



Basi di Dati

Appunti del corso Basi di Dati 1

▼ Lezione del 27/09/2023

Definizioni Generali introduttive

Un database è un'insieme di file organizzati da strutture dati che facilitano la loro creazione, accesso e aggiornamento.

L'accesso agli elementi delle strutture dati è possibile mediante query o interrogazioni.

Un Sistema di Informazioni (Information System) è un insieme di dati organizzati fisicamente e organizzati tramite software in modo da poterli creare, visionare ed aggiornare.

Esistono diversi modelli per organizzare database, i due principali sono:

- Modelli **Logici** indipendenti dalle strutture fisiche
- Modelli **Concettuali** indipendenti dai modelli di realizzazione (struttura dati)

Esempi pratici di modelli di organizzazione dei database:

Mesh Model

I dati sono rappresentati come una collezione di record di dati omogenei, le relazioni binarie sono rappresentate come collegamenti (implementati con puntatori), sono rappresentati quindi come una struttura a grafo.

Hierarchical Model

Un tipo ristretto di Mesh Model, rappresenta un grafico mediante grafi implementati tramite alberi (o foreste).

Relational Model

I dati e le relazioni sono rappresentati come valori, non ci sono esplicativi collegamenti tra i dati (es non ci sono puntatori).

Un esempio di realizzazione di database mediante modello relazionale può essere la seguente:

STUDENTS			
Matric	Last name	Name	Birthday
276545	Smith	Mary	25/11/1980
485745	Black	Anna	23/04/1981
200768	Greens	Paolo	12/02/1981
587614	Smith	Lucy	10/10/1980
937653	Brown	Mavis	01/12/1980

COURSES			EXAMS		
Code	Title	Tutor	Stud	Vote	Course
01	Physics	Grant	276545	C	01
03	Chemistry	Beale	276545	B	04
04	Chemistry	Clark	937653	B	01

I tre livelli astratti di un Database:

- Schema Esterno:

Descrizione di una porzione del database in un modello logico attraverso "viste" parziali o derivate che possono fornire organizzazioni dei dati diverse da quelle utilizzate nello schema logico e che riflettono le esigenze e i privilegi di accesso di tipi specifici di utenti; più di uno schema esterno può essere associato a uno schema logico.

- Schema Logico:

Descrizione dell'intero database nel modello logico principale del DBMS, ad esempio la struttura delle tabelle.

- Schema Fisico:

Rappresentazione dello schema logico attraverso strutture di archiviazione fisiche, cioè file.

Indipendenza dei dati:

- Indipendenza Fisica:

I livelli logici ed esterni sono indipendenti dal livello fisico, ciò significa che le relazioni usate nel database sono indipendenti dalla loro realizzazione fisica (organizzazione dei file o la loro allocazione) inoltre l'implementazione fisica può cambiare senza avere ripercussioni sul programma.

- Indipendenza Logica:

Il livello esterno è indipendente dal livello logico, ciò significa che sono possibili aggiunte o modifiche alle viste non richiedono modifiche al livello logico.

Le modifiche allo schema logico che non influenzano lo schema esterno rimangono trasparenti.

In ogni Database esistono:

- Lo schema: sostanzialmente invariabile nel tempo che descrive la struttura del database stesso(nel modello relazionale rappresenta l'intestazione delle tabelle).
- Le istanze: i valori che correnti dei dati archiviati che possono cambiare nel tempo.

NAME	SURNAME	BIRTH	TOWN
Piero	Naples	22-10-63	Bari
Marco	Bianchi	01-05-54	Rome
Maria	Rossi	09-02-68	Milan
Maria	Bianchi	07-12-70	Bari
Paolo	Sossi	15-03-75	Palermo

Parte in bianco: Schema | Parte in giallo: istanze.

Integrità e Sicurezza

Integrità:

I dati devono soddisfare "vincoli" che esistono nel contesto di interesse:

- Uno studente risiede in una sola città (vincoli di dipendenza funzionale).
- Il numero di matricola identifica univocamente uno studente (vincoli di chiave).
- Un voto è un numero intero positivo compreso tra 18 e 30 (vincoli di dominio).
- Le ore straordinarie di un dipendente sono date dal prodotto del numero di ore e del salario orario.
- Il salario di un dipendente non può diminuire (vincoli dinamici).

Sicurezza

- I dati devono essere protetti dall'accesso non autorizzato.

L'amministratore del database (DBA) deve prendere in considerazione:

- Il valore attuale delle informazioni per l'organizzazione.
- Chi può accedere a quali dati e in quale modo.

Successivamente, il DBA deve prendere decisioni riguardanti:

- La regolamentazione dell'accesso.
 - Gli effetti di una violazione.
-

Modello Relazionale

Introduzione

Le relazioni, che sono un insieme di coppie ordinate i cui elementi appartengono a domini differenti, si traducono naturalmente in tabelle.

I dati e le relazioni/associazioni tra dati di diversi insiemi (relazioni/tabelle) sono rappresentati come valori.

Bisogna fare attenzione alle diverse accezioni di "relazione" e "relazione" nell'ambito matematico e informatico.

Definizione n°1, Il Dominio



Dominio: un insieme (possibilmente infinito) di valori.

Esempi:

- L'insieme degli interi è un dominio.
- L'insieme dei numeri decimali è un dominio.
- L'insieme delle stringhe di caratteri di lunghezza = 20 è un dominio.
- $\{0,1\}$ è un dominio

Definizione n°2



Siano D_1, D_2, \dots, D_k domini, non necessariamente distinti; il prodotto cartesiano di questi domini, indicato da:

$D_1 \times D_2 \times \dots \times D_k$

ed è l'insieme:

$$\{(V_1, V_2, \dots, V_k) \mid V_1 \in D_1, V_2 \in D_2, \dots, V_k \in D_k\}$$

Definizione n°3



Una relazione matematica è qualsiasi sottoinsieme del prodotto cartesiano di uno o più domini.

Una relazione che è un sottoinsieme del prodotto cartesiano di k domini è detta di grado k .

Gli elementi di una relazione sono chiamati tuple (o ennuple): il numero di tuple di una relazione è la sua cardinalità.

Ciascuna tupla di una relazione di grado k ha k componenti ordinate (il valore i -esimo proviene dal dominio i -esimo), ma non c'è un ordine tra le tuple.

Le tuple di una relazione sono tutte distinte (almeno per un valore)

Esempio:

Supponiamo $k=2$

$D_1 = \{\text{bianco}, \text{nero}\}$, $D_2 = \{0, 1, 2\}$

$D_1 \times D_2 = \{(\text{bianco}, 0), (\text{bianco}, 1), (\text{bianco}, 2), (\text{nero}, 0), (\text{nero}, 1), (\text{nero}, 2)\}$

Quindi $\{(\text{bianco}, 0), (\text{nero}, 0), (\text{nero}, 2)\}$ è una relazione di grado 2 (poiché due elementi per coppia) di cardinalità 3 (poiché presenti 3 coppie).

Il numero massimo di relazioni per quest' esempio è di 2^6 .

Notazione

Data r , una relazione di grado k

Data t , una tupla in r

Se i è un intero in $\{1, \dots, k\}$

$t[i]$ indica l' i -esimo componente di t .

Esempio:

$$r = \{(0,a), (0,c), (1,b)\}$$

$$t = (0,a)$$

$$t[2] = a$$

$$t[1,2] = (0,a)$$

Relazioni e Tabelle

Un attributo è definito dal nome A e dal suo dominio, che denotiamo con $\text{dom}(A)$.

Supponiamo che R sia un insieme di attributi. Una tupla su R è una funzione definita su R che associa a ciascun attributo A in R un elemento di $\text{dom}(A)$.

Se t è una tupla in R e A è un attributo in R, allora con $t(A)$ denotiamo il valore assunto dalla funzione t sull'attributo A. In altre parole, $t(A)$ rappresenta il valore dell'attributo A nella tupla t.

Le tuple sono un sottoinsieme dell'istanza dello schema.

Gli attributi rappresentano il “titolo” di ogni colonna della mia tabella.

Il relation name rappresenta il tipo di dato che sto immagazzinando nel database

L'istanza è l'insieme di dati che sono contenuti nelle varie tuple.

relation schema

relation name	attributes			
	Name	Surname	Exams	Avg
Student				

Student	Name	Surname	Exams	Avg
tuples	Paolo	Rossi	2	26.5
	Mario	Bianchi	10	28.7

relation instance

Dato $t1=\{\text{name}, \text{surname}, \text{exams}, \text{avg}\}$ questo insieme rappresenta gli attributi del mio database, il cui dominio è formato da String \cup String \cup Int \cup Reale.

Riassumendo

Una relazione può essere considerata come una tabella in cui ogni riga rappresenta una tupla distinta e ogni colonna corrisponde a un componente (con valori omogenei, cioè provenienti dallo stesso dominio).

Le colonne corrispondono ai domini D_1, D_2, \dots, D_k e hanno nomi univoci che sono autoesplicativi all'interno della tabella.

La coppia (nome attributo, dominio) è chiamata attributo; l'insieme di attributi di una relazione è chiamato schema.

Se R è una relazione e i suoi attributi sono:

A_1, A_2, \dots, A_k , lo schema è spesso indicato come:

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

Schemi e Istanze

Schema di una relazione: un nome di relazione R con un insieme di nomi di attributi.

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

Lo schema di una relazione è invariante nel tempo e descrive la sua struttura (aspetto intenzionale).

L'istanza di una relazione con schema $R(X)$ è rappresentata da un insieme r di tuple su X .

L'istanza contiene i valori attuali, che possono cambiare rapidamente (aspetto estensionale).

In altre parole, lo schema fornisce la struttura concettuale della relazione, mentre l'istanza rappresenta i dati concreti e attuali contenuti nella relazione.

Relazioni e Database

Schema del database:

un insieme di schemi di relazione con nomi diversi. In altre parole, il database schema è la descrizione complessiva della struttura e delle relazioni tra le tabelle (relazioni) all'interno di un database, e ciascun schema di relazione definisce la struttura specifica di una tabella all'interno del database.

Schema del database relazionale: un insieme di schemi R_1, R_2, \dots, R_n di relazioni.

Un database relazionale con lo schema R_1, R_2, \dots, R_n è definito come un insieme di relazioni

r_1, r_2, \dots, r_n , dove ogni r_i è un'istanza di relazione con schema R_i .

Nella definizione di un modello relazionale, i componenti di una relazione vengono indicati dai nomi degli attributi, piuttosto che dalla loro posizione.

$t[A_i]$ indica il valore dell'attributo con il nome A_i della tupla t .

Se t è la seconda tupla nell'esempio precedente, allora $t[Region] = Lombardia$.

Se Y è un sottoinsieme degli attributi dello schema X di una relazione ($Y \subseteq X$), allora $t[Y]$ è il sottoinsieme dei valori nella tupla t che corrispondono agli attributi in Y (chiamato anche restrizione di t).

Nel modello relazionale, i riferimenti tra dati in diverse relazioni vengono rappresentati mediante i valori di dominio che compaiono nelle tuple.

▼ Lezione del 28/09/2023

Il valore Null



Definizione

I valori **NULL** rappresentano la mancanza di informazioni o il fatto che l'informazione non è applicabile.

Ad esempio, per un numero di telefono posso utilizzare il valore Null poichè:

- La persona non ha un telefono.
- Non so se la persona ha un telefono.
- La persona ha un telefono, ma non conosco il numero.

In alcune situazioni, è necessario inserire un valore (la tupla deve essere conforme allo schema). In questi casi, è possibile definire un valore predefinito.

Tuttavia, è importante notare che alcuni attributi non dovrebbero mai assumere valori NULL, come l'ID studente o i voti degli esami, poiché questi dovrebbero essere sempre presenti e noti.

Il problema delle "unused" (non utilizzate) valori di dominio è importante da considerare nel design di un database.

Questi valori potrebbero effettivamente essere utilizzati in futuro, e potrebbero influenzare calcoli come il salario di un lavoratore.

Pertanto, è importante decidere attentamente come trattare tali valori nel contesto specifico del tuo sistema.

Il valore speciale **NULL** è un concetto importante nei database relazionali. Viene utilizzato come un valore polimorfico che **non appartiene a nessun dominio** specifico ma può sostituire valori in qualsiasi dominio.

È importante notare che due valori **NULL**, anche nello stesso dominio, sono considerati diversi. Questo significa che due valori **NULL** in una colonna possono rappresentare concetti diversi o mancanza di informazioni diversificate.

La gestione dei valori **NULL** e dei valori non utilizzati dovrebbe essere pianificata attentamente per garantire che il database rappresenti accuratamente i dati e possa eseguire le operazioni desiderate in modo efficace.

Ricorda Null NON E' 0

students	Matric	Surname	Name	Date of birth
	6554	Rossi	Mario	05/12/1978
	9283	Greens	Luisa	12/11/1979
	NULL	Rossi	Maria	01/02/1978

exams		
Student	Vote	Course
NULL	30	NULL
NULL	24	02
9283	28	01

courses		
Course	Title	Lecturer
01	NULL Math	NULL
02	Chemis try	02
04		01

In questo Database sono presenti troppi valori null.

Integrity Constraints

Integrity constraints sono regole o condizioni che vengono applicate ai dati all'interno di un database relazionale per garantire l'integrità dei dati e la coerenza delle informazioni.

Queste regole vengono imposte per garantire che i dati soddisfino determinati standard di qualità e correttezza.

Queste proprietà devono essere soddisfatte da ogni istanza nel Database.

Esistono due tipi di constraints:

- intra-relational constraints: definiti su singoli domini o su valori di stesse tuple o tuple di stesse istanze.
- interrelational constraints: definite tra più relazioni.

Keys - Chiavi

Le chiavi sono fondamentali per identificare le tuple di un istanza.

Definizione informale:



Una chiave relazionale (non necessariamente unica) è un attributo o un insieme di attributi che identificano unicamente una tupla.

Definizione formale:



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 due tuple distinte t1 e t2 che abbiano gli stessi valori per tutti gli attributi in X, cioè tali che $t1[X] = t2[X]$.
2. Non esiste un sottoinsieme proprio di X che soddisfa la condizione 1).

Definizione matematica:



- 1) $X \subseteq R (A_1, \dots, A_n)$
 $\forall t_1, t_2 \in R$
 $t_1[x] = t_2[x] \Rightarrow t_1 = t_2$
- 2) $\forall x' \subseteq X$
 $\exists t_1, t_2 \in r$ tale che:
 $t_1[x] = t_2[x] \wedge t_1 \neq t_2$

TAXCODE	CODE	SURNAME	NAME	ROLE	HIRING
CSR...	COD1	Rossi	Mario	Analyst	1995
BA...	COD2	Bianchi	Peter	Analyst	1990
NRI...	COD3	Neri	Paolo	Admin	1985

Una chiave è tale quindi se non esistono 2 attributi uguali al suo interno, in Role ho 2 volte Analyst e per questo NON è una chiave.

La definizione delle chiavi in questo schema (ID, SURNAME, NAME, HIRING) vale però solo per questa istanza d'esempio poiché sarebbe plausibile trovare in un vero Database più persone con lo stesso nome.

Una relazione può avere diverse chiavi alternative.

Tuttavia, tra queste chiavi alternative, se ne sceglie una come chiave primaria (primary key). La chiave primaria è solitamente quella più utilizzata o composta da un numero minore di attributi rispetto alle altre chiavi alternative.

La chiave primaria ha alcune caratteristiche importanti:

1. **Non Ammette Valori NULL**: La chiave primaria non consente valori NULL. Ogni tupla all'interno della relazione deve avere un valore univoco per ciascun attributo della chiave primaria.
2. **Identificazione Unica**: La chiave primaria deve essere in grado di identificare univocamente ogni tupla nella relazione. Questo significa che non possono esistere due tuple con gli stessi valori per tutti gli attributi della chiave primaria.
3. **Almeno una Chiave**: Ogni relazione deve avere almeno una chiave, poiché non possono esistere tuple identiche all'interno di una relazione. Quindi, anche se non viene esplicitamente definita una chiave primaria, ce ne sarà sempre almeno una.

Le chiavi, inclusa la chiave primaria, sono importanti perché consentono di stabilire relazioni tra le tabelle in un database. Ad esempio, le chiavi esterne in una tabella possono fare riferimento alla chiave primaria in un'altra tabella, consentendo di collegare dati tra tabelle diverse.

Interrelational Constraints

I vincoli di integrità referenziale tra le relazioni "Road violations" e "Officers" e tra "Road violations" e "Cars" possono essere definiti come segue:

- L'attributo "Officer" nella relazione "Road Violations" e l'attributo "ID" (che è la chiave) nella relazione "Agenti".
- Gli attributi "Prov" e "Plate" delle "Road Violations" e gli attributi "Prov" e "Plate" (che costituiscono la chiave) della relazione Cars.

<u>Code</u>	Date	Officer	Pro	Plate
34321	1/2/95	3987	MI	39548K
Cars				
53524	4/3/95	3295	TO	E39548
64521	5/4/96	3295	PR	839548
	Prov	Plate	Surname	Name
34321	MI	E39548	Rossi	Mario
	TO	F34268	Rossi	Mario
	PR	839548	Neri	Luigi

WARE OF
CONSTRAINTS
MULTIPLE
ATTRIBUTES

Svolgono un ruolo "chiave" nel concetto di un "modello basato su valori" (il modello relazionale).

Nella presenza di valori NULL, i vincoli possono essere resi meno restrittivi.

È possibile definire "azioni" compensatorie in seguito a violazioni.

Dipendenze funzionali

Una dipendenza funzionale stabilisce un collegamento semantico tra due insiemi non vuoti di attributi, X e Y, appartenenti a uno schema R.

Questo vincolo viene scritto come $X \rightarrow Y$ e viene letto come: "X determina Y".



Definizione

Sono dei modi per esprimere vincoli, fondamentali nella teoria relazionale.

Esempio di dipendenze funzionali:

Supponiamo di avere uno schema di relazione "Flights" (Codice, Giorno, Pilota, Ora) con i seguenti vincoli basati sul buon senso:

- Un volo con un determinato codice parte sempre alla stessa ora.
- C'è solo un volo con un dato pilota, in un dato giorno, a un dato orario.
- C'è solo un pilota per un dato volo in un dato giorno.

Questi vincoli generano le seguenti dipendenze funzionali:

- Code → Time (Il codice determina l'ora di partenza del volo.)
- {Day, Pilot, Time} → Code (Il giorno, il pilota e l'orario determinano il codice del volo.)
- {Code, Day} → Pilot (Il codice e il giorno determinano il pilota del volo.)

Diciamo che un' istanza r con schema R soddisfa la dipendenza funzionale X

→ Y se:



1)

La dipendenza funzionale $X \rightarrow Y$ è applicabile a R nel senso che sia X che Y sono sottoinsiemi dell'insieme di attributi R, cioè entrambi X e Y sono attributi presenti nello schema R.

2)

Nelle tuple r che sono identiche sull'insieme di attributi X, devono anche essere identiche sull'insieme di attributi Y.

$$t1[X] = t2[X] \Rightarrow t1[Y] = t2[Y].$$

La freccia verso destra significa che se le tuple sono uguali su X allora devono essere uguali su Y.

X e Y sono sottoinsiemi di attributi di tutti gli attributi dello schema.

▼ Lezione del 11/10

L'algebra relazionale

Definiamo come algebra relazionale una **notazione algebrica specifica** utilizzata per realizzare **query** su un database relazionale, composta da un insieme di operatori unari e binari che, se applicati rispettivamente ad una o due istanze di relazioni, generano una **nuova istanza**.

In particolare, individuiamo i seguenti quattro tipi di operatore:

1. **Rimozione** di specifiche parti di una relazione (Proiezione e Selezione)
2. **Operazioni insiemistiche** (Unione, Intersezione e Differenza)
3. **Combinazione delle tuple di due relazioni** (Prodotto cartesiano e Join)
4. **Ridenominazione** di attributi.

L'algebra relazionale è un **linguaggio procedurale**, ossia descrive l'esatto ordine con cui gli operatori devono essere applicati.

Proiezione e Selezione

Proiezione



Definizione

Sia R una relazione con istanza r e sia A_1, \dots, A_k un insieme di attributi.

