



SAPIENZA
UNIVERSITÀ DI ROMA

“SAPIENZA” UNIVERSITÀ DI ROMA
INGEGNERIA DELL'INFORMAZIONE,
INFORMATICA E STATISTICA
DIPARTIMENTO DI INFORMATICA

Basi di Dati I

Appunti integrati con il libro “Principles of Database & Knowledge-Base
Systems - Vol. 1”, J. D. Ullman

Autore
Simone Bianco

23 maggio 2025

Indice

Informazioni e Contatti	1
1 Database e DBMS	2
1.1 Database come sistema informativo	2
1.2 Modellazione dei dati	3
1.3 Linguaggi, Integrità e Transazioni	4
2 Modello relazionale	6
2.1 Da relazioni a tabelle	8
2.2 Riferimenti e Valori nulli	10
2.3 Vincoli di integrità	11
2.3.1 Chiavi primarie e secondarie	13
2.4 Anomalie di un database	15
3 Algebra relazionale	17
3.1 Proiezione e Selezione	17
3.2 Unione, Intersezione e Differenza	20
3.3 Ridenominazione e Prodotto Cartesiano	25
3.4 Join Naturale e Theta Join	27
3.5 Quantificazione universale	32
4 Teoria della normalizzazione	34
4.1 Dipendenze funzionali	34
4.1.1 Chiusura di F	35
4.2 Assiomi di Armstrong	37
4.2.1 Chiusura di X	39
4.2.2 $F^+ = F^A$	40
4.3 Terza Forma Normale (3NF)	43
4.4 Calcolare X^+	47
4.4.1 Trovare le chiavi di uno schema	51
4.5 Decomposizione di uno schema	55
4.5.1 Preservazione di F	58
4.5.2 Join senza perdita	63
4.5.3 Copertura minimale	68
4.5.4 Algoritmo di decomposizione	73

5	Organizzazione fisica	76
5.1	Hardware di archiviazione e progettazione fisica	77
5.1.1	Passaggio dai concetti logici ai concetti fisici	79
5.2	Organizzazione dei file	81
5.2.1	Organizzazione con file heap e file sequenziali	81
5.2.2	Organizzazione con file hash	82
5.2.3	Organizzazione con file indicizzati sequenziali (ISAM)	85
5.2.4	Organizzazione con B-Tree	88

Informazioni e Contatti

Appunti e riassunti personali raccolti in ambito del corso di *Basi di Dati I* offerto dal corso di laurea in Informatica dell'Università degli Studi di Roma "La Sapienza".

Ulteriori informazioni ed appunti possono essere trovati al seguente link:

<https://github.com/Exyss/university-notes>. Chiunque si senta libero di segnalare incorrettezze, migliorie o richieste tramite il sistema di Issues fornito da GitHub stesso o contattando in privato l'autore :

- Email: bianco.simone@outlook.it
- LinkedIn: [Simone Bianco](#)

Gli appunti sono in continuo aggiornamento, pertanto, previa segnalazione, si prega di controllare se le modifiche siano già state apportate nella versione più recente.

Prerequisiti consigliati per lo studio:

Nessuno

Licence:

These documents are distributed under the [GNU Free Documentation License](#), a form of copyleft intended for use on a manual, textbook or other documents. Material licensed under the current version of the license can be used for any purpose, as long as the use meets certain conditions:

- All previous authors of the work must be **attributed**.
- All changes to the work must be **logged**.
- All derivative works must be **licensed under the same license**.
- The full text of the license, unmodified invariant sections as defined by the author if any, and any other added warranty disclaimers (such as a general disclaimer alerting readers that the document may not be accurate for example) and copyright notices from previous versions must be maintained.
- Technical measures such as DRM may not be used to control or obstruct distribution or editing of the document.

1

Database e DBMS

1.1 Database come sistema informativo

Nei dispositivi elettronici, l'informazione può essere registrata sottoforma di **dati strutturati**, ossia oggetti rappresentati da piccole stringhe di simboli e numeri, oppure sottoforma di **dati de-strutturati**, ossia testi scritti in linguaggio naturale descriventi qualcosa.

Tali informazioni vengono immagazzinate in **sistemi informativi**, ad esempio un grande archivio, cartaceo o digitale che sia, utilizzato per acquisire, processare e condividere informazioni all'interno di un'organizzazione.

Agli albori dell'era digitale, ogni software registrava informazioni all'interno di **file** strettamente associato al software stesso. Quest'ultimo veniva pertanto scritto tramite un linguaggio di programmazione basato sulla gestione dei file, in modo da poter ottimizzare l'accesso ai dati richiesti. Ovviamente, tale soluzione presenta degli svantaggi, tra cui:

- **Ridondanza**: se due applicazioni usano gli stessi dati, essi devono essere replicati su entrambi i file
- **Inconsistenza**: l'aggiornamento di un dato potrebbe avvenire su una sola copia del dato condiviso
- **Dipendenza dei dati**: ogni applicazione organizza i dati in base alle proprie necessità, senza seguire uno standard

Nel corso degli anni, venne sviluppato il concetto di **Database (DB)**, ossia un insieme di file interconnessi tra loro, dove i dati sono organizzati in differenti strutture dati che ne facilitano l'utilizzo e ne ottimizzano la gestione.

Per facilitare maggiormente l'uso di tali DB, vennero sviluppati dei software chiamati **Database Management System (DBMS)** in grado di gestire di grandi quantità di

dati fornendo all'utente una visione astratta dei dati, svolgendo le operazioni richieste dall'utente senza che quest'ultimo si preoccupi di interrogare direttamente il DB.

La struttura dell'informazione dipende dai suoi usi ed è soggetta a cambiamenti nel corso del tempo. Ad esempio, i dati di una persona (nome, cognome, data di nascita, codice fiscale, ...) possono cambiare nel tempo.

L'obiettivo è quindi quello di semplificare l'elaborazione dei dati strutturati sfruttandone le proprietà:

- Gli accessi individuali agli elementi della struttura vengono realizzati tramite delle **query** (o interrogazioni)
- Le relazioni tra dati individuali vengono rappresentate tramite un **record** strutturato

In un'organizzazione, ogni componente di essa è interessato ad accedere ad una porzione del sistema informativo, richiedendo quindi che esso venga **condiviso**, riducendo la ridondanza dei dati. La condivisione di tale accesso ai dati deve essere **regolamentata**, richiedendo che ogni componente possa accedere solo ai dati di sua competenza. Tuttavia, la condivisione dei dati stessi può generare problemi di **concorrenza**, dovuta all'accesso simultaneo al sistema da parte di più componenti.

1.2 Modellazione dei dati

I dati vengono quindi organizzati concettualmente in **aggregati di informazioni omogenee** che costituiscono i componenti del sistema informativo. Ogni operazione di aggiornamento dei dati è mirata ad singolo insieme aggregato, mentre una query potrebbe coinvolgere più di un insieme aggregato. Nel caso particolare dei database, gli aggregati di informazioni omogenee vengono costituiti da file, i quali vengono indicizzati da **indici**, ossia ulteriori file che permettono di accedere rapidamente alle informazioni contenute nei file "principali".

