposto che è **SEMPRE** possibile ottenere una valida decomposizione che soddisfi le condizioni. Anche in questo caso ci serviremo di un algoritmo. È però importante sapere che la decomposizione ottenuta tramite l'algoritmo non è l'unica possibile che soddisfi le condizioni richieste. Lo stesso algoritmo a seconda dell'input di partenza può fornire risultati diversi. Proprio perché **non esiste LA decomposizione giusta**, l'algoritmo di decomposizione non può essere usato come un algoritmo di verifica.

Prima di procedere all'algoritmo dobbiamo introdurre il concetto di **copertura minimale**, che sarà la base di partenza dell'algoritmo di decomposizione.

DEFINIZIONE

Sia F un insieme di dipendenze funzionali. Una copertura minimale di F è un insieme G di dipendenze funzionali equivalente ad F tale che:

I. Per ogni dipendenza funzionale in G, la parte destra è un **singleton**, cioè è costituita da un unico attributo (ogni attributo nella parte **destra** non è ridondante)

II. **Per nessuna** dipendenza funzionale $X \to A$ in G , $\nexists\, X' \subset X : G \equiv G - \{X \to A\} \cup \{X' \to A\}$
(ogni attributo nella parte **sinistra** non è ridondante)

III. **Per nessuna** dipendenza funzionale $X \to A$ in G, $G \equiv G - \{X \to A\}$
(ogni **dipendenza** non è ridondante)

CALCOLO

Per ogni insieme di dipendenze funzionali F esiste una copertura minimale che può essere ottenuta in tempo polinomiale in tre passaggi. Per semplificare la comprensione affianchiamo la spiegazione ad un esempio.

IMPORTANTE : l'ordine dei passaggi non va invertito!

Dati:

$\boldsymbol{R} = \{\boldsymbol{A},\ \boldsymbol{B},\ \boldsymbol{C},\ \boldsymbol{D},\ \boldsymbol{E},\ \boldsymbol{H}\}$
$\boldsymbol{F} = \{\boldsymbol{AB} \to \boldsymbol{CD},\ \boldsymbol{C} \to \boldsymbol{E},\ \boldsymbol{AB} \to \boldsymbol{E},\ \boldsymbol{ABC} \to \boldsymbol{D}\ \}$

Trovare la copertura minimale G di F:

I. **Usando la regola della composizione le parti destre vengono ridotte a singleton**.

$$G = \{\ AB \to C,\ AB \to D,\ C \to E,\ AB \to E,\ ABC \to D\ \}$$

II. **Verificare che nelle dipendenze non ci siano ridondanze nei determinanti (parti sinistre).**

- Analizziamo $\boldsymbol{AB} \to \boldsymbol{C}$, controlliamo se $A \to C \in F^+$ e quindi se $C \in (A)_F^+$.
  A non compare mai da solo nei determinanti, quindi abbiamo
  $$(A)_F^+ = \{A\}$$
  lo stesso vale per B
  $$(B)_F^+ = \{B\}$$
  siccome $C \notin (A)_F^+$ e $C \notin (B)_F^+$, non possiamo modificare la parte sinistra.
  Lo stesso vale anche per $AB \to D$ e $AB \to E$.

- $C \to E$ non è ridondante, quindi non la controlliamo.

- Analizziamo $ABC \to D$
  $$(AB)_F^+ = \{ABCDE\}$$
  siccome $D \in (AB)_F^+$ posso eliminare C.
  Inoltre posso eliminare completamente la dipendenza in quanto $AB \to D$ fa già parte di F.

Abbiamo quindi
$$G = \{\ AB \to C,\ AB \to D,\ C \to E,\ AB \to E\}$$

III. **Vediamo se l'insieme contiene delle dipendenze ridondanti.**

- Possiamo subito notare che C viene determinato unicamente da AB, non possiamo quindi eliminare questa dipendenza in quanto non riusciremo più ad arrivare a C tramite $(AB)_g^+$ (con $g = G - \{AB \to C\}$).
  Lo stesso vale per D.

- Proviamo ad eliminare $C \to E$, vediamo se E sarebbe comunque presente nella chiusura di C rispetto al nuovo insieme (controlliamo se $E \in (C)_g^+$ con $g = \{\ AB \to C,\ AB \to D,\ AB \to E\}$).
  $$(C)_g^+ = \{C\}$$
  siccome $E \notin (C)_g^+$ non possiamo rimuovere $C \to E$.

- Proviamo ad eliminare $AB \to E$, controlliamo nuovamente se, eliminando tale dipendenza, il dipendente (E) sarebbe comunque presente nella chiusura del determinante (AB) rispetto al nuovo insieme ($g = \{\ AB \to C,\ AB \to D,\ C \to E\}$).
  $$(AB)_g^+ = \{ABCDE\}$$
  Siccome $E \in (AB)_g^+$ possiamo eliminare la dipendenza $AB \to E$ in quanto comunque preservata nella chiusura del nuovo insieme.

Abbiamo quindi che $\boldsymbol{G} = \{\boldsymbol{AB} \to \boldsymbol{C},\ \boldsymbol{AB} \to \boldsymbol{D},\ \boldsymbol{C} \to \boldsymbol{E}\}$ che è equivalente ad F in quanto $G^+ = F^+$.

ESEMPI

1) Dato

$\boldsymbol{F = \{\, BC \to DE,\ C \to D,\ B \to D,\ E \to L,\ D \to A,\ BC \to AL \}}$

determinare una copertura minimale G di F.
(per calcolare la copertura minimale non occorre conoscere R)

PASSAGGI:

I. $\boldsymbol{G = \{\, BC \to D,\ BC \to E,\ C \to D,\ B \to D,\ E \to L,\ D \to A,\ BC \to A,\ BC \to L \}}$

II. $\boldsymbol{BC \to D}$
$D \in (B)_F^+ = \{\, ABD \,\} \Rightarrow$ *posso eliminare C*
$B \to D$ esiste già, quindi posso eliminare l'intera dipendenza

$\boldsymbol{BC \to E}$
$E \notin (B)_F^+$ *e* $E \notin (C)_F^+ = \{\, ACD \,\}$
non posso modificare la dipendenza

$\boldsymbol{BC \to A}$
$A \in (B)_F^+ \Rightarrow$ *posso eliminare C*

$\boldsymbol{BC \to L}$
$L \notin (B)_F^+$ *e* $L \notin (C)_F^+$
non posso modificare la dipendenza

$\boldsymbol{G = \{\, BC \to E,\ C \to D,\ B \to D,\ E \to L,\ D \to A,\ B \to A,\ BC \to L \}}$

III. $\boldsymbol{BC \to E}$
$E \notin (BC)_g^+ = \{ABCDL\}$
non posso eliminare la dipendenza

$\boldsymbol{C \to D}$
$D \notin (C)_g^+ = \{C\}$
non posso eliminare la dipendenza

$\boldsymbol{B \to D}$
$D \notin (B)_g^+ = \{AB\}$
non posso eliminare la dipendenza

$\boldsymbol{E \to L}$
$L \notin (E)_g^+ = \{E\}$
non posso eliminare la dipendenza

$\boldsymbol{D \to A}$
$A \notin (D)_g^+ = \{D\}$
non posso eliminare la dipendenza

$\boldsymbol{B \to A}$
$A \in (B)_g^+ = \{ABD\}$
posso eliminare la dipendenza

$\boldsymbol{BC \to L}$
$L \in (BC)_g^+ = \{ABCDEL\}$
posso eliminare la dipendenza

$\boldsymbol{G = \{\, BC \to E,\ C \to D,\ B \to D,\ E \to L,\ D \to A \}}$

2) Dato

$\boldsymbol{F = \{\, AB \to C,\ A \to E,\ E \to D,\ D \to C,\ B \to A \,\}}$

PASSAGGI:

I. Tutti i dipendenti sono già singleton.

II. $\boldsymbol{AB \to C}$
$C \in (A)_F^+ = \{\, ACDE \,\}$ *e* $C \in (B)_F^+ = \{\, ABCDE \,\}$
possiamo sostituire sia con $A \to C$ che con $B \to C$

$\boldsymbol{G = \{\, A \to C,\ A \to E,\ E \to D,\ D \to C,\ B \to A \,\}}$

III. $\boldsymbol{A \to C}$
$(A)_g^+ = \{\, ACDE \,\}$
possiamo eliminare la dipendenza

Notiamo che tutti i restanti dipendenti compaiono solo una volta nell'insieme, non possiamo quindi eliminare altre dipendenze.

$$\boldsymbol{G = \{ A \to E,\ \ E \to D,\ \ D \to C,\ \ B \to A \}}$$

Per altri esercizi clicca qui.

DECOMPOSIZIONE DI UNO SCHEMA

Dato uno schema di relazione R e un insieme di dipendenze funzionali F su R esiste sempre una decomposizione $\rho = \{R_1, R_2, \dots, R_k\}$ di R tale che:

- per ogni $i = 1, \dots, k$, R_i è in 3NF
- ρ preserva F
- ρ ha un join senza perdita

Una tale decomposizione può essere calcolata, a partire da una copertura minimale, in tempo polinomiale.

ALGORITMO

Input	schema copertura minimale su R	$\boldsymbol{R}$ $\boldsymbol{F}$
Output	decomposizione di R	$\boldsymbol{\rho}$

Inizio

$S = \emptyset$

for every A ∈ R tale che A non è coinvolto in nessuna dipendenza funzionale in F:

$S = S \cup \{A\}$

if $\boldsymbol{S \neq \emptyset}$:

$R = R - S$

$\rho = \rho \cup \{S\}$

if esiste una dipendenza funzionale in F che coinvolge tutti gli attributi di R :

$\rho = \rho \cup \{R\}$

else :

for every $X \to A \in F$:

$\rho = \rho \cup \{XA\}$

return ρ

Fine

Al fine di assicurare il join senza perdite, a tale decomposizione verrà aggiunto un sottoschema K tale che K è una chiave per R.

Per consultare dimostrazioni e teoremi di questo algoritmo clicca qui.

ESEMPI

1) Dati

$$\boldsymbol{R = \{ A,\ \ B,\ \ C,\ \ D,\ \ E,\ \ H \}}$$
$$\boldsymbol{F = \{ AB \to CD,\ \ C \to E,\ \ AB \to E,\ \ ABC \to D\ \}}$$

   a. Verificare che ABH è una chiave per R
   b. Sapendo che ABH è l'unica chiave per R, verificare che R non è in 3NF
   c. Trovare una copertura minimale G di F
   d. Trovare una decomposizione ρ di R tale che preserva G e ogni schema in ρ è in 3NF
   e. Trovare una decomposizione ρ di R tale che preserva G, ha un join senza perdita e ogni schema di ρ è in 3NF

$\boldsymbol{R = \{ A,\ B,\ C,\ D,\ E,\ H \}}$ $\quad$ $\boldsymbol{F = \{ AB \rightarrow CD, C \rightarrow E,\ AB \rightarrow E,\ ABC \rightarrow D \}}$

a. **Verificare che ABH è una chiave** significa verificare che:
   - $ABH \rightarrow R$

Inizio
$Z = ABH$
$S = \{ L : Y \rightarrow V \in F \wedge L \in V \wedge Y \subseteq ABH \}$ $\quad$ $S = CDE$ per $AB \rightarrow CD,\ AB \rightarrow E$

while $S \not\subset Z$: $\quad$ $CDE \not\subset ABH$
$Z = Z \cup S$ $\quad$ $Z = ABCDEH$
$S = \{ A : Y \rightarrow V \in F \wedge A \in V \wedge Y \subseteq Z \}$ $\quad$ $S = ABCDEH$

while $S \not\subset Z$: $\quad$ $ABCDEH \subseteq ABCDEH$

return Z
Fine

   - Nessun sottoinsieme di ABH determina funzionalmente R

     Nessun attributo in ABH compare come dipendente e singolarmente non determinano nulla, quindi devono per forza far parte tutti e 3 della chiave.

b. Verificare che lo schema non è in terza forma normale

   Basta osservare la presenza delle dipendenze parziali $AB \rightarrow CD$ e $AB \rightarrow E$.

c. Trovare la copertura minimale

   I. $\boldsymbol{G = \{ AB \rightarrow C,\ AB \rightarrow D,\ C \rightarrow E,\ AB \rightarrow E,\ ABC \rightarrow D \}}$

   II. $\boldsymbol{AB \rightarrow C}$
   A e B singolarmente non determinano nulla, abbiamo quindi
   $(A)_F^+ = \{A\}$
   $(B)_F^+ = \{B\}$
   $\boldsymbol{ABC \rightarrow D}$
   $D \in (AB)_F^+ = \{ ABCDE \}$
   possiamo eliminare D, ma $AB \rightarrow D$ appartiene già a G, quindi possiamo eliminare tutta la dipendenza.

   $\boldsymbol{G = \{ AB \rightarrow C,\ AB \rightarrow D,\ C \rightarrow E,\ AB \rightarrow E \}}$

   III. C e D sono determinati unicamente da AB, quindi non possiamo eliminare queste dipendenze.

   $\boldsymbol{C \rightarrow E}$
   $E \notin (C)_g^+ = \{C\} \Rightarrow$ non possiamo eliminare $C \rightarrow E$
   $\boldsymbol{AB \rightarrow E}$
   $E \in (AB)_g^+ = \{ABCDE\} \Rightarrow$ possiamo eliminare $AB \rightarrow E$

   $\boldsymbol{G = \{ AB \rightarrow C,\ AB \rightarrow D,\ C \rightarrow E \}}$

d. Calcolare la decomposizione

Inizio
$S = \emptyset$

for every $A \in R$ tale che A non è coinvolto in nessuna dipendenza funzionale in G:
$S = S \cup \{A\}$ $\quad$ $S = \emptyset \cup H$

if $S \neq \emptyset$:
$R = R - S$ $\quad$ $R = ABCDEH - H$
$\rho = \{S\}$ $\quad$ $\rho = \{H\}$

if esiste una dipendenza funzionale in G che coinvolge tutti gli attributi di R :
$\rho = \rho \cup \{R\}$
else :
for every $X \rightarrow A \in G$:
$\rho = \rho \cup \{XA\}$ $\quad$ $\rho = \{H, ABC, ABD, CE\}$
return ρ
Fine

e. Per avere una decomposizione senza perdita aggiungiamo alla decomposizione ottenuta un sottoschema che contenga la chiave ABH.