Una proiezione

su R è un operatore unario che effettua una restrizione con A_1, \dots, A_k su tutte

le tuple di R :

$$\pi_{A_1, \dots, A_k}(R) := \{t[A_1, \dots, A_k] \mid t \in r\}$$

In altre parole, una proiezione effettua un "**taglio verticale**" su una relazione, selezionando solo le colonne A_1, \dots, A_k

Esempio:

- Supponiamo di voler ottenere una lista di tutti i clienti presenti in questa relazione:

Customers		
Name	C#	Town
Rossi	C1	Roma
Rossi	C2	Milano
Rossi	C3	Milano
Bianchi	C4	Roma
Verdi	C5	Roma

- Proviamo ad effettuare una proiezione $\pi_{Name}(Customers)$, il cui output sarà:

$\pi_{Name}(Customers)$
Name
Rossi
Bianchi
Verdi

- La nuova relazione generata dalla proiezione segue comunque le regole dell'insiemistica, dunque non possono esserci tuple uguali. Difatti, notiamo come nell'output sia presente una sola tupla contenente il nome "Rossi", nonostante nella relazione iniziale ve ne fossero tre.
- Per prevenire tale perdita di informazione, proviamo ad effettuare la proiezione $\pi \text{ Name, Town}(\text{Customers})$, il cui output sarà:

$\pi_{\text{Name, Town}}(\text{Customers})$	
Name	Town
Rossi	Roma
Rossi	Milano
Bianchi	Roma
Verdi	Roma

- La situazione è migliorata rispetto alla prima proiezione, tuttavia vi è stata comunque una perdita di informazione.
- Per prevenire totalmente la perdita di informazione, quindi, sfruttiamo l'unicità della chiave C#, effettuando la proiezione $\pi \text{ Name, C\#}(\text{Customers})$, il cui output sarà:

$\pi_{\text{Name, C\#}}(\text{Customers})$	
Name	C#
Rossi	C1
Rossi	C2
Rossi	C3
Bianchi	C4
Verdi	C5

- Abbiamo quindi ottenuto una lista completa dei nostri clienti senza alcuna perdita di informazioni

Selezione



Definizione

Data una relazione R con istanza r e schema R(X), una selezione su R è un operatore unario che data una proposizione logica ' σ ' seleziona tutte le tuple di R per cui ' σ ' è rispettata:

$$\sigma(R) := \{t \in r \mid \sigma \text{ è valida}\}$$

Le proposizioni logiche associate ad una selezione corrispondono ad un composto di espressioni Booleane (dunque utilizzando operatori come \wedge , \neg e \neg) i cui termini appaiono nelle forme $A \wedge B$ o $A \wedge a$, dove:

- $A, B \in R(X) \mid \text{dom}(A) = \text{dom}(B)$
- $A \in R(X), a \in \text{dom}(A)$
- \emptyset è un operatore di comparazione ($<, =, -, >$)

In altre parole, una selezione effettua un "taglio orizzontale" su una relazione, selezionando solo alcune tuple di essa.

Esempio:

- Supponiamo di voler ottenere tutte le informazioni riguardo i clienti provenienti da Roma presenti in questa relazione:

Customers		
Name	C#	Town
Rossi	C1	Roma
Rossi	C2	Milano
Rossi	C3	Milano
Bianchi	C4	Roma
Verdi	C5	Roma

- Possiamo effettuare una selezione $\sigma_{\text{Town}='Roma'}(\text{Customers})$ per ottenere le tuple richieste:

$\sigma_{\text{Town}='Roma'}(\text{Customers})$		
Name	C#	Town
Rossi	C1	Roma
Bianchi	C4	Roma
Verdi	C5	Roma

- Vogliamo ora ottenere tutte le informazioni riguardo i clienti chiamati "Rossi" provenienti da Roma. Possiamo effettuare una selezione $\sigma_{Nome='Rossi' \wedge Town='Roma'}(Customers)$ per ottenere le tuple richieste:

$\sigma_{Nome='Rossi' \wedge Town='Roma'}(Customers)$		
Name	C#	Town
Rossi	C1	Roma

Unione, Intersezione e Differenza

Compatibilità di unione

Data una relazione R1 con schema R1(A1, . . . , Ak) ed una relazione R2 con schema

R2(a1, . . . , Ah), tali relazioni vengono dette compatibili in unione se e solo se:

- k = h, ossia hanno lo stesso numero di attributi
- $\forall i \in [1, k], \text{dom}(R1.A_i) = \text{dom}(R2.A_i)$, ossia ogni attributo corrispondente ha lo stesso dominio

Unione



Definizione

Date due relazioni compatibili in unione R1 e R2 con rispettive istanze r1 e r2,

l'unione tra R1 e R2 è un operatore binario che restituisce una nuova relazione

contenente tutte le tuple presenti in almeno una relazione tra R1 e R2.

$$R1 \cup R2 := \{t \mid t \in r1 \vee t \in r2\}$$

Affinché sia possibile utilizzare l'operatore di unione, non è necessario che gli attributi corrispondenti delle due relazioni abbiano lo stesso nome, ma solo lo stesso dominio (nonostante spesso non abbia alcun senso unire due relazioni aventi attributi con nomi diversi).

Esempio:

- Consideriamo le seguenti due relazioni, descriventi gli insegnanti e i responsabili di vari dipartimenti di una scuola:

Teachers		
Name	Code	Department
Rossi	C1	Math
Rossi	C2	Italian
Bianchi	C3	Math
Verdi	C4	English

Admins		
Name	AdminCode	Department
Esposito	C1	English
Riccio	C2	Math
Pierro	C3	Italian
Verdi	C4	English
Bianchi	C5	English

- Effettuando l'unione tra *Teachers* e *Admins*, otteniamo una nuova relazione contenente tutti i membri dello staff:

Teachers \cup Admins		
Name	Code	Department
Rossi	C1	Math
Rossi	C2	Italian
Bianchi	C3	Math
Verdi	C4	English
Esposito	C1	English
Riccio	C2	Math
Pierro	C3	Italian
Bianchi	C5	English

- Consideriamo le seguenti due relazioni:

Teachers		
Name	Code	Department
Rossi	C1	Math
Rossi	C2	Italian
Bianchi	C3	Math
Verdi	C4	English

Admins			
Name	Code	Department	Salary
Esposito	C1	English	1250
Riccio	C2	Math	2000
Pierro	C3	Italian	1000

- È impossibile applicare l'operatore unione tra le due relazioni *Teachers* e *Admins*, poiché non possiedono lo stesso numero di attributi
- Per risolvere il problema, possiamo effettuare prima una proiezione su *Admins*, per poi effettuare l'unione con *Teachers*:

Teachers $\cup \pi_{\text{Name}, \text{AdminCode}, \text{Department}}(\text{Admins})$		
Name	Code	Department
Rossi	C1	Math
Rossi	C2	Italian
Bianchi	C3	Math
Verdi	C4	English
Esposito	C1	English
Riccio	C2	Math
Pierro	C3	Italian

- Consideriamo ora le seguenti due relazioni:

Teachers		
Name	Code	Department
Rossi	C1	Math
Rossi	C2	Italian
Bianchi	C3	Math
Verdi	C4	English

Admins		
Name	Code	Service Years
Esposito	C1	3
Riccio	C2	13
Pierro	C3	7

- È impossibile applicare l'operatore unione tra le due relazioni *Teachers* e *Admins*, poiché $\text{dom}(\text{Teachers}.\text{Department}) \neq \text{dom}(\text{Admins}.\text{ServiceYears})$. Tuttavia, possiamo effettuare una proiezione su entrambe le relazioni, per poi unirle:

$\pi_{\text{Name}, \text{Code}}(\text{Teachers}) \cup \pi_{\text{Name}, \text{Code}}(\text{Admins})$	
Name	Code
Rossi	C1
Rossi	C2
Bianchi	C3
Verdi	C4
Esposito	C1
Riccio	C2
Pierro	C3

Intersezione



Definizione

Date due relazioni **compatibili** in unione R_1 e R_2 con rispettive istanze r_1 e r_2 ,

l'intersezione tra R_1 e R_2 è un **operatore binario** che restituisce una nuova relazione

contenente tutte le tuple presenti sia in R_1 sia in R_2 .

$$R_1 \cap R_2 := \{t \mid t \in r_1 \wedge t \in r_2\}$$

Differenza



Definizione

Date due relazioni compatibili in unione R1 e R2 con rispettive istanze r1 e r2,

la differenza tra R1 e R2 è un operatore binario che restituisce una nuova relazione

contenente tutte le tuple presenti in R1 ma non in R2.

$$R1 - R2 := \{t \mid t \in r1 \wedge t \notin r2\}$$

Contrariamente da unione e intersezione, l'operatore di **differenza non è commutativo**.

Esempio:

- Consideriamo le seguenti due relazioni, descriventi gli insegnanti e i responsabili di vari dipartimenti di una scuola:

Teachers		
Name	Code	Department
Rossi	C1	Math
Rossi	C2	Italian
Verdi	C3	English
Bianchi	C4	English

Admins		
Name	AdminCode	Department
Esposito	C1	English
Riccio	C2	Math
Verdi	C3	English
Bianchi	C4	English

- Effettuando **l'intersezione** tra *Teachers* e *Admins*, otteniamo una nuova relazione

contenente tutti gli insegnanti che sono anche responsabili:

Teachers \cap Admins		
Name	Code	Department
Verdi	C4	English
Bianchi	C5	English

- Effettuando la **differenza** tra *Teachers* e *Admins*, otteniamo una nuova relazione contenente tutti gli insegnanti che non sono responsabili:

Teachers – Admins		
Name	Code	Department
Rossi	C1	Math
Rossi	C2	Italian

- Analogamente, effettuando la differenza tra *Admins* e *Teachers*, otteniamo una nuova relazione contenente tutti i responsabili che non sono insegnanti:

Admins – Teachers		
Name	AdminCode	Department
Esposito	C1	English
Riccio	C2	Math

Ridenominazione



Definizione

Sia R una relazione con istanza r e schema $R(A_1, \dots, A_k)$. Una **ridenominazione** su R è un operatore unario che restituisce una nuova relazione R' con istanza r' e schema $R'(B_1, \dots, B_k)$, dove le tuple di r' sono identiche alle tuple di r :

$$\rho_{B_1, \dots, B_k} \leftarrow A_1, \dots, A_k(R) := \{t' \mid t'[B_i] = t[A_i], t \in r, \forall i \in [1, k]\}$$

In altre parole, una ridenominazione **modifica** il nome di un attributo della relazione.

Prodotto Cartesiano



Definizione

Siano R_1 ed R_2 due relazioni con rispettive istanze r_1 e r_2 . Il **prodotto cartesiano** di R_1 e R_2 è un operatore binario che restituisce una relazione contenente tutte le possibili combinazioni tra le tuple di r_1 e le tuple di r_2 :

$$R_1 \times R_2 := \{(t_1, t_2) \mid t_1 \in r_1, t_2 \in r_2\}$$

Esempio:

- Consideriamo le seguenti relazioni:

Customers		
Name	C#	Town
Rossi	C1	Roma
Rossi	C2	Milano
Bianchi	C3	Roma
Verdi	C4	Roma

Orders			
O#	C#	A#	Qnty
O1	C1	A1	100
O2	C2	A2	200
O3	C3	A2	150
O4	C4	A3	200
O1	C1	A2	200
O1	C1	A3	100

- Vogliamo ottenere l'elenco di tutti i clienti e gli ordini da loro effettuati.
- Prima di poter effettuare il prodotto cartesiano tra le due relazioni, è necessario ridenominare uno dei due attributi C# presenti in entrambe le relazioni

$$\text{OrdersR} = \rho_{C\# \leftarrow CC\#}(\text{Orders})$$

- Successivamente, effettuando il prodotto cartesiano Customers \times OrdersR, otteniamo:

Customers × OrdersR						
Name	C#	Town	O#	CC#	A#	Qty
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
Rossi	C2	Milano	O2	C2	A2	200
⋮	⋮	⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮	⋮	⋮
Verdi	C4	Roma	O4	C4	A3	200
Verdi	C4	Roma	O1	C1	A2	200

- A questo punto, però, notiamo la presenza di alcune incorrettezze. Ad esempio, il cliente "Rossi", avente codice C1, non ha mai effettuato l'ordine "(O2, C2, 200)", il quale invece è stato effettuato dal cliente avente codice C2.
- Possiamo risolvere tale problema effettuando una selezione dopo aver effettuato il prodotto cartesiano:

$\sigma_{C\#=CC\#}(Customers \times OrdersR)$						
Name	C#	Town	O#	CC#	A#	Qty
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

- Infine, per via del select svolto, le colonne C# e CC# risultano essere uguali, dunque potremmo rimuovere una delle due effettuando una proiezione. La query finale, quindi risulta essere:

$$\text{OrdersR} = \rho_{C\#} \leftarrow CC\#(\text{Orders})$$
$$\begin{aligned} \text{CustomerOrders} = & \pi_{\text{Name}, C\#, \text{Town}, O\#, A\#, \text{Qnty}} (\sigma_{C\#=CC\#} \\ & (\text{Customers} \times \text{OrdersR})) \end{aligned}$$

CustomerOrders					
Name	C#	Town	O#	A#	Qnty
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

Join Naturale



Definizione

Siano R1 e R2 due relazioni con rispettive istanze r1 e r2 e rispettivi schemi R1(X) e R2(Y). Il join naturale tra R1 e R2 è un operatore binario equivalente a:

$$R1 \bowtie R2 = \pi_{X, (Y - X)} (\sigma_{\varphi}(R1 \times R2))$$

dove dato un insieme di attributi A1, ..., Ak | 8i 2 [1, k], Ai 2 X \ Y si ha che:

- $\varphi := R1.A1 = R2.A1 \wedge \dots \wedge R1.Ak = R2.Ak$

In altre parole, il **join naturale** tra R1 ed R2 restituisce l'insieme di tutte le combinazioni tra tuple di r1 ed r2 che sono uguali per i loro attributi in comune.

Esempi:

- 1-Riprendiamo l'esempio visto per il prodotto cartesiano: date le seguenti due relazioni, vogliamo ottenere un elenco di tutti i clienti e gli ordini da

loro effettuati

Customers		
Name	C#	Town
Rossi	C1	Roma
Rossi	C2	Milano
Bianchi	C3	Roma
Verdi	C4	Roma

Orders			
O#	C#	A#	Qnty
O1	C1	A1	100
O2	C2	A2	200
O3	C3	A2	150
O4	C4	A3	200
O1	C1	A2	200
O1	C1	A3	100

- Notiamo come la soluzione già vista sia equivalente ad un join naturale tra le due relazioni:

Customers ⚠ Orders					
Name	C#	Town	O#	A#	Qnty
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

- Oltre alle due precedenti relazioni, consideriamo anche la seguente relazione:

Articles		
A#	Label	Price
A1	Plate	3
A2	Glass	2
A3	Mug	4

- Vogliamo ottenere una lista dei nomi e delle città dei clienti che hanno ordinato più di 100 pezzi di articoli che costano più di due euro.
- Prima di tutto, effettuiamo un join naturale tra le tre relazioni:

$$\text{CustOrdArt} = (\text{Customers} \bowtie \text{Orders}) \bowtie \text{Articles}$$

CustOrdArt							
Name	C#	Town	O#	A#	Qnty	Label	Price
Rossi	C1	Roma	O1	A1	100	Plate	3
Rossi	C1	Roma	O1	A2	200	Glass	2
Rossi	C1	Roma	O1	A3	100	Mug	4
Rossi	C2	Milano	O2	A2	200	Glass	2
Bianchi	C3	Roma	O3	A2	150	Glass	2
Verdi	C4	Roma	O4	A3	200	Mug	4

- Successivamente, selezioniamo le tuple che rispettano le condizioni che ci interessano:

$\sigma_{\text{Qnty} > 100 \wedge \text{Price} > 2}(\text{CustOrdArt})$							
Name	C#	Town	O#	A#	Qnty	Label	Price
Verdi	C4	Roma	O4	A3	200	Mug	4

- Infine, effettuiamo una proiezione sul nome e la città delle tuple ottenute:

$\pi_{\text{Name}, \text{Town}}(\sigma_{\text{Qnty} > 100 \wedge \text{Price} > 2}(\text{CustOrdArt}))$	
Name	Town
Verdi	Roma

- 3- Oltre alla soluzione appena vista, possiamo ottenere lo stesso risultato selezionando prima le tuple che rispettano le condizioni e i

dati che ci interessano, per poi effettuare il join tra loro, rendendo così la query più efficiente, poiché la mole di dati su cui vengono applicati gli operatori è minore:

$$\begin{aligned} & \pi_{\text{Name}, \text{Town}}((\text{Customer} \bowtie \sigma_{\text{Qnty} > 100}(\text{Order})) \\ & \bowtie \sigma_{\text{Price} > 2}(\pi_{\text{A}\#}(\text{Price}(\text{Article}))) \end{aligned}$$

Casi Speciali join naturale

Siano R1 e R2 due relazioni con rispettivi schemi R1(X) e R2(Y).

1. Se R1 ed R2 hanno un insieme di attributi in comune ma nessun valore in comune

per tali attributi, allora il risultato sarà un insieme vuoto:

$$\begin{aligned} Z \subseteq X \cap Y, \forall t_1 \in R_1 \wedge t_2 \in R_2 \mid t_1[Z] = t_2[Z] \Rightarrow R_1 \bowtie R_2 \\ = \emptyset \end{aligned}$$

2. Se R1 ed R2 non hanno degli attributi in comune, allora il join naturale degenererà

in un prodotto cartesiano:

$$X \cap Y = \emptyset \Rightarrow R_1 \bowtie R_2 = R_1 \times R_2$$

Posso usare la rinomina.

▼ Lezione del 12/10

Theta Join



Definizione

Siano R1 e R2 due relazioni con rispettive istanze r1 e r2 e rispettivi schemi R1(X) e R2(Y).

Il join naturale tra R1 e R2 è un operatore binario equivalente a:

$$R1 \triangleright\triangleleft A \theta B \quad R2 = \sigma_{A \theta B}(R1 \times R2)$$

dove:

- $A \in R1(X), B \in R2(Y)$
- $\text{dom}(A) = \text{dom}(B)$
- θ è un operatore di confronto ($<, =, >$)

In altre parole, un theta join tra R1 ed R2 restituisce tutte le combinazioni tra le tuple di r1 e r2 che rispettano una condizione su un attributo in comune

In alcuni casi può essere necessario effettuare il join tra una relazione e se stessa (**self join**), in modo da ottenere combinazioni di tuple della stessa relazione.

Esempio:

- Data la seguente relazione, vogliamo ottenere una lista dei codici e dei nomi dei dipendenti aventi un salario maggiore dei loro supervisori

Employees				
Name	C#	Section	Salary	Supervisor#
Rossi	C1	B	100	C3
Pirlo	C2	A	200	C3
Bianchi	C3	A	500	NULL
Verdi	C4	B	200	C2
Neri	C5	B	150	C1
Tosi	C6	B	100	C1

- In tal caso, la soluzione migliore risulta essere una selezione effettuata su self join di *Employees*, in modo da poter accoppiare ogni dipendente al suo supervisore. Possiamo ottenere un self join eseguendo una delle seguenti query:

- Creiamo una copia della relazione per poi effettuare un theta join tra il codice dei dipendenti e il codice dei loro supervisori, specificando la relazione di appartenenza di ognuno dei due attributi confrontati:

EmployeesC = Employees

EmpSup1 = EmployeesC \bowtie EmployeesC.Supervisor# =
Employees.C# Employees

- Effettuiamo un theta join tra la relazione ed una sua copia rinominata, senza dover specificare la relazione di appartenenza degli attributi confrontati:

$X = \{\text{Name, C\#, Section, Salary, Supervisor\#}\}$

$Y = \{\text{CName, CC\#, CSec, CSal, CSup\#}\}$

EmpSup2 = Employees \bowtie Supervisor# = C# $\rho X \leftarrow Y$ (Employees)

- Attenzione: utilizzare il join naturale al posto del theta join in una delle tre soluzioni genererebbe una relazione identica a *Employees*, poiché ogni tupla verrebbe joinata con se stessa scartando automaticamente gli attributi doppi.
- Successivamente, eseguiamo la selezione richiesta, mantenendo solo le tuple dove il salario del dipendente è maggiore del salario del suo supervisore:

$\sigma_{\text{Employees.Salary} > \text{EmployeesC.Salary}}(\text{EmpSup1})$

oppure:

$\sigma_{\text{Salary} > \text{CSalary}}(\text{EmpSup2})$

$\sigma_{\text{Salary} > \text{CSal}}(\text{EmpSup}_2)$										
Name	C#	Section	Salary	Supervisor#	CName	CC#	CSec	CSal	CSup#	
Verdi	C4	B	200	C2	Pirlo	C2	A	200	C3	
Neri	C5	B	150	C1	Rossi	C1	B	100	C3	
Tosi	C6	B	100	C1	Rossi	C1	B	100	C3	

- Infine, effettuiamo una proiezione sul nome e il codice del dipendente:

$\pi_{\text{Employees.Name}, \text{Employees.C\#}}(\sigma_{\text{Employees.Salary} > \text{EmployeesC.Salary}}(\text{EmpSup1}))$
oppure:
 $\pi_{\text{Name}, \text{C\#}}(\#_{\text{Salary} > \text{CSal}}(\text{EmpSup2}))$

$\pi_{\text{Name}, \text{C\#}}(\sigma_{\text{Salary} > \text{CSal}}(\text{EmpSup}_2))$	
Name	C#
Verdi	C4
Neri	C5
Tosi	C6

Quantificazione universale

Fino ad ora, abbiamo visto solo query inerenti la **quantificazione esistenziale** (indicata col simbolo \exists), ossia la selezione di oggetti che soddisfino almeno una volta una determinata condizione.