A seconda dell'ambito, quindi, i dati assumono **due modelli** diversi:

- **Modelli logici**: indipendenti dalla struttura fisica dei dati e gestiti dal DBMS stesso. Il modello logico che verrà approfondito dettagliatamente in questo corso è il **modello relazionale**, basato sul concetto di relazione matematica. Altri modelli logici utilizzati sono il modello gerarchico, ad oggetti o a rete.
- **Modelli concettuali**: indipendenti dalle modalità di realizzazione. Hanno lo scopo di rappresentare le entità del mondo reale e le relazioni tra di essi. Il modello concettuale legato al modello logico relazionale è il **modello di entità-relazione (E-R)**.

Possiamo riassumere i concetti descritti attraverso un'**astrazione a tre livelli**:

- **Schema esterno**: descrizione di una porzione del database in un modello logico attraverso "viste" parziali ed un'organizzazione dei dati diversa o coincidente da

quella dello schema logico. Tale schema non dipende dallo schema fisico e gli accessi al database possono essere effettuati solo attraverso esso.

- **Schema logico:** descrizione dell'intero database nel principale modello logico del DBMS, ad esempio la struttura delle tabelle. Non dipende dallo schema esterno e da quello fisico.
- **Schema fisico:** rappresentazione dello schema logico attraverso il salvataggio in strutture fisiche, ad esempio in file.

In ogni database vengono definiti uno **schema**, descrivente la struttura dei dati salvati al suo interno e le **istanze** di tale schema, ossia i valori correnti, i quali possono cambiare anche molto velocemente.

Nelle sezioni successive vedremo meglio come all'interno del **modello relazionale** lo schema e le istanze vengano definiti tramite **tabelle**, dove la loro intestazione corrisponde allo schema e il loro contenuto corrisponde alle istanze di tale schema. Ad esempio, nella tabella sottostante lo schema corrisponde a (NAME, SURNAME, BIRTH, TOWN), mentre le righe della tabella corrispondono alle istanze:

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

1.3 Linguaggi, Integrità e Transazioni

Per la gestione e la descrizione di un database, vengono impiegati dei linguaggi specifici:

- **Data definition language (DDL)**, per la definizione degli schemi
- **Data manipulation language (DML)**, per le interrogazioni e gli aggiornamenti dei dati
- **Structured Query Language (SQL)**, un linguaggio standardizzato che racchiude DDL e DML, basato sul modello relazionale.

Tramite tali linguaggi è possibile definire quelli che vengono chiamati **vincoli di integrità**, ossia delle imposizioni che essi devono rispettare a seconda del contesto. Alcuni esempi sono:

- **Dipendenze funzionali:** uno studente può risiedere in una sola città
- **Vincoli di chiave:** il numero di matricola identifica univocamente uno studente
- **Vincoli di dominio:** un voto può essere solo un numero intero tra 18 e 30
- **Vincoli di dinamicità:** il salario di un impiegato non può diminuire

Fondamentale nell'ambito dei database è il concetto di **transazione**, ossia una sequenza di singole operazioni che possono avere solo due esiti possibili: eseguita completamente (**committed**) oppure annullata (**rolled back**). Ad esempio, la transazione corrispondente a "Trasferisci 1000€ dall'account C100 all'account C200" sarà:

1. Cerca C100
2. Sottrai 1000€ al bilancio di C100
3. Cerca C200
4. Aggiungi 1000€ al bilancio di C200

Nel caso in cui l'account C200 non venga trovato, è necessario effettuare un **rollback della transazione**, annullando le due operazioni precedentemente effettuate. Affinché ciò sia possibile, è necessario un **transaction log**, contenente i valori precedenti e successivi alle modifiche effettuate.

Oltre alla possibilità di annullare le operazioni svolte tramite le transazioni, è necessario che all'interno di un database venga gestita la **competizione** tra le varie transazioni.

Ad esempio, immaginiamo vengano eseguite in contemporanea le seguenti query:

- Transazione 1: "Accredita 1000€ al conto C1"
- Transazione 2: "Accredita 500€ al conto C1"

Transazione 1	Tempo	Transazione 2
Ricerca C1	T1	Ricerca C1
Cambia bilancio in: Bilancio+1000	T2	
	T3	
	T4	
		Cambia bilancio in: Bilancio+500

In tal caso, dato un bilancio iniziale pari a 2500, per via della **competizione** il bilancio finale sarà 3000 e non 4000.

2

Modello relazionale

Il modello relazionale, proposto per la prima volta da E. F. Codd nel 1970, è un modello di database basato sul concetto matematico di **relazione**, le quali vengono tradotte in tabelle memorizzate all'interno del database. In particolare, i dati e le relazioni (o associazioni) tra essi e dati di insiemi esterni vengono rappresentati come valori.

Prima di procedere, è necessario introdurre alcune definizioni:

Definizione 2.1: Dominio

Definiamo come **dominio** un insieme (possibilmente finito) di valori utilizzabili. Ad esempio, l'insieme dei numeri interi e l'insieme di tutte le stringhe contenenti 20 caratteri sono due domini.

Definizione 2.2: Prodotto cartesiano

Siano D_1, D_2, \dots, D_k dei domini non necessariamente distinti tra loro. Il **prodotto cartesiano** di tali domini corrisponde all'insieme delle liste ordinate dei valori appartenenti ai vari domini:

$$D_1 \times D_2 \times \dots \times D_k : \{(v_1, v_2, \dots, v_k) \mid v_1 \in D_1, v_2 \in D_2, \dots, v_k \in D_k\}$$

Definizione 2.3: Relazione

Definiamo come **relazione** un qualsiasi sottoinsieme di un prodotto cartesiano.

$$r \subseteq D_1 \times \dots \times D_k$$

Se P è il prodotto cartesiano di k domini, allora $r \subseteq P$ viene detta **relazione di grado k** .

Definizione 2.4: Tuple di una relazione

Data una relazione r e un elemento $t \in r$, tale elemento viene detto **tupla**. La cardinalità di una relazione $|r|$, quindi, corrisponde al numero di tuple appartenenti ad essa.

Se r è una relazione di grado k , allora ogni tupla $t \in r$ possiede k valori al suo interno.

Osservazione 2.1

Poiché $r \subseteq P$, dove P è un prodotto cartesiano, per definizione matematica di sottoinsieme si ha che le tuple di una relazione sono **distinte tra di loro di almeno un valore**.