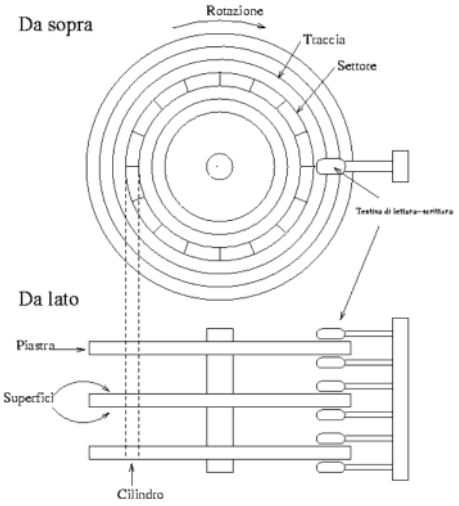
$$\boldsymbol{\rho} = \{\boldsymbol{H}, \boldsymbol{ABC}, \boldsymbol{ABD}, \boldsymbol{CE}, \boldsymbol{ABH}\}$$

5 - Organizzazione fisica dei dati

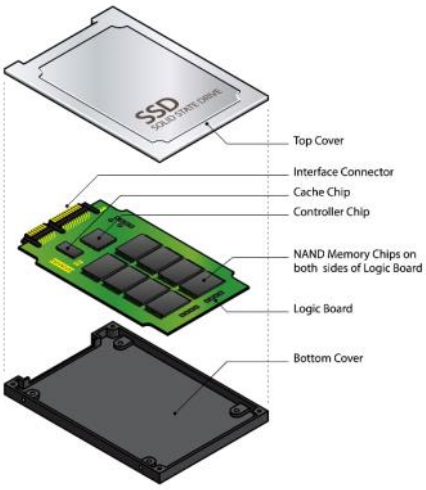
Un requisito fondamentale di un sistema di gestione di basi di dati è l'efficienza, cioè la capacità di rispondere alle richieste dell'utente il più rapidamente possibile.
Poiché una particolare organizzazione fisica dei dati può rendere efficiente l'elaborazione di particolari richieste, l'amministratore della base di dati durante il progetto fisico della base di dati deve tener presente quali operazioni saranno effettuate più frequentemente.

Generalmente ad ogni oggetto base di un modello logico (schemi di relazione nel modello relazionale, segmenti nel modello gerarchico, tipi di record nel modello a rete) corrisponde un file di record che hanno tutti lo stesso formato, cioè gli stessi campi (che corrispondono agli attributi nel caso del modello relazionale e ai campi di segmenti e di tipi di record negli altri due modelli).

I file sono memorizzati su disco; un disco è partizionato in blocchi di formato fisso (compreso tra 2^9 e 2^{12} byte); un blocco corrisponde generalmente ad un settore di una traccia; la divisione in blocchi (o pagine) avviene al momento della formattazione.
Il costo di un accesso a disco, cioè del trasferimento di un blocco da/a memoria secondaria è molto superiore a quello di elaborazione di un blocco in memoria principale o in una SSD (Solid State Driver). Pertanto la valutazione del tempo di risposta del sistema ad una richiesta dell'utente viene effettuata in termini di numero di accessi a disco. In realtà non sempre quando si deve elaborare un blocco è necessario effettuare un accesso a disco, in quanto il sistema quando trasferisce un blocco in memoria principale lo memorizza in un buffer e mantiene i blocchi trasferiti fin quando è possibile ricordando quali blocchi sono nei buffer (memoria virtuale). Inoltre il costo di un accesso a disco non è fisso, ma dipende dalla posizione del blocco rispetto all'ultimo blocco trasferito in quanto da ciò dipendono sia lo spostamento della testina da un cilindro all'altro sia la rotazione del disco. Tuttavia per valutare i costi delle operazioni elementari su file con determinata organizzazione fisica assumeremo che ogni lettura e scrittura di un blocco richieda il trasferimento del blocco da o a memoria secondaria e che ogni accesso a memoria secondaria abbia costo costante.



Un disco rigido, hard disk, o HDD (hard disk drive), è un dispositivo di memoria di massa di tipo magnetico, che utilizza uno o più dischi magnetizzati per l'archiviazione di dati e applicazioni. Il tempo di accesso ad un disco rigido si attesta tra i 5 e 10 millesimi di secondo, oltre 50 volte maggiore delle memorie a stato solido (SSD).



Un'unità di memoria a stato solido, SSD (Solid State Drive) è un dispositivo di memoria di massa basato su un semiconduttore, che utilizza una memoria allo stato solido, in particolare memoria flash NAND, per l'archiviazione di dati. L'accesso in memoria richiede un tempo nell'ordine dei decimi di millesimi di secondo.

Nel caso del modello relazionale, **ad ogni relazione corrisponde un file di record**, che hanno tutti lo **stesso tipo** (numero e tipo dei campi). **Ad ogni attributo corrisponde un campo**, mentre **ad ogni tupla corrisponde un record**. In ogni file ci sono quindi record appartenenti ad un'unica relazione (ad una **tabella**).

RECORD

Come abbiamo detto i **record fisici contengono campi che corrispondono ad attributi di relazioni**. Oltre a questi campi, però, un record fisico può contenere campi che forniscono informazioni sul record stesso o che "puntano" ad altri record.

Un puntatore ad un record/blocco è un dato che permette di accedere rapidamente a quel record/blocco. Pertanto un puntatore può essere l'indirizzo dell'inizio (primo byte) del record su disco. Tale scelta però può non essere adeguata se si vuole avere piena libertà di muovere un record.

Nel secondo caso è preferibile assumere come puntatore una coppia (b, k) dove:

- b è l'indirizzo del blocco che contiene il record
- k è il valore di uno o più campi che servono come chiave nel file a cui il record appartiene.

In tal modo è possibile spostare il record all'interno del blocco.

All'inizio di un record ci possono essere campi che non contengono dati ma informazioni sul record.
Ad esempio:

- Alcuni byte possono essere utilizzati per **specificare il tipo del record** (è necessario quando in uno stesso blocco sono memorizzati record di tipo diverso)
- Uno o più byte possono essere utilizzati per **specificare la lunghezza del record** se il record ha campi a lunghezza variabile
- Un byte può contenere un **bit di cancellazione** (serve nel caso che il record sia puntato; in tal caso infatti lo spazio occupato dal record cancellato non può essere riutilizzato) o un **bit di usato/non usato** per specificare se in quello spazio c'è un record oppure è vuoto (contiene valori casuali).

Per poter accedere ad un campo di un record contenente un dato è necessario sapere qual è il primo byte del campo nel record.

Se tutti i campi del record hanno lunghezza fissa, basta ordinarli; infatti, una volta ordinati, l'inizio di ciascun campo sarà sempre ad un numero fisso di byte dall'inizio del record: il numero di byte del record che precedono il campo è detto **offset del campo**.
Se il record contiene campi a lunghezza variabile, allora l'offset di un campo può variare da un record all'altro. In tal caso si possono usare due strategie:

- All'inizio di ogni campo c'è un contatore che specifica la lunghezza del campo in numero di byte
- All'inizio del record ci sono dei puntatori (qui un puntatore è inteso come l'offset del campo) all'inizio di ciascun campo a lunghezza variabile (tutti i campi a lunghezza fissa precedono quelli a lunghezza variabile).

Nel primo caso per individuare la posizione di un campo bisogna esaminare i campi precedenti per vedere quanto sono lunghi; quindi la prima strategia è meno efficiente della seconda.

BLOCCHI

Analogamente a quanto accade per un record, anche in un blocco può essere riservato spazio per: memorizzare informazioni sul blocco stesso; far in modo che gli offset di campi siano multipli di 4 (spazio "sprecato"); collegarlo tramite puntatore ad altri blocchi in una lista.

Se un blocco contiene solo record di lunghezza fissa allora il blocco può essere suddiviso in aree (sottoblocchi) di lunghezza fissa ciascuna delle quali può contenere un record.
Se devo inserire un record nel blocco devo cercare un area non usata; se il bit "usato/non usato" è in ciascun record ciò può richiedere la scansione di tutto il blocco, per evitare ciò si possono raccogliere tutti i bit "usato/non usato" in uno o più byte all'inizio del blocco.

Se un blocco contiene **record di lunghezza variabile**, si possono usare **2 strategie**:

- Si pone in ogni record un **campo che ne specifica la lunghezza** in termini di numero di byte
- Si pone **all'inizio del blocco una directory contenente i puntatori** (offset) ai record nel blocco.
  Poiché il numero di record che possono entrare in un blocco è in questo caso variabile, la directory può essere realizzata in uno dei modi seguenti:
  - la directory è preceduta da un campo che specifica quanti sono i puntatori nella directory
  - la directory è una lista di puntatori (la fine della lista è specificata da uno 0).
  - la directory ha dimensione fissa (può contenere un numero fisso di puntatori) e contiene il valore 0 negli spazi che non contengono puntatori.

FILE

In questo paragrafo esamineremo diversi tipi di organizzazione fisica di file che consentono la ricerca di record in base al valore di uno o più campi chiave.
Il termine “chiave” non va inteso nel senso in cui viene usato nella teoria relazionale, in quanto un valore della chiave non necessariamente identifica univocamente un record nel file.

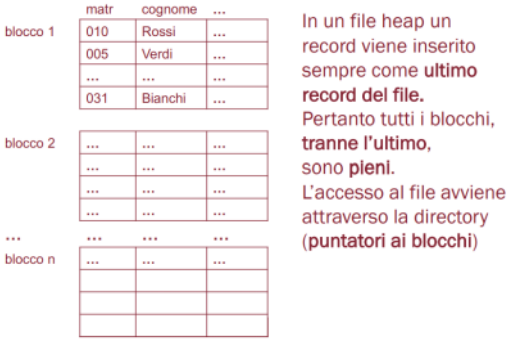
Le operazioni elementari su questi tipi di file sono dunque:

- la **ricerca** di uno o più record con un dato valore per la chiave;
- l’**inserimento** di un record con un dato valore per la chiave;
- la **cancellazione** di uno o più record con un dato valore per la chiave;
- la **modifica** di uno o più record con un dato valore per la chiave.

HEAP

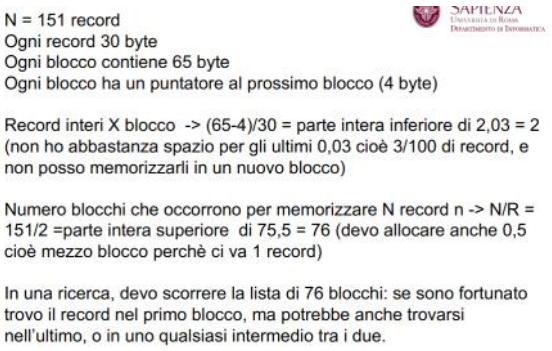
In questo tipo di organizzazione i **record sono memorizzati nei blocchi senza alcun ordine, al di fuori da quello di inserimento** (ogni nuovo record viene sempre inserito come ultimo record del file).
Ciò comporta prestazioni peggiori in termini di ricerca ma, se sono ammessi duplicati, nettamente migliori in termini di inserimento .

L'accesso al file avviene attraverso una directory che contiene i puntatori ai blocchi; se le dimensioni lo consento, tale directory può essere mantenuta in memoria principale durante l'utilizzo dei file, altrimenti saranno necessari ulteriori accessi per portare in memoria principale i necessari blocchi della directory.



RICERCA

Se si vuole cercare un record occorre scandire tutto il file, iniziando dal primo record fino ad incontrare il record desiderato.



Se la chiave di ricerca identifica univocamente un solo record nel file:

- N = numero di record nel file
- R = numero di record che possono essere memorizzati in un blocco
- $n = \left\lceil \frac{N}{R} \right\rceil$ = numero di blocchi del file

$$\frac{R*1+R*2+\cdots+R*i+\cdots+R*n}{N} = \frac{R*(1+2+\cdots+i+\cdots+n)}{N} = \frac{R}{N} * \frac{n\,(n+1)}{2} = \frac{1}{n} * \frac{n(n+1)}{2} \approx \left\lceil \frac{n}{2} \right\rceil$$

Per ottenere il costo medio occorre sommare i costi per accedere ai singoli record e quindi dividere tale somma per il numero dei record.
Per ognuno degli R record nell’i-esimo blocco sono necessari i accessi.

La **ricerca di un record con un dato valore per la chiave richiede in media** $\left\lceil \frac{n}{2} \right\rceil$ **accessi**.

Se la chiave di ricerca non identifica univocamente un record (sono ammessi duplicati), dovremmo comunque accedere ad n blocchi, perché non possiamo sapere se abbiamo trovato tutte le occorrenze della chiave fin quando non abbiamo analizzato ogni record.

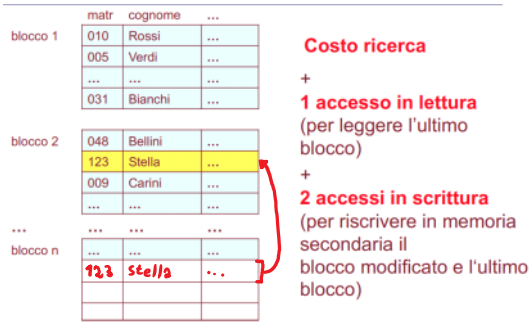
INSERIMENTO

L'inserimento richiede la lettura dell'ultimo blocco del file, se in tale blocco c'è spazio sufficiente per memorizzare il nuovo record, il record viene inserito, altrimenti viene chiesto un nuovo blocco al file system. Dopo l'inserimento del record nel blocco, il blocco deve essere scritto su memoria secondaria. Sono quindi sufficienti 2 accessi (uno per lettura e uno per scrittura).