Tuttavia, utilizzando solo gli operatori visti precedentemente, non abbiamo un modo per poter effettuare query inerenti alla **quantificazione universale** (indicata col simbolo \forall), ossia la selezione di oggetti che soddisfino **sempre** una determinata condizione.

Nella logica del primo ordine, la negazione di "Per ogni oggetto x la condizione 'è vera'" non corrisponde a "Per ogni oggetto x la condizione 'è falsa'", bensì corrisponde a "Esiste almeno un oggetto x per cui la condizione 'è falsa'".

In simboli, diremmo che:

$$\neg(\forall x, \varphi(x)) \neq \forall x, \neg\varphi(x)$$

ma bensì:

$$\neg(\exists x, \varphi(x)) = \forall x, \neg\varphi(x)$$

Per selezionare tutte le tuple di una relazione per cui una determinata condizione è sempre valida, quindi, ci basta scartare tutte le tuple per cui almeno una volta la condizione non è valida.

Esempi:

- 1- Data la seguente relazione, vogliamo ottenere un elenco di tutti i nomi e la città di provenienza dei clienti che hanno sempre effettuato un ordine di più di 100 pezzi.

Customers			Orders			
Name	C#	Town	O#	C#	A#	Qty
Rossi	C1	Roma	O1	C1	A1	100
Rossi	C2	Milano	O2	C2	A2	200
Bianchi	C3	Roma	O3	C3	A2	150
Verdi	C4	Roma	O4	C4	A3	200
			O1	C1	A2	200
			O1	C1	A3	100

- Per ottenere l'elenco richiesto, ci basta scartare l'elenco dei nomi e delle città che almeno una volta non hanno acquistato più di 100 pezzi dall'elenco totale dei nomi e delle città:

$$\text{ElencoNonValidi} = \pi_{\text{Name}, \text{Town}}(\sigma_{\neg(\text{Qty} > 100)}(\text{Customers} \bowtie \text{Orders}))$$

$$R = \pi_{\text{Name}, \text{Town}}(\text{Customers}) - \text{ElencoNonValidi}$$

oppure, direttamente:

$$R = \pi_{\text{Name}, \text{Town}}(\text{Customers}) - \pi_{\text{Name}, \text{Town}}(\sigma_{\neg(\text{Qty} > 100)}(\text{Customers} \bowtie \text{Orders}))$$

R	
Name	Town
Bianchi	Roma
Verdi	Roma

- 2-Data la seguente relazione, vogliamo ottenere una lista dei codici e dei nomi dei supervisori aventi un salario maggiore di tutti i loro dipendenti

Employees				
Name	C#	Section	Salary	Supervisor#
Rossi	C1	B	100	C3
Pirlo	C2	A	200	C3
Bianchi	C3	A	500	NULL
Verdi	C4	B	200	C2
Neri	C5	B	150	C1
Tosi	C6	B	100	C1

- Anche in questo caso, per ottenere l'elenco richiesto ci basta scartare i supervisori aventi il salario minore di almeno un dipendente:

$$\text{EmployeesC} = \text{Employees}$$

$$\text{EmpSup} = \text{EmployeesC} \bowtie_{\text{EmployeesC.Supervisor\#} = \text{Employees.C\#}} \text{Employees}$$

$$\text{Invalid} = \pi_{\text{Name}, \text{ C\#}}(\sigma_{\neg(\text{Employees.Salary} < \text{EmployeesC.Salary})}(\text{EmpSup}))$$

$$R = \pi_{\text{Name}, \text{ C\#}}(\text{EmpSup}) - \text{Invalid}$$

R	
Name	C#
Bianchi	C3

▼ Lezione del 19/10

Teoria della Normalizzazione

Dipendenze funzionali



Sia R uno schema con istanza r e siano $X, Y \subseteq R$.

Definiamo come **dipendenza funzionale** tra X e Y, indicata come $X \rightarrow Y$ e detta "X determina Y", un vincolo di integrità che impone ad ogni coppia di tuple in r aventi valori uguali su X di avere valori uguali anche su Y:

$$\forall t_1, t_2 \in r, t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$$

Attenzione: notiamo come nella condizione non vi sia un "se e solo se", bensì solo un "se". Dunque, $X \rightarrow Y$ non implica che ogni coppia di tuple in r aventi valori uguali su Y debba avere valori uguali anche su X:

Esempi:

- Supponiamo di avere il seguente schema $\text{Flights}(\text{Code}, \text{Day}, \text{Pilot}, \text{Time})$. Viene naturale considerare i seguenti vincoli:

- Un volo con un certo codice partì sempre allo stesso orario
- C'è un solo volo con un certo pilota ad un certo orario in un certo giorno
- C'è un solo pilota di un certo volo in un certo giorno

Dunque, imponiamo le seguenti dipendenze funzionali sullo schema:

- $Code \rightarrow Time$
- $(Day, Pilot, Time) \rightarrow Code$
- $(Code, Day) \rightarrow Pilot$

Istanza legale



Dato uno schema R e un insieme F di dipendenze funzionali definite su R, diciamo che un'istanza di R è **legale su F** se soddisfa tutte le dipendenze funzionali in F

Esempi:

- Consideriamo la seguente relazione su cui sono definite le seguenti dipendenze funzionali:

$$F = \{A \rightarrow B\}$$

A	B	C	D
a ₁	b ₁	c ₁	d ₁
a ₁	b ₁	c ₂	d ₂
a ₂	b ₂	c ₁	d ₃

Tale istanza è *legale* su F, poiché soddisfa le dipendenze funzionali in F: tutte le tuple aventi $t[A] = a_1$ hanno anche $t[B] = b_1$, così come tutte le tuple (nonostante sia solo una in questo caso) aventi $t[A] = a_2$ hanno anche $t[B] = b_2$

- Consideriamo la seguente situazione:

$$F = \{A \rightarrow B\}$$

A	B	C	D
a_1	b_1	c_1	d_1
a_2	b_1	c_2	d_2
a_2	b_2	c_1	d_3

Tale istanza è *illegal*e su F, poiché non soddisfa le dipendenze funzionali in F: la seconda e la terza tupla hanno lo stesso valore in A ma non in B

Chiusura di F



Sia R uno schema e sia F un insieme di dipendenze funzionali definite su R.

Definiamo come **chiusura di F**, indicata con F^+ , l'insieme di **tutte** le dipendenze funzionali, incluse quelle non in F, soddisfatte da ogni istanza di R legale su F.

$$F^+ = \cap_{r \in L} \{f \text{ dipendenza funzionale } | r \text{ soddisfa } f\}$$

dove $L = \{r \text{ istanza di } R | r \text{ legale su } F\}$. In generale, quindi, si ha che $F \subseteq F^+$.

Chiavi



Dato uno schema di relazione R e un insieme F di dipendenze funzionali definite su R:

Un sottoinsieme K di uno schema di relazione R è una chiave di R se:

1. $K \rightarrow R \in F^+$
2. non esiste un sottoinsieme proprio $K' \subseteq K$ tale che $K' \rightarrow R$

Chiave Primaria



Dato uno schema R e un insieme F di dipendenze funzionali, è possibile che ci siano diverse chiavi di R. In SQL, ne verrà scelta una come chiave primaria (la quale non può essere assegnata un valore nullo).

Esempio:

- Consideriamo la seguente relazione su cui sono definite le seguenti dipendenze funzionali:

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

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_1	c_1	d_2
a_2	b_2	c_1	d_3

Tale istanza è legale su F, poiché soddisfa tutte le dipendenze funzionali in F. Inoltre, è soddisfatta anche la dipendenza funzionale $A \rightarrow C$, che tuttavia non è in F. Dunque, si ha che $A \rightarrow B, B \rightarrow C, A \rightarrow C \in F^+$

Dato uno schema R e un insieme F di dipendenze funzionali definite su R, si ha che:

$$Y \subseteq X \subseteq R \Rightarrow X \rightarrow Y \in F^+$$

Tali dipendenze funzionali vengono dette **banali**, poiché soddisfatte da ogni istanza di R.

Preposizione:

Sia R uno schema e sia F un insieme di dipendenze funzionali definite su R.

Dati X, Y \subseteq R, si ha che:

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

Dimostrazione:

- Siano X, Y \subseteq R, dove Y = {A₁, ..., A_k} .
- Data r una qualsiasi istanza di R, si ha che:

$$\begin{aligned} \forall A_i \in Y, X \rightarrow A \in F^+ &\Leftrightarrow \left\{ \begin{array}{l} \forall t_1, t_2 \in r, t_1[X] = t_2[X] \implies t_1[A_1] = t_2[A_1] \\ \vdots \\ \forall t_1, t_2 \in r, t_1[X] = t_2[X] \implies t_1[A_k] = t_2[A_k] \end{array} \right. \Leftrightarrow \\ &\Leftrightarrow \forall t_1, t_2 \in r, t_1[X] = t_2[X] \implies t_1[\{A_1, \dots, A_k\}] = t_2[\{A_1, \dots, A_k\}] \\ &\Leftrightarrow \forall t_1, t_2 \in r, t_1[X] = t_2[X] \implies t_1[Y] = t_2[Y] \Leftrightarrow X \rightarrow Y \in F^+ \end{aligned}$$

Esempio:

- Consideriamo la seguente istanza di uno schema R = {A,B,C}:

A	B	C
a ₁	b ₁	c ₁
a ₁	b ₁	c ₁
a ₂	b ₂	c ₁

Dato F un insieme di dipendenze funzionali definite su R , notiamo facilmente, che tutte le tuple di tale istanza soddisfano la dipendenza $ABC \rightarrow ABC \in F^+$.

Notiamo inoltre che tutte le tuple di tale istanza in cui A e B sono uguali anche A è uguale, dunque $AB \rightarrow A \in F^+$.

Procedendo analogamente, in definitiva si ha che:

$$\left. \begin{array}{l} ABC \rightarrow ABC, ABC \rightarrow AB, ABC \rightarrow AC, ABC \rightarrow BC, \\ ABC \rightarrow A, ABC \rightarrow B, ABC \rightarrow C, AB \rightarrow AB, AB \rightarrow A, \\ AC \rightarrow A, AC \rightarrow C, BC \rightarrow A, BC \rightarrow C, A \rightarrow A, B \rightarrow B, C \rightarrow C \end{array} \right\} \in F^+$$

▼ Lezione del 25/10

Assiomi di Armstrong



Sia R uno schema e sia F un insieme di dipendenze funzionali definite su R .

Definiamo come F^A l'insieme di tutte le dipendenze funzionali ottenibili partendo da F applicando i seguenti **Assiomi di Armstrong**:

- **Inclusione iniziale** ($F \subseteq F^A$):

$$X \rightarrow \in F \Rightarrow X \rightarrow \in F^A$$

- **Assioma di riflessività**:

$$Y \subseteq X \subseteq R \Rightarrow X \rightarrow Y \in F^A$$

- **Assioma di aumento**:

$$Z \subseteq R, X \rightarrow Y \in F^A \Rightarrow XZ \rightarrow YZ \in F^A$$

- **Assioma di transitività**:

$$X \rightarrow Y \in F^A \wedge Y \rightarrow Z \in F^A \Rightarrow X \rightarrow Z \in F^A$$

Regole secondarie di Armstrong



Dato uno schema R e un insieme F di dipendenze funzionali definite su R, tramite gli assiomi di Armstrong è possibile ricavare le seguenti regole aggiuntive:

-Regola dell'unione:

$$X \rightarrow Y \in F^A \wedge X \rightarrow Z \in F^A \Rightarrow X \rightarrow YZ \in F^A$$

-Regola della decomposizione:

$$Z \subseteq Y, X \rightarrow Y \in F^A \Rightarrow X \rightarrow Z \in F^A$$

-Regola della pseudo-transitività:

$$X \rightarrow Y \in F^A \wedge WY \rightarrow Z \in F^A \Rightarrow WX \rightarrow Z \in F^A$$

Dimostrazione:

• **Regola dell'unione:**

- Siano $X \rightarrow Y, X \rightarrow Z \in F^A$.
- Per assioma di aumento, si ha che:

$$X \subseteq R, X \rightarrow Y \in F^A \implies XX \rightarrow XY = X \rightarrow XY \in F^A$$

- Analogamente, si ha che:

$$Y \subseteq R, X \rightarrow Z \in F^A \implies XY \rightarrow ZY = XY \rightarrow YZ \in F^A$$

- Infine, per assioma di transitività si ha che:

$$X \rightarrow XY \in F^A \wedge XY \rightarrow YZ \in F^A \implies X \rightarrow YZ \in F^A$$

• **Regola della decomposizione:**

- Sia $Z \subseteq Y \subseteq R$ e sia $X \rightarrow Y \in F^A$.
- Per assioma di riflessività, si ha che:

$$Z \subseteq Y \subseteq R \implies Y \rightarrow Z \in F^A$$

- Infine, per assioma di transitività si ha che:

$$X \rightarrow Y \in F^A \wedge Y \rightarrow Z \in F^A \implies X \rightarrow Z \in F^A$$

- **Regola della pseudo-transitività:**

- Sia $X \rightarrow Y, YW \rightarrow Z \in F^A$.
- Per assioma di aumento, si ha che:

$$W \subseteq R, X \rightarrow Y \in F^A \implies XW \rightarrow YW \in F^A$$

- Infine, per assioma di transitività si ha che:

$$XW \rightarrow YW \in F^A \wedge YW \rightarrow Z \in F^A \implies XW \rightarrow Z \in F^A$$

Proposizione

Dato uno schema R e un insieme F di dipendenze funzionali definite su R , si ha che:

$$X \rightarrow Y \in F^A \iff \forall A \in Y, X \rightarrow A \in F^A$$

Dimostrazione:

- Siano $X, Y \subseteq R$, dove $Y = \{A_1, \dots, A_k\}$.
- Per la regola dell'unione, si ha che:

$$\forall A_i \in Y, X \rightarrow A \in F^A \implies X \rightarrow \{A_1, \dots, A_k\} = Y \in F^A$$

- Per la regola della decomposizione, invece si ha che:

$$X \rightarrow Y \in F^A \implies \forall A \in Y, X \rightarrow A \in F^A$$

Chiusura di X

Chiusura di un insieme di attributi



Sia R uno schema e sia F un insieme di dipendenze funzionali definite su R .

Dato $X \subseteq R$, definiamo come **chiusura di X rispetto ad F** , indicata con X_F^+ (o solo con X^+ se F è l'unico insieme di dipendenze su R), il seguente insieme:

$$X_F^+ = \{ A \in R | X \rightarrow A \in F^A \}$$

Dove $A \in R$ implica che A sia un singolo attributo di R

Rappresenta quindi l'insieme degli elementi funzionalmente determinati da X anche dopo aver applicato i teoremi di Armstrong

Lemma 1



Sia R uno schema e sia F un insieme di dipendenze funzionali definite su R .

Dato $X, Y \subseteq R$, si ha che:

$$X \rightarrow Y \in F^A \iff Y \subseteq X^+$$

Dimostrazione:

- Dato $Y = \{A_1, \dots, A_k\}$, si ha che:

$$X \rightarrow Y \in F^A \iff X \rightarrow A \in F^A, \forall A \in Y \iff A \in X^+, \forall A \in Y \iff Y \subseteq X^+$$

Corollario 1



Sia R uno schema e sia F un insieme di dipendenze funzionali definite su R .

Dato $X \subseteq R$, si ha che $X \rightarrow X \in F^A$. Dunque ne segue che:

$$X \rightarrow X \in F^A \iff X \subseteq X^+$$

In altre parole, **ogni insieme di attributi è un elemento della sua chiusura**.

$$F^+ = F^A$$

Teorema 1: $F^+ = F^A$



Dato uno schema R e un insieme F di dipendenze funzionali definite su R , si ha che:

$$F^+ = F^A$$

Dimostrazione:

DIMOSTRAZIONE PER INDUZIONE SU NOTE

Osservazione



Poiché $F^+ = F^A$, per **calcolare** F^+ ci basta applicare gli assiomi di Armstrong sulle dipendenze in F in modo da trovare F^A .

Tuttavia, calcolare $F^+ = F^A$ richiede **tempo esponenziale** quindi $\Omega(2^R)$: considerando anche solo l'assioma di riflessività, siccome ogni possibile sottoinsieme di R genera una dipendenza e siccome i sottoinsiemi possibili di R sono 2^R allora ne segue che $|F^+| >> 2^R$

▼ Lezione del 26/10

Terza Forma Normale (3NF)

A questo punto, possiamo sfruttare la definizione di dipendenza funzionale per dare una definizione più rigorosa di chiave:

Chiave e Primo



Sia R uno schema e sia F un insieme di dipendenze funzionali definite su R .

Definiamo il sottoinsieme di attributi $K \subseteq R$ come **Chiave** di R se:

- $K \rightarrow R \in F^+$
- $\nexists K' \subset K | K' \rightarrow R \in F^+$

Se K è una chiave di R , ogni attributo $A \in K$ viene detto **primo**.

Esempio:

- Consideriamo lo schema $\text{Student}(\text{Matr}, \text{LastName}, \text{FirstName}, \text{BirthD})$
- In questo caso, è ovvio imporre la seguente dipendenza funzionale in F :

$$\text{Matr} \rightarrow \{\text{LastName}, \text{FirstName}, \text{BirthD}\} \in F \subseteq F^+$$

poiché ogni tupla avente matricola uguale deve anche avere informazioni uguali.

- Siccome $\text{Matr} \subseteq \text{Matr}^+ \iff \text{Matr} \rightarrow \text{Matr} \in F^+ = F^A$, per unione si ha che:

$$\text{Matr} \rightarrow \{\text{Matr}, \text{LastName}, \text{FirstName}, \text{BirthD}\} \in F^+ = F^A$$

dunque Matr è superchiave di Student poiché determina tutto il suo schema

- Siccome non esiste alcun sottoinsieme di Matr , allora possiamo concludere che Matr sia chiave di Student

Superchiave



Sia R uno schema e sia F un insieme di dipendenze funzionali definite su R .

Definiamo il sottoinsieme di attributi $K \subseteq R$ come **Superchiave** di R se:

- $K \rightarrow R \in F^+$
- $\exists K' \subseteq K | K' \rightarrow R \in F^+ \wedge \nexists K'' \subset K' | K'' \rightarrow R \in F^+$
ossia contiene una chiave

Osservazione



Sia R uno schema e sia F un insieme di dipendenze funzionali definite su R .

Se $X \subseteq R$ è chiave di R , allora essa è anche superchiave, poiché $\exists X \subseteq X$ tale che X chiave di R

$$X \text{ chiave di } R \Rightarrow X \text{ superchiave di } R$$

Terza Forma Normale (3NF)



Sia R uno schema e sia F un insieme di dipendenze funzionali definite su R .

Lo schema R viene detto in **terza forma normale (3NF)** se:

$$\forall X \rightarrow A \in F^+, A \in R - X, \exists K \subseteq R \text{ chiave} \mid K \subseteq X \vee A \in K$$

In altre parole, uno schema viene detto in terza forma normale se per ogni dipendenza funzionale **non banale** $X \rightarrow A \in F^+$, il determinante di X è superchiave o il determinante di A è primo.