Esempio:

- Siano $D_1 : \{\text{white}, \text{black}\}$, $D_2 : \{0, 1, 2\}$. Il loro prodotto cartesiano sarà:

$$D_1 \times D_2 : \{(\text{white}, 0), (\text{white}, 1), (\text{white}, 2), (\text{black}, 0), (\text{black}, 1), (\text{black}, 2)\}$$

- La seguente relazione

$$r_1 \subseteq D_1 \times D_2 : \{(\text{white}, 1), (\text{black}, 0), (\text{black}, 1)\}$$

è una relazione di grado 2 e cardinalità 3

- La seguente relazione

$$r_2 \subseteq D_1 \times D_2 : \{(\text{black}, 0), (\text{black}, 2)\}$$

è una relazione di grado 2 e cardinalità 2

- Per comodità, vengono utilizzati domini già presenti all'interno della maggior parte dei linguaggi di programmazione. Ad esempio, la seguente relazione è un sottoinsieme del prodotto cartesiano tra i domini `String` \times `String` \times `Integer` \times `Real`:

$$r : \{(\text{Paolo}, \text{Rossi}, 2, 26.5), (\text{Mario}, \text{Bianchi}, 10, 28.7), \}$$

Come già accennato, le relazioni possono essere tradotte in **tabelle** di un database, dove ogni riga rappresenta una tupla:

Paolo	Rossi	2	26.5
Mario	Bianchi	10	28.7

Introduciamo quindi la seguente notazione:

Definizione 2.5: Elemento di una tupla

Data una relazione $r \subseteq D_1 \times \dots \times D_k$ di grado k e data una tupla $t \in r$, indichiamo il suo **i-esimo elemento** come $t[i]$, dove $i \in [1, k]$

Esempio:

- Se $r : \{(0, a), (0, c), (1, b)\}$ e $t \in r : (0, a)$, allora $t[1] = 0$, $t[2] = a$ e $t[1, 2] = (0, a)$

2.1 Da relazioni a tabelle

Abbiamo già visto come una relazione possa essere rappresentata sottoforma di tabella.

Tuttavia, rimane un problema da risolvere: come facciamo ad interpretare i dati contenuti in una tabella? Ci basta **assegnare un nome ad ogni colonna della tabella**, trasformando tali dati in informazione.

Introduciamo quindi ulteriori definizioni:

Definizione 2.6: Attributo

Definiamo come **attributo** la coppia $(A, dom(A))$, dove A corrisponde al nome dell'attributo e $dom(A)$ corrisponde al dominio ad esso associato.

Definizione 2.7: Schema relazionale

Definiamo come **schema relazionale**, indicato come R , l'insieme di tutti gli attributi $A \in R$ descrittivi la relazione associata allo schema stesso, indicato come:

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

Lo schema descrivente la struttura di una relazione è **invariante nel tempo**.

Definizione 2.8: Istanza di una relazione

Sia R uno schema relazionale. Definiamo come **istanza di una relazione** $R(X)$, dove $X \subseteq R$ è un sottoinsieme di attributi, un **insieme di tuple** r su X dove $\forall t \in r$ gli attributi di t corrispondono a X .

Un'istanza di una relazione contiene i **valori attualmente memorizzati**, i quali possono anche cambiare rapidamente.

Ogni tupla t definita su R può essere quindi vista come una funzione $t : R \rightarrow dom(A) : A \mapsto t(A)$ che associa ad ogni attributo $A \in R$ l'elemento $t(A) \in dom(A)$.

Esempio:

- Le seguenti due tabelle r_1 ed r_2 sono due istanze del seguente schema R :

$$R = \{(\text{Nome}, \text{String}), (\text{Cognome}, \text{String}), (\text{Esami sostenuti}, \text{Integer}), (\text{Media}, \text{Real})\}$$

Nome	Cognome	Esami sostenuti	Media
Paolo	Rossi	2	26.5
Mario	Bianchi	10	28.7

Nome	Cognome	Esami sostenuti	Media
Giada	Verdi	3	24.3
Luigi	Neri	14	29.8

Osservazione 2.2

Una **relazione** può essere quindi vista come una tabella in cui ogni riga corrisponde ad una tupla distinta ed ogni colonna corrisponde ad un componente con valori omogenei, ossia provenienti dallo stesso dominio.

Definizione 2.9: Schema di un database e Database relazionale

Definiamo come **schema di un database** un insieme di relazioni distinte

$$(R_1, R_2, \dots, R_n)$$

Definiamo come **database relazionale** uno schema di un database (R_1, R_2, \dots, R_n) su cui sono definite le istanze (r_1, r_2, \dots, r_n) , dove $\forall i \in [1, n]$ si ha che r_i è un'istanza di R_i .

Esempio:

- Schema relazione: Info_City(City, Region, Population)
- Istanza relazione:

City	Region	Population
Rome	Lazio	3000000
Milan	Lombardy	1500000
Genoa	Liguria	800000
Pisa	Tuscany	150000

Osservazione 2.3

Rispetto all'uso dell'indice relativo alla loro posizione all'interno di una tupla, nell'ambito del modello relazionale gli elementi di una tupla $t \in r$, dove r è istanza di R , vengono indicati tramite il nome dell'attributo ad essi associato.

Esempio:

- Considerando la tabella dell'esempio precedente, se $t_1 \in r$ è la prima tupla dell'istanza e $t_2 \in r$ è la seconda tupla dell'istanza, allora:

$$t_1[\text{City}] = t_1[1] = \text{Rome}$$

$$t_2[\text{Region}] = t_2[2] = \text{Lombardy}$$

Definizione 2.10: Restrizione di una tupla

Sia $Y \subseteq X$ un sottoinsieme di attributi dello schema X e sia r un'istanza di X .

Indichiamo come $t[Y]$ la **restrizione** di $t \in r$, ossia il sottoinsieme di valori di t corrispondenti agli attributi in Y .

- Considerando ancora la tabella dell'esempio precedente, se $t \in r$ è la seconda tupla dell'istanza e $Y = \{\text{City}, \text{Population}\}$, allora:

$$t[Y] = (\text{Milan}, 1500000)$$

2.2 Riferimenti e Valori nulli

Nel modello relazionale, i riferimenti reciproci tra dati presenti all'interno di diverse relazioni viene effettuato tramite valori.

Consideriamo le seguenti relazioni:

Students			
Matricola	Surname	Name	Date of birth
6554	Rossi	Mario	05/12/1978
8765	Neri	Paolo	03/11/1976
9283	Greens	Luisa	12/11/1979
3456	Rossi	Maria	01/02/1978

Exams		
Student	Grade	Course
3456	30	04
3456	24	02
9283	28	01

Courses		
Code	Title	Lecture
01	Chemistry	Mario
02	Math	Bruni
04	Chemistry	Verdi

- Notiamo come le tre tabelle possiedono dei valori che fanno **riferimento ad altre tuple di altre tabelle**.
- Ad esempio, la prima tupla t della tabella **Exams** contiene **due riferimenti**: $t[\text{Student}]$ fa riferimento al campo **Matricola** dell'ultima tupla della tabella **Students**, mentre $t[\text{Course}]$ fa riferimento all'ultima tupla della tabella **Courses**.
- Considerando i due riferimenti, quindi, riusciamo a **ricostruire l'informazione completa**: la studentessa Maria Rossi, nata il 01/02/1978 e avente matricola 3456, ha ottenuto un voto pari a 30 all'esame di Chimica, tenuto dal docente Verdi

Spesso può capitare di non avere ancora tutte le informazioni relative ad una determinata tupla. Immaginiamo di avere una tabella **Students**, i cui attributi comprendono anche un campo **Phone number**. Potrebbe verificarsi una delle seguenti tre situazioni:

- Lo studente non ha un numero di telefono
- Non sappiamo se lo studente abbia un numero di telefono
- Lo studente ha un numero di telefono, ma non sappiamo quale sia

Poiché la tupla deve aderire allo schema della relazione imposto, non possiamo non inserire un valore all'interno di tale campo. Dunque, inseriamo un **valore di default**, chiamato **Null**.