Se il nostro inserimento non ammette duplicati, l'inserimento deve essere preceduto dalla ricerca, quindi dobbiamo aggiungere una media di $\left\lceil \frac{n}{2} \right\rceil$ accessi per verificare che non esista già un secondo record con la chiave data.

CANCELLAZIONE

La cancellazione di un record richiede $\left\lceil \frac{n}{2} \right\rceil$ accessi per la ricerca del record da cancellare e 1 accesso in scrittura.
Se si vuole recuperare lo spazio occupato dal record cancellato e i record hanno lunghezza fissa, si può trasferire un record che si trova nell'ultimo blocco (restituendo al sistema il blocco se non vi sono altri record) al posto di quello cancellato.
Se i record hanno lunghezza variabile, si possono far slittare i record nel blocco e, se in tal modo si è ottenuto uno spazio sufficiente, si può procedere come nel caso precedente.

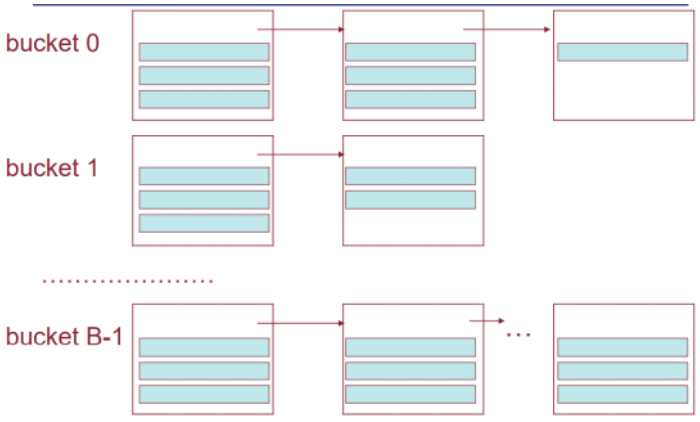


MODIFICA

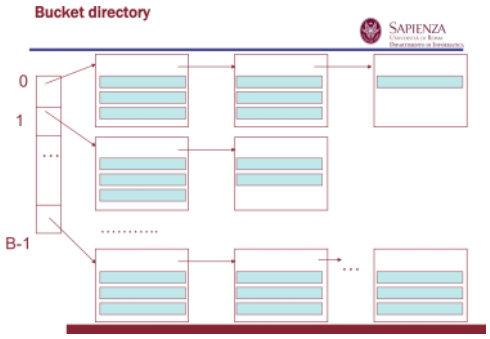
La modifica di un record richiede $\left\lceil \frac{n}{2} \right\rceil$ accessi per la ricerca del record da e 1 accesso in scrittura.
Se ammettiamo duplicati queste operazioni vanno ripetute fin quando non si abbiano cercato e modificato tutte le occorrenze.

HASH

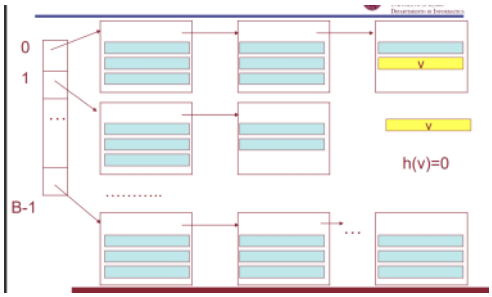
Il file è **suddiviso in bucket** (secchi) numerati da 0 a $B-1$. **Ciascun bucket è costituito da uno o più blocchi** (collegati mediante puntatori) ed è **organizzato come un heap**.



L'**accesso ai bucket avviene attraverso la bucket directory che contiene B elementi**. L'i-esimo elemento contiene l'indirizzo (bucket header) del primo blocco dell'i-esimo bucket.



Dato un valore v per la chiave, il numero del bucket in cui deve trovarsi un record con chiave v è calcolato mediante una funzione che prende il nome di **funzione hash**.



Una qualsiasi operazione (ricerca, inserimento, cancellazione, modifica) su un file hash richiede:

- valutazione di $h(v)$ per **individuare il bucket**
- **esecuzione dell'operazione sul bucket** che è organizzato come un heap

Poiché l'**inserimento di un record viene effettuato sull'ultimo blocco del bucket**, è opportuno che la bucket directory contenga anche, per ogni bucket, un **puntatore all'ultimo record del bucket**.

Pertanto, se la funzione hash distribuisce uniformemente i record nei bucket: **ogni bucket è costituito da $\frac{n}{B}$ $\left(\frac{numero\ di\ blocchi\ nel\ file}{numero\ di\ bucket}\right)$ blocchi** e quindi il **costo richiesto di un'operazione è circa $\frac{1}{B-esimo}$ del costo** della stessa operazione se il file fosse organizzato come un heap.

Una funzione hash per essere "buona" deve ripartire uniformemente i record nei bucket, cioè al variare del valore della chiave, **deve assumere con la "stessa" probabilità uno dei valori compresi tra 0 e $B-1$**.

> In genere, una funzione hash trasforma la chiave in un intero, divide questo intero per B, e fornisce il resto della divisione come numero del bucket in cui deve trovarsi il record con quel valore della chiave.

Da quanto detto appare evidente che quanti più sono i bucket tanto più basso è il costo di ogni operazione.

D'altra parte limitazioni al numero dei bucket derivano dalle seguenti considerazioni:

- Ogni bucket deve avere **almeno un blocco**.
- Se le dimensioni della bucket directory sono tali che non può essere mantenuta in memoria principale durante l'utilizzo del file, ulteriori accessi sono necessari per leggere i blocchi della bucket directory.

ESEMPIO 1

Supponiamo di avere un file di $250.000\ record$ dove:

- Ogni record occupa $300\ byte$, di cui 75 per il campo chiave.
- Ogni blocco contiene $1024\ byte$.
- Un puntatore a blocco occupa $4\ byte$.

a. Se usiamo una organizzazione hash con $1200\ bucket$, **quanti blocchi occorrono per la bucket directory**?

b. **Quanti blocchi occorrono per i bucket**, assumendo una distribuzione uniforme dei record nei bucket?

c. Assumendo ancora che tutti i bucket contengano il numero medio di record, qual è il **numero medio di accessi a blocco per ricercare un record presente nel file**?

d. **Quanti bucket dovremmo creare per avere un numero medio di accessi a blocco inferiore o uguale a 10**, assumendo comunque una distribuzione uniforme dei record nei bucket?

Numero record	$NR = 250.000$
Taglia record	$R = 300\ byte$
Dimensione campo chiave	$K = 75\ byte$
Capacità blocco	$CB = 1024\ byte$
Taglia puntatore	$P = 4\ byte$

a. Consideriamo:

$$B = numero\ di\ bucket = 1200$$
$$BD = numero\ di\ blocchi\ per\ la\ bucket\ directory$$

Vediamo quanti puntatori possono entrare in un blocco:

$$PB = \left\lfloor \frac{CB}{P} \right\rfloor = \left\lfloor \frac{1024}{4} \right\rfloor = 256$$

Prendiamo la parte intera inferiore perché assumiamo che i record siano contenuti interamente nel blocco, i record devono essere interi. In questo caso CB è multiplo di P ma non è sempre vero.

Ci occorreranno

$$BD = \left\lceil \frac{B}{PB} \right\rceil = \left\lceil \frac{1200}{256} \right\rceil = \lceil 4,69 \rceil = 5$$

blocchi per la bucket directory (prendiamo la parte intera superiore perché, non essendo stato specificato diversamente dall'esercizio, i blocchi vengono allocati interamente).

NB:

Se viene chiesto che nella bucket directory venga memorizzato anche il puntatore all'ultimo blocco del bucket occorre considerare coppie intere di puntatori

$$\mathbf{PB} = \left\lfloor \frac{\mathbf{CB}}{\mathbf{2 * P}} \right\rfloor$$

b. Abbiamo record a lunghezza fissa, quindi supponiamo di non avere una directory di record all'inizio del blocco. Serve però un puntatore per ogni blocco per linkare i blocchi dello stesso bucket. In un blocco dobbiamo quindi memorizzare il maggior numero possibile di record e in più un puntatore per un eventuale prossimo blocco nel bucket.

Se indichiamo con RBL il massimo numero di record memorizzabili in un blocco, avremo

$$R * RBL + P \leq CB \Rightarrow 300RBL + 4 \leq 1024 \Rightarrow RBL \leq 1020/300 = 3,4$$

M deve essere intero, perché non essendo stato detto altrimenti nella traccia, i record non possano trovarsi a cavallo di due blocchi, quindi assumiamo

$$RBL = 3$$

In alternativa

$$RBL = \left\lfloor \frac{CB - P}{R} \right\rfloor = \left\lfloor \frac{1020}{300} \right\rfloor = \lfloor 3,4 \rfloor = 3$$

NB:

Si potrebbe chiedere che ogni blocco abbia anche un puntatore al blocco precedente quindi

$$RBL * R + 2 * P \leq CB$$

oppure

$$RBL = \left\lfloor \frac{CB - 2 * P}{R} \right\rfloor$$

Se la distribuzione dei record nei bucket è uniforme, indicando con RB il numero di record in un bucket, avremo

$$RB = \left\lceil \frac{NR}{B} \right\rceil = \left\lceil \frac{250.000}{1200} \right\rceil = \lceil 208,3 \rceil = 209$$

record per ogni bucket (prendiamo la parte intera superiore perché i record devono essere inseriti tutti).
Indichiamo con NB il numero di blocchi per ogni bucket, abbiamo

$$NB = \left\lceil \frac{RB}{RBL} \right\rceil = \left\lceil \frac{209}{3} \right\rceil = 70 .$$

Indichiamo con BB il numero complessivo di blocchi per il file hash, abbiamo

$$BB = NB * B = 70 * 1200 = 84.000 .$$

c. Se la distribuzione dei record nei bucket è uniforme, in un bucket avremo come detto

$$NB = \left\lceil \frac{RB}{RBL} \right\rceil = 70$$

blocchi per bucket. Poiché la ricerca avviene solo sul bucket individuato in base al risultato dell'applicazione della funzione hash alla chiave del record, avremo un numero di accessi a blocco pari a quello che si avrebbe su un heap della stessa dimensione del bucket (cioè $1/B$ rispetto alla dimensione originale). In media accederemo alla metà di questi blocchi, quindi indicando con MA il numero medio di accessi, avremo

$$MA = \left\lceil \frac{NB}{2} \right\rceil = \left\lceil \frac{70}{2} \right\rceil = 35$$

a questi occorrerà aggiungerne 1 se il blocco della bucket directory relativo al bucket in cui si trova il record non si trova già in memoria principale.

d. Per avere un numero di accessi a blocco inferiore o al massimo uguale a 10, riscriviamo l'espressione di MA in modo che vi compaia esplicitamente il numero di bucket B, e tralasciando per semplicità gli arrotondamenti che abbiamo effettuato via via nei calcoli tranne l'ultimo.

Avremo

$$MA = \left\lceil \frac{NB}{2} \right\rceil = \left\lceil \frac{\left(\frac{RB}{RBL}\right)}{2} \right\rceil = \left\lceil \frac{\left(\frac{NR}{B}\right)}{\frac{RBL}{2}} \right\rceil = \left\lceil \frac{NR}{2(B * M)} \right\rceil$$

Vogliamo calcolare quindi B in modo che

$$\left\lceil \frac{NR}{2(B * M)} \right\rceil \leq 10 \quad \Rightarrow \quad B \geq \frac{NR}{20M} \quad \Rightarrow \quad B \geq \frac{250\,000}{20 * 3} = 4167$$

(l'inversione della disuguaglianza deriva da fatto che il numero medio di accessi decresce con l'aumentare del numero di bucket).

Verifichiamo infatti che in questo caso avremo

$$RB = \left\lceil \frac{NB}{2} \right\rceil = \left\lceil \frac{250\,000}{4167} \right\rceil = 60$$

record per ogni bucket, quindi

$$\mathrm{NB} = \left\lfloor \frac{\mathrm{RB}}{RBL} \right\rfloor = 20$$

record per bucket, e infine

$$MA = \left\lceil \frac{NB}{2} \right\rceil = 10$$

accessi a blocco.

Si poteva anche ragionare in un altro modo.
Siccome

$$\boldsymbol{MA = \left\lceil \frac{NB}{2} \right\rceil}$$

per avere $MA \leq 10$ dobbiamo fare in modo che sia $NB \leq 20$ (NB è il numero di blocchi in un bucket). Abbiamo allora

$$\mathbf{RB} = \boldsymbol{RBL} * \mathbf{NB} \leq \mathbf{M} * \mathbf{20}$$

cioè nel nostro caso $RB \leq 60$ (ricordiamo che RB è il numero di record per bucket). Per avere un numero di record per bucket inferiore a 60, deve essere

$$\boldsymbol{\frac{NR}{B} \leq 60}$$

e quindi

$$\boldsymbol{B \geq \left(\frac{250\,000}{60}\right)}$$

ottenendo lo stesso risultato.

ESEMPIO 2

Supponiamo di avere un file di $\boldsymbol{780.000\ record}$ dove:

- Ogni record occupa $\boldsymbol{250\ byte}$.
- Ogni blocco contiene $\boldsymbol{1024\ byte}$.
- Un puntatore a blocco occupa $\boldsymbol{4\ byte}$.
- Usiamo una organizzazione hash con $\boldsymbol{2500\ bucket}$.

a. Quanti blocchi dobbiamo utilizzare complessivamente per la bucket directory e per i bucket, assumendo una distribuzione uniforme dei record nei bucket?

b. Quanti blocchi dobbiamo utilizzare complessivamente per i bucket, assumendo che il 30% dei record sia distribuito in modo uniforme su 1000 bucket, e che il restante 70% dei record sia distribuito in modo uniforme sui 1500 bucket rimanenti?