Se uno scema è in 3NF, la quantità di **anomalie** e di **ridondanze** dei dati è **estremamente ridotta**.

Esempio:

1.
 - Sia $R = ABCD$ uno schema e sia $F = \{A \rightarrow B, B \rightarrow CD\}$ un insieme dipendenze funzionali su R

- Applicando gli assiomi di Armstrong, si ha che:

- Per riflessività:

$$A \subseteq A \implies A \rightarrow A \in F^A$$

- Per transitività:

$$A \rightarrow B, B \rightarrow CD \in F^A \implies A \rightarrow CD \in F^A$$

- Per unione:

$$A \rightarrow A, A \rightarrow B, A \rightarrow CD \in F^A \implies A \rightarrow ABCD = R \in F^A$$

- Dunque, siccome $A \rightarrow R \in F^A = F^+$ e siccome A non ha sottoinsiemi, allora A è chiave di R (in particolare, A è l'unica chiave di R)
 - Verifichiamo quindi se R sia in 3NF:
 - $A \rightarrow B \in F^+$ rispetta la definizione di 3NF, poiché il determinante A è chiave (e quindi anche una superchiave di se stessa)
 - $B \rightarrow CD \in F^+$ non è una dipendenza da controllare, poiché CD sono un sottoinsieme di attributi e non un singolo attributo
 - Tuttavia, per decomposizione abbiamo che $B \rightarrow CD \in F^A = F^+ \Rightarrow B \rightarrow C, B \rightarrow D \in F^A = F^+$
 - Per entrambe si ha che B non è superchiave, poiché $A \not\subseteq B$, mentre C e D non sono primi, poiché $C, D \notin A$, dunque concludiamo che R non sia in 3NF
2. • Sia $R = ABCD$ e sia $F = \{AC \rightarrow B, B \rightarrow AD\}$ un insieme di dipendenze funzionali su R .
- Applicando gli assiomi di Armstrong, si ha che:
 - Per riflessività:
$$A, C, AC \subseteq AC \Rightarrow AC \rightarrow A, AC \rightarrow C, AC \rightarrow AC \in F^A$$

$$B, C, BC \subseteq BC \Rightarrow BC \rightarrow B, BC \rightarrow C, BC \rightarrow BC \in F^A$$
 - Per transitività:
$$AC \rightarrow B, B \rightarrow AD \in F^A \Rightarrow AC \rightarrow AD \in F^A$$

$$BC \rightarrow B, B \rightarrow AD \in F^A \Rightarrow BC \rightarrow AD \in F^A$$
 - Per unione:
$$AC \rightarrow C, AC \rightarrow B, AC \rightarrow AD \in F^A \Rightarrow AC \rightarrow ABCD = R \in F^A$$

$$BC \rightarrow B, BC \rightarrow C, BC \rightarrow AD \in F^A \Rightarrow BC \rightarrow ABCD \in F^A$$
 - Per aumento:
$$AC \rightarrow ABCD = R \in F^A \Rightarrow ABC \rightarrow ABCD = ABCD = R \in F^A$$
- Deduciamo quindi che AC e BC siano chiave di R , mentre ABC è una superchiave di R
 - Verifichiamo quindi se R sia in 3NF:
 - $AC \rightarrow B \in F^A = F^+$ rispetta la definizione di 3NF, poiché AC è chiave
 - $B \rightarrow AD \in F^A = F^+$ non va controllato, ma per decomposizione si ha che $B \rightarrow A, B \rightarrow D \in F^A = F^+$

- $B \rightarrow A \in F^+$ rispetta la definizione di 3NF, poiché $A \in AC$ e dunque primo, mentre $B \rightarrow D \in F^+$ non rispetta la definizione di 3NF, poiché né B è superchiave né D è primo, dunque concludiamo che R non sia in 3NF
3. • Sia $R = ABCD$ uno schema e sia $F = \{AB \rightarrow CD, BC \rightarrow A, D \rightarrow AC\}$ un insieme di dipendenze funzionali su R
- In tal caso si ha che AB, BC e BD sono chiavi di R
 - Verifichiamo quindi se R sia in 3NF:
 - $AB \rightarrow CD \in F^+$ rispetta la definizione di 3NF, poiché AB è chiave
 - $BC \rightarrow A \in F^+$ rispetta la definizione di 3NF, poiché BC è chiave e A è primo
 - $D \rightarrow AC \in F^+ \implies D \rightarrow A, D \rightarrow C \in F^+$, i quali rispettano entrambi la definizione di 3NF, poiché A e C sono entrambi primi
 - Dunque, concludiamo che R sia in 3NF

Dipendenza parziale



Sia R uno schema e sia F un insieme di dipendenze funzionali definite su R .

Definiamo $X \rightarrow A \in F^+$, dove $A \notin X$, come **dipendenza parziale** su R se:

- A **non** è primo
- $\exists K \subseteq R$ chiave di R tale che $X \subset K$ (quindi in particolare $X \neq K$)

Dipendenza transitiva



Sia R uno schema e sia F un insieme di dipendenze funzionali definite su R .

Definiamo $X \rightarrow A \in F^+$, dove $A \notin X$, come **dipendenza transitiva** in R se:

- A non è primo
- $\forall K \subseteq R$ chiave di R si ha che $X \not\subset K$ e $K - X \neq 0$

Corollario 2: Definizione alternativa di 3NF



Sia R uno schema e sia F un insieme di dipendenze funzionali definite su R .

Lo schema di R viene detto **in terza forma normale (3NF)** se non esistono dipendenze parziali o transitive su F

$\nexists X \rightarrow Y \in F | X \rightarrow Y$ dipendenza parziale o transitiva.

▼ Lezione del 02/11

A seguito esempi di decomposizione in 3NF

Uno schema che non è in terza forma normale (3NF) può essere decomposto in molteplici modi in un insieme di schemi in 3NF.

Ad esempio, lo schema $R = ABC$ con l'insieme di dipendenze funzionali $F = \{A \rightarrow B, B \rightarrow C\}$ non è in 3NF a causa della presenza nella chiusura transitiva F^+ della dipendenza $B \rightarrow C$ (la chiave è A).

R può essere decomposto in due modi diversi:

1. $R_1 = AB$ con $A \rightarrow B$ e $R_2 = BC$ con $B \rightarrow C$.
2. $R_1 = AB$ con $A \rightarrow B$ e $R_2 = AC$ con $A \rightarrow C$.

Entrambi gli schemi sono in 3NF, ma la seconda soluzione potrebbe non essere soddisfacente, probabilmente a causa di una ridondanza di dati.

Se consideriamo due istanze legittime degli schemi ottenuti:

R1	A	B	R2	A	C
	a1	b1		a1	c1
	a2	b1		a2	c2

L'istanza dello schema R originale che posso ricostruire da questo (l'unico modo è ricostruirla tramite un "natural join") è:

R	A	B	C
	a1	b1	c1
	a2	b1	c2

Ma non è un'istanza legale di R, poiché non soddisfa la dipendenza funzionale $B \rightarrow C$. Vogliamo preservare **TUTTE** le dipendenze in F+.

In conclusione, ci sono altri due requisiti dello schema decomposto che devono essere tenuti a mente quando si suddivide uno schema per ottenere uno in terza forma normale (3NF):

- Dobbiamo preservare le dipendenze funzionali che si applicano a ciascuna istanza legale dello schema originale.
- Dobbiamo consentire di ricostruire, tramite un "natural join," ogni istanza legale dello schema originale (senza aggiungere alcuna informazione aggiuntiva).

Forma Normale di Boyce-Codd



Sia R uno schema e sia F un insieme di dipendenze funzionali definite su R .

Lo schema R viene detto in forma normale di **Boyce-Codd (BCNF)** se:

$$\forall X \rightarrow Y \in F \Rightarrow X \text{ superchiave}$$

Uno schema in BCNF è anche uno schema in 3NF, poiché la BCNF è una versione **più restrittiva della 3NF**.

Non è sempre possibile decomporre uno schema in BCNF in più sottoschemi in BCNF preservando tutte le dipendenze. Tuttavia, ciò non vale per uno schema in 3NF, il quale, invece, è sempre decomponibile in più sottoschemi in 3NF preservando tutte le dipendenze.

Dunque, **si preferisce l'uso della 3NF rispetto alla BCNF**

▼ Lezione del 15/11

Calcolare X^+

Algoritmo 1: Algoritmo per la chiusura di un insieme di attributi



Sia R uno schema e sia F un insieme di dipendenze funzionali definite su R .

Dato un qualsiasi insieme di attributi $X \subseteq R$, è possibile calcolare tutti gli elementi appartenenti a X_F^+ tramite il seguente algoritmo:

```
def closureX(R: schema, F: set of dependencies, X: subset of R):
    Z := X
    S := { A | ∃Y → V ∈ F, A ∈ V ⊆ R, Y ⊆ Z }
    while S ⊈ Z do:
        Z := X ∪ S
        S := { A | ∃Y → V ∈ F, A ∈ V ⊆ R, Y ⊆ Z }
    X+ := Z
    return X+
```

Tale algoritmo viene eseguito in tempo polinomiale, ossia $O(n^k)$

Esempio:

- Dato lo schema $R = ABCDEHL$ e l'insieme di dipendenze funzionali $F = \{AB \rightarrow C, B \rightarrow D, AD \rightarrow E, CE \rightarrow H\}$ definite su R , vogliamo calcolare AB^+ .
- Utilizzando l'algoritmo, ad ogni iterazione si ha che:

1. Inizialmente si ha che $Z := AB$ e $S := CD$, poiché:

- $AB \subseteq AB \wedge AB \rightarrow C \implies C \in S$
- $B \subseteq AB \wedge B \rightarrow D \implies D \in S$

Notiamo come tramite l'algoritmo stiamo implicitamente utilizzando gli assiomi di Armstrong per aggiungere C e D a $Z = AB$:

- $A, B \in AB, AB \rightarrow A, AB \rightarrow B \in F^A$
- $AB \rightarrow C, B \rightarrow D \in F \implies AB \rightarrow C, B \rightarrow D \in F^A$
- $B \subseteq AB \implies AB \rightarrow B \in F^A$
- $AB \rightarrow B, B \rightarrow D \in F^A \implies AB \rightarrow D \in F^A$
- $AB \rightarrow A, AB \rightarrow B, AB \rightarrow C, AB \rightarrow D \in F^A \iff A, B, C, D \in AB^+$

2. Siccome $C, D \in S \wedge C, D \notin Z \implies S \not\subseteq Z$, procediamo ponendo $Z := Z \cup S = ABCD$ e $S := CDE$, poiché:

- $AB \subseteq ABCD \wedge AB \rightarrow C \implies C \in S$
- $B \subseteq ABCD \wedge B \rightarrow D \implies D \in S$
- $AD \subseteq ABCD \wedge AD \rightarrow E \implies E \in S$

Anche in questo caso, stiamo implicitamente utilizzando gli assiomi di Armstrong per aggiungere E a $Z = ABCD$:

- $B \rightarrow D \in F^A \implies AB \rightarrow AD \in F^A$
- $AD \rightarrow E \in F \implies AD \rightarrow E \in F^A$
- $AB \rightarrow AD, AD \rightarrow E \in F^A \implies AB \rightarrow E \in F^A \iff E \in AB^+$

3. Siccome $E \in S \wedge E \notin Z \implies S \not\subseteq Z$, procediamo ponendo $Z := Z \cup S = ABCDE$ e $S := CDEH$, poiché:

- $AB \subseteq ABCDE \wedge AB \rightarrow C \implies C \in S$
- $B \subseteq ABCDE \wedge B \rightarrow D \implies D \in S$
- $AD \subseteq ABCDE \wedge AD \rightarrow E \implies E \in S$
- $CE \subseteq ABCDE \wedge CE \rightarrow H \implies H \in S$

Anche in questo caso, stiamo implicitamente utilizzando gli assiomi di Armstrong per aggiungere H a $Z = ADCDE$:

- $AB \rightarrow C, AB \rightarrow E \in F^A \implies AB \rightarrow CE \in F^A$
- $CE \rightarrow H \in F \implies CE \rightarrow H \in F^A$
- $AB \rightarrow CE, CE \rightarrow H \in F^A \implies AB \rightarrow H \in F^A \iff H \in AB^+$

4. Siccome $H \in S \wedge H \notin Z \implies S \not\subseteq Z$, procediamo ponendo $Z := Z \cup S = ABCDEH$ e $S := CDEH$, poiché:

- $AB \subseteq ABCDEH \wedge AB \rightarrow C \implies C \in S$
- $B \subseteq ABCDEH \wedge B \rightarrow D \implies D \in S$
- $AD \subseteq ABCDEH \wedge AD \rightarrow E \implies E \in S$
- $CE \subseteq ABCDEH \wedge CE \rightarrow H \implies H \in S$

In questo caso, quindi, S rimane inalterato

5. Infine, siccome $S \subseteq Z$, l'output dell'algoritmo sarà $AB^+ = ABCDEH$

Teorema 2: Correttezza dell'algoritmo



Sia R uno schema e sia F un insieme di dipendenze funzionali definite su R .

Dato un qualsiasi insieme di attributi $X \subseteq R$, l'algoritmo $\text{closure}_X(R, F, X)$ restituisce X_F^+

Dimostrazione:

PRESENTA SU APPUNTI IPAD NOTA "DIMOSTRAZIONE ALGORITMO CHIUSURA X+"

Trovare Chiavi di uno Schema

Dato lo schema seguente: $R = ABCDEH$
e il seguente insieme di dipendenze funzionali:
 $F = \{ AB \rightarrow CD, C \rightarrow E, AB \rightarrow E, ABC \rightarrow D \}$

calcola la chiusura di ABH

applicando l'algoritmo, iniziamo con $Z=ABH$ e nella prima iterazione del ciclo while aggiungiamo gli attributi C, D ed E a Z,

grazie alle dipendenze funzionali in cui la parte sinistra è

contenuta in ABH, ovvero $AB \rightarrow CD$ e $AB \rightarrow E$

effettivamente, a questo punto potremmo fermarci, abbiamo già aggiunto tutti gli attributi dello schema, cioè abbiamo verificato che $(ABH)^+ = \{A, B, C, D, E, H\}$

$ABH^+ = ABCDEH = R$

è una chiave?

Definizione di Chiave



Dato uno schema di relazione R e un insieme F di dipendenze funzionali, un sottoinsieme K di uno schema di relazione R è una chiave di R se:

1. $K \rightarrow R \in F^+$

2. non esiste un sottoinsieme proprio K' di K tale che $K' \rightarrow R \in F^+$

ATTENZIONE: è sempre necessario verificare che nessun sottoinsieme di K determini

funzionalmente R:

· utilizziamo ancora l'algoritmo sui sottoinsiemi di K!

$R = ABCDEH$, $F = \{ AB \rightarrow CD, C \rightarrow E, AB \rightarrow E, ABC \rightarrow D \}$

· calcoliamo la chiusura dei sottoinsiemi di ABH:

- **osservazione 1:** è meglio iniziare da quelli con cardinalità più alta... se la loro chiusura non contiene R, non è necessario controllare i loro sottoinsiemi
- **osservazione 2:** gli attributi che non compaiono mai a destra delle dipendenze funzionali di F non sono determinati funzionalmente da nessun altro attributo, quindi devono essere presenti in tutte le chiavi dello schema

- **osservazione 3:** un approccio brute force (provare TUTTI i sottoinsiemi comunque) non è sbagliato ma può essere inefficiente
- **osservazione 4:** negli esercizi, ogni shortcut nei calcoli deve essere giustificato

Riprendendo l'esempio precedente:

$$R = ABCDEH, F = \{ AB \rightarrow CD, C \rightarrow E, AB \rightarrow E, ABC \rightarrow D \}$$

Noto che:

- H non è determinato da altri attributi, deve far parte di ogni chiave
- i sottoinsiemi di cardinalità 2 da verificare sono AH e BH



Dato uno schema R e dato un insieme F di dipendenze funzionali definite su R, si ha

che:

$$\nexists X \rightarrow Y \in F | A \in Y \Rightarrow A \in K \subseteq R | K \text{ chiave di } R$$

In altre parole, se A non è determinato da nessuna dipendenza funzionale in F (non è alla destra di nessuna freccia) allora A apparterrà ad ogni chiave di R.

Dato uno schema R e un insieme di dipendenze funzionali su R, possono **esistere diverse chiavi** di R.



Dato uno schema R e dato un insieme F di dipendenze funzionali definite su R, si ha che:

$$X \subseteq R \text{ superchiave di } R \iff X^+ = R$$

Dato uno schema R e dato un insieme F di dipendenze funzionali definite su R, si ha che:

$$X \subseteq R \text{ chiave di } R \iff X^+ = R \wedge \nexists Y \subset X | Y^+ = R$$

Esempio:

- Dato lo schema $R = ABCDEGH$ e il seguente insieme di dipendenze funzionali $F = \{AB \rightarrow D, G \rightarrow A, G \rightarrow B, H \rightarrow E, H \rightarrow G, D \rightarrow H\}$
- Siccome C non è determinato da alcuna dipendenza, allora esso sarà in ogni chiave di R. Tuttavia, si ha che $C^+ = \emptyset \neq R$, dunque C non è chiave di R
- Applichiamo quindi l'algoritmo per calcolare le chiusure di ogni insieme di attributi costituiti da C e da un determinato di una dipendenza funzionale in F:
 - $ABC^+ = R$
 - $GC^+ = R$
 - $DC^+ = R$
 - $HC^+ = R$
- Siccome gli unici sottoinsiemi di GC, DC, HC contenenti anche C sono loro stessi, allora tutti e tre sono chiavi di R
- Quanto ad ABC, è necessario applicare l'algoritmo sui sottoinsiemi AC e BC, ottenendo che $AC^+ = AC$ e che $BC^+ = BC$, implicando quindi che ABC sia chiave di R

Teorema: Test dell'Unicità



Sia R uno schema e sia F un insieme di dipendenze funzionali definite su R .