È necessario sottolineare alcuni aspetti riguardo al **valore Null**:

- Il valore Null **non corrisponde ad uno zero**
- Non dovrebbe mai essere utilizzato per campi fondamentali, ad esempio una matricola
- È un valore *polimorfo*, ossia non appartiene ad alcun dominio ma può rimpiazzare un valore di qualsiasi dominio
- Due valori Null, anche se sullo stesso dominio, vengono considerati diversi l'uno dall'altro
- Utilizzare troppi valori Null viene considerata una pessima abitudine

2.3 Vincoli di integrità

Consideriamo il seguente database:

Employees					
Code	Surname	Name	Role	Hiring	Department
COD1	Rossi	Mario	Analyst	1795	01
COD2	Bianchi	Luigi	Analyst	1990	05
COD2	Neri	Paolo	Admin	1985	01

Departments	
Number	Name
01	Management
02	Administration

Nonostante tale database risulti essere sintatticamente corretto, risultano alcune **incoerenze** nella tabella **Employees**:

- La prima tupla contiene il valore 1795 nell'attributo **Hiring** indicante l'anno di assunzione del dipendente, il che non ha senso logico
- La seconda tupla fa riferimento al dipartimento numero 05, il quale non esiste nella tabella **Departments**
- La seconda e la terza tupla contengono lo stesso valore nell'attributo **Code**, il quale invece dovrebbe rappresentare in modo univoco un dipendente

Per evitare errori di questo tipo, imponiamo sul database dei **vincoli di integrità**

Definizione 2.11: Vincolo di integrità

Definiamo come **vincolo di integrità** delle proprietà che devono essere soddisfatte da ogni istanza di un database. Un database viene detto **corretto** se soddisfa tutti i vincoli di integrità associati al suo schema.

I vincoli di integrità possono essere di due tipologie:

- **Intrarelazionale**, ossia definiti su una singola relazione, in particolare sugli attributi del suo schema o tra le tuple della sua istanza.
- **Interrelazionale**, ossia definiti tra più relazioni.

Proposizione 2.1

In particolare, individuiamo i seguenti vincoli di integrità:

- **Vincoli di dominio**, ossia delle restrizioni imposte sul dominio di un attributo della relazione
- **Vincoli di tupla**, ossia delle proprietà che devono essere rispettate da ogni tupla appartenente ad un'istanza di una relazione
- **Vincoli di unicità**, ossia l'impossibilità di avere due tuple con lo stesso valore per un determinato attributo
- **Vincoli di esistenza del valore**, ossia l'impossibilità per un attributo di una tupla di poter essere impostato su Null

Esempio:

- Considerando il database dell'esempio precedente, possiamo imporre i seguenti vincoli di integrità:

- `Hiring > 1980` (vincolo di dominio)
- `Name, Surname Not Null` (vincolo di esistenza del valore)
- `Unique Employees.Code` (vincolo di unicità)

2.3.1 Chiavi primarie e secondarie

Definizione 2.12: Chiave relazionale

Un **insieme X di attributi** appartenenti ad una relazione R viene detto **chiave di R** se:

1. Per ogni istanza r di R , si ha che $\forall i \neq j, \nexists t_i, t_j \in r \mid t_i[X] = t_j[X]$, ossia non esistono tuple distinte aventi gli stessi valori per tutti gli attributi di X (vincolo di unicità)
2. Non esiste un sottoinsieme $Y \subset X$ che soddisfa la prima condizione

Definizione 2.13: Chiave primaria

Data una relazione R , definiamo come **chiave primaria** la chiave più utilizzata (o consistente di un minor numero di attributi).

Una chiave primaria, oltre al vincolo di unicità, deve rispettare anche il vincolo di esistenza del valore (**vincolo di chiave primaria**)

Esempio:

- Consideriamo la seguente relazione:

Employees						
Tax Code	Code	Surname	Name	Role	Hiring	Department
RSS...	COD1	Rossi	Mario	Analyst	1795	01
BNC...	COD2	Bianchi	Luigi	Analyst	1990	05
NRI...	COD3	Neri	Paolo	Admin	1985	01

- Cerchiamo di individuare alcune possibili chiavi all'interno di essa:
 - L'attributo `Tax Code` non è una chiave poiché, nonostante sia raro, potrebbe accadere che più dipendenti abbiano lo stesso codice fiscale
 - L'insieme (`Surname, Name, Role, Hiring`) può essere una chiave
 - L'insieme (`Surname, Role, Hiring, Department`) può essere una chiave

- L'attributo **Code** può essere una chiave, implicando che qualsiasi insieme di attributi X comprendente anche **Code** non possa essere chiave, poiché non soddisferebbe la seconda condizione.
- Siccome **Code** è la chiave più piccola individuata, allora essa sarà la chiave primaria di tale relazione, indicata come

Employees(Code, Tax Code, Surname, Name, Role, Hiring, Department)

Definizione 2.14: Vincolo di chiave esterna (o di riferimento)

Siano R_1, R_2 due relazioni, dove $X \subseteq R_1$ è una chiave primaria di R_1 . Definiamo come **vincolo di chiave esterna** (o di riferimento) l'obbligo per un insieme di attributi $Y \subseteq R_2$ di assumere valori presenti all'interno di X .

In altre parole, se r_1 ed r_2 sono rispettivamente un'istanza di R_1 ed R_2 si ha che:

$$\forall t_2 \in r_2, \exists t_1 \in r_1 \mid t_2[Y] = t_1[X]$$

Esempio:

- Consideriamo le seguenti relazioni:

Road Violations				
Code	Date	Officer	Prov	Plate
34321	1/2/95	3987	MI	39548K
53524	4/3/95	3295	TO	E39548
64521	5/4/96	3295	PR	839548
73321	5/2/98	9345	PR	839548

Officers		
ID	Surname	Name
3987	Bianchi	Luca
3295	Gialli	Piero
9345	Rossi	Mario
7543	Verdi	Luigi

Cars			
Prov	Plate	Surname	Name
MI	39548K	Perini	Paolo
TO	E39548	Ascani	Marco
RM	M2931D	Rossi	Mario

- Individuiamo le seguenti chiavi primarie:
 - **RoadViolation**(Code, Date, Officer, Prov, Plate)
 - **Officers**(ID, Surname, Name)
 - **Cars**(Prov, Plate, Surname, Name)
- Applichiamo quindi i seguenti vincoli di chiave secondaria:
 - **Road_Violations.Officer** References **Officers.ID**
 - **Road_Violations.(Prov, Plate)** References **Cars.(Prov, Plate)**