Numero record	$\boldsymbol{NR = 780.000}$
Taglia record	$\boldsymbol{R = 250\ byte}$
Taglia puntatore	$\boldsymbol{P = 4\ byte}$
Capacità blocco	$\boldsymbol{CB = 1024\ byte}$
Numero bucket	$\boldsymbol{B = 2500}$

a. Calcoliamo innanzitutto quanti record sono presenti in ciascun bucket

$$\boldsymbol{RB = \left\lceil \frac{NR}{B} \right\rceil = \left\lceil \frac{780.000}{2500} \right\rceil = \lceil 312 \rceil = 312}$$

Si prende la parte intera superiore perché tutti i record devono essere memorizzati nei bucket.

Vediamo ora **quanti record possono essere memorizzati in un blocco** (ogni blocco contiene anche un puntatore al blocco successivo).

$$\boldsymbol{RBL = \left\lfloor \frac{CB - P}{R} \right\rfloor = \left\lfloor \frac{1020}{250} \right\rfloor = \lfloor 4,08 \rfloor = 4}$$

Si prende la parte intera inferiore perché i record devono essere memorizzati per intero.

Calcoliamo infine quanti blocchi occorrono per ogni bucket

$$\boldsymbol{BB = \left\lceil \frac{RB}{RBL} \right\rceil = \left\lceil \frac{312}{4} \right\rceil = \lceil 78 \rceil = 78}$$

Si prende la parte intera superiore perché i blocchi devono essere allocati per intero
Complessivamente ci occorrono

$$\boldsymbol{BD + BB * B = 10 + 78 * 2500 = 195.010}$$

blocchi.

b. I calcoli per la bucket directory e per il numero di record che possono essere memorizzati in un blocco rimangono validi anche in questo caso. Cambia la distribuzione dei record nei bucket, e quindi il numero di blocchi che occorrono per ogni bucket. Il 30% dei record è distribuito uniformemente su 1000 bucket, il rimanente 70% dei record è distribuito uniformemente su 1500 bucket. Quindi il numero di record che va distribuito uniformemente su 1000 bucket è

$$\boldsymbol{N1000 = \frac{780.000 * 30}{100} = 234.000}$$

Per ogni bucket dei 1000 avremo quindi

$$\boldsymbol{RB1000 = \left\lfloor \frac{N1000}{1000} \right\rfloor = 234}$$

record.

Quindi per ognuno di questi 1000 bucket occorrono

$$\boldsymbol{B1000 = \left\lceil \frac{RB1000}{RBL} \right\rceil = \left\lceil \frac{234}{4} \right\rceil = \lceil 58,5 \rceil = 59}$$

blocchi.

l numero di record che va distribuito uniformemente su *1500* bucket è

$$\boldsymbol{N1500 = N - N1000 = 780.000 - 234.000 = 546.000}$$

record.

Calcoliamo direttamente per differenza i record rimanenti che sono comunque quelli da memorizzare per ogni bucket dei *1500* avremo quindi

$$\boldsymbol{RB1500 = \left\lceil \frac{RB1500}{1500} \right\rceil = 364}$$

record.

Quindi per ognuno di questi *1500* bucket occorrono quindi

$$\boldsymbol{B1500 = \left\lceil \frac{RB1500}{RBL} \right\rceil = \left\lceil \frac{364}{4} \right\rceil = \lceil 91 \rceil = 91}$$

blocchi.

In totale avremo

$$\boldsymbol{B1000 * 1000 + B1500 * 1500 = 59 * 1000 + 91 * 1500 = 195.500}$$

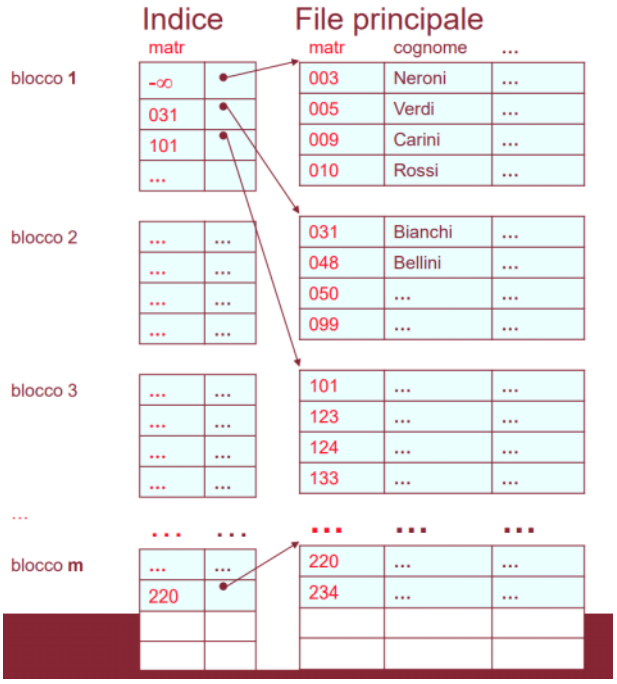
blocchi per i bucket.

Notare che a parità di spazio totale occupato, nella prima parte del file ho un tempo di ricerca medio che è quasi la metà che nella seconda parte ($\left\lceil \frac{59}{2} \right\rceil$ contro $\left\lceil \frac{91}{2} \right\rceil$).

FILE CON INDICE

Nella presentazione di questo tipo di organizzazione, spesso chiamata **ISAM** (Indexed Sequential Access Method), assumeremo che il **valore della chiave identifica univocamente un record del file**. Inoltre, inizialmente, assumeremo che i record non siano puntati.

Inizialmente il file viene ordinato in base al valore (crescente) della chiave e viene memorizzato lasciando in ogni blocco una certa percentuale (ad esempio il 20%) di spazio non utilizzato per l'inserimento di nuovi record .Quindi viene creato il file indice (anche in questo caso viene lasciato spazio libero per l'inserimento di nuovi record) e infine la directory del file indice che generalmente ha dimensioni tali da poter essere mantenuta in memoria principale durante l'utilizzo del file.
Un file indice è un file i cui record consistono di coppie del tipo (v, b), in cui v è un valore della chiave e b è l'indirizzo di un blocco del file principale: il valore della chiave di ogni record nel blocco di indirizzo b è maggiore o uguale a v e il valore della chiave di ogni record che si trova in un blocco che precede quello di indirizzo b è minore di v. Il file indice viene creato nel modo seguente.



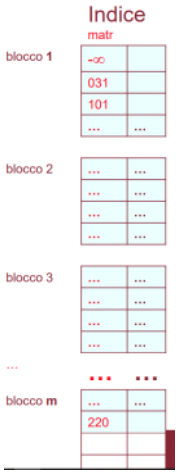
Per ogni blocco del file principale viene creata una coppia costituita dal valore della chiave del primo record nel blocco e dall'indirizzo del blocco; l'unica eccezione è rappresentata dal primo blocco per il quale viene preso anziché il valore della chiave del primo record, un valore (denotato con $-\infty$) che è più piccolo di qualsiasi valore che può essere assunto dalla chiave.

RICERCA

La chiave del file principale è una chiave anche per il file indice; pertanto i record del file indice possono a loro volta essere ordinati in base al valore della chiave. Il fatto che il file indice sia ordinato in base al valore della chiave consente di usare metodi più efficienti della scansione completa per ricercare sul file indice un valore della chiave che ricopra un dato valore v.
Per ricercare un record con valore della chiave k occorre ricercare sul file indice un valore k' della chiave che ricopre k, cioè tale che:

- $k' \leq k$
- se il record con chiave k' non è l'ultimo record del file indice e k'' è il valore della chiave nel record successivo $k < k''$

Esempi:



- Il record con chiave 090 deve trovarsi nel blocco del file principale che contiene 031 in quanto i valori della chiave nei blocchi precedenti < 031 e quelli nei blocchi successivi sono ≥ 101 $(031 \leq 090 < 101)$

- Il record con chiave 234 deve trovarsi nell'ultimo blocco del file principale in quanto i valori della chiave nei blocchi precedenti sono < 220 $(220 \leq 234)$

2 metodi più comuni per effettuare la ricerca su file con indice:

- **Ricerca binaria**

  Si fa un accesso in lettura al blocco $\left(\frac{m}{2}\right) + 1$ e si confronta k con $k1$ (prima chiave del blocco). Se $k = k1$ abbiamo finito. Se $k < k1$ allora si ripete il procedimento sui blocchi da 1 a $\left(\frac{m}{2}\right)$ altrimenti si ripete il procedimento sui blocchi da $\left(\frac{m}{2}\right) + 1$ ad m (il blocco $\left(\frac{m}{2}\right) + 1$ va riconsiderato perché abbiamo controllato solo la prima chiave …)
  Ci si ferma quando lo spazio di ricerca è ridotto ad un unico blocco, quindi dopo $\boldsymbol{log_2 m}$ **accessi**.

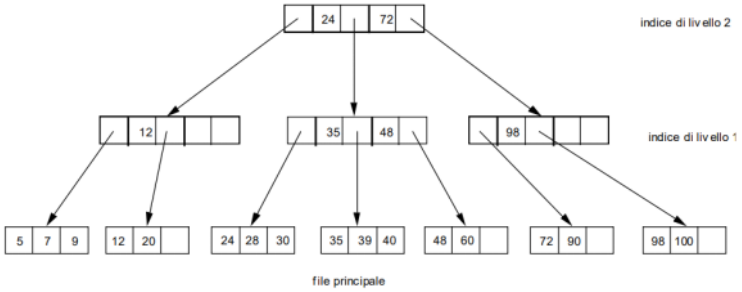
- **Ricerca per interpolazione**

  basata sulla **conoscenza della distribuzione dei valori della chiave**: deve essere disponibile una funzione f che dati tre valori $k1$, $k2$, $k3$ della chiave fornisce un valore che è la frazione dell'intervallo di valori della chiave compresi tra $k2$ e $k3$ in cui deve trovarsi $k1$ cioè la chiave che stiamo cercando (nella ricerca binaria questa frazione è sempre ½).
  Esempio: quando cerchiamo in un elenco telefonico non partiamo sempre da metà.

  $k1$ deve essere confrontato con il valore k della chiave nel primo record del blocco i (del file indice), dove $i = f(k1, k2, k3) * m$; analogamente a quanto accade nella ricerca binaria.
  Se $k1$ è minore di tale valore allora il procedimento deve essere ripetuto sui blocchi $1, 2, \dots, i-1$, mentre se è maggiore il procedimento deve essere ripetuto sui blocchi $i, i+1, \dots, m$, finchè la ricerca si restringe ad un unico blocco.
  La ricerca per interpolazione richiede circa $1 + log_2(log_2 m)$ accessi, che è molto più veloce ma è anche molto difficile conoscere f, inoltre la distribuzione dei dati potrebbe cambiare nel tempo.

B-TREE

Questa organizzazione è una generalizzazione del file con indice. In un B-tree si accede al file attraverso una gerarchia di indici. L'indice a livello più alto nella gerarchia (la radice) è costituito da un unico blocco, può quindi risiedere in memoria principale durante l'utilizzo del file.
Ogni blocco di un file indice è costituito di record contenenti una coppia (v, b) dove v è il valore della chiave del primo record della porzione del file principale che è accessibile attraverso il puntatore b; b può essere un puntatore ad un blocco del file indice a livello immediatamente più basso oppure (nei record del file indice nel più basso livello della gerarchia) ad un blocco del file principale.
Il primo record indice di ogni blocco indice contiene solo un puntatore ad un blocco le cui chiavi sono minori di quelle nel blocco puntato dal secondo record indice.
Ogni blocco di un B-tree (indice o file principale) deve essere pieno almeno per metà (della taglia!) tranne eventualmente la radice (più importante che sia un unico blocco).



Come possiamo vedere nell'esempio qui sopra, il puntatore prima della chiave 24 punta al sottoalbero di chiavi minori di 24; il puntatore tra 24 e 72 punta al sottoalbero di chiavi comprese tra 24 (incluso) e 72 (escluso); il puntatore dopo il 72 punta al sottoalbero di chiavi maggiori di 72.

Se i blocchi sono completamente pieni un inserimento può richiedere una modifica dell'indice ad ogni livello e in ultima ipotesi può far crescere l'altezza dell'albero di un livello.

RICERCA

Durante la ricerca di un record con un dato valore per la chiave si accede agli indici a partire da quello a livello più alto; a mano a mano che si scende nella gerarchia di indici si restringe la porzione (insieme di blocchi) del file principale in cui deve trovarsi il record desiderato, fino a che, nell'ultimo livello (il più basso nella gerarchia) tale porzione è ristretta ad un unico blocco.

Più in dettaglio, per ricercare il record del file principale con un dato valore v per la chiave si procede nel modo seguente. Si parte dall'indice a livello più alto (che è costituito da un unico blocco) e ad ogni passo si esamina un unico blocco. Se il blocco esaminato è un blocco del file principale, tale blocco è quello in cui deve trovarsi il record desiderato; se, invece, è un blocco di un file indice, si cerca in tale blocco un valore della chiave che ricopre v e si segue il puntatore associato.

Per la ricerca sono quindi necessari $h + 1$ accessi dove h è l'altezza dell'albero.
Più sono pieni i blocchi più h è piccolo (e quindi meno costa la ricerca).

MASSIMO VALORE DI h

Siano:

- $\boldsymbol{N}$ numero di record nel file principale
- $\boldsymbol{2e-1}$ numero massimo di record del file principale che possono essere memorizzati in un blocco
- $\boldsymbol{2d-1}$ numero massimo di record del file indice che possono essere memorizzati in un blocco

L'assunzione che il numero di record del file principale e del file indice che possono essere memorizzati in un blocco sia dispari viene fatta esclusivamente per rendere semplici i calcoli.

Poiché i blocchi devono essere pieni almeno per metà:

- ogni blocco del file principale deve contenere almeno e record
- ogni blocco del file indice deve contenere almeno d record

L'**altezza massima** dell'albero denotata con k si ha quando i blocchi sono pieni al minimo, cioè quando

- **ogni blocco del file principale contiene esattamente e record**
- **ogni blocco del file indice contiene esattamente d record**

ESEMPIO 1

Supponiamo di avere un file di 170.000 record. Ogni record occupa 200 byte, di cui 20 per il campo chiave. Ogni blocco contiene 1024 byte. Un puntatore a blocco occupa 4 byte.