Posto:

$$X := \bigcap_{V \rightarrow W \in F} R - (W - V)$$

Si ha che:

- $X^+ = R \Rightarrow X$ è l'unica chiave di R
- $X^+ \neq R \Rightarrow$ esistono più chiavi in R e X non è superchiave di R

Esempi:

1.
 - Dato lo schema $R = ABCDEH$ e l'insieme di dipendenze funzionali $F = \{AB \rightarrow CD, C \rightarrow E, AB \rightarrow E, ABC \rightarrow D\}$, vogliamo determinare se R sia in 3NF
 - Utilizziamo il test dell'unicità determinare la quantità di chiavi in R :
 - Siccome $AB \rightarrow CD \in F$, allora consideriamo l'insieme di attributi $R - (CD - AB) = R - CD + AB = ABEH$
 - Siccome $C \rightarrow E \in F$, allora consideriamo l'insieme di attributi $R - (E - C) = R - E + C = ABCDH$
 - Siccome $AB \rightarrow E \in F$, allora consideriamo l'insieme di attributi $R - (E - AB) = R - E + AB = ABCDH$
 - Siccome $ABC \rightarrow D \in F$, allora consideriamo l'insieme di attributi $R - (D - ABC) = R - D + ABC = ABCEH$
 - A questo punto, consideriamo l'intersezione degli insiemi di attributi determinati:

$$\bigcap_{V \rightarrow W \in F} R - (W - V) = ABEH \cap ABCDH \cap ABCDH \cap ABCEH = ABH$$

– Siccome $ABH^+ = R$, allora ABH è l'unica chiave di R

- Per verificare che R sia in 3NF, ci basta vedere che:

$$AB \rightarrow CD \in F \implies AB \rightarrow CD \in F^A \implies AB \rightarrow C, AB \rightarrow D \in F^A = F^+$$

- Siccome $AB \rightarrow C, AB \rightarrow D \in F^+$ sono dipendenze parziali, allora R non è in 3NF

2. • Dato lo schema $R = ABCDEGH$ e l'insieme di dipendenze funzionali $F = \{AB \rightarrow CD, EH \rightarrow D, D \rightarrow H\}$, vogliamo determinare se R sia in 3NF
• Utilizziamo il test dell'unicità determinare la quantità di chiavi in R :

$$\bigcap_{V \rightarrow W \in F} R - (W - V) = ABEGH \cap ABCEGH \cap ABCDEG = ABEG$$

- Siccome $ABEG^+ = R$, allora $ABEG$ è l'unica chiave di R , implicando che R non sia in 3NF (basta considerare la dipendenza $AB \rightarrow CD \in F$)

3. • Dato lo schema $R = ABCDE$ e l'insieme di dipendenze funzionali $F = \{AB \rightarrow C, AC \rightarrow B, B \rightarrow E\}$, vogliamo determinare se R sia in 3NF
• Utilizziamo il test dell'unicità determinare la quantità di chiavi in R :

$$\bigcap_{V \rightarrow W \in F} R - (W - V) = ABDE \cap ACDE \cap ABCD = AD$$

- Siccome $AD^+ = AD \neq R$, allora esistono più chiavi in R e AD non è superchiave di R .
- Siccome A e D non sono determinati da alcuna dipendenza in F , allora sappiamo che essi devono appartenere ad ogni chiave di R .
- Osservando i determinanti delle dipendenze in F , notiamo che aggiungendo B all'insieme di attributi AD potremmo raggiungere anche C ed E tramite l'algoritmo conosciuto. Difatti, si ha che $ABD^+ = R$, implicando che ABD sia chiave di R , poiché l'unico sottoinsieme di ABD contenente anche AD è AD stesso, il quale sappiamo non essere superchiave.
- Analogamente, osserviamo che aggiungendo C all'insieme di attributi AD potremmo raggiungere B ed E . Difatti, si ha che $ACD^+ = R$, implicando che anche ACD sia chiave di R
- Siccome $B \rightarrow E \in F$ è una dipendenza parziale, allora R non è in 3NF

Non è in 3NF poiché in $AB \rightarrow C$ AB non è chiave e C non è primo (non contenuto in una chiave)

Decomposizione di uno schema



Sia R uno schema. Definiamo come decomposizione di R l'insieme di sottoschemi $\varphi = R_1, \dots, R_k$ che ricoprono R , ossia tali che:

$$R = \bigcup_{i=0}^k R_i$$

In altre parole, R_1, \dots, R_k sono un insieme di schemi tramite cui è possibile ricostruire R effettuando un join naturale tra essi

Decomporre uno schema R in più sottoschemi R_1, \dots, R_k risulta utile nel caso in cui:

- R non sia in 3NF, poiché è più probabile che i suoi sottoschemi siano in 3NF
- Si vuole ottenere un'efficienza maggiore, poiché in alcuni casi potrebbe essere necessaria solo una parte dell'informazione totale, rendendo quindi necessario effettuare una query solo tra alcuni sottoschemi invece che su tutto R . Inoltre, essendo le tuple dei sottoschemi più piccole rispetto a quelle di R , possiamo caricarne di più in memoria.

Buona decomposizione di uno schema

Sia R uno schema con decomposizione $\rightarrow = R_1, \dots, R_k$ e sia F un insieme di dipendenze funzionali su R .

Definiamo φ come una buona decomposizione di R se:

- Ogni sottoschema $R_1, \dots, R_k \in \varphi$ è in Terza Forma Normale
- φ permette di mantenere soddisfatta ogni dipendenza in F per ogni istanza legale r di R ricostruita attraverso un join naturale tra tutte le istanze r_1, \dots, r_k rispettivamente di R_1, \dots, R_k (preservazione di F)
- φ permette di ricostruire senza perdita di informazioni le tuple di ogni istanza legale r di R ricostruita attraverso un join naturale tra tutte le istanze r_1, \dots, r_k rispettivamente di R_1, \dots, R_k (join senza perdita)

Esempio:

- Consideriamo lo schema $R = ABC$ e l'insieme di dipendenze funzionali $F = \{A \rightarrow B, B \rightarrow C\}$.
 - Notiamo facilmente che A è l'unica chiave di R , implicando che R non sia in 3NF poiché $B \rightarrow C$ non rispetta la definizione di 3NF.
 - Proviamo quindi a decomporre R in $\rho = \{AB, AC\}$, dove $F_1 = \{A \rightarrow B\}$ è l'insieme di dipendenze di $R_1 := AB$ mentre $F_2 = \{A \rightarrow C\}$ è l'insieme di dipendenze di $R_2 := AC$
 - Entrambi i sottoschemi risultano essere in 3NF, tuttavia la decomposizione risulta essere incorretta:
 - Consideriamo due istanze legali dei due sottoschemi ottenuti:

R_1		R_2	
A	B	A	C
a_1	b_1	a_1	c_1
a_2	b_1	a_2	c_2

- Effettuando il join naturale tra R_1 ed R_2 , otteniamo che:

$R = R_1 \bowtie R_2$		
A	B	C
a_1	b_1	c_1
a_2	b_1	c_2

- Considerando l'insieme iniziale di dipendenze funzionali $F = \{A \rightarrow B, B \rightarrow C\}$, notiamo come $B \rightarrow C$ non **preservata dalla decomposizione**, ossia non sia più soddisfatta da tale istanza di R , rendendola quindi illegale.
- Dunque, tale decomposizione non risulta essere una buona decomposizione

Dato uno schema R con decomposizione $\rightarrow = R_1, \dots, R_k$ ed istanza r , ogni istanza r_1, \dots, r_k rispettivamente di R_1, \dots, R_k corrisponde ad una proiezione di r sugli attributi di R_i :

$$r_i = \pi_{R_i}(r)$$

dove $i \in [1, k]$.

Di conseguenza, le singole proiezioni hanno l'effetto di eliminare i duplicati che potrebbero essere generati da due tuple distinte aventi una porzione comune che

ricade nello stesso sottoschema, riducendo la memoria necessaria a conservare le informazioni.

Preservazione di F

Equivalenza tra insiemi di dipendenze



Sia R uno schema e siano F e G due insiemi di dipendenze funzionali su R. Tali insiemi vengono detti equivalenti, indicato come $F \equiv G$, se $F^+ = G^+$

Inclusione delle chiusure



Dato uno schema R e due insiemi F e G di dipendenze funzionali su R, si ha che:

$$F \subseteq G^+ \iff F^+ \subseteq G^+$$

Dimostrazione su appunti

Preservazione di F



Sia R uno schema con decomposizione $\varphi = R_1, \dots, R_k$ e sia F un insieme di dipendenze funzionali su R .

Si ha che φ preserva F se:

$$F \equiv G := \bigcup_{i=0}^k \pi_{R_i}(F)$$

dove $\pi_{R_i}(F) = \{X \rightarrow Y \in F^+ | XY \subseteq R\}$ viene detta **proiezione** di F su R_i , ossia l'insieme di tutte le dipendenze in F^+ tali che il determinante e il determinato sono entrambi sottoinsiemi di R_i

Dato uno schema R con decomposizione $\varphi = R_1, \dots, R_k$, dato un insieme F di dipendenze

funzionali su R e posto:

$$G := \bigcup_{i=0}^k \pi_{R_i}(F)$$

si ha che φ preserva F se $F \subseteq G^+$, poiché:

$$F \subseteq G^+ \Rightarrow F \equiv G$$

Algoritmo 2: Verifica di $F_1 \subseteq F_2^+$



Dato uno schema R e dati due insiemi F_1 e F_2 di dipendenze funzionali su R , il seguente algoritmo determina se $F_1 \subseteq F_2^+$:

```
def F1_in_F2 (R: schema, F1: set of dep., F2: set of dep.):
    for X → Y ∈ F1 :
        if Y ⊈ XF2+ :
            return False
    return True
```

Per applicare tale algoritmo durante la verifica della preservazione di F , è necessario prima calcolare F^+ , in modo da poter calcolare G e successivamente ogni X_G^+ richiesto per verificare che $F \subseteq G^+$. Tuttavia, siccome il calcolo di F^+ richiede tempo esponenziale, allora è necessario calcolare i vari X_G^+ tramite un metodo alternativo.

Algoritmo 3: Calcolo di X_G^+ tramite F



Dato uno schema R con decomposizione $\rightarrow = R_1, \dots, R_k$, dato un insieme F di dipendenze funzionali su R e posto:

$$G := \bigcup_{i=0}^k \pi_{R_i}(F)$$

preso $X \subseteq R$, il seguente algoritmo calcola X_G^+ tramite F:

```
def X_G^+ _with_F(R: schema, F: set of dependencies, X: set of attributes):
    Z := X
    S := ∅
    for i in range(1, k):
        S ∪ ((Z ∩ R_i)^+_F ∩ R_i)
        while S ⊈ Z:
            Z := Z ∪ S
            for i in range(1, k):
                S := S ∪ ((Z ∩ R_i)^+_F ∩ R_i)
    X_G^+ = Z
    return X_G^+
```

Esempi:

1.
 - Dato lo schema $R = ABC$ e l'insieme di dipendenze funzionali $F = \{A \rightarrow B, B \rightarrow C\}$, vogliamo vedere se la decomposizione $\rho = \{AB, AC\}$ preserva F .
 - Per il corollario precedentemente visto, sappiamo che è sufficiente utilizzare l'algoritmo in grado di verificare se $F \subseteq G^+$, richiamante a sua volta l'algoritmo del calcolo di X_G^+ tramite F
 - Siccome $A \rightarrow B \in \pi_{AB}(F) \subseteq G \subseteq G^+$, non è necessario verificare tramite l'algoritmo se tale dipendenza venga preservata.
 - Dunque, l'unica dipendenza da verificare è $B \rightarrow C$. Calcoliamo quindi B_G^+ :
 - Inizializzando l'algoritmo, dunque ponendo $Z := B$ otteniamo che:
 - * $S_1 = S_0 \cup ((B \cap AB)_F^+ \cap AB) = \emptyset \cup (B_F^+ \cap AB) = \emptyset \cup (BC \cap AB) = B$
 - * $S_2 = S_1 \cup ((B \cap AC)_F^+ \cap AC) = B \cup ((\emptyset)_F^+ \cap AC) = B$
 - Siccome $S_2 = B \subseteq Z = B$, allora l'algoritmo termina con $B_G^+ = Z = B$, implicando a sua volta che $C \notin B_G^+ \iff B \rightarrow C \notin G^A = G^+$
 - Dunque, siccome tutte non tutte le dipendenze di F sono in G^+ , l'algoritmo terminerà concludendo che $F \not\subseteq G^+$, implicando che $F \not\equiv G$ e quindi che F non venga preservato

2.
 - Dato lo schema $R = ABCD$ e l'insieme di dipendenze funzionali $F = \{AB \rightarrow C, D \rightarrow C, D \rightarrow B, D \rightarrow A, C \rightarrow B\}$, vogliamo vedere se la decomposizione $\rho = \{ABC, ABD\}$ preserva F .
 - Siccome $AB \rightarrow C, C \rightarrow B \in \pi_{ABC}(F) \subseteq G \subseteq G^+$, non è necessario verificare tramite l'algoritmo se tali dipendenze vengano preservate.

Analogamente, siccome $D \rightarrow B, D \rightarrow A \in \pi_{ABD}(F) \subseteq G \subseteq G^+$, non è necessario verificare se tali dipendenze vengano preservate.

 - Dunque, l'unica dipendenza da verificare è $D \rightarrow C$. Calcoliamo quindi D_G^+ :
 - Inizializzando l'algoritmo, dunque ponendo $Z := D$ otteniamo che:
 - * $S_1 = S_0 \cup ((D \cap ABC)_F^+ \cap ABC) = \emptyset \cup ((\emptyset)_F^+ \cap ABC) = \emptyset$
 - * $S_2 = S_1 \cup ((D \cap ABD)_F^+ \cap ABD) = \emptyset \cup (D_F^+ \cap ABC) = \emptyset \cup (ABCD \cap ABD) = ABD$
 - Siccome $S_2 = ABD \not\subseteq D$, allora entriamo nel ciclo while ponendo $Z := Z \cup S_2 = ABD$ e ripetendo il procedimento:
 - * $S_3 = S_2 \cup ((ABD \cap ABC)_F^+ \cap ABC) = ABD \cup (AB_F^+ \cap ABC) = ABD \cup (ABC \cap ABC) = ABCD$
 - * $S_4 = S_3 \cup ((ABD \cap ABD)_F^+ \cap ABD) = ABCD \cup (ABD_F^+ \cap ABD) = ABCD \cup (ABCD \cap ABD) = ABCD$

- Siccome $S_4 = ABCD \not\subseteq ABD$, allora poniamo $Z := Z \cup S_4 = ABCD$ e ripetiamo il procedimento:
 - * $S_5 = S_4 \cup ((ABCD \cap ABC)_F^+ \cap ABC) = ABCD \cup (ABC_F^+ \cap ABC) = ABD \cup (ABC \cap ABC) = ABCD$
 - * $S_6 = S_5 \cup ((ABCD \cap ABD)_F^+ \cap ABD) = ABCD \cup (ABD_F^+ \cap ABD) = ABCD \cup (ABCD \cap ABD) = ABCD$
- Siccome $S_6 = ABCD \subseteq ABCD = Z$, allora l'algoritmo termina con $D_G^+ = Z = ABCD$, da cui si ha che $D_G^+ = ABCD \implies D \rightarrow C \in G^+$, dunque si ha che $F \subseteq G^+ \implies F \equiv G$ e quindi che ρ preservi F

Terema: Correttezza dell'algoritmo X_G^+ _with_F



Sia R uno schema con decomposizione $\rightarrow = R_1, \dots, R_k$ e sia F un insieme di dipendenze funzionali definite su R .

Posto:

$$G := \bigcup_{i=0}^k \pi_{R_i}(F)$$

e dato un qualsiasi insieme di attributi $X \subseteq R$, l'algoritmo X_G^+ _with_F(R, F, X) restituisce X^+

Dimostrazione negli appunti

▼ Lezione del 22/11

Join Senza Perdita



Sia R uno schema con decomposizione $\rho = R_1, \dots, R_k$ e sia F un insieme di dipendenze funzionali su R . La decomposizione ρ presenta un join senza perdita se per ogni istanza legale r di R si ha che:

$$r = m_\rho(r) := \rho_{R1}(r) \bowtie \dots \bowtie \pi_{Rk}(r)$$

Sia R uno schema con decomposizione $\rho = R_1, \dots, R_k$ e sia F un insieme di dipendenze funzionali su R .

Posto $m_\varphi(r) := \pi_{R1}(r) \bowtie \dots \bowtie \pi_{Rk}(r)$, per ogni istanza legale r di R si ha che:

1. $r \subseteq m_\varphi(r)$
2. $\pi_{Ri}(m_\varphi(r)) = \pi_{Ri}(r) \forall R_i \in \varphi$
3. $m_\varphi(m_\varphi(r)) = m_\varphi(r)$

Dimostrazioni:

Su appunti

Algoritmo 4: Controllo presenza del join senza perdita



Dato uno schema $R = A_1, \dots, A_n$ con decomposizione $\rho = R_1, \dots, R_k$ e un insieme F di dipendenze funzionali su R , presa l'istanza legale di r di R dove

$\forall i \in [1, k], \forall j \in [1, n]$ si ha:

$$r_{i,j} = \begin{cases} "a" & \text{se } A_j \in R_i \\ "b_i" & \text{se } A_j \notin R_i \end{cases}$$

il seguente algoritmo determina se φ presenta un join senza perdita.

```
def has_lossless_join(R: schema, F: set of dep., ϕ: decomposition):
    unchanged := False
    while not unchanged:
        unchanged := True
        for X → Y ∈ F :
            for t1 in r:
                for t2 in r:
                    if t1[X] == t2[X] && t1[Y] != t2[Y]:
                        unchanged = False
                    for Aj ∈ Y :
                        if t1[Aj] == "a":
                            t2[Aj] := t1[Aj]
                        else:
                            t1[Aj] := t2[Aj]
        if ∃t ∈ r | t = ("a", . . . , "a"):
            return True
    else: return False
```

Commenti sull'algoritmo:

- L'algoritmo modifica l'istanza di partenza r in modo che tutte le dipendenze di F vengano soddisfatte
- Ogni volta che l'algoritmo trova due tuple aventi stesso valore nel determinante ma valori differenti, quest'ultimo viene modificato, in modo che essi siano uguali

- Nel fare ciò, il simbolo "a" viene considerato prioritario (difatti, "a" non può mai diventare "b", mentre "b" può diventare "a")
- Se due tuple hanno stesso valore nel determinante ma valori differenti nel determinato e se solo una delle due tuple ha un valore "a" nel determinato, viene cambiato il valore "b" dell'altra tupla in una "a"
- Se due tuple hanno stesso valore nel determinante ma valori differenti nel determinato ma nessuna delle due tuple ha un valore "a" nel determinato, viene cambiato il valore "b" di una delle due tuple in modo che esse abbiano lo stesso valore "b"
- Due valori vengono considerati uguali se sono entrambi una "a" (indipendentemente dal pedice che hanno) o se entrambi hanno una "b" con lo stesso identico pedice
- L'algoritmo termina quando tutte le coppie di tuple soddisfano le dipendenze di F
- Infine, quindi, r diventa un'istanza legale di R
- Una volta terminato l'algoritmo, se esiste almeno una tupla avente tutti valori "a" al suo interno, allora φ presenta un join senza perdita, altrimenti no

Esempio:

- Dato lo schema $R = ABCDE$ con decomposizione $\rho = \{AC, ADE, CDE, AD, B\}$ e con insieme di dipendenze $F = \{C \rightarrow D, AB \rightarrow E, D \rightarrow B\}$, vogliamo verificare se ρ presenti un join senza perdita
- Partiamo costruendo l'istanza di r secondo le regole date:

	A	B	C	D	E
AC	<i>a</i>	b_1	<i>a</i>	b_1	b_1
ADE	<i>a</i>	b_2	b_2	<i>a</i>	<i>a</i>
CDE	b_3	b_3	<i>a</i>	<i>a</i>	<i>a</i>
AD	<i>a</i>	b_4	b_4	<i>a</i>	$b_{4,5}$
B	b_5	<i>a</i>	b_5	b_5	b_5

- Effettuiamo quindi la prima iterazione del ciclo:
 - Considerando $C \rightarrow D \in F$, notiamo che la prima e la terza tupla sono uguali nel determinante C , dunque modifichiamo $b_1 \rightarrow a$ affinché esse siano uguali anche nel determinante D
 - Considerando $AB \rightarrow E \in F$, notiamo che tale dipendenza è già soddisfatta
 - Considerando $D \rightarrow B \in F$, notiamo che la prima (poiché abbiamo già modificato $b_1 \rightarrow a$), la seconda, la terza e la quarta tupla sono uguali nel determinante D , dunque modifichiamo $b_2 \rightarrow b_1, b_3 \rightarrow b_1$ e $b_4 \rightarrow b_1$ affinché esse siano uguali anche nel determinante B

	A	B	C	D	E
AC	<i>a</i>	b_1	<i>a</i>	$b_1 \rightarrow a$	b_1
ADE	<i>a</i>	$b_2 \rightarrow b_1$	b_2	<i>a</i>	<i>a</i>
CDE	b_3	$b_3 \rightarrow b_1$	<i>a</i>	<i>a</i>	<i>a</i>
AD	<i>a</i>	$b_4 \rightarrow b_1$	b_4	<i>a</i>	b_4
B	b_5	<i>a</i>	b_5	b_5	b_5

- Siccome la tabella è stata modificata, allora effettuiamo un'altra iterazione del ciclo:
 - Considerando $C \rightarrow D \in F$, notiamo che tale dipendenza è già soddisfatta
 - Considerando $AB \rightarrow E \in F$, notiamo la prima, la seconda e la terza tupla sono uguali nel determinante AB , dunque modifichiamo $b_1 \rightarrow a$ e $b_4 \rightarrow a$ in modo che siano uguali nel determinato E
 - Considerando $D \rightarrow B \in F$, notiamo che tale dipendenza è già soddisfatta

	A	B	C	D	E
AC	a	b_1	a	a	$b_1 \rightarrow a$
ADE	a	b_1	b_2	a	a
CDE	b_3	b_1	a	a	a
AD	a	b_1	b_4	a	$b_4 \rightarrow a$
B	b_5	a	b_5	b_5	b_5

- Siccome la tabella è stata modificata, allora effettuiamo un'altra iterazione del ciclo:
 - Considerando $C \rightarrow D \in F$, notiamo che tale dipendenza è già soddisfatta
 - Considerando $AB \rightarrow E \in F$, notiamo che tale dipendenza è già soddisfatta
 - Considerando $D \rightarrow B \in F$, notiamo che tale dipendenza è già soddisfatta

	A	B	C	D	E
AC	a	b_1	a	a	a
ADE	a	b_1	b_2	a	a
CDE	b_3	b_1	a	a	a
AD	a	b_1	b_4	a	a
B	b_5	a	b_5	b_5	b_5

- Siccome la tabella non è stata modificata, allora l'algoritmo termina stabilendo che ρ non presenta un join senza perdita, poiché non esiste alcuna riga contenente tutti valori "a"

Correttezza algoritmo has_lossless_join



Sia R uno schema con decomposizione $\rho = R_1, \dots, R_k$ e sia F un insieme di dipendenze funzionali definite su R .
 L'algoritmo $\text{has_lossless_join}(R, F, \rho)$ determina che φ presenta un join senza perdita se e solo se esiste una tupla in r contente tutte "a" una volta terminato

Dimostrazione:

Su appunti

▼ Lezione del 23/11

Copertura Minimale



Sia R uno schema con decomposizione $\rho = R_1, \dots, R_k$ e sia F un insieme di dipendenze funzionali definite su R .