- Una volta applicati tali vincoli, notiamo come alcune tuple di **Road Violations** rendano tale istanza invalide:
 - Se r è l'istanza di **Road Violations**, c è l'istanza di **Cars** e t_3, t_4 sono rispettivamente la terza e la quarta tupla di r , allora si ha che:

$$\nexists t \in c \mid t_3[\text{Prov}, \text{Plate}] = t[\text{Prov}, \text{Plate}]$$

$$\nexists t \in c \mid t_4[\text{Prov}, \text{Plate}] = t[\text{Prov}, \text{Plate}]$$
 - In altre parole, non esiste una tupla nell'istanza di **Cars** avente (PR, 839548) come valori di (Prov, Plate)

2.4 Anomalie di un database

Proposizione 2.2: Anomalie di un database

Se uno schema relazionale è **mal progettato**, potrebbero verificarsi le seguenti anomalie:

- **Anomalie di inserimento**, dove non è possibile inserire un dato senza inserire altre informazioni non vere o non disponibili
- **Anomalie di cancellazione**, dove la cancellazione di un record potrebbe portare alla perdita di altre informazioni importanti
- **Anomalie di aggiornamento**: dove l'aggiornamento di un attributo potrebbe richiedere un costo elevato in quanto andrebbe aggiornato per ogni tupla contenente lo stesso valore in tale attributo
- **Ridondanza dei dati**, dove vi sono molti dati ripetuti

Esempio:

- Consideriamo il seguente schema **Università**(Matr, Nome, Città, Prov, C#, Titolo, Docente, C_laurea, Data, Voto)
- Dunque, una tupla rappresenta uno studente con matricola m e nome n , residente nella città c che si trova in provincia di p , avente sostenuto l'esame del corso con codice c e titolo t tenuto dal docente d del corso di laurea l in data d , ottenendo la votazione v .
- Notiamo la presenza di alcune anomalie all'interno di tale relazione:
 - Anomalie di inserimento, poiché non è possibile inserire i dati di uno studente senza che esso abbia sostenuto un esame
 - Anomalie di cancellazione, poiché nel caso in cui i dati di un corso debbano essere cancellati (ad esempio durante la rimozione dell'insegnamento di un corso dal piano di studi) e ci sia uno studente avente sostenuto solo l'esame di quel corso, allora verranno perse le informazioni relative allo studente

- Anomalie di aggiornamento, poiché nel caso in cui si debba modificare il docente di un corso è necessario modificarlo per ogni tupla in cui tale corso compare
- Ridondanza dei dati, poiché le informazioni sui corsi e sugli studenti sono ripetute molteplici volte
- Tali anomalie derivano da una mal progettazione della relazione, poiché essa esprime più concetti al suo interno
- Per risolvere tale anomalie, sarebbe più opportuno decomporre tale schema in più sottoschemi, ognuno rappresentante un concetto: uno schema `Studente(Mat, Nome, Città, Prov)`, uno schema `Corso(C#, Titolo, Docente, C_laurea)` ed uno schema `Esame(Mat, C#, Data, Voto)`

3

Algebra relazionale

Definizione 3.1: 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.

3.1 Proiezione e Selezione

Definizione 3.2: Proiezione

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 $\pi_{\text{Name}}(\text{Customers})$, il cui output sarà:

$\pi_{\text{Name}}(\text{Customers})$
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:

$\pi_{\text{Name}, \text{C\#}}(\text{Customers})$	
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

Definizione 3.3: Selezione

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

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

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

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

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 $\sigma_{\text{Nome}='Rossi' \wedge \text{Town}='Roma'}(\text{Customers})$ per ottenere le tuple richieste:

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

3.2 Unione, Intersezione e Differenza

Definizione 3.4: Compatibilità in unione

Data una relazione R_1 con schema $R_1(A_1, \dots, A_k)$ ed una relazione R_2 con schema $R_2(A_1, \dots, A_h)$, tali relazioni vengono dette **compatibili in unione** se e solo se:

- $k = h$, ossia hanno lo stesso numero di attributi
- $\forall i \in [1, k], \text{dom}(R_1.A_i) = \text{dom}(R_2.A_i)$, ossia ogni attributo corrispondente ha lo stesso dominio

Definizione 3.5: Unione

Date due relazioni **compatibili in unione** R_1 e R_2 con rispettive istanze r_1 e r_2 , l'**unione** tra R_1 e R_2 è un operatore binario che restituisce una nuova relazione contenente tutte le tuple presenti in almeno una relazione tra R_1 e R_2 .

$$R_1 \cup R_2 := \{t \mid t \in r_1 \vee t \in r_2\}$$

Osservazione 3.1

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)

Esempi:

1. • 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

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

3. • 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	ServiceYears
Esposito	C1	3
Riccio	C2	13
Pierro	C3	7

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

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

Definizione 3.6: Intersezione

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 .

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

Definizione 3.7: Differenza

Date due relazioni **compatibili in unione** R_1 e R_2 con rispettive istanze r_1 e r_2 , la **differenza** tra R_1 e R_2 è un operatore binario che restituisce una nuova relazione contenente tutte le tuple presenti in R_1 ma non in R_2 .

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

Osservazione 3.2

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	C3	English
Bianchi	C4	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

3.3 Ridenominazione e Prodotto Cartesiano

Definizione 3.8: Ridenominazione

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.

Definizione 3.9: Prodotto Cartesiano

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#	Qty
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.

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

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

Customers \times 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:

$\sigma_{\text{C}\#=\text{CC}\#}(\text{Customers} \times \text{OrdersR})$						
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:

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

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

CustomerOrders					
Name	C#	Town	O#	A#	Qty
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

3.4 Join Naturale e Theta Join

Definizione 3.10: Join Naturale

Siano R_1 e R_2 due relazioni con rispettive istanze r_1 e r_2 e rispettivi schemi $R_1(X)$ e $R_2(Y)$. Il **join naturale** tra R_1 e R_2 è un operatore binario equivalente a:

$$R_1 \bowtie R_2 = \pi_{X, (Y-X)}(\sigma_{\varphi}(R_1 \times R_2))$$

dove dato un insieme di attributi $A_1, \dots, A_k \mid \forall i \in [1, k], A_i \in X \cap Y$ si ha che:

$$\varphi := R_1.A_1 = R_2.A_1 \wedge \dots \wedge R_1.A_k = R_2.A_k$$

In altre parole, il join naturale tra R_1 ed R_2 restituisce l'insieme di tutte le combinazioni tra tuple di r_1 ed r_2 che sono uguali per i loro attributi in comune.

Esempi:

- 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#	Qty
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 \bowtie Orders					
Name	C#	Town	O#	A#	Qty
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

2. • 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#	Qty	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:

$\sigma_{Qnty>100 \wedge Price>2}(CustOrdArt)$							
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:

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

- 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:

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

Proposizione 3.1: Casi speciali del join naturale

Siano R_1 e R_2 due relazioni con rispettivi schemi $R_1(X)$ e $R_2(Y)$.