Numero record	$\boldsymbol{NR = 170.000}$
Taglia record	$\boldsymbol{R = 200\ byte}$
Dimensione campo chiave	$\boldsymbol{K = 20\ byte}$
Capacità blocco	$\boldsymbol{B = 1024\ byte}$
Taglia puntatore	$\boldsymbol{P = 4\ byte}$

a) Usiamo un B-tree e assumiamo che sia i blocchi indice che i blocchi del file sono **pieni al minimo**: quanti blocchi vengono usati per il livello foglia (file principale) e quanti per l'indice, considerando tutti i livelli non foglia ?
b) Qual è il costo di una ricerca in questo caso?

a. Per quanto riguarda i blocchi di dati nel livello foglia (file principale) indichiamo il numero massimo di record memorizzabili con MRP.

$$\boldsymbol{MRP = \left\lfloor \frac{B}{R} \right\rfloor = \left\lfloor \frac{1024}{200} \right\rfloor = 5}$$

Abbiamo quindi

$$\boldsymbol{MRP = 2e - 1 \quad \Rightarrow \quad 2e - 1 = 5 \quad \Rightarrow \quad e = 3}$$

Quindi un blocco dati non conterrà mai meno di $e = 3$ record, o più di $2e - 1 = 5$ record. L'esercizio indica che i blocchi sono pieni al minimo, quindi ogni record contiene e record. A questo punto possiamo calcolare il numero di blocchi nel livello foglia (file principale)

$$\boldsymbol{NBF = \left\lceil \frac{NR}{e} \right\rceil = \left\lceil \frac{170.000}{3} \right\rceil = 56.667}$$

A questo punto dobbiamo calcolare la capacità dei blocchi indice. Nei livelli indice del B-tree, ogni record, eccetto il primo, contiene una coppia (chiave, puntatore); il primo record invece contiene solo un puntatore (relativo al blocco dati che contiene i record con chiave minore della chiave contenuta nel record indice successivo). Indichiamo il numero massimo di record del file indice memorizzabili in un blocco con MRI.

$$(MRI - 1) * K + MRI * P \leq B$$
$$(MRI - 1) * 20 + MRI * 4 \leq 1024$$
$$MRI * 20 - 20 + MRI * 4 \leq 1024$$
$$MRI * 24 \leq 1044$$
$$\boldsymbol{MRI \leq \frac{1044}{24} = \lfloor 43.5 \rfloor = 43}$$

Possiamo anche calcolarlo come

$$\boldsymbol{MRI = \left\lfloor \frac{B - P}{K + P} \right\rfloor + 1 = \left\lfloor \frac{1020}{24} \right\rfloor + 1 = \lfloor 42,5 \rfloor + 1 = 43}$$

NB!
In questo secondo caso sottraiamo P per preservare lo spazio del primo puntatore, contiamo quante coppie chiave/puntatore entrano nei restanti byte, poi sommiamo 1 in quanto consideriamo il puntatore come un record (la cui dimensione va comunque considerata come quella di un puntatore, altrimenti il numero massimo di byte occupati sforerebbe la capacità del blocco).

Calcoliamo ora l'occupazione minima dei blocchi nel file indice

$$\boldsymbol{d = \left\lceil \frac{\frac{B}{2} - P}{K + P} \right\rceil + 1 = \left\lceil \frac{508}{24} \right\rceil + 1 = \lceil\, 21,16 \,\rceil + 1 = 43}$$

prendiamo la parte intera superiore in quanto i blocchi, per definizione, devono essere **pieni almeno per metà**, se considerassimo la parte inferiore staremo al disotto della metà.

Andiamo adesso a calcolare il numero complessivo di blocchi nel file indice: dobbiamo contare tutti i blocchi di ogni singolo livello.

$$\boldsymbol{NB1 = \left\lceil \frac{NBF}{d} \right\rceil = \left\lceil \frac{56.667}{23} \right\rceil = \lceil\, 2.463,78 \,\rceil = 2.464}$$
$$\boldsymbol{NB2 = \left\lceil \frac{NB1}{d} \right\rceil = \left\lceil \frac{2.464}{23} \right\rceil = \lceil\, 107.13 \,\rceil = 108}$$
$$\boldsymbol{NB3 = \left\lceil \frac{NB2}{d} \right\rceil = \left\lceil \frac{108}{23} \right\rceil = 5}$$
$$\boldsymbol{NB4 = \left\lceil \frac{NB3}{d} \right\rceil = \left\lceil \frac{5}{23} \right\rceil = 1}$$

Il totale di tutti i blocchi del file indice è dato dalla somma dei blocchi presenti in tutti i livelli, avremo quindi

$$\boldsymbol{NBI = \sum_{i=1}^{4} NBi = 2464 + 108 + 5 + 1 = 2578}$$

b. Il costo di una ricerca sarà quindi di 5 accessi: un blocco per ognuno dei 4 livelli + 1 blocco del file principale.

ESEMPIO 2

Considerati gli stessi dati dell'esercizio precedente:

a) Usiamo un B-tree e assumiamo che sia i blocchi indice che i blocchi del file sono **pieni al massimo**: quanti blocchi vengono usati per il livello foglia (file principale) e quanti per l'indice, considerando tutti i livelli non foglia ?
b) Qual è il costo di una ricerca in questo caso?

a. $MRP = massimo\ numero\ di\ record\ nei\ blocchi\ del\ file\ principale$
$MRI = massimo\ numero\ di\ record\ (chiave, puntatore)\ nei\ blocchi\ del\ file\ indice$
$NBF = numero\ di\ blocchi\ nel\ livello\ foglia\ (file\ principale)$
$NBI = numero\ di\ blocchi\ nel\ file\ indice$

$$MRP = \left\lfloor \frac{B}{R} \right\rfloor = \left\lfloor \frac{1024}{200} \right\rfloor = 5$$
$$NBF = \left\lceil \frac{NR}{MRP} \right\rceil = \left\lceil \frac{170.000}{5} \right\rceil = 34.000$$
$$MRI = \left\lfloor \frac{B - P}{K + P} \right\rfloor + 1 = \left\lfloor \frac{1020}{24} \right\rfloor + 1 = 43$$
$$NB1 = \left\lceil \frac{NBF}{MRI} \right\rceil = \left\lceil \frac{34.000}{43} \right\rceil = 791$$
$$NB2 = \left\lceil \frac{NB1}{MRI} \right\rceil = \left\lceil \frac{791}{43} \right\rceil = 19$$
$$NB3 = \left\lceil \frac{NB2}{MRI} \right\rceil = \left\lceil \frac{19}{43} \right\rceil = 1$$
$$NBI = NB1 + NB2 + NB3 = 791 + 19 + 1 = 811$$

b. $CR = costo\ di\ ricerca\ in\ termini\ di\ accesso\ ai\ blocchi$
$CR = 3\ (livelli\ file\ indice) + 1\ (livello\ foglia) = 4$

6 - Appendice

Definizione di F^+

Se $\boldsymbol{F}$ **è un insieme di dipendenze funzionali** su R ed $\boldsymbol{r}$ **è un'istanza** di R che **soddisfa tutte le dipendenze in** $\boldsymbol{F}$, diciamo che r è un'**istanza legale di** $\boldsymbol{R}$.
Dato uno schema di relazione R e un insieme F di dipendenze funzionali su R la **chiusura di** $\boldsymbol{F}$ è l'**insieme delle dipendenze funzionali che sono soddisfatte da ogni istanza legale di** $\boldsymbol{R}$.

Definizione di F^A

Denotiamo con $\boldsymbol{F^A}$ **l'insieme di dipendenze funzionali**, ottenuto mediante l'**applicazione ricorsiva dei seguenti assiomi di Armstrong**:

- Se $Y \subseteq X \subseteq R \Rightarrow X \to Y \in F^A$ assioma della **riflessività**
- Se $X \to Y \in F^A \Rightarrow XZ \to YZ \in F^A \ \forall\ Z \subseteq R$ assioma dell'**aumento**
- Se $X \to Y \in F^A$ e $Y \to Z \in F^A \Rightarrow X \to Z \in F^A$ assioma della **transitività**

Di seguito tre regole conseguenza degli assiomi che consentono di derivare da dipendenze funzionali in F^A altre dipendenze funzionali in F^A.

a. Se $X \to Y \in F^A$ e $X \to Z \in F^A \Rightarrow X \to YZ \in F^A$ regola dell'**unione**
b. Se $X \to Y \in F^A$ e $Z \subseteq Y \Rightarrow X \to Z \in F^A$ regola della **decomposizione**
c. Se $X \to Y \in F^A$ e $WY \to Z \in F^A \Rightarrow WX \to Z \in F^A$ regola della **pseudotransitività**

Dimostrazione

a) Se $X \to Y \in F^A$, per l'assioma dell'aumento si ha $X \to XY \in F^A$.
Analogamente, se $X \to Z \in F^A$, per l'assioma dell'aumento si ha $XY \to YZ \in F^A$.
Quindi, poiché $X \to XY \in F^A$ e $XY \to YZ \in F^A$, per l'assioma della transitività si ha $X \to YZ \in F^A$.

b) Se $Z \subseteq Y$ allora, per l'assioma della riflessività, si ha $Y \to Z \in F^A$.
Quindi, poiché $X \to Y \in F^A$ e $Y \to Z \in F^A$. Per l'assioma della transitività si ha $X \to Z \in F^A$.

c) Se $X \to Y \in F^A$, per l'assioma dell'aumento si ha $WX \to WY \in F^A$. Quindi, poiché $WX \to WY \in F^A$ e $WY \to Z \in F^A$, per l'assioma della transitività si ha $WX \to Z \in F^A$.

Teorema: chiusura di un insieme di attributi

Siano $\boldsymbol{R}$ **uno schema di relazione**, $\boldsymbol{F}$ **un insieme di dipendenze funzionali** su R e $\boldsymbol{X}$ **un sottoinsieme di** $\boldsymbol{R}$.
La chiusura di X rispetto ad F, denotata con X_F^+ è definito nel modo seguente:

$$X_F^+ = \{A : X \to A \in F^A\}$$

In pratica **fanno parte della chiusura** di un insieme di attributi X, tutti gli **attributi che sono determinati funzionalmente da** $\boldsymbol{X}$ **eventualmente applicando gli assiomi di Armstrong** $(X \subseteq X_F^+)$.

Lemma

Siano R uno schema di relazione ed F un insieme di dipendenze funzionali su R. Si ha che:

$$\boldsymbol{X \to Y \in F^A \ \Leftrightarrow\ Y \subseteq X^+}$$

Dimostrazione

Sia $Y = A_1, A_2, \dots, A_n$:

Parte se

Poiché $Y \subseteq X^+$, $\forall\, i = 1, \dots, n$ si ha che $X \to A_i \in F^A$.
Pertanto, per la **regola dell'unione**, $X \to Y \in F^A$.

- **Parte solo se**

  Poiché $X \to Y \in F^A$, per la **regola della decomposizione** si ha che,
  $\forall\, i = 1, \dots, n,\ X \to A_i \in F^A$, cioè $A_i \in X^+$, quindi $Y \subseteq X^+$.

Teorema: $F^+ = F^A$

Siano R uno schema di relazione ed F un insieme di dipendenze funzionali su R, si ha che

$$F^+ = F^A .$$

Dimostrazione (doppia inclusione)

$F^+ \supseteq F^A$

Sia $X \to Y$ una dipendenza funzionale in F^A.
Dimostriamo che $X \to Y \in F^+$ per induzione sul numero i di applicazioni di uno degli assiomi di Armstrong.

Passo base $i = 0$

In tal caso non è stato applicato ancora nessun assioma, quindi abbiamo che $X \to Y \in F$ e banalmente $X \to Y \in F^+$.

Ipotesi induttiva $i = n - 1$

Ogni dipendenza funzionale ottenuta a partire da F applicando gli assiomi di Armstrong un numero di volte minore o uguale a $n-1$ è in F^+ .

Passo induttivo $i = n$

Dimostriamo l'ipotesi induttiva per $i = n$.

Si possono presentare tre casi:

1) $X \to Y$ è stata ottenuta mediante l'assioma della **riflessività**, in tal caso $Y \subseteq X$.
   Sia r un'istanza di R e siano t_1 e t_2 due tuple di r tali che $t_1[X] = t_2[X]$; banalmente si ha $t_1[Y] = t_2[Y]$.
2) $X \to Y$ è stata ottenuta applicando l'assioma dell'**aumento** ad una dipendenza funzionale $V \to W$ in F^A, ottenuta a sua volta applicando ricorsivamente gli assiomi di Armstrong un numero di volte minore o uguale a $n-1$ (quindi per l'ipotesi induttiva $V \to W \in F^+$); sarà quindi $X = VZ$ e $Y = WZ$, per qualche $Z \subseteq R$.
   Sia r un'istanza legale di R e siano t_1 e t_2 due tuple di r tali che $t_1[X] = t_2[X]$, banalmente si ha che $t_1[V] = t_2[V]$ e $t_1[Z] = t_2[Z]$. Per l'ipotesi induttiva da $t_1[V] = t_2[V]$ segue $t_1[W] = t_2[W]$; da $t_1[W] = t_2[W]$ e $t_1[Z] = t_2[Z]$ segue $t_1[Y] = t_2[Y]$.
3) $X \to Y$ è stata ottenuta applicando l'assioma della **transitività** a due dipendenze funzionali $X \to Z$ e $Z \to Y$ in F^A, ottenute a loro volta applicando ricorsivamente gli assiomi di Armstrong un numero di volte minore o uguale a $n-1$ (quindi per l'ipotesi induttiva $X \to Z \in F^+$ e $Z \to Y \in F^+$).
   Sia r un'istanza legale di R e siano t_1 e t_2 due tuple di r tali che $t_1[X] = t_2[X]$. Per l'ipotesi induttiva da $t_1[X] = t_2[X]$ segue $t_1[Z] = t_2[Z]$; da $t_1[Z] = t_2[Z]$, segue $t_1[Y] = t_2[Y]$.