Definiamo come **copertura minimale** di F un insieme di dipendenze G tale che:

- $G \equiv F$
- $\forall X \rightarrow A \in G$ si ha che $A \in G$, ossia il determinato di ogni dipendenza è un attributo
- $\forall X \rightarrow A \in G, \exists X' \subset X | G \equiv (G - \{X \rightarrow A\}) \cup \{X' \rightarrow A\}$, ossia non è possibile determinare funzionalmente A tramite un sottoinsieme proprio di X (rimozione determinanti ridondanti)
- $\exists X \rightarrow A \in G | G \equiv G - \{X \rightarrow A\}$, ossia non è possibile determinare funzionalmente A tramite altre dipendenze (rimozione dipendenze ridondanti)

Algoritmo 5: Calcolare una copertura minimale



Sia R uno schema con decomposizione $\rho = R_1, \dots, R_k$ e sia F un insieme di dipendenze funzionali definite su R .

Per calcolare la copertura minima di F sono sufficienti i seguenti tre step:

1. Usando la regola della decomposizione, tutte le dipendenze vengono ridotte ad avere un singolo attributo come determinante
2. Ogni dipendenza funzionale $X \rightarrow A \in F$ dove $\exists X' \subset X | X \rightarrow A \in F$ tale che $G \equiv (G - \{X \rightarrow A\}) \cup \{X' \rightarrow A\}$ viene rimpiazzata direttamente con $X' \rightarrow A$ (o scartata se $X' \rightarrow A$ è già in F), ripetendo tale processo finché possibile.
3. Ogni dipendenza $X \rightarrow A$ tale $G \equiv (G - \{X \rightarrow A\})$ viene scartata

Osservazioni

Durante lo **step 2**, chiamiamo F l'insieme di dipendenze originale, dunque contenente $X \rightarrow A$, mentre chiamiamo F_r l'insieme ridotto, dunque contenente $X' \rightarrow A$.

Siccome gli insiemi differiscono di una sola dipendenza, è sufficiente verificare che $X \rightarrow A \in F_r^+$ e che $X' \rightarrow A \in F^+$ affinché $F \equiv F_r$

Tuttavia, non è necessario verificare se $X \rightarrow A \in F_r^+$, poiché:

$$X' \subset X \Rightarrow X \rightarrow X' \in F_r^A = F_r^+$$

e conseguentemente che:

$$X \rightarrow X', X' \rightarrow A \in F_r^A = F_r^+ \Rightarrow X \rightarrow A \in F_r^A = F_r^+$$

Dunque, affinché $F \equiv F_r$ è **sufficiente verificare** che $X' \rightarrow A \in F^+$

Durante lo **step 2**, denotiamo come F l'insieme di dipendenze originale e come F_k l'insieme di dipendenze ridotto dopo aver effettuato k riduzioni.

Siccome affinché F_1, \dots, F_k siano riduzioni valide è necessario che $F_1 \equiv$

$F, \dots, F_k \equiv F$, allora è sufficiente verificare che $F_k \equiv F_{k-1}$ affinché F_k sia una riduzione valida.

Inoltre, è necessario sottolineare che $X_{F_{k-1}}^+ = X_{F_k}^+$ dunque **non è necessario ricalcolare le chiusure ad ogni riduzione**

Durante lo **step 3**), denotiamo come F l'insieme contenente tutte le dipendenze $X \rightarrow A$ da scartare e F_r l'insieme in cui esse sono scartate.

Siccome $F_r \subseteq F \subseteq F_r^+ \iff F_r^+ \subseteq F^+$, è sufficiente verificare che ogni dipendenza $X \rightarrow A \in F$ scartata sia ancora in F_r^+ dunque che $X \rightarrow A \in F_r^+ \iff A \in X_{F_r}^+$ per affermare che $F \subseteq F_r^+ \iff F^+ \subseteq F_r^+$ e quindi conseguentemente che $F \equiv F_r$.

Inoltre, ogni dipendenza $X \rightarrow A \in F$ tale che $\exists Y \neq X \subseteq R | Y \neq A \in F$ ovviamente risulta non essere ridondante, poiché altrimenti A non sarebbe determinato più da alcuna dipendenza, dunque **non è necessario** effettuare lo step 3) su tali dipendenze.

Durante lo **step 3**), denotiamo come F l'insieme di dipendenze originale e come F_k l'insieme di dipendenze ridotto dopo aver effettuato k riduzioni.

Siccome affinché F_1, \dots, F_k siano riduzioni valide è necessario che $F_1 \equiv F, \dots, F_k \equiv F$,

allora è sufficiente verificare che $F_k \equiv F_{k-1}$ affinché F_k sia una riduzione valida.

Tuttavia, è necessario sottolineare che in tal caso $X_{F_{k-1}}^+ \neq X_{F_k}^+$, poiché scartando le dipendenze cambia il comportamento dell'algoritmo per il calcolo di X^+ , dunque ad ogni riduzione **è necessario ricalcolare le chiusure ad ogni riduzione**.

Esempi:

- Vogliamo trovare la copertura minima del seguente insieme di dipendenze funzionali $F = \{AB \rightarrow CD, C \rightarrow E, AB \rightarrow E, ABC \rightarrow D\}$
 - Prima di tutto, scomponiamo le dipendenze con la regola della decomposizione, ottenendo che:

$$F = \{AB \rightarrow C, AB \rightarrow D, C \rightarrow E, AB \rightarrow E, ABC \rightarrow D\}$$

- Consideriamo quindi la dipendenza $AB \rightarrow C \in F$ e verifichiamo se $A \rightarrow C, B \rightarrow C \in F^+$, dunque se $C \in A_F^+$ e se $C \in B_F^+$:
 - $A_F^+ = A \implies C \notin A_F^+$
 - $B_F^+ = B \implies C \notin B_F^+$
 dunque, non possiamo rimpiazzare la dipendenza $AB \rightarrow C$
- Procedendo analogamente, verifichiamo che $A \rightarrow E, B \rightarrow E, A \rightarrow D, B \rightarrow D \notin F^+$, dunque ne traiamo che $AB \rightarrow E, AB \rightarrow D \in F$ non possano essere rimpiazzate
- Infine, considerando $ABC \rightarrow D \in F$, sappiamo già che $AB \rightarrow D \in F \subseteq F^+$, dunque tale dipendenza può essere rimpiazzata data $AB \rightarrow D$ (e di conseguenza rimossa, poiché già presente in F)

- Al termine dello step 2), quindi, abbiamo che

$$F = \{AB \rightarrow C, AB \rightarrow D, C \rightarrow E, AB \rightarrow E\}$$

- Applichiamo quindi lo step 3)

- C e D sono determinati rispettivamente solo da $AB \rightarrow C$ e $AB \rightarrow D$, dunque tali dipendenze non possono essere rimosse
- Considerando $C \rightarrow E \in F$, verifichiamo se $E \in C_{F-\{C \rightarrow E\}}^+$ affinché $C \rightarrow E \in F - \{C \rightarrow E\}^+$. Siccome $E \notin C_{F-\{C \rightarrow E\}}^+ = C$, ne segue che $C \rightarrow E$ non possa essere scartata
- Considerando $AB \rightarrow E \in F$, verifichiamo se $E \in AB_{F-\{AB \rightarrow E\}}^+$ affinché $AB \rightarrow E \in F - \{AB \rightarrow E\}^+$. Siccome $E \in AB_{F-\{AB \rightarrow E\}}^+ = ABCDE$, ne segue che $AB \rightarrow E$ possa essere scartata

- Infine, otteniamo che $F = \{AB \rightarrow C, AB \rightarrow D, C \rightarrow E\}$ è la copertura minimale di F

2.
 - Vogliamo trovare la copertura minimale del seguente insieme di dipendenze funzionali $F = \{BC \rightarrow DE, C \rightarrow D, B \rightarrow D, E \rightarrow L, D \rightarrow A, BC \rightarrow AL\}$
 - Prima di tutto, decomponiamo le dipendenze:

$$F = \{BC \rightarrow D, BC \rightarrow E, C \rightarrow D, B \rightarrow D, E \rightarrow L, D \rightarrow A, BC \rightarrow A, BC \rightarrow L\}$$

- Siccome BC è l'unico determinante su cui potremmo applicare lo step due, calcoliamo $B_F^+ = ABD$ e $C_F^+ = ACD$, da cui otteniamo che:
 - $BC \rightarrow D$ può essere scartata direttamente, poiché sia $B \rightarrow D$ sia $C \rightarrow D$ sono entrambe già in F
 - $E \notin B_F^+ \implies B \rightarrow E \notin F^+$ (e anche $E \notin C_F^+ \implies C \rightarrow E \in F^+$), dunque $BC \rightarrow E$ non può essere rimpiazzata
 - $A \in B_F^+ \implies B \rightarrow A \in F^+$ e $A \in C_F^+ \implies C \rightarrow A \in F^+$, dunque $BC \rightarrow A$ può essere rimpiazzata sia da $B \rightarrow A$ sia da $C \rightarrow A$ (dunque, in base alla scelta, otterremo due coperture minimali diverse). Nel nostro caso, sceglieremo $B \rightarrow A$
 - Siccome $L \notin B_F^+ \implies B \rightarrow L \notin F^+$ e siccome $L \notin C_F^+ \implies C \rightarrow L \notin F^+$, la dipendenza $BC \rightarrow L$ non può essere rimpiazzata

- Al termine dello step 2), quindi, abbiamo che

$$F = \{BC \rightarrow E, C \rightarrow D, B \rightarrow D, E \rightarrow L, D \rightarrow A, B \rightarrow A, BC \rightarrow L\}$$

- Procediamo quindi con lo step 3):

- Siccome E è determinato solo da $BC \rightarrow E$, allora non possiamo scartare tale dipendenza

- Considerando $C \rightarrow D$, si ha che $D \notin C_{F-\{C \rightarrow D\}}^+ = C \iff C \rightarrow D \notin F - \{C \rightarrow D\}^+$, dunque $C \rightarrow D$ non può essere scartata
- Considerando $B \rightarrow D$, si ha che $D \notin B_{F-\{B \rightarrow D\}}^+ = BA \iff B \rightarrow D \notin F - \{B \rightarrow D\}^+$, dunque $B \rightarrow D$ non può essere scartata
- Considerando $E \rightarrow L$, si ha che $L \notin E_{F-\{E \rightarrow L\}}^+ = E \iff E \rightarrow L \notin F - \{E \rightarrow L\}^+$, dunque $E \rightarrow L$ non può essere scartata
- Considerando $D \rightarrow A$, si ha che $A \notin D_{F-\{D \rightarrow A\}}^+ = D \iff D \rightarrow A \notin F - \{D \rightarrow A\}^+$, dunque $D \rightarrow A$ non può essere scartata
- Considerando $B \rightarrow A$, si ha che $A \in B_{F-\{B \rightarrow A\}}^+ = BDA \iff B \rightarrow A \in F - \{B \rightarrow A\}^+$, dunque $B \rightarrow A$ può essere scartata
- Considerando $BC \rightarrow L$, si ha che $L \in BC_{F-\{B \rightarrow A, BC \rightarrow L\}}^+ = BCEDLA \iff BC \rightarrow L \in F - \{B \rightarrow A, BC \rightarrow D\}^+$, dunque $BC \rightarrow L$ può essere scartata
- Infine, otteniamo che $F = \{BC \rightarrow E, C \rightarrow D, B \rightarrow D, E \rightarrow L, D \rightarrow A\}$ è la copertura minimale

▼ Lezione del 29/11

Algoritmo di Decomposizione



Dato uno schema R e un insieme F una **copertura minimale** di dipendenze funzionali su R , il seguente algoritmo calcola in tempo polinomiale, $O(n^k)$, una decomposizione ρ di R tale che ogni sottoschema è in 3NF e ρ preserva F :

```
def decompose_R(R: set of attributes, F: minimal cover of dependencies):
    S, ρ := ;
    for A ∈ R | ∃X → B ∈ F, A ∈ X ∨ A = B :
        S := S ∪ A
    if S ≠ ∅ :
        R := R - S
        ρ := ρ ∪ {s}
    if ∃X → A ∈ F | X ∪ A = R :
        ρ := ρ ∪ {R}
    else:
        for X → A ∈ F :
            ρ := ρ ∪ {xA}
    return ρ
```

Teorema 8: Correttezza algoritmo decompose_R



Sia R uno schema e sia F una copertura minimale di dipendenze funzionali definite su R .

L'algoritmo `decompose_R(R, F)` restituisce una decomposizione ρ tale che ogni sottoschema di ρ è in 3NF e ρ preserva F

Dimostrazione:

Su Appunti

Ottenerne una buona decomposizione

Sia R uno schema e sia F una copertura minimale di dipendenze funzionali definite su R .

Data la decomposizione ρ ottenuta applicando l'algoritmo `decompose_R(R, F)`, se

$\exists K$ chiave di R tale che $K \subseteq R_i$ dove $R_i \in \rho$, allora ρ è una buona decomposizione (ossia preserva F , ha un join senza perdita ed è composta da sottoschemi in 3NF).

Di conseguenza, è sempre possibile ottenere una buona decomposizione di uno schema applicando l'algoritmo `decompose_R(R, F)` e aggiungendo una chiave di R alla decomposizione (a meno che non esista già un sottoschema contenente una chiave).

Esempio:

Esempio:

- Consideriamo lo schema $R = ABCDEH$ e l'insieme di dipendenze $F = \{AB \rightarrow CD, C \rightarrow E, AB \rightarrow E, ABC \rightarrow D\}$
- Prima di tutto, verifichiamo l'unicità della chiave dello schema:
 1. Troviamo prima l'intersezione

$$ABEH \cap ABCDH \cap ABCDH \cap ABCEH = ABH$$

2. Vediamo se la chiusura dell'intersezione coincide con R

$$ABH^+ = ABCDEH = R \implies ABH \text{ è l'unica chiave di } R$$

- Poiché ABH è l'unica chiave di R , notiamo subito che tale schema non è in 3NF, poiché la dipendenza $C \rightarrow E$ viola la condizione richiesta in quanto C non è una superchiave ed E non è un primo
- Cerchiamo quindi una copertura minimale di F , in modo da poter utilizzare l'algoritmo di decomposizione:

1. Decomponiamo tutte le dipendenze di F :

$$F = \{AB \rightarrow C, AB \rightarrow D, C \rightarrow E, AB \rightarrow E, ABC \rightarrow D\}$$

2. Cerchiamo i determinanti ridondanti:

- $A \rightarrow C, B \rightarrow C \notin F^+$ poiché $C \notin A_F^+ = A$ e $C \notin B_F^+,$ dunque $AB \rightarrow C$ non può essere ridotta
- $A \rightarrow D, B \rightarrow D \notin F^+$ poiché $D \notin A_F^+ = A$ e $D \notin B_F^+,$ dunque $AB \rightarrow D$ non può essere ridotta
- $A \rightarrow E, B \rightarrow E \notin F^+$ poiché $E \notin A_F^+ = A$ e $E \notin B_F^+,$ dunque $AB \rightarrow E$ non può essere ridotta

- $AB \rightarrow D, C \rightarrow D \in F \implies AB \rightarrow D, C \rightarrow D \in F^+$, dunque $ABC \rightarrow D$ può essere ridotta sia in $AB \rightarrow D$ sia in $C \rightarrow D$ e dunque scartata poiché entrambe sono già in F
- L'insieme di dipendenze senza determinanti ridondanti quindi sarà:

$$F = \{AB \rightarrow C, AB \rightarrow D, C \rightarrow E, AB \rightarrow E\}$$

3. Cerchiamo quindi le dipendenze ridondanti:

- C e D sono determinati solo da $AB \rightarrow C$ e $AB \rightarrow D$, dunque non possono essere rimosse
- Considerando $C \rightarrow E$, si ha che $E \notin D_{F-\{C \rightarrow E\}}^+ = C \iff C \rightarrow E \notin F - \{C \rightarrow E\}^+$, dunque $C \rightarrow E$ non può essere scartata
- Considerando $AB \rightarrow E$, si ha che $E \in AB_{F-\{AB \rightarrow E\}}^+ = ABCDE \iff AB \rightarrow E \in F - \{AB \rightarrow E\}^+$, dunque $AB \rightarrow E$ può essere scartata

4. La copertura minimale di F quindi sarà:

$$F = \{AB \rightarrow C, AB \rightarrow D, C \rightarrow E\}$$

- A questo punto, possiamo applicare l'algoritmo di decomposizione:

1. L'attributo H non compare in alcuna dipendenza, dunque $S := \{H\}$
2. Siccome $S = \{H\} \neq \emptyset$, allora $R := R - S = ABCDEH - H = ABCDE$ e $\rho := \rho \cup S = \{H\}$
3. Siccome $\nexists X \rightarrow A \in F \mid X \cup A = R = ABCDE$, allora entriamo nel ciclo for, dunque $\rho := \rho \cup \{ABC, ABD, CE\} = \{H, ABC, ABD, CE\}$
4. La decomposizione ρ ottenuta preserva F e tutti i suoi sottoschemi sono in 3NF

- Infine, affinché ρ sia una buona decomposizione, dunque presenti anche un join senza perdita, è sufficiente aggiungere un sottoschema composto dalla chiave ABH alla decomposizione ρ .

Dunque, una buona decomposizione di R con copertura minimale $F = \{AB \rightarrow C, AB \rightarrow D, C \rightarrow E\}$ risulta essere:

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

▼ Lezione del 30/11

Organizzazione fisica

Abbiamo già accennato la differenza tra livello fisico e livello logico di un database. In questo capitolo verrà discusso il funzionamento interno del livello fisico con particolare attenzione sull'organizzazione fisica dei dati.

Prima di tutto, è necessario descrivere le sotto-aree che compongono il livello fisico:

- **Hardware di archiviazione e progettazione fisica:**

- Gerarchia di archiviazione
- Interni di un Hard Disk
- Passaggio dai concetti logici ai concetti fisici

- **Organizzazione dei record:**

- Puntatori
- Liste

- **Organizzazione dei file:**

- Organizzazione con file heap
- Organizzazione sequenziale dei file
- Organizzazione casuale dei file (Hashing)
- Organizzazione indicizzata sequenziale dei file
- Organizzazione dei dati in lista
- Indici secondari e File invertiti
- B-trees e B+-trees

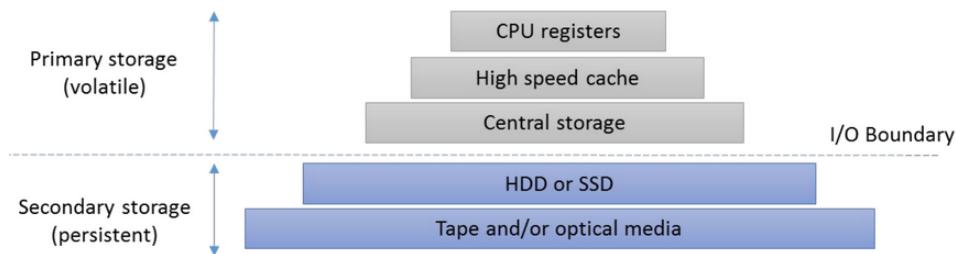
Hardware di archiviazione e progettazione fisica

Gerarchia dell'archiviazione



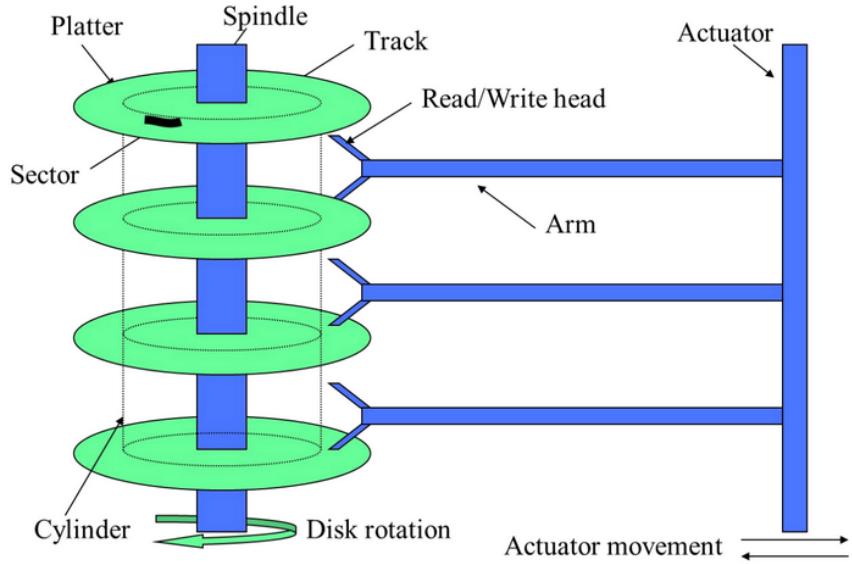
L'**Archiviazione primaria** di un DBMS contiene i buffer del database e i codici in esecuzione delle applicazioni e del DBMS. Essa è composta da CPU, Cache, Memoria principale (composta a sua volta da vari banchi di RAM).