1. Se R_1 ed R_2 hanno un insieme di attributi in comune ma nessun valore in comune per tali attributi, allora il risultato sarà un insieme vuoto:

$$Z \subseteq X \cap Y, \nexists t_1 \in R_1 \wedge t_2 \in R_2 \mid t_1[Z] = t_2[Z] \implies R_1 \bowtie R_2 = \emptyset$$

2. Se R_1 ed R_2 non hanno degli attributi in comune, allora il join naturale degenererà in un prodotto cartesiano:

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

Definizione 3.11: Theta Join

Siano R_1 e R_2 due relazioni con rispettive istanze r_1 e r_2 e rispettivi schemi $R_1(X)$ e $R_2(Y)$. Il **theta join** tra R_1 e R_2 è un operatore binario equivalente a:

$$R_1 \bowtie_{A\theta B} R_2 = \sigma_{A\theta B}(R_1 \times R_2)$$

dove:

- $A \in R_1(X), B \in R_2(Y)$
- $\text{dom}(A) = \text{dom}(B)$
- θ è un operatore di confronto ($<, \leq, =, \geq, >$)

In altre parole, un theta join tra R_1 ed R_2 restituisce tutte le combinazioni tra le tuple di r_1 e r_2 che rispettano una condizione su un attributo in comune

Osservazione 3.3

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:

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

$$\text{EmpSup}_1 = \text{EmployeesC} \bowtie_{\text{EmployeesC.Superisor\#} = \text{Employees.C\#}} \text{Employees}$$

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

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

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

$$\text{EmpSup}_2 = \text{Employees} \bowtie_{\text{Supervisor\#} = \text{C\#}} \rho_{X \leftarrow Y}(\text{Employees})$$

- *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 doppiati
- Successivamente, eseguiamo la selezione richiesta, mantenendo solo le tuple dove il salario del dipendente è maggiore del salario del suo supervisore:

$$\sigma_{\text{Employees.Salary} > \text{EmployeesC.Salary}}(\text{EmpSup}_1)$$

oppure

$$\sigma_{\text{Salary} > \text{CSal}}(\text{EmpSup}_2)$$

$\sigma_{\text{Salary} > \text{CSal}}(\text{EmpSup}_2)$									
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:

$$\pi_{\text{Employees.Name}, \text{Employees.C\#}}(\sigma_{\text{Employees.Salary} > \text{EmployeesC.Salary}}(\text{EmpSup}_1))$$

oppure

$$\pi_{\text{Name}, \text{C\#}}(\sigma_{\text{Salary} > \text{CSal}}(\text{EmpSup}_2))$$

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

3.5 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.

Osservazione 3.4

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

In simboli, diremmo che:

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

ma bensì:

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

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

Esempi:

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

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

- Per 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 supervisor 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 supervisor aventi il salario minore di almeno un dipendente:

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

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

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

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

R	
Name	C#
Bianchi	C3

Teoria della normalizzazione

4.1 Dipendenze funzionali

Definizione 4.1: Dipendenza funzionale

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 letta " 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 :

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

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:

1. • Supponiamo di avere il seguente schema `Flights(Code, Day, Pilot, Time)`. Viene naturale considerare i seguenti vincoli:
 - Un volo con un certo codice partirà 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:
 - $\text{Code} \rightarrow \text{Time}$
 - $(\text{Day}, \text{Pilot}, \text{Time}) \rightarrow \text{Code}$
 - $(\text{Code}, \text{Day}) \rightarrow \text{Pilot}$

Definizione 4.2: 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:

1. • 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 illegale 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

4.1.1 Chiusura di F **Definizione 4.3: 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 .

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

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

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 $A \rightarrow B, B \rightarrow C, A \rightarrow C \in F^+$

Osservazione 4.1

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

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

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

Proposizione 4.1

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

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

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

Dimostrazione:

- Siano $X, Y \subseteq R$, dove $Y = \{A_1, \dots, A_k\}$.
- Data r una qualsiasi istanza di R , si ha che:

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

□

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:

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

4.2 Assiomi di Armstrong

Definizione 4.4: 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** ($F \subseteq F^A$):

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

- **Assioma di riflessività**:

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

- **Assioma di aumento**:

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

- **Assioma di transitività**:

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

Proposizione 4.2: 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:**

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

- **Regola della decomposizione:**

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

- **Regola della pseudo-transitività:**

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

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 4.3

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:

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

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

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

□

4.2.1 Chiusura di X

Definizione 4.5: 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 a F**, indicata con X_F^+ (o solo X^+ se F è l'unico insieme di dipendenze su R), il seguente insieme:

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

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

Lemma 4.1

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 4.1

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.

4.2.2 $F^+ = F^A$ **Teorema 4.1: $F^+ = F^A$**

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

$$F^+ = F^A$$

Dimostrazione:

- Dimostriamo che $F^A \subseteq F^+$:
 - **Caso base** ($n = 0$): se $X \rightarrow Y \in F^A$ senza aver applicato alcun assioma di Armstrong, allora l'unica possibilità è che:

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

Siccome $X \rightarrow Y \in F$, allora

$$X \rightarrow Y \in F \implies X \rightarrow Y \in F^+$$

- **Ipotesi induttiva forte:** ogni dipendenza funzionale in F^A ottenuta da F applicando $k \leq n$ assiomi di Armstrong è anche in F^+

$$X \rightarrow Y \in F^A \text{ tramite } k \leq n \text{ assiomi} \implies X \rightarrow Y \in F^+$$

- **Passo induttivo** ($n > 0$): è necessario dimostrare che se $X \rightarrow Y \in F^A$ dopo aver applicato $n + 1$ assiomi di Armstrong, allora $X \rightarrow Y \in F^+$.

È possibile ritrovarsi in uno dei seguenti tre casi:

1. Se l'($n + 1$)-esimo assioma applicato è l'assioma di riflessività, allora l'unica possibilità è che $X \rightarrow Y \in F^A \iff Y \subseteq X \subseteq R$.

Dunque, poiché, $Y \subseteq X \subseteq R$, per ogni istanza legale di R si ha che:

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

da cui ne segue automaticamente che $X \rightarrow Y \in F^+$:

2. Se l'($n + 1$)-esimo assioma applicato è l'assioma di aumento, allora è obbligatoriamente necessario che:

* $\exists V, W \subseteq R \mid \exists V \rightarrow W \in F^A$, ottenuta applicando $j \leq n$ assiomi di Armstrong

* $\exists Z \subseteq R \mid X := VZ, Y := WZ$

affinché si abbia che:

$$Z \subseteq R, V \rightarrow W \implies VZ \rightarrow WZ = X \rightarrow Y \in F^A$$

Siccome per ipotesi induttiva si ha $V \rightarrow W \in F^A \implies V \rightarrow W \in F^+$ e siccome $Z \subseteq Z \implies Z \rightarrow Z \in F^+$, si vede facilmente che:

$$\begin{aligned} \left\{ \begin{array}{l} V \rightarrow W \in F^+ \\ Z \rightarrow Z \in F^+ \end{array} \right. &\implies \left\{ \begin{array}{l} \forall t_1, t_2 \in r, t_1[V] = t_2[V] \implies t_1[W] = t_2[W] \\ \forall t_1, t_2 \in r, t_1[Z] = t_2[Z] \implies t_1[Z] = t_2[Z] \end{array} \right. \implies \\ &\implies \forall t_1, t_2 \in r, t_1[VZ] = t_2[VZ] \implies t_1[WZ] = t_2[WZ] \implies \\ &\implies \forall t_1, t_2 \in r, t_1[X] = t_2[X] \implies t_1[Y] = t_2[Y] \implies X \rightarrow Y \in F^+ \end{aligned}$$