$F^+ \subseteq F^A$

Supponiamo per assurdo che esista una dipendenza funzionale $X \to Y \in F^+$ tale che $X \to Y \notin F^A$.
Useremo una particolare istanza legale di R per dimostrare che questa supposizione porta ad una contraddizione.

Consideriamo la seguente istanza r di R:

	X^+				$R\text{-}X^+$			
r	1	1	…	1	1	1	…	1
	1	1	…	1	0	0	…	0

L'istanza ha solo due tuple, uguali sugli attributi in X^+ e diverse in tutti gli altri $(R - X^+)$.
Utilizzando questa istanza dimostreremo che se $X \to Y \in F^+$ NON può succedere che $X \to Y \notin F^A$.
Dimostriamo quindi le due proposizioni.

r è un'istanza legale di R

Sia $V \to W$ una dipendenza funzionale in F e supponiamo per assurdo che non sia soddisfatta da r (r non è legale). In tal caso le due tuple di r devono avere gli stessi valori per V e differenti valori per W; ciò implica che $V \subset X^+$ e $W \cap (R - X^+) \neq \emptyset$. Poiché $V \subset X^+$, per il Lemma, si ha che $X \to V \in F^A$; pertanto, per l'assioma della transitività ($X \to V$ e $V \to W$), $X \to W \in F^A$ e quindi, di nuovo per il Lemma, $W \subseteq X^+$ (che contraddice $W \cap (R - X^+) \neq \emptyset$). Quindi $V \to W$ è soddisfatta.
Siamo arrivati ad una contraddizione, poiché abbiamo considerato una generica dipendenza in F, r le soddisfa tutte, quindi è legale.

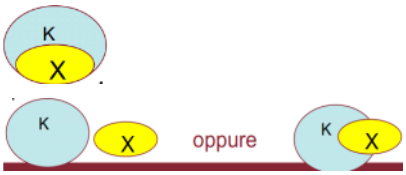
$X \to Y \in F^+ \Rightarrow X \to Y \in F^A$

Supponiamo per assurdo che $X \to Y \in F^+$ e $X \to Y \notin F^A$. Abbiamo mostrato che la nostra r è un'istanza legale. Poiché le dipendenze in F^+ sono soddisfatte da ogni istanza legale, allora r soddisfa $X \to Y$.
Poiché $X \subseteq X^+$ (per l'assioma della riflessività e il Lemma) le due tuple di r coincidono sugli attributi X e quindi, poiché r soddisfa $X \to \mathrm{Y}$, devono coincidere anche sugli attributi di Y.
Questo implica che $Y \subseteq X^+$ e, per il Lemma, che $X \to \mathrm{Y} \in F^A$, arrivando ad una contraddizione.

Terza forma normale

Dati uno schema di relazione R e un insieme di dipendenze funzionali F su R, diciamo che:

- Un attributo A di R è **primo** se **appartiene** ad una chiave di R
- Un sottoinsieme X di R è una **superchiave** se **contiene** una chiave di R
- $X \to A \in F^+$ è una **dipendenza parziale** su R se A non è primo ed X è contenuto propriamente in una chiave di R $(X \subset R)$
- $X \to A \in F^+$ è una **dipendenza transitiva** su R se A non è primo e $\forall\ K$ di R si ha che $X \not\subset K$ e $K - X \neq \emptyset$



R è in 3NF se $\forall\ X \to A \in F^+ : A \notin X$ si ha che:

- **A è primo**

  oppure (inclusivo)

- **X è una superchiave**

R è in 3NF se e solo se F non contiene dipendenze parziali o transitive!

Dimostrazione

Solo se

Lo schema R è in 3NF, quindi $\forall\ X \to A \in F^+ : A \notin X$

- A appartiene ad una chiave (è primo) oppure
- X contiene una chiave (è una superchiave)

Se A è parte di una chiave (primo), viene a mancare la prima condizione per avere una dipendenza parziale o transitiva.

Se A non è primo (non fa parte di nessuna chiave), allora X è superchiave. In quanto tale può contenere una chiave, ma NON essere contenuto propriamente, quindi la dipendenza non può essere parziale.
Inoltre, essendo superchiave non può verificarsi che per ogni chiave K di R, X non è contenuto propriamente in K e $K - X \neq \emptyset$ (ne contiene almeno una completamente). Quindi la dipendenza non può essere transitiva.

Se

Non esistono né dipendenze parziali né dipendenze transitive in R.

Supponiamo per assurdo che R non sia in 3NF; in tal caso esiste una dipendenza funzionale $X \to A \in F^+ : A\ non\ è\ primo\ e\ X\ non\ è\ una\ superchiave$. Poiché X non è una superchiave sono possibili due casi (mutuamente esclusivi) :

per ogni chiave K di R, X non è contenuto propriamente in K e $K - X \neq \emptyset$; in tal caso $X \to A$ è una dipendenza transitiva su R (contraddizione),

oppure

esiste una chiave K di R tale che $X \subset K$; in tal caso $X \to A$ è una dipendenza parziale su R (contraddizione).

Teorema del calcolo di X^+

L'Algoritmo Calcolo di X^+ calcola correttamente la chiusura di un insieme di attributi X rispetto ad un insieme F di dipendenze funzionali.

Dimostrazione

Indichiamo con Z^0 il valore iniziale di Z ($Z^0 = X$) e con Z^i ed S^i, $i \geq 1$, i valori di Z ed S dopo l'$i - esima$ esecuzione del corpo del ciclo; è facile vedere che $Z^i \subseteq Z^{i+1}$, per ogni i.

Ricordiamo: In Z^i ci sono gli attributi aggiunti a Z fino alla $i - esima$ iterazione.
Alla fine di ogni iterazione aggiungiamo sempre qualcosa a Z.

Sia j tale che $S^j \subseteq Z^j$ (cioè, Z^j è il valore di Z quando l'algoritmo termina), proveremo che:

$$\boldsymbol{A \in Z^j}\ \textbf{se e solo se}\ \boldsymbol{A \in X^+}$$

Solo se

Mostreremo per induzione su i che $Z^i \subseteq X^+\ \forall\ i$ (in particolare $Z^j \subseteq X^+$).

Passo base: $i = 0$

Poiché $Z^0 = X$ e $X \subseteq X^+$, si ha $Z^0 \subseteq X^+$.

Ipotesi induttiva: $i > 0$

$Z^{i-1} \subseteq X^+$

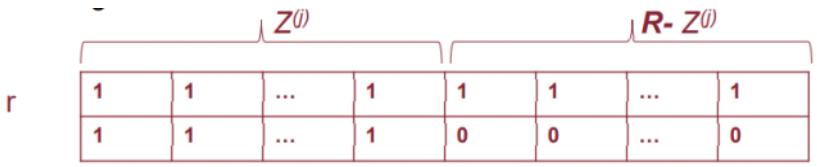
Passo induttivo:

Sia A un attributo in $Z^i - Z^{i-1}$ (aggiunto all'$i - esima$ iterazione in quanto non era in Z^{i-1}), deve esistere una dipendenza $Y \to V \in F$ tale che $Y \subseteq Z^{i-1}$ e $A \in V$.

Poiché $Y \subseteq Z^{i-1}$, per l'ipotesi induttiva si ha che $Y \subseteq X^+$, pertanto, per il Lemma, $X \to Y \in F^A$. Poiché $X \to Y \in F^A$ e $Y \to V \in F$, per l'assioma della transitività si ha $X \to V \in F^A$ e quindi, per il Lemma , $V \subseteq X^+$. Pertanto, $\forall\ A \in Z^i - Z^{i-1}$ si ha $A \in X^+$ e, per l'ipotesi induttiva, che $Z^i \subseteq X^+$.

Se (NON CHIEDE)

Sia A un attributo in X^+ e sia j tale che $S^j \subseteq Z^j$ (Z^j è il valore di Z quando l'algoritmo si termina). Mostreremo che $A \in Z^j$. Poiché $A \in X^+$, si ha $X \to A \in F^+$ (per il Teorema), pertanto $X \to A$ deve essere soddisfatta da ogni istanza legale di R.
Si consideri la seguente istanza r di R:



Mostriamo prima che r è un'istanza legale di R.
Se, per assurdo, esistesse in F una dipendenza funzionale $V \to W$ non soddisfatta da r, si dovrebbe avere $V \subseteq Z^j$ (i valori delle due tuple sono uguali SOLO in quel sottoinsieme di R), e serve che siano uguali per poter dire che la dipendenza NON E' soddisfatta) e
W $\cap (R - Z^j) \neq \emptyset$, (i valori delle due tuple sono diversi SOLO in quel sottoinsieme di R) ma, in tal caso, si avrebbe $S^j \not\subset Z^j$ (contraddizione).

Abbiamo assunto che j è il valore per cui $S^j \subseteq Z^j$ (all'iterazione j non abbiamo aggiunto NIENTE di nuovo, e quindi da lì in poi non potremo farlo neppure continuando). Se fosse $V \subseteq Z^j$ e $W \cap (R - Z^j) \neq \emptyset$ avrei qualche elemento di W non è ancora in Z^j; applicando l'algoritmo alla iterazione $j+1$ potrei ancora raccogliere in S questi NUOVI elementi tramite $V \to W$ e poi inserirli in Z^{j+1}. L'algoritmo però si ferma solo quando non è più possibile inserire nuovi elementi in Z, che significa che da quel punto in poi continueremmo ad ottenere sempre lo stesso insieme S che è stato già aggiunto a Z, e quindi siamo ad una contraddizione

Poiché r è un'istanza legale di R deve soddisfare $X \to A$; (che è in F^+) (se esistono due tuple uguali su X devono essere uguali anche su A). Esistono due tuple uguali su X? CERTO!
$X = Z^0 \subseteq Z^j$, e le due tuple sono uguali su TUTTI gli attributi in Z^j; ma allora le due tuple DEVONO essere uguali anche su A, che, allora, deve essere in Z^j .

Lemma su chiusure

Siano F e G due insiemi di dipendenze funzionali. F e G sono equivalenti ($F \equiv G$) se $F^+ = G^+$. (F e G non contengono le stesse dipendenze, ma le loro chiusure si).
Verificare l'equivalenza di due insiemi F e G di dipendenze funzionali richiede dunque che venga verificata l'uguaglianza di F^+ e G^+, cioè che $F^+ \subseteq G^+$ e che $F^+ \supseteq G^+$.

Lemma

Siano F e G due insiemi di dipendenze funzionali. Se $F \subseteq G^+ \Rightarrow F^+ \subseteq G^+$.

Sia $f \in F^+ - F$, siccome f è derivabile da F mediante gli assiomi di Armstrong (per il teorema $F^+ = F^A$), e ogni dipendenza funzionale in F è derivabile da G mediante gli assiomi di Armstrong (per l'ipotesi F si trova in G^+), f è derivabile da G mediante gli assiomi di Armstrong.

Abbiamo quindi che

$$G \overset{A}{\to} F \overset{A}{\to} F^+$$

Con $\overset{A}{\to}$ denotiamo la derivazione tramite assiomi di Armstrong.

Definizione: Decomposizione

Sia R uno schema di relazione. Una decomposizione di R è una famiglia $\rho = \{R_1, R_2, \dots, R_k\}$ di sottoinsiemi di R che ricopre R ($\bigcup_{i=1}^{k} R_i = R$), i sottoinsiemi possono avere intersezione non vuota.

Decomposizione che preserva F

Dati uno schema di relazione R, un insieme di dipendenze funzionali F su R, una decomposizione $\rho = \{R_1, R_2, \dots, R_k\}$ di R: diciamo che ρ preserva F se

$$F \equiv \bigcup_{i=1}^{k} \pi_{R_i}(F) \text{ dove } \pi_{R_i}(F) = \{X \to Y : X \to Y \in F^+ \wedge XY \subseteq R_i\}.$$

NB:

- $\bigcup_{i=1}^{k} \pi_{R_i}(F)$ è un insieme di dipendenze funzionali
- Ogni $\pi_{R_i}(F)$ è un insieme di dipendenze funzionali dato dalla proiezione dell'insieme di dipendenze funzionali F sul sottoschema R_i

Supponiamo di avere già una decomposizione e voler verificare che essa preservi le dipendenze funzionali. Bisogna verificare l'equivalenza tra F e $G = \bigcup_{i=1}^{k} \pi_{R_i}(F)$ tramite la doppia inclusione delle loro chiusure.

$$F^+ \subseteq G^+ \text{ e } F^+ \supseteq G^+.$$

Per come G è stato definito **in questo caso**, sarà sicuramente $\boldsymbol{F^+ \supseteq G}$ (ogni proiezione di F, che viene inclusa per definizione in G, è un sottoinsieme di F^+), per il **lemma delle chiusure** abbiamo che $\boldsymbol{F^+ \supseteq G^+}$.

È quindi sufficiente verificare che $F \subseteq G^+$ (che implica $F^+ \subseteq G^+$). Tale verifica viene fatta tramite il relativo algoritmo di **verifica del contenimento di** F in G^+.

Teorema: calcolo di X_G^+ a partire da $\boldsymbol{F}$

Sia R uno schema di relazione, F un insieme di dipendenze funzionali su R, $\rho = \{R_1, R_2, \dots, R_k\}$ una decomposizione di R e X un sottoinsieme di R. L'Algoritmo dato calcola correttamente X_G^+, dove

$$\mathrm{G} = \cup_{i=1}^{k} \pi_{R_i}(F).$$

Dimostrazione

Indichiamo con Z^0 il valore iniziale di Z ($Z^0 = X$) e con Z^i, $i \geq 1$, il valore di Z dopo l'i-esima esecuzione dell'assegnazione $Z = Z \cup S$; è facile vedere che $Z^i \subseteq Z^{i+1}$, per ogni i. Sia Z^f il valore di Z quando l'algoritmo termina; proveremo che:

$$\boldsymbol{A \in Z^f \Leftrightarrow A \in X_G^+}.$$

Parte solo se

Mostreremo per induzione su i che $Z^i \in X_G^+ \ \forall\, i$ (in particolare per $i = f$).