L'**Archiviazione secondaria** si occupa dell'archiviazione permanente dei dati attraverso vari Hard Disk (HDD) e Solid-state drive (SSD), memorizzando i file fisici contenenti i dati del database



In particolare, ci concentreremo sul funzionamento interno di un Hard Disk, il quale è costituito da:

- Un controller interno che gestisce gli altri componenti e che gestisce le richieste di lettura/scrittura, mettendole in coda
- Piatti circolari magnetici assicurati su uno **spindle** (ossia un mandrino), il quale ruota a velocità costante
- Delle **testine di lettura e scrittura** posizionate sui bracci di un attuatore. Quest'ultimo, muovendosi, posizionerà le testine sui piatti, andando a leggere o a scrivere dei dati.



Leggere o scrivere un blocco di dati dai piatti, quindi, implica:

- Il posizionamento corretto dell'attuatore, il cui tempo impiegato viene detto **Seek time (Seek)**
- L'attesa della rotazione del disco affinché il settore richiesto si trovi sotto la testina di lettura e scrittura, il cui tempo impiegato viene detto **Rotation time (ROT)**
- Il trasferimento dei dati, il cui tempo viene detto **Transfer time**, dipendente dalla grandezza del blocco da leggere, ossia il **block size (BS)**, e il rateo di trasferimento dei dati, ossia il **trasfer rate (TR)**, influenzato dalla densità delle particelle magnetiche presenti sul disco e la velocità di rotazione del disco stesso

Random Block Access e Sequential Block Access



Il tempo **totale** impiegato per completare una lettura o una scrittura viene detto **Service time**

$$\text{Service time} = \text{Seek} + \text{ROT} + \text{Transfertime} = \text{Seek} + \text{ROT} + \frac{BS}{TR}$$

Il tempo totale impiegato dall'hard disk a restituire una risposta è detto **Response time** ed è quindi composto dal Service time e il tempo necessario a mettere in coda la richiesta (**Queueing time**):

$$\text{Response time} = \text{Service time} + \text{Queueing time}$$

Poiché il tempo di lettura e di scrittura dipende anche dalla posizione delle testine,

distinguiamo in:

-Il tempo impiegato ad effettuare un **Random Block Access**, ossia il tempo

impiegato ad accedere ad un blocco indipendentemente dal precedente accesso,

dunque indipendentemente dalla posizione attuale della testina

$$T_{RBA} = \text{Seek} + \frac{\text{ROT}}{2} + \frac{BS}{TR}$$

-Il tempo impiegato ad effettuare un **Sequential Block Access**, ossia il tempo

impiegato ad accedere sequenzialmente ad un blocco con la testina già posizionata

nel settore corretto

$$T_{SBA} = \frac{\text{ROT}}{2} + \frac{BS}{TR}$$

Esempio:

- Vogliamo sapere il tempo impiegato dalla lettura di un blocco 4096 Byte da HDD possedente le seguenti specifiche:
 - Seek time medio: 8.9 ms
 - Velocità dello spindle: 7200 rpm (rotazioni per minuto)
 - Transfer rate: 150 MBps (MB al secondo)
- Prima di tutto, calcoliamo il Rotation time, prima in minuti e poi in millisecondi:

$$ROT = \frac{1}{7200 \text{ rpm}} \text{ minutes} = \frac{60 \cdot 1000}{7200 \text{ rpm}} \text{ ms} \approx 8.33 \text{ ms}$$

- Successivamente, calcoliamo il tempo impiegato per un RBA ed un SBA:

$$\begin{aligned} T_{RBA} &\approx 8.9 \text{ ms} + \frac{8.33}{2} \text{ ms} + \frac{4096 \text{ B}}{150 \cdot 2^{20} \text{ Bps}} \text{ s} = 8.9 \text{ ms} + \frac{8.33}{2} \text{ ms} + 2.6 \cdot 10^{-5} \text{ s} \approx \\ &\approx 8.9 \text{ ms} + 4.165 \text{ ms} + 2.6 \cdot 10^{-2} \text{ ms} \approx 13.09 \text{ ms} \end{aligned}$$

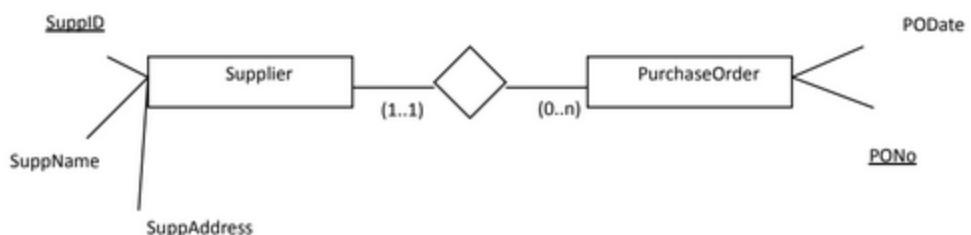
$$T_{SBA} \approx \frac{8.33}{2} \text{ ms} + \frac{4096 \text{ B}}{150 \cdot 2^{20} \text{ Bps}} \text{ s} \approx 4.165 \text{ ms} + 2.6 \cdot 10^{-2} \text{ ms} \approx 4.19 \text{ ms}$$

Passaggio dai concetti logici ai concetti fisici

La seguente tabella racchiude i concetti inerenti alla "traduzione" dal modello concettuale di un database alla sua implementazione prima a livello relazionale e successivamente alla sua rappresentazione fisica.

Modello concettuale	Modello relazionale	Modello interno
Tipo attributo e valore	Nome colonna e cella	Dati oggetti o campi (bit o caratteri rappresentanti uno specifico valore)
(Entità) Record	Riga o tupla	Record archiviato (collezione di dati oggetto)
(Entità) Tipo record	Tabella o relazione	File fisico o insieme di dati
Insieme di tipi di record	Insieme di tabelle	Database fisico (collezione di file)
Strutture dati logiche	Chiavi esterne	Strutture di archiviazione

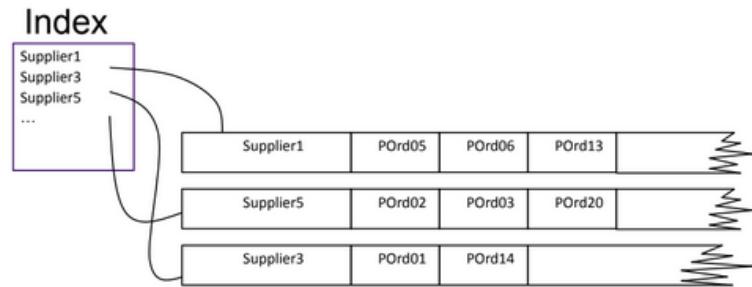
Modello concettuale:



Modello Logico:

`Supplier(SupplID,SuppName,SuppAddress)`
`PurchaseOrder(PONo,PODate,SupplID)`

Modello Interno:



Riassumendo, quindi, si ha che:

- Un database consiste in un insieme di file
- Ogni file può essere visto come una collezione di pagine di dimensione fissa
- Ogni pagina archivia più record (corrispondenti a tuple logiche)
- Un record consiste di più campi di dimensione fissa a variabile, rappresentanti gli attributi delle tuple

Organizzazione dei file

Una volta studiato il comportamento di un HDD, vogliamo organizzare i file fisici sui piatti in modo da minimizzare al massimo il Seek time e ridurre il più possibile il Rotation time.

Per ottenere ciò, ci basta minimizzare gli RBA e massimizzare gli SBA.

Organizzazione con file heap e file sequenziali

Metodo 1: Organizzazione con file heap



- È il modello di organizzazione dei file primaria più basilare
- I nuovi record vengono **inseriti alla fine del file**
- Non vi è relazione tra gli attributi dei record e la loro locazione fisica
- L'unica opzione per il recupero dei record è la **ricerca lineare** (ossia controllando sequenzialmente ogni record)
- Ad ogni record è associata una **chiave di ricerca** che lo identifica
- Per un file con N blocchi, il **tempo medio impiegato per trovare un record** in base alla sua chiave univoca di ricerca è $|\frac{N}{2}|SBA$
- Cercare record in base a chiavi di ricerca non univoche richiede la lettura dell'intero file, in modo da selezionare il record giusto

Metodo 2: Organizzazione con file sequenziali



- I record vengono **archiviati in ordine crescente** (o discendente) in base al valore della loro **chiave di ricerca**
- Essendo i record ordinati in base alla loro chiave di ricerca, viene utilizzata la **ricerca binaria**, rendendo il numero atteso di accessi ai blocchi necessari per recuperare un record pari a $|\log_2(N)| RBA$, poiché i blocchi su cui si accede non sono sequenziali per natura stessa dell'algoritmo di ricerca binaria.
- Può essere utilizzata anche la **ricerca lineare**, rendendo il numero atteso di accessi ai blocchi necessari per recuperare un record pari a $|\frac{N}{2}|SBA$, risultando tuttavia meno efficiente
- Aggiornare i file sequenziali è più laborioso rispetto all'aggiornamento di un file heap, poiché richiede il **riordinamento delle chiavi**. Per tale motivo, spesso vengono fatti più aggiornamenti simultaneamente

Esempio:

- Dato un numero di record (NR) pari a 30000, un block size (BS) pari a 2048 Byte e un record size (RS) pari a 100 Byte, la quantità di record per blocco (RpB) è pari a:

$$RpB = \left\lfloor \frac{BS}{RS} \right\rfloor = \left\lfloor \frac{2048}{100} \right\rfloor = 20$$

- Dunque, il numero di blocchi (NB) necessari per archiviare i record è:

$$NB = \left\lceil \frac{NR}{RpB} \right\rceil = \left\lceil \frac{30000}{20} \right\rceil = 15000$$

- Se viene utilizzata la ricerca lineare, il valore atteso del numero di accessi ai blocchi (NA) sarà:

$$NA = \left\lceil \frac{NB}{2} \right\rceil = \left\lceil \frac{1500}{2} \right\rceil = 750 \text{ SBA}$$

- Se viene utilizzata la ricerca binaria, il valore atteso del numero di accessi ai blocchi (NA) sarà:

$$NA = \lceil \log_2(NB) \rceil = \lceil \log_2(1500) \rceil = 11 \text{ RBA}$$

- Dunque, nonostante gli SBA richiedano molto meno tempo dei RBA, in questo caso il numero di RBA è talmente basso da rendere comunque più efficiente la ricerca binaria

Organizzazione con file hash

Metodo 3: Organizzazione con file hash



- Viene utilizzato un **algoritmo di hashing** per effettuare una conversione da chiave all'indirizzo fisico dove il record è archiviato
 - Risulta più efficiente quando viene utilizzata una chiave primaria
 - Poiché viene utilizzata una funzione di hash, non è garantito che tutte le chiavi vengano mappate a valori hash diversi, per via delle possibili collisioni, dunque vengono utilizzati dei "recipienti", detti **bucket**, che contengono tutti i record il cui hash generato coincide.
 - Viene mantenuta salvata in memoria principale una **bucket directory**, ossia un insieme di record dove ogni entrata corrisponde ad un puntatore al primo blocco di un bucket associato. Ogni entrata è indicizzata da un valore assumibile dalla funzione di hash.
 - L'algoritmo di hashing deve distribuire i record il più possibile in modo **uniforme**, spesso realizzata tramite l'operatore modulo (ad es: $\text{address}(\text{key}_i) = \text{key}_i \bmod M$)

Esempio:

- Consideriamo la seguente serie di chiavi, dove ognuna incrementa di 1 rispetto alla precedente, analizzando il loro comportamento utilizzando il mod 20 e il mod 23:

Chiavi	Resto in mod 20	Resto in mod 23
3000	00	10
3001	01	11
3002	02	12
3003	03	13
3004	04	14
3005	05	15
3006	06	16
3007	07	17
3008	08	18
3009	09	19
3010	10	20
3011	11	21
3012	12	22

- In tal caso, quindi, le chiavi risultano essere ben distribuite uniformemente nei vari recipienti
- Tuttavia, se l'incremento tra una chiave e l'altra fosse di 25, otterremmo una distribuzione poco uniforme nel caso del mod 20, poiché solo i recipienti 0, 5, 10 e 15 verrebbero utilizzati:

Chiavi	Resto in mod 20	Resto in mod 23
3000	00	10
3025	05	12
3050	10	14
3075	15	16
3100	00	18
3125	05	20
3150	10	22
3175	15	01
3200	00	03
3225	05	05
3250	10	07
3275	15	09
3300	00	11

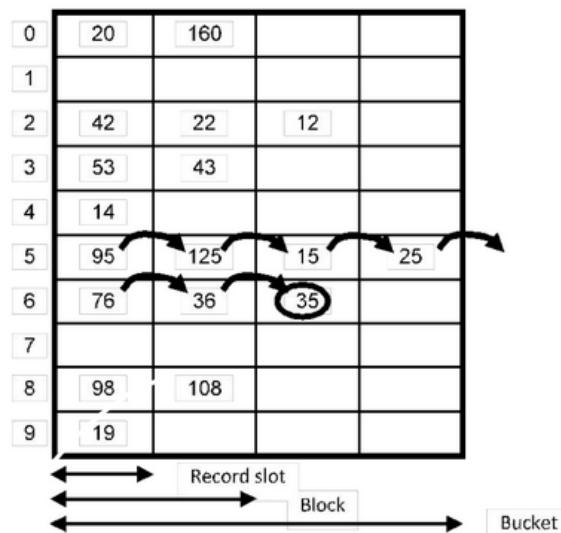
Se si ha una quantità di record mappati allo stesso bucket maggiore della capienza stabilita per il bucket stesso, allora il record viene considerato in **overflow**. La tecnica più utilizzata per la gestione degli overflow è la tecnica di **indirizzamento aperto**, dove gli overflow vengono archiviati nel primo slot disponibile

L'efficienza dell'algoritmo di hash utilizzato è misurata in base al numero atteso di RBA e di SBA:

- **Recuperare un record non in overflow** richiede un singolo RBA per raggiungere l'indirizzo del primo blocco del bucket, ottenuto dall'algoritmo di hash, per poi (potenzialmente) eseguire uno o più SBA, poiché i record nello stesso bucket sono archiviati in modo sequenziale
- **Recuperare un record in overflow** richiede accessi ai blocchi aggiuntivi in base alla percentuale di record in overflow, dipendente dalla tecnica di hashing utilizzata, e in base alla tecnica adottata per gestirli

La scelta di un bucket size maggiore comporta la presenza di meno overflow, aumentando tuttavia la quantità di SBA da effettuare per recuperare i record non in overflow.

Dunque, è necessario effettuare un **compromesso** tra le due cose.



Esempio:

- Supponiamo di avere un file di 1857000 record (NR). Ogni record occupa 256 byte (RS). Ogni blocco contiene 4096 byte (BS). Un puntatore a blocco occupa 5 byte (PS). Il file viene organizzato con una struttura hash

con 300 bucket (NBk). La funzione hash è valutata su un campo che è chiave.

- La quantità di puntatori per blocco della bucket directory corrisponde a:

$$PpB = \left\lceil \frac{BS}{PS} \right\rceil = \left\lceil \frac{4096}{5} \right\rceil = 819$$

- Il numero di blocchi necessari per la bucket directory sarà:

$$NBpBD = \left\lceil \frac{NBk}{PpB} \right\rceil = \left\lceil \frac{300}{819} \right\rceil = 1$$

- La quantità di record per blocco di ogni bucket corrisponde a:

$$RpB = \left\lceil \frac{BS - PS}{RS} \right\rceil = \left\lceil \frac{4096 - 5}{256} \right\rceil = 15$$

- Il numero di record che verrà inserito in ogni bucket corrisponde a:

$$RpBk = \left\lceil \frac{NR}{NB} \right\rceil = \left\lceil \frac{1857000}{300} \right\rceil = 6190$$

- Il numero di blocchi necessari per ogni bucket sarà:

$$NBpBk = \left\lceil \frac{RpBk}{RpB} \right\rceil = \left\lceil \frac{1857000}{300} \right\rceil = 413$$

- Il numero di blocchi complessivi necessari per poter contenere la bucket directory e tutti i bucket corrisponde a:

$$NB_{TOT} = NBk \cdot NBpBk + NBpBD = 300 \cdot 413 + 1 = 123901$$

- Il numero medio di accessi per effettuare una ricerca sarà:

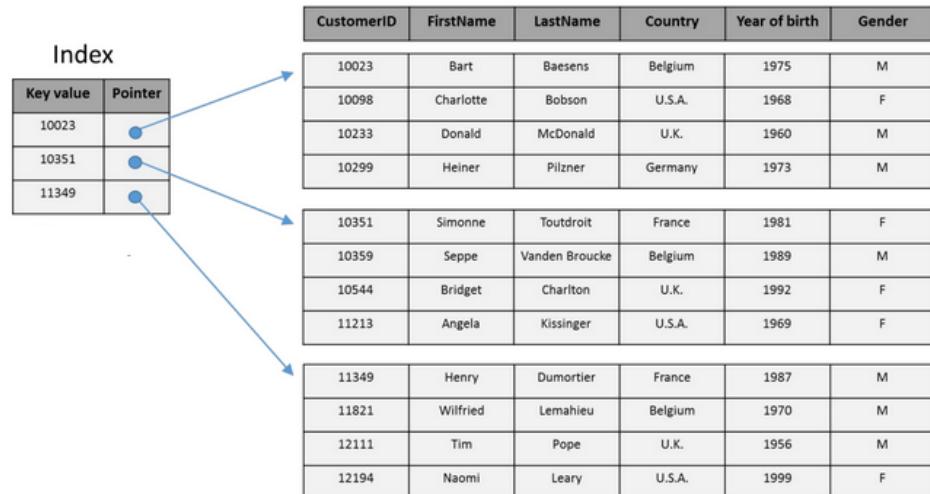
$$NA = \left\lceil \frac{NBpBk}{2} \right\rceil = \left\lceil \frac{413}{2} \right\rceil = 207$$

Organizzazione con file indicizzati sequenziali (ISAM)

Metodo 4: Organizzazione ISAM



- Il **file principale** contenente i record è diviso in **partizioni**, ognuna corrispondente ad un blocco, ognuna rappresentata da **un'entrata <Chiave di ricerca primo record, Puntatore indirizzo fisico primo record>**, le quali vengono archiviate in un **file indice**.
- Il file principale e il file indice sono entrambi **ordinati** in base alle chiavi di ricerca presenti nelle loro entrate
- Il file indice può essere di tipo **denso**, dove esiste un'entrata per ogni possibile valore della chiave di ricerca, o di tipo **sparso**, dove esiste solo l'entrata del primo record di ogni partizione, implicando quindi che ogni entrata faccia riferimento ad un gruppo di record



Dato un numero di blocchi del file principale pari a NB_{FP} ed un numero di blocchi del file indice pari a NB_{FI} , l'**efficienza delle ricerche** risulta essere:

- **Ricerca lineare sul file principale:** NB_{FP} SBA
- **Ricerca binaria sul file principale:** $\lceil \log_2(NB_{FP}) \rceil$ RBA
- **Ricerca binaria tramite file indice:** $\lceil \log_2(NB_{FI}) \rceil + 1$ RBA, dove l'ultimo RBA è dovuto all'accesso al blocco dell'entrata trovata

Effettuando la **ricerca binaria sul file indice**, se k è la chiave del record che si sta cercando, a e b sono gli estremi dell'intervallo di chiavi del file indice che si

sta considerando attualmente e m è il valore medio di tale intervallo, allora:

- Se $k < m$ allora l'algoritmo controllerà l'insieme di chiavi compreso tra $[a, m-1]$, poiché la chiave k **sicuramente non si trova all'interno del blocco m**
- Se $k = m$ allora l'algoritmo selezionerà tale blocco
- Se $k > m$ allora l'algoritmo controllerà l'insieme di chiavi compreso tra $[m, b]$, poiché la chiave k **potrebbe essere all'interno del blocco m**

Esempio:

- È dato un file di 1750000 record (NR). Ogni record occupa 130 byte (RS), di cui 35 per la chiave (KS). Un puntatore a blocco occupa 5 byte (PS). Un blocco di memoria contiene 2048 byte (BS). Utilizziamo un indice ISAM per organizzare i record di tale file.
- La quantità di record per blocco del file principale corrisponde a:

$$RpB = \left\lfloor \frac{BS}{RS} \right\rfloor = \left\lfloor \frac{2048}{130} \right\rfloor = 15$$

- Il numero di blocchi necessari per il file principale corrisponde a:

$$NBpFP = \left\lceil \frac{NR}{RpB} \right\rceil = \left\lceil \frac{1750000}{15} \right\rceil = 116667$$

- La dimensione di ogni entrata del file indice corrisponde a:

$$ES = KS + PS = 35 + 5 = 40$$

- La quantità di entrate per blocco corrisponde a:

$$EpB = \left\lfloor \frac{BS}{ES} \right\rfloor = \left\lfloor \frac{2048}{40} \right\rfloor = 51$$

- Il numero di blocchi necessari per il file indice corrisponde a:

$$NBpFI = \left\lceil \frac{NBpFP}{EpB} \right\rceil = \left\lceil \frac{116667}{51} \right\rceil = 2288$$

- Numero massimo di accessi necessari per effettuare una ricerca utilizzando la ricerca tramite file indice corrisponde a:

$$NA = \lceil \log_2(NBpFI) \rceil + 1 = \lceil \log_2(2288) + 1 \rceil + 1 = 12 + 1 = 13$$

▼ Lezione del 06/12

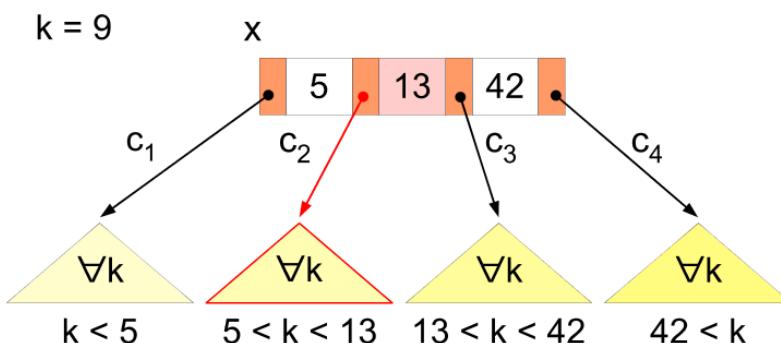
Organizzazione con B-Tree

Albero di ricerca ad m-vie



Un **albero di ricerca ad m-vie** è una generalizzazione del normale albero binario di ricerca, avente le seguenti proprietà:

- Ogni nodo può avere da **1 ad $m - 1$ valori chiave**
- Il **numero di sotto-alberi** di ogni nodo può variare da 0 a $i + 1$, dove i è il numero di valori chiave nel nodo.
- Il grado di ogni nodo (ossia il numero di figli) è al **massimo m**
- Le **chiavi interne al sottoalbero** puntato da un puntatore compreso tra due chiavi k e k_0 di un nodo possono assumere solo valori nell'intervallo aperto (k, k_0)



B-Tree



Un **B-Tree** è un albero di ricerca ad m-vie avente le seguenti proprietà aggiuntive:

- Ogni nodo ha al **massimo** m nodi figli e **massimo** $m - 1$ **chiavi**
- Ogni nodo (eccetto la radice e le foglie) ha **minimo** $d := \lceil \frac{m}{2} \rceil$ nodi figli e **minimo** $d-1$ **chiavi**, mentre la radice ha **minimo** due nodi figli (a meno che non sia essa stessa una foglia)
- Tutti i nodi foglia **sono allo stesso livello**

Implementazione dei B-Tree

Posso implementare gli alberi Bilanciati in diversi modi, uno di questi è con una **Linked List**

E' utile usare la **ricerca binaria** con gli alberi Binari di Ricerca

Un **albero Binario** è un albero con al massimo 2 figli.

Metodo 5: Organizzazione con B-Tree

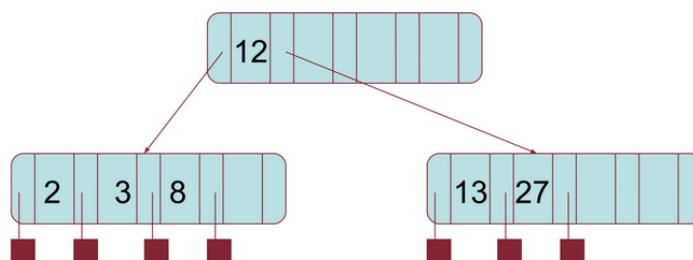


Un B-Tree può essere utilizzato come un **file indice strutturato ad albero**:

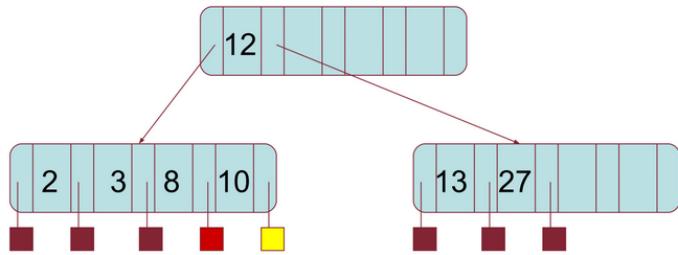
- Ogni nodo corrisponde ad un **blocco** e, per via dei vincoli imposti dalle proprietà,
essi sono sempre **carichi almeno a metà** (ossia almeno metà della loro capienza
è occupata), implicando che esistano almeno d nodi figli
- Ogni nodo contiene:
 - Un **insieme di chiavi di ricerca**
 - Un **insieme di puntatori ai nodi figli**
 - Un **insieme di puntatori ai dati** che fanno riferimento ai blocchi del file principale, contenente i record
- Per comodità, i **blocchi del file principale** vengono tutti puntati dai nodi posti
al livello più basso dell'albero (i blocchi del file principale

Esempio:

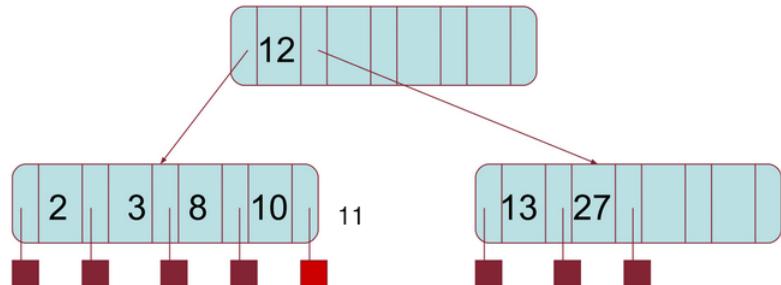
- Consideriamo un B-Tree a 5 vie, implicando che il numero massimo di nodi figli per ogni nodo sarà 5, mentre il numero minimo di nodi figli per ogni nodo sia $\lfloor \frac{5}{2} \rfloor = 3$ (fatta eccezione per la radice, la quale avrà minimo 2 figli, a meno che essa non sia una foglia)



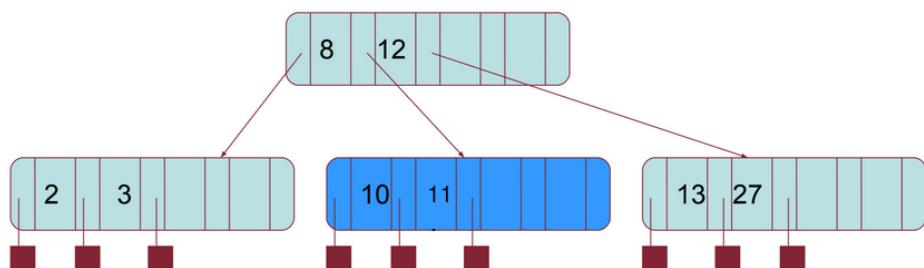
- Inserendo la chiave 10, procediamo come un normale inserimento di un albero binario di ricerca, aggiungendo quindi 10 al figlio sinistro della radice



- Nel caso in cui volessimo inserire la chiave 11, essa verrebbe posta all'interno del figlio sinistro, il quale tuttavia può avere una capienza massima pari a 5, dunque l'inserimento della chiave 11 porterebbe il nodo ad essere sovraccarico

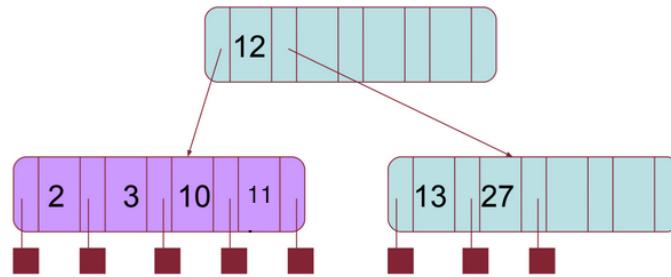


- Per poter inserire la chiave 11, quindi, è necessario separare il nodo a metà, aggiungendo un nuovo nodo figlio alla radice e muovendo i valori affinché le regole di un normale albero binario vengano rispettate (ad esempio, il puntatore al figlio destro di 8 e sinistro di 12 conterrà tutti i valori compresi tra 8 e 12)



- Nel caso in cui volessimo rimuovere la chiave 8 dalla radice, saremmo costretti a spostare nella radice la chiave 3 o la chiave 10, in modo che

possano rimpiazzare l'8 mantenendo le proprietà dell'albero di ricerca. Tuttavia, muovendo una delle due chiavi si otterrebbe che uno dei due nodi figli abbia meno del numero minimo di figli, ossia 3. Di conseguenza, l'unica soluzione è fondere i due nodi figli, senza spostare una delle due chiavi nella radice



Ricerca dei record in un B-Tree

La **ricerca di record** all'interno di un B-Tree risulta essere simile a quella di un albero

di ricerca normale:

- Partendo dalla radice, la ricerca viene effettuata **ricorsivamente**
- Se il valore chiave X desiderato viene trovato in un nodo, ad esempio X corrisponde alla chiave K_i , allora il record corrispondente può essere acceduto tramite il puntatore ai dati P_{di}
- Se invece il valore non è presente nel nodo, allora si procede ricorsivamente tramite il puntatore al figlio P_i , dove i è il valore più piccolo per cui $X < K_i + 1$.
Se $X > K_j \forall K_j$ in un nodo, allora si procede col puntatore figlio P_{i+1}

Poiché nel caso peggiore viene traversato un intero ramo dell'albero fino al raggiungimento di una foglia, il **costo della ricerca risulta essere $O(h - 1 + 1)$** , dove h è **l'altezza dell'albero**, il -1 è dovuto alla presenza della radice nella

RAM (dunque richiedendo nessun accesso ai blocchi) e il +1 è dovuto all'accesso ai blocchi finale al record puntato dal puntatore dati corrispondente alla chiave trovata

▼ Lezione del 7/12

Altezza di un B-Tree

Dato un B-Tree avente N chiavi, posto m il numero massimo di figli che un nodo possa avere e $d := \lfloor \frac{m}{2} \rfloor$ il numero minimo di figli che un nodo possa avere, l'**altezza** h del B-Tree è compresa tra:

$$|\log_m(N+1)| \leq h \leq |\log_d(\frac{N+1}{2})| + 1$$

Dunque, l'**altezza massima** viene raggiunta quando l'albero possiede il **minimo numero di nodi**, mentre l'**altezza minima** quando possiede il massimo numero di nodi

Dimostrazione:

Su appunti

Performance di un B-Tree

Data l'altezza h di un B-Tree, si ha che:

- L'operazione $get(k)$ impiega massimo h accessi al disco
- L'operazione $put(k)$ impiega massimo $3h + 1$ accessi al disco
- L'operazione $remove(k)$ impiega massimo $3h$ accessi al disco

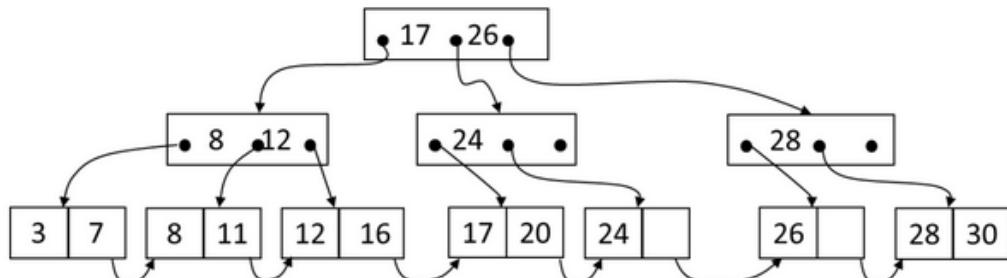
B^+ -Tree



Un B^+ -Tree è un B-Tree avente le seguenti proprietà aggiuntive:

- Tutti i valori chiave presenti in nodi non foglia sono **ripetuti** nei nodi foglia, in modo che ogni possibile chiave dell'albero sia presente nei nodi foglia stessi
- I nodi di livello più alto contengono sottoinsiemi delle chiavi presenti nei nodi foglia
- Ogni nodo foglia ha un **puntatore aggiuntivo**, il quale punta ad un nodo foglia adiacente

Per via di tali proprietà, un B+-Tree risulta essere più efficiente rispetto ad un normale B-Tree, poiché la sua altezza è generalmente inferiore



Esempi:

1

- Supponiamo di avere un file di 1700000 record (NR). Ogni record occupa 250 byte (RS), di cui 45 per il campo chiave (KS). Ogni blocco contiene 2048 byte (BS). Un puntatore a blocco occupa 4 byte (PS). Utilizzando un'organizzazione B-tree, vogliamo sapere il minimo numero di accessi per effettuare una ricerca.
- Il minimo numero di accessi è raggiunto quanto l'altezza dell'albero è minima, dunque quando ogni nodo possiede il massimo numero di figli,

implicando che ogni blocco sia carico al 100%

- Sia d il numero minimo di figli di un nodo e sia $k_{\min} := d-1$ il numero minimo di chiavi in un nodo
- Poiché il minimo numero di accessi necessari è raggiunto quando l'altezza dell'albero è minima, è necessario che l'albero sia carico al massimo. Sia m il numero massimo di figli di un nodo e sia $k_{\max} := m - 1$ il numero massimo di chiavi in un nodo
- Il massimo numero di chiavi per nodo (dunque per blocco) corrisponde a:

$$BS = k_{\max}(KS + PS) + PS \implies k_{\max} = \left\lfloor \frac{BS - PS}{KS + PS} \right\rfloor$$

da cui traiamo che il massimo numero di figli per nodo corrisponde a:

$$m = \left\lfloor \frac{BS - PS}{KS + PS} \right\rfloor + 1 = \left\lfloor \frac{2048 - 4}{45 + 4} \right\rfloor + 1 = 41 + 1 = 42$$

- La quantità di record per blocco del file principale corrisponde a:

$$RpB = \left\lfloor \frac{BS}{RS} \right\rfloor = \left\lfloor \frac{2048}{250} \right\rfloor = 8$$

- Il numero di blocchi necessari per il file principale corrisponde a:

$$NBpFP = \left\lceil \frac{NR}{RpB} \right\rceil = \left\lceil \frac{1700000}{8} \right\rceil = 21250$$

- Il numero di nodi/blocchi del livello L0 del file indice (ossia il più basso) corrisponde a:

$$NBpL0 = \left\lceil \frac{NBpFP}{m} \right\rceil = \left\lceil \frac{21250}{42} \right\rceil = 506$$

- Il numero di nodi/blocchi del livello L1 del file indice (ossia il penultimo) corrisponde a:

$$NBpL1 = \left\lceil \frac{NBpL0}{m} \right\rceil = \left\lceil \frac{506}{42} \right\rceil = 13$$

- Il numero di nodi/blocchi del livello L2 del file indice corrisponde a:

$$NBpL2 = \left\lceil \frac{NBpL1}{m} \right\rceil = \left\lceil \frac{13}{42} \right\rceil = 1$$

- Il livello L2, quindi, corrisponderà con la radice dell'albero, implicando che l'albero abbia un'altezza pari a $h = 3$. Dunque, il costo minimo di una ricerca sarà:

$$NA_{min} = O(h) = 3$$

- Il numero di blocchi necessari per il file indice sarà:

$$NBpFI = NBpL0 + NBpL1 + NBpL2 = 21250 + 506 + 13 + 1 = 21770$$

2

- Supponiamo di voler calcolare il massimo numero di accessi necessari ad effettuare una ricerca con gli stessi dati dell'esercizio precedente.
- Il massimo numero di accessi è raggiunto quanto l'altezza dell'albero è massima, dunque quando ogni nodo possiede il minimo numero di figli, implicando che ogni blocco sia carico al 50%
- Sia d il numero minimo di figli di un nodo e sia $k_{min} := d-1$ il numero minimo di chiavi in un nodo
- Il minimo numero di chiavi per nodo (dunque per blocco) corrisponde a:

$$BS \cdot 0.5 = k_{min}(KS + PS) + PS \implies k_{min} = \left\lceil \frac{BS \cdot 0.5 - PS}{KS + PS} \right\rceil$$

da cui traiamo che il minimo numero di figli per nodo corrisponde a:

$$d = \left\lceil \frac{BS \cdot 0.5 - PS}{KS + PS} \right\rceil + 1 = \left\lceil \frac{2048 \cdot 0.5 - 4}{45 + 4} \right\rceil + 1 = 21 + 1 = 22$$

- La quantità di record per blocco del file principale corrisponde a:

$$RpB = \left\lfloor \frac{BS \cdot 0.5}{RS} \right\rfloor = \left\lfloor \frac{2048 \cdot 0.5}{250} \right\rfloor = 4$$

- Il numero di blocchi necessari per il file principale corrisponde a:

$$NBpFP = \left\lceil \frac{NR}{RpB} \right\rceil = \left\lceil \frac{1700000}{4} \right\rceil = 42500$$

- Il numero di nodi/blocchi del livello L0 del file indice (ossia il più basso) corrisponde a:

$$NBpL0 = \left\lceil \frac{NBpFP}{m} \right\rceil = \left\lceil \frac{42500}{22} \right\rceil = 1932$$

- Il numero di nodi/blocchi del livello L1 del file indice (ossia il penultimo) corrisponde a:

$$NBpL1 = \left\lceil \frac{NBpL0}{m} \right\rceil = \left\lceil \frac{1932}{22} \right\rceil = 88$$

- Il numero di nodi/blocchi del livello L2 del file indice corrisponde a:

$$NBpL2 = \left\lceil \frac{NBpL1}{m} \right\rceil = \left\lceil \frac{88}{22} \right\rceil = 4$$

- Il numero di nodi/blocchi del livello L3 del file indice corrisponde a:

$$NBpL3 = \left\lceil \frac{NBpL2}{m} \right\rceil = \left\lceil \frac{4}{22} \right\rceil = 1$$

- Il livello L3, quindi, corrisponderà con la radice dell'albero, implicando che l'albero abbia un'altezza pari a $h = 4$. Dunque, il costo minimo di una ricerca sarà:

$$NA_{max} = O(h) = 4$$

- Il numero di blocchi necessari per il file indice sarà:

$$NBpFI = NBpL0 + NBpL1 + NBpL2 + NBpL3 = 42500 + 1932 + 88 + 4 + 1 = 44525$$

Altri esercizi nel fogli negli appunti