3. Se l'($n + 1$)-esimo assioma applicato è l'assioma di transitività, allora è obbligatoriamente necessario che $\exists X \rightarrow Z, Z \rightarrow Y \in F^A$, ottenute con $k \leq n$ assiomi di Armstrong, affinché si abbia che:

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

Siccome per ipotesi induttiva $X \rightarrow Z \in F^A \implies X \rightarrow Z \in F^+$ e $Z \rightarrow Y \in F^A \implies Z \rightarrow Y \in F^+$, si vede facilmente che:

$$\begin{aligned} \left\{ \begin{array}{l} X \rightarrow Z \in F^+ \\ Z \rightarrow Y \in F^+ \end{array} \right. &\implies \forall t_1, t_2 \in r, t_1[X] = t_2[X] \\ &\implies t_1[Z] = t_2[Z] \implies t_1[Y] = t_2[Y] \implies X \rightarrow Y \in F^+ \end{aligned}$$

- Dimostriamo che $F^+ \subseteq F^A$:

– Sia $X \subseteq R$ e sia r istanza di $R(X^+, R - X^+)$ tale che

X^+			$R - X^+$		
A_1	\dots	A_i	A_j	\dots	A_n
1	\dots	1	1	\dots	1
1	\dots	1	0	\dots	0

dunque tale che $\forall t_1, t_2 \in r$ si ha:

- * $t_1[X^+] = (1, \dots, 1) = t_2[X^+]$
- * $t_1[R - X^+] = (1, \dots, 1) \neq (0, \dots, 0) = t_2[R - X^+]$

– Notiamo che $\forall V, W \subseteq R \mid V \rightarrow W \in F$ si ha che:

- * Se $V \cap R - X^+ \neq \emptyset$ (dunque anche se $V \subseteq R - X^+$) allora $t_1[V] \neq t_2[V]$, dunque r soddisfa $V \rightarrow W \in F$
- * Se invece $V \subseteq X^+$, per il lemma precedentemente visto si ha che

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

Siccome $V \rightarrow W \in F \implies V \rightarrow W \in F^A$, per transitività si ha che

$$X \rightarrow V \in F^A \wedge V \rightarrow W \in F^A \implies X \rightarrow W \in F^A \iff W \subseteq X^+$$

Dunque, siccome $V, W \subseteq X^+$, in definitiva si ha che

$$t_1, t_2 \in r, t_1[V] = (1, \dots, 1) = t_2[V] \wedge t_1[W] = (1, \dots, 1) = t_2[W]$$

e quindi r soddisfa ogni $V \rightarrow W \in F$

- Siccome in entrambi i casi r soddisfa ogni $V \rightarrow W \in F$, allora r è legale.
- A questo punto, una qualsiasi dipendenza $X \rightarrow Y \in F^+$ deve essere soddisfatta da qualsiasi istanza legale di R , inclusa r stessa
- Poiché $X \subseteq X^+$, ne segue che la dipendenza non può essere soddisfatta a vuoto poiché $t_1[X] = t_2[X]$. Dunque, l'unica possibilità affinché $X \rightarrow A \in F^+$ sia soddisfatta da r è che $Y \subseteq X^+$ in modo che si abbia $t_1[Y] = t_2[Y]$
- A questo punto, per il lemma si ha che $Y \subseteq X^+ \iff X \rightarrow Y \in F^A$
- Dunque, siccome $X \rightarrow Y \in F^+ \implies X \rightarrow Y \in F^A$, concludiamo che $F^+ \subseteq F^A$

□

Osservazione 4.2

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 $|F^+| \gg 2^{|R|}$

4.3 Terza Forma Normale (3NF)

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

Definizione 4.6: 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:

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

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

Esempio:

- Consideriamo lo schema `Student(Matricola, LastName, FirstName, 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**

Definizione 4.7: 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:

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

Osservazione 4.3

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

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

Definizione 4.8: 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 X è superchiave o il determinato A è primo.

Se uno schema è in 3NF, la quantità di **anomalie** e di **ridondanze** dei dati (sezione 2.4) è **estremamente ridotta**.

Esempio:

1. • Sia $R = ABCD$ uno schema e sia $F = \{A \rightarrow B, B \rightarrow CD\}$ un insieme di 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 $B \rightarrow CD \in F^A = F^+ \implies B \rightarrow C, B \rightarrow D \in F^A = F^+$
 - 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:

$$A, C, AC \subseteq AC \implies AC \rightarrow A, AC \rightarrow C, AC \rightarrow AC \in F^A$$

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

$$AC \rightarrow B, B \rightarrow AD \in F^A \implies AC \rightarrow AD \in F^A$$

$$BC \rightarrow B, B \rightarrow AD \in F^A \implies BC \rightarrow AD \in F^A$$
 - Per unione:

$$AC \rightarrow C, AC \rightarrow B, AC \rightarrow AD \in F^A \implies AC \rightarrow ABCD = R \in F^A$$

$$BC \rightarrow B, BC \rightarrow C, BC \rightarrow AD \in F^A \implies BC \rightarrow ABCD \in F^A$$
 - Per aumento:

$$AC \rightarrow ABCD = R \in F^A \implies ABC \rightarrow ABBCD = ABCD = R \in F^A$$
- 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

Definizione 4.9: 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 $X \neq K$)

Definizione 4.10: 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** su R se:

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

Corollario 4.2: Definizione alternativa di 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 non esistono dipendenze parziali o transitive in F .

$$\nexists X \rightarrow Y \in F \mid X \rightarrow Y \text{ dip. parziale o transitiva}$$

Definizione 4.11: 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

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

Osservazione 4.4

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.

4.4 Calcolare X^+

Algoritmo 4.1: 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 4.2: Correttezza dell'algoritmo closureX

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 `closureX(R,F,X)` restituisce X_F^+

Dimostrazione:

- Siano $Z_0, Z_1, \dots, Z_i, \dots$ e $S_0, S_1, \dots, S_i, \dots$ gli insiemi calcolati ad ogni iterazione del ciclo while dell'algoritmo
- Osserviamo che $Z_i \subseteq Z_{i+1} \forall i \in \mathbb{N}$, dunque $Z_0, Z_1, \dots, Z_i, \dots$ è una sequenza monotona limitata da R , implicando che $\exists f \in \mathbb{N} \mid Z_f = Z_{f+1}$
- Siccome ciò può accadere solo se $S_f \subseteq Z_f$, ossia quando l'algoritmo termina, si ha che Z_f è l'output dell'algoritmo
- Dimostriamo quindi per induzione che $Z_f \subseteq X^+$:
 - **Caso base ($i = 0$):** Alla 0-esima iterazione del while (ossia prima di esso) si ha $Z_0 = X \subseteq X^+$
 - **Ipotesi induttiva:** Per ogni $i \in \mathbb{N}$ si ha che $Z_i \subseteq X^+$
 - **Passo induttivo ($i > 0$):** Dato $A \in Z_{i+1} := Z_i \cup S_i$, si ha che $A \in Z_i \vee A \in S_i$. Dunque, si possono verificare due casi:

- * Se $A \in Z_i$, allora per ipotesi si ha che $A \in Z_i \subseteq X^+$
- * Se $A \in S_i$, allora $\exists Y \rightarrow V \in F \mid A \in V \subseteq R, Y \subseteq Z_i$.