Passo base: $i = 0$.

Poiché $Z^0 = X$ e $X \subseteq X^+$, si ha $Z^0 \subseteq X_G^+$.

Ipotesi induttiva: $i > 0$

$\mathrm{Z}^{\mathrm{i}-1} \subseteq X_G^+$.

Induzione: $i > 0$

Sia A un attributo in $Z^i - Z^{i-1}$ in tal caso deve esistere un indice j (della decomposizione) tale che $A \in \left(Z^{i-1} \cap R_j\right)_F^+ \cap R_j$. Poiché $A \in \left(Z^{i-1} \cap R_j\right)_F^+$ si ha $\left(Z^{i-1} \cap R_j\right) \to A \in F^+$ (per il lemma sulla chiusura di un insieme di attributi e il Teorema $F^+ = F^A$).

Poiché $\left(Z^{i-1} \cap R_j\right) \to A \in F^+$, $A \in R_j$ e $Z^{i-1} \cap R_j \subseteq R_j$ si ha, per definizione di G, che $\left(\mathrm{Z}^{\mathrm{i}-1} \cap R_j\right) \to A \in G$.

Per l'ipotesi induttiva si ha $X \to Z^{i-1} \in G^+$, per la regola di decomposizione si ha anche che $\mathrm{X} \to \left(Z^{i-1} \cap R_j\right) \in G^+$ e quindi, per l'assioma della transitività, che $X \to A \in G^+$, cioè $A \in X_G^+$.

Quindi $Z^i \subseteq X_G^+$.

Parte se (NON CHIEDE)

Mostreremo che $X_G^+ \subseteq Z^f$.

Presi comunque due insiemi di attributi X e Y e un insieme di dipendenze funzionali F si ha (per la definizione di chiusura di un insieme di attributi): se $X \subseteq Y$ allora $X_F^+ \subseteq Y^+$. Poichè $X = Z^0 \subseteq Z^f$, segue che $X_G^+ \subseteq \left(Z^f\right)_G^+$. Mostreremo che $Z^f = \left(Z^f\right)_G^+$ da cui segue $X_G^+ \subseteq Z^f$.

Supponiamo per assurdo che $Z^f \neq \left(Z^f\right)_G^+$. Consideriamo l'Algoritmo 1 che, per evitare ambiguità, consideriamo la variabile W invece della variabile Z e la variabile U invece della variabile S.

Se eseguiamo tale algoritmo fornendo in input l'insieme di attributi Z^f e l'insieme di dipendenze funzionali G, al termine la variabile W conterrà $\left(Z^f\right)_G^+$. Se, come abbiamo supposto per assurdo, $Z^f \neq \left(Z^f\right)_G^+$, deve esistere un attributo B che appartiene a U^0 e non appartiene a $W^0 = Z^f$ (altrimenti si avrebbe $Z^f = \left(Z^f\right)_G^+$. D'altra parte si ha: $U^0 = \{\, \mathrm{A} : \mathrm{Y} \to \mathrm{V} \in \mathrm{G} \wedge \mathrm{A} \in \mathrm{V} \wedge \mathrm{Y} \subseteq W^0 \,\}$. Quindi, per la definizione di G, deve esistere j tale che: $B \in \{\, \mathrm{A} : \mathrm{Y} \to \mathrm{V} \in \mathrm{F}^+ \wedge \mathrm{A} \in \mathrm{V} \wedge \mathrm{Y} \subseteq W^0 \wedge \mathrm{YW} \subseteq \mathrm{R_j} \}$. Da $\mathrm{Y} \subseteq W^0 \wedge \mathrm{YW} \subseteq \mathrm{R_j}$ segue che: $\mathrm{Y} \subseteq W^0 \cap \mathrm{R_j} = Z^f \cap \mathrm{R_j}$.

Inoltre dal fatto che $\mathrm{Y} \to \mathrm{V} \in \mathrm{F}^+$ segue, per il Lemma delle chiusure, che $V \subseteq Y_F^+$, quindi si ha che $V \subseteq \left(Z^f \cap R_j\right)_F^+$. Infine poiché $\mathrm{YV} \subseteq \mathrm{R_j}$ si ha: $V \subseteq \left(Z^f \cap R_j\right)_F^+ \cap R_j$.

Poiché $B \in V$ si ha che $B \in \left(Z^f \cap R_j\right)_F^+ \cap R_j$ e quindi $B \in S^{(f)}$. Poiché B non appartiene a Z^f (per l'ipotesi per assurdo), Z^f non può essere il valore finale di Z (contraddizione).

Definizione: decomposizione con join senza perdita

Se si decompone uno schema di relazione R, si vuole che la decomposizione $\{R_1, R_2, \dots, R_k\}$ ottenuta sia tale che ogni istanza legale r di R sia ricostruibile mediante un join naturale ($\bowtie$) da un'istanza legale $\{r_1, r, \dots, r_k\}$ dello schema decomposto $\{R_1, R_2, \dots, R_k\}$.

Poiché per ricostruire una tupla t di r è necessario che $t[R_i] \in r_i \ \forall \ i = 1, 2, \dots, k$, si deve avere

$$\mathbf{r_i} = \boldsymbol{\pi_{R_i}(r)} \ \forall \ \boldsymbol{i = 1, 2, \dots, k}$$

Definizione

Sia R uno schema di relazione. Una decomposizione $\rho = \{R_1, R_2, \dots, R_k\}$ di R ha un join senza perdita se per ogni istanza legale di r si ha $\mathbf{r} = \boldsymbol{\pi_{R_1}(r) \bowtie \pi_{R_2}(r) \bowtie \dots \bowtie \pi_{R_k}(r)}$.

Unendo quindi tutte le istanze delle proiezioni deve tornarci esattamente l'istanza iniziale.
Prima di procedere introduciamo un nuovo operatore, il **join delle proiezioni**: $\boldsymbol{m_\rho}$, indicato come

$$\boldsymbol{m_\rho(\mathbf{r}) = \pi_{R_1}(r) \bowtie \pi_{R_2}(r) \bowtie \dots \bowtie \pi_{R_k}(r)} .$$

Per ogni istanza legale r di R si ha:

i. $\boldsymbol{r \subseteq m_\rho(\mathbf{r})}$

ii. $\boldsymbol{\pi_{R_i}\left(m_\rho(\mathbf{r})\right) = \pi_{R_i}(r)}$

iii. $\boldsymbol{m_\rho\left(m_\rho(\mathbf{r})\right) = m_\rho(\mathbf{r})}$

Teorema: decomposizione con join senza perdita

Sia R uno schema di relazione e $\rho = \{R_1, R_2, \dots, R_k\}$ una decomposizione di R. Per ogni istanza legale r di R, indicato con $m_\rho(\mathrm{r}) = \pi_{R_1}(r) \bowtie \pi_{R_2}(r) \bowtie \dots \bowtie \pi_{R_k}(r)$, si ha:

a. $\boldsymbol{r \subseteq m_\rho(\mathbf{r})}$

b. $\boldsymbol{\pi_{R_i}\left(m_\rho(\mathbf{r})\right) = \pi_{R_i}(r)}$

c. $\boldsymbol{m_\rho\left(m_\rho(\mathbf{r})\right) = m_\rho(\mathbf{r})}$

Dimostrazione

Prova di a. $\boldsymbol{r \subseteq m_\rho(\mathbf{r})}$

Sia t una tupla di r. Per ogni i, $i = 1, \dots, k$, $t[R_i] \in \pi_{R_i}(r)$ e quindi $t \in m_\rho(\mathrm{r})$

Consideriamo
$R = ABC$
$F = \{A \to B, C \to B\}$
$\rho = \{AB, BC\}$

R

A	B	C
a1	b1	c1
a2	b1	c1
a3	b1	c2

$\pi_{R1}(r)$

A	B
a1	b1
a2	b1
a3	b1

$\pi_{R2}(r)$

B	C
b1	c1
b1	c2

Se però applicassimo il join naturale alla decomposizione effettuata otterremmo il seguente risultato

$$\boldsymbol{m_\rho(\mathbf{r}) = \pi_{R_1}(r) \bowtie \pi_{R_2}(r)}$$

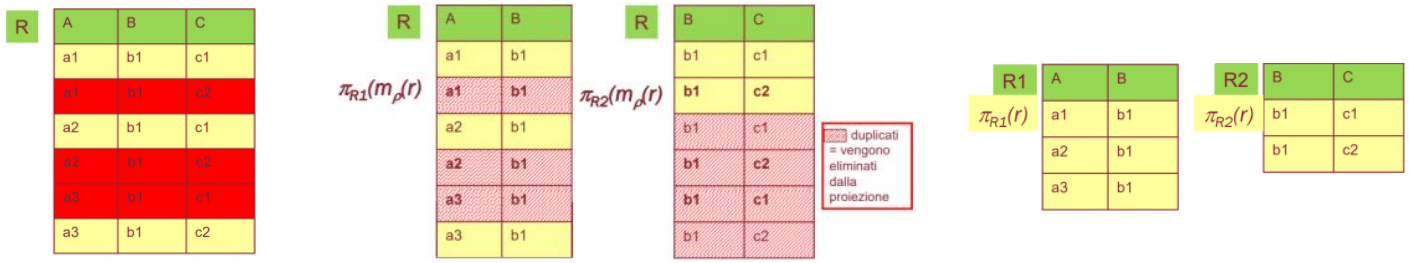
R

A	B	C
a1	b1	c1
a1	b1	c2
a2	b1	c1
a2	b1	c2
a3	b1	c1
a3	b1	c2

Occorre garantire che il join delle istanze della decomposizione non riveli perdita di informazione.

Prova di b. $\boldsymbol{\pi_{R_i}\left(m_\rho(\mathrm{r})\right) = \pi_{R_i}(r)}$

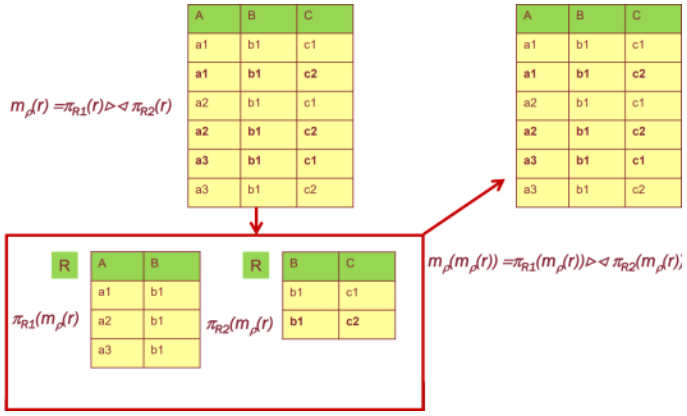
Per a) si ha $r \subseteq m_\rho(\mathrm{r})$ e, quindi, $\pi_{R_i}(r) \subseteq \pi_{R_i}(m_\rho(\mathrm{r}))$. E' sufficiente, pertanto, mostrare che $\pi_{R_i}(r) \supseteq \pi_{R_i}(m_\rho(\mathrm{r}))$. Banalmente, per ogni tupla $t \in m_\rho(\mathrm{r})$ e per ogni i, $i = 1, \ldots, k$, deve esistere una tupla $t' \in r$ tale che $t[R_i] = t'[R_i]$. Se tale tupla non esistesse, non troveremmo i valori di t su $\mathrm{R_i}$ $(t[R_i])$ nella proiezione $\pi_{R_i}(r)$ e di conseguenza non li avremmo nel join naturale.



Prova di c. $\boldsymbol{m_\rho\left(m_\rho(\mathrm{r})\right) = m_\rho(\mathrm{r})}$

Per b) abbiamo che $\boldsymbol{\pi_{R_i}\left(m_\rho(\mathrm{r})\right) = \pi_{R_i}(r)}$, pertanto, applicando la definizione di $\boldsymbol{m_\rho}$ avremo

$$\boldsymbol{m_\rho\left(m_\rho(\mathrm{r})\right) = \pi_{R_1}\left(m_\rho(\mathrm{r})\right) \bowtie \pi_{R_2}\left(m_\rho(\mathrm{r})\right) \bowtie \ldots \bowtie \pi_{R_k}\left(m_\rho(\mathrm{r})\right) =}$$
$$\boldsymbol{= \pi_{R_1}(r) \bowtie \pi_{R_2}(r) \bowtie \ldots \bowtie \pi_{R_k}(r) = m_\rho(\mathrm{r})}$$



Teorema: algoritmo di verifica della presenza del join senza perdita

Sia R uno schema di relazione, F un insieme di dipendenze funzionali su R e $\rho = \{R_1, R_2, \ldots, R_k\}$ una decomposizione di R. L'Algoritmo di verifica decide correttamente se ρ ha un join senza perdita.

Dimostrazione

Occorre dimostrare che:

ρ ha un join senza perdita ($m_\rho(\mathrm{r}) = r$ per ogni r legale)

se e solo se

quando l'algoritmo termina la tabella r ha una tupla con tutte 'a'.

Parte solo se

Supponiamo per assurdo che ρ abbia un join senza perdita ($m_\rho(\mathrm{r}) = r$) e che quando l'algoritmo termina la tabella r non abbia una tupla con tutte 'a'. La tabella r può essere interpretata come un'istanza legale di R, in quanto l'algoritmo termina quando non ci sono più violazioni delle dipendenze in F. Poiché nessun simbolo 'a' che compare nella tabella costruita inizialmente viene mai modificato dall'algoritmo, per ogni i, $\mathrm{i} = 1, \ldots, \mathrm{k}$, $\pi_{R_i}(r)$ contiene (fin dall'inizio) una tupla con tutte 'a' (quella ottenuta proiettando l'istanza r sugli attributi di R_i e precisamente nella riga corrispondente al sottoschema R_i), pertanto $m_\rho(\mathrm{r})$ contiene sicuramente una tupla con tutte 'a' e, quindi, $m_\rho(\mathrm{r}) \neq r$ (contraddizione).

Corollario

Sia R uno schema di relazione, F un insieme di dipendenze funzionali su R e $\rho = \{R_1, R_2, \ldots, R_k\}$ una decomposizione di R.

Se $R1 \cap R2 \rightarrow R1 - R2$ o $R1 \cap R2 \rightarrow R2 - R1$ allora ρ ha un join senza perdita.

Definizione: copertura minimale

Sia F un insieme di dipendenze funzionali. Una copertura minimale di F è un insieme G di dipendenze funzionali equivalente ad F tale che:

I. Per ogni dipendenza funzionale in G, la parte destra è un **singleton**, cioè è costituita da un unico attributo (ogni attributo nella parte **destra** non è ridondante)

II. **Per nessuna** dipendenza funzionale $X \to A$ in G , $\exists\, X' \subset X : G \equiv G - \{X \to A\} \cup \{X' \to A\}$
(ogni attributo nella parte **sinistra** non è ridondante)

III. **Per nessuna** dipendenza funzionale $X \to A$ in G, $G \equiv G - \{X \to A\}$
(ogni **dipendenza** non è ridondante)

Per ogni insieme di dipendenze funzionali F esiste una copertura minimale equivalente ad F che può essere ottenuta in tempo polinomiale in tre passi:

- Usando la regola della decomposizione, le parti destre delle dipendenze funzionali vengono ridotte a singleton.
- Ogni dipendenza funzionale $A_1A_2 \dots A_{i-1}A_iA_{i+1} \dots A_n \to A$ in F tale che $F \equiv F - \{A_1A_2 \dots A_{i-1}\boldsymbol{A_i}A_{i+1} \dots A_n \to A\} \cup \{A_1A_2 \dots \boldsymbol{A_{i-1}A_{i+1}} \dots A_n \to A\}$ viene sostituita appunto da $A_1A_2 \dots \boldsymbol{A_{i-1}A_{i+1}} \dots A_n \to A$; se quest'ultima appartiene già ad F la dipendenza originaria viene semplicemente eliminata, altrimenti il processo viene ripetuto ricorsivamente su $A_1A_2 \dots \boldsymbol{A_{i-1}A_{i+1}} \dots A_n \to A$; il passo termina quando nessuna dipendenza funzionale può più essere ridotta (tutti gli attributi delle parti sinistre delle dipendenze funzionali risultano non ridondanti).
- Ogni dipendenza funzionale $X \to A$ in F tale che $F \equiv F - \{X \to A\}$ viene eliminata, in quanto risulta ridondante; in questo modo minimizziamo il numero di dipendenze funzionali.

Teorema: algoritmo di decomposizione

Teorema Sia R uno schema di relazione ed F un insieme di dipendenze funzionali su R, che è una copertura minimale. L'Algoritmo di decomposizione permette di calcolare in tempo polinomiale una decomposizione ρ di R tale che:

- ogni schema di relazione in ρ è in 3NF
- ρ preserva F.

Dimostrazione (due proprietà della decomposizione)

ρ preserva F

Sia $G = \cup_{i=1}^{k} \pi_{R_i}(F)$. Poiché per ogni dipendenza funzionale $X \to A \in F$ si ha che $XA \in \rho$ (è proprio uno dei sottoschemi), si ha che questa dipendenza di F sarà sicuramente in G, quindi $G \supseteq F$ e, quindi $G^+ \supseteq F^+$. L'inclusione $G^+ \subseteq F^+$ è banalmente verificata in quanto, per definizione, $G \subseteq F^+$.

Ogni schema di relazione in ρ è in 3NF

Analizziamo i diversi casi che si possono presentare

1. Se $S \in \rho$, ogni attributo in S fa parte della chiave e quindi, banalmente, S è in 3NF.
2. Se $R \in \rho$ esiste una dipendenza funzionale in F che coinvolge tutti gli attributi in R. Poiché F è una copertura minimale tale dipendenza avrà la forma $R - A \to A$; poiché F è una copertura minimale, non ci può essere una dipendenza funzionale $X \to A$ in F^+ tale che $X \subset R - A$ e, quindi, $R - A$ è chiave in R.
   Sia $Y \to B$ una qualsiasi dipendenza in F; se $B = A$ allora, poiché F è una copertura minimale, $Y = R - A$ (cioè, Y è una superchiave); se $B \neq A$ allora $B \in R - A$ e quindi B è primo.
3. Se $XA \in \rho$, poiché F è una copertura minimale, non ci può essere una dipendenza funzionale $X' \to A$ in F^+ tale che $X' \subset X$ e, quindi, X è chiave in XA. Sia $Y \to B$ una qualsiasi dipendenza in F tale che $YB \subseteq XA$; se $B = A$ allora, poiché F è una copertura minimale, $Y = X$ (cioè, Y è una superchiave); se $B \neq A$ allora $B \in X$ e quindi B è primo.

   Nota: Possiamo avere $1 + 2$ (R residuo), oppure $1 + 3$, oppure solo 3

Teorema: $\rho \cup \{K\}$

Sia R uno schema di relazione, F un insieme di dipendenze funzionali su R, che è una copertura minimale e ρ la decomposizione di R prodotta dall'Algoritmo di decomposizione.
La decomposizione $\sigma = \rho \cup \{K\}$, dove K è una chiave per R, è tale che:

- Ogni schema di relazione in σ è in 3NF
- σ preserva F
- σ ha un join senza perdita.

Dimostrazione

σ preserva F

Poiché ρ preserva F anche σ preserva F.
Stiamo aggiungendo un nuovo sottoschema, quindi alla nuova G' dobbiamo aggiungere una proiezione di F, cioè $G' = G \cup \pi_K(F)$ quindi $G' \supseteq G \supseteq F$ e quindi $G'^{+} \supseteq G^{+} \supseteq F^{+}$. L'inclusione $G'^{+} \subseteq F^{+}$ è di nuovo banalmente verificata in quanto, per definizione, $G \subseteq F^{+}$.

Ogni schema di relazione in σ è in 3NF

Poiché $\sigma = \rho \cup \{K\}$, è sufficiente verificare che anche lo schema di relazione K è in 3NF. Mostriamo che K è chiave anche per lo schema K. Supponiamo per assurdo che K non sia chiave per lo schema K; allora esiste un sottoinsieme proprio K' di K che determina tutto lo schema K, cioè tale che $K' \rightarrow K \in F^{+}$ (più precisamente alla chiusura di $\pi_K(F)$, ma poiché $\pi_K(F) \subset F^{+}$ allora $(\pi_K(F))^{+} \subset F^{+}$). Poiché K è chiave per lo schema R, $K \rightarrow R \in F^{+}$; pertanto per transitività $K' \rightarrow R \in F^{+}$, che contraddice il fatto che K è chiave per lo schema R (verrebbe violato il requisito di minimalità). Pertanto, K è chiave per lo schema K e quindi per ogni dipendenza funzionale $X \rightarrow A$ in F^{+} con $XA \subseteq K$, A è primo.

σ ha un join senza perdita

Supponiamo che l'ordine in cui gli attributi in $R - K$ vengono aggiunti a Z dall'Algoritmo che calcola la chiusura di un insieme di attributi (in questo caso K^{+}) sia $A_1, A_2, \dots, A_n$, e supponiamo che per ogni i, $i = 1, \dots, n$, l'attributo Ai venga aggiunto a Z a causa della presenza in F (copertura minimale!) della dipendenza $Y_i \rightarrow A_i$ (quindi avremo anche un sottoschema $Y_i \rightarrow A_i$) dove:

$$Y_i \subseteq Z^{i-1} = KA_1A_2 \dots A_{i-1} \subseteq K^{+}.$$

Per dimostrare che σ ha un join senza perdita mostreremo che quando l'Algoritmo per la verifica del join senza perdita è applicato a σ viene prodotta una tabella che ha una riga con tutte 'a'.

Senza perdita di generalità, supponiamo che l'Algoritmo che verifica il join senza perdita esamini le dipendenze funzionali $Y_1 \rightarrow A_1, Y_2 \rightarrow A_2, \dots, Y_n \rightarrow A_n$ in questo ordine. Dimostreremo per induzione su i che dopo che è stata considerata la dipendenza funzionale $Y_i \rightarrow A_i$ nella riga che corrisponde allo schema di relazione K c'è una 'a' in ogni colonna j con $j \leq i$.

Passo base: $i = 1$.

Poiché $Y_1 \subseteq Z^0 = K$, sia nella riga che corrisponde allo schema di relazione Y_1A_1 (per costruzione) che in quella che corrisponde allo schema di relazione K (perché $Y_1 \subseteq K$) ci sono tutte 'a' in corrispondenza degli attributi in Y_1; inoltre nella riga che corrisponde allo schema di relazione Y_1A_1c'è una 'a' in corrispondenza ad A_1 (è un attributo dello schema).
Pertanto l'Algoritmo pone una 'a' in corrispondenza ad A_1 nella riga che corrisponde allo schema di relazione K per fare in modo che venga soddisfatta $Y_1 \rightarrow A_1$

Induzione: $i > 1$.

Per l'ipotesi induttiva, nella riga che corrisponde allo schema di relazione K c'è una 'a' in corrispondenza di ogni attributo A_j con $j \leq i - 1$.

Poiché $Y_i \subseteq KA_1A_2 \dots A_{i-1}$, sia nella riga che corrisponde allo schema di relazione Y_iA_i che in quella che corrisponde allo schema di relazione K ci sono tutte 'a' in corrispondenza agli attributi in Y_i ; (per la costruzione iniziale della riga corrispondente a K, per l'ipotesi induttiva e perché Y_i è contenuto in questo insieme) inoltre nella riga che corrisponde allo schema di relazione Y_iA_i c'è una 'a' in corrispondenza ad A_i (fa parte dello schema). Pertanto l'Algoritmo pone una 'a' in corrispondenza ad A_i nella riga che corrisponde allo schema di relazione K.

Algoritmi

Calcolo di X_F^+

Input schema R
insieme dipendenze funzionali su R F
sottoinsieme di R X

Output chiusura di X rispetto ad F Z

Inizio

```
Z = X
S = { A : Y → V ∈ F ∧ A ∈ V ∧ Y ⊆ X }

    while S ⊄ Z :
        Z = Z ∪ S
        S = { A : Y → V ∈ F ∧ A ∈ V ∧ Y ⊆ Z }

    return Z
```

Fine

Contenimento di F in G^+

Input due insiemi di dipendenze funzionali F e G

Output valore booleano

Inizio

```
for every X → Y ∈ F :
    calcola X_G^+
    if Y ⊄ X_G^+:
        return False
return True
```

Fine

Calcolo di X_G^+ a partire da F (ρ preserva F?)

Input schema R
insieme dipendenze funzionali su R F
decomposizione ρ
sottoinsieme di R X

Output chiusura di X rispetto a $G = \bigcup_{i=1}^{k} \pi_{R_i}(F)$ Z

Inizio

```
Z = X
S = ∅
for i = 1 to k :
    S = S ∪ (X ∩ ρ_i)_F^+ ∩ ρ_i
while S ⊄ Z :
    Z = Z ∪ S
    for i = 1 to k:
        S = S ∪ (Z ∩ ρ_i)_F^+ ∩ ρ_i
return Z
```

Fine

Verifica della presenza di join senza perdite in una decomposizione

Input schema $\boldsymbol{R}$
insieme dipendenze funzionali su R $\boldsymbol{F}$
decomposizione $\boldsymbol{\rho}$

Output valore booleano

Inizio

Costruisci una tabella r nel modo seguente:

- r ha |R| colonne e $|\rho|$ righe
- all'incrocio tra l' i-esima riga e alla j-esima colonna metti:
  - "a_j" se l'attributo $A_j \in R_i$
  - "b_{ij}" altrimenti

for every $X \to Y \in F$:

if $\exists\, t_1, t_2 \in r : t_1[X] = t_2[X] \wedge t_1[Y] \neq t_2[Y]$:

for every A_j **in** Y:

if $t_1[A_j] = a_j$:
$t_2[A_j] = t_1[A_j]$

else:
$t_1[A_j] = t_2[A_j]$

until r ha una riga con tutte 'a' **or** r non è cambiato

if r ha una riga con tutte 'a':
ρ ha un join senza perdita

else:
ρ non ha un join senza perdita

Fine

Calcolo della copertura minimale

Sia F un insieme di dipendenze funzionali. Una copertura minimale di F è un insieme G di dipendenze funzionali equivalente ad F tale che:

I. Per ogni dipendenza funzionale in G, la parte destra è un **singleton**, cioè è costituita da un unico attributo (ogni attributo nella parte **destra** non è ridondante)

I. **Per nessuna** dipendenza funzionale $X \to A$ in G , $\exists\, X' \subset X : G \equiv G - \{X \to A\} \cup \{X' \to A\}$
(ogni attributo nella parte **sinistra** non è ridondante)

II. **Per nessuna** dipendenza funzionale $X \to A$ in G, $G \equiv G - \{X \to A\}$
(ogni **dipendenza** non è ridondante)

Calcolo della decomposizione

Input	schema	$\boldsymbol{R}$
	copertura minimale su R	$\boldsymbol{F}$
Output	decomposizione di R	$\boldsymbol{\rho}$

Inizio

$S = \emptyset$

for every $A \in R$ tale che A non è coinvolto in nessuna dipendenza funzionale in F:

$S = S \cup \{A\}$

if $\boldsymbol{S} \neq \emptyset$:

$R = R - S$

$\rho = \rho \cup \{S\}$

if esiste una dipendenza funzionale in F che coinvolge tutti gli attributi di R :

$\rho = \rho \cup \{R\}$

else :

for every $X \rightarrow A \in F$:

$\rho = \rho \cup \{XA\}$

return ρ

Fine

Al fine di assicurare il join senza perdite, a tale decomposizione verrà aggiunto un sottoschema K tale che K è una chiave per R.