Siccome per ipotesi si ha $Z_i \subseteq X^+$ e siccome $Y \subseteq Z_i$, allora $Y \subseteq Z_i \subseteq X^+ \iff X \rightarrow Y \in F^A$ e siccome $Y \rightarrow V \in F \implies Y \rightarrow V \in F^A$, allora per transitività si ha che

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

Dunque, si ha che $A \in V \subseteq X^+$

- * Siccome in entrambi i casi $A \in Z_{i+1} \implies A \in X^+$, allora concludiamo che $Z_{i+1} \subseteq X^+$
- Dimostriamo ora che $X^+ \subseteq Z_f$:

- Sia $X \subseteq R$ e sia r istanza di $R(Z_f, R - Z_f)$ tale che

Z_f			$R - Z_f$		
A_1	\dots	A_i	A_j	\dots	A_n
1	\dots	1	1	\dots	1
1	\dots	1	0	\dots	0

dunque tale che per $t_1, t_2 \in r$ si ha:

- * $t_1[Z_f] = (1, \dots, 1) = t_2[Z_f]$
- * $t_1[R - Z_f] = (1, \dots, 1) \neq (0, \dots, 0) = t_2[R - Z_f]$
- Notiamo che $\forall V, W \subseteq R \mid V \rightarrow W \in F$ si ha che:
 - * Se $V \cap R - Z_f \neq \emptyset$ (dunque anche se $V \subseteq R - Z_f$) allora $t_1[V] \neq t_2[V]$, dunque r soddisfa $V \rightarrow W \in F$
 - * Se invece $V \subseteq Z_f$, allora $W \subseteq S_f$, poiché, per come viene calcolato S_f , si ha che:

$$V \rightarrow W \in F, V \subseteq Z_f, B \in W \subseteq R \implies B \in S_f \implies W \subseteq S_f$$

e dunque, siccome $S_f \subseteq Z_f$ è la condizione che termina l'algoritmo, allora $W \subseteq S_f \subseteq Z_f$

- * Siccome $V, W \subseteq Z_f$, in definitiva si ha che

$$t_1, t_2 \in r, t_1[V] = (1, \dots, 1) = t_2[V] \wedge t_1[W] = (1, \dots, 1) = t_2[W]$$

e quindi r soddisfa $V \rightarrow W \in F$

- Siccome in entrambi i casi r soddisfa ogni $V \rightarrow W \in F$, allora r è legale.
- A questo punto, dato $A \in X^+$ si ha che $X \rightarrow A \in F^A = F^+$ deve essere soddisfatta da qualsiasi istanza legale di R , inclusa r stessa

- Poiché $X = Z_0 \subseteq Z_f$, ne segue che la dipendenza non può essere soddisfatta a vuoto poiché $t_1[X] = t_2[X]$. Dunque, l'unica possibilità affinché $X \rightarrow A \in F^+$ sia soddisfatta da r è $A \in Z_f$ in modo che si abbia $t_1[A] = t_2[A]$
- Dunque, siccome $A \in X^+ \implies A \in Z_f$, concludiamo che $X^+ \subseteq Z_f$

□

4.4.1 Trovare le chiavi di uno schema

Proposizione 4.4

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

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

Dimostrazione:

- Sia $R = \{A_1, \dots, A_k\}$ e sia $X \subseteq R$. In tal caso, si ha che:

$$\begin{aligned} X \rightarrow R \in F^+ &\iff \forall i \in [1, k], X \rightarrow A_i \in F^+ = F^A \iff \\ &\iff \forall i \in [1, k], A_i \in X^+ \iff X^+ = \{A_i, \dots, A_k\} = R \end{aligned}$$
- Se $X \rightarrow R \in F^+$, le uniche possibilità sono:
 - $\exists Y \subseteq X \mid Y \text{ chiave di } R \implies X \text{ superchiave di } R$
 - $\nexists Y \subset X \mid Y \rightarrow R \in F^+ \implies X \text{ chiave di } R \implies X \text{ superchiave di } R$
- Dunque, possiamo concludere che $X \subseteq R \text{ superchiave di } R \iff X^+ = R$

□

Corollario 4.3

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

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

Proposizione 4.5

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

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

In altre parole, se A non è determinato da nessuna dipendenza funzionale in F , allora A apparterrà ad ogni chiave di R

Dimostrazione:

- Sia $R = A_1, \dots, A_j, \dots, A_k$ e sia F in insieme di dipendenze funzionali su R dove $\nexists X \rightarrow Y \in F \mid A \in Y$.
- Se $K = R$ fosse chiave di R , allora necessariamente si avrebbe che $A \in K = R$
- Supponiamo quindi per assurdo che $\exists K \subset R \mid K$ chiave di $R, A \notin K$
- Siccome K è chiave di R , allora $K^+ = R = A_1, \dots, A_j, \dots, A_k$, implicando necessariamente $K \rightarrow A_j \in F^A$.
- Tuttavia, si verifica che:
 - $K \rightarrow A_j \in F^A$ non può essere ottenuta tramite l'inclusione di F , poiché $K \rightarrow A_j \notin F$, siccome A_j non è determinato da alcuna dipendenza in F
 - $K \rightarrow A_j \in F^A$ non può essere ottenuta tramite riflessività, poiché implicherebbe necessariamente che $A_j \in K$
 - $K \rightarrow A_j \in F^A$ non può essere ottenuta tramite aumento, poiché implicherebbe necessariamente che $A_j \in K$
 - L'unica possibilità, quindi, è che $K \rightarrow A_j \in F^A$ sia ottenuta tramite transitività, implicando l'esistenza di $Y \subseteq R$ tale che

$$K \rightarrow Y, Y \rightarrow A_j \in F^A \implies K \rightarrow A_j \in F^A$$

- Affinché $Y \rightarrow A_j \in F^A$, si ha necessariamente che $\exists V \rightarrow W \in F \mid A_j \in V \vee A_j \in W$, poiché altrimenti non sarebbe possibile ricavare $Y \rightarrow A_j \in F^A$ applicando gli assiomi di Armstrong, contraddicendo l'ipotesi per cui A non appartiene a nessun determinante e nessun determinato di ogni dipendenza funzionale in F

□

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 trovare le chiavi del seguente schema
 - Siccome non esistono dipendenze funzionali in F per cui A, B ed H appaiono come determinato, necessariamente per ogni K chiave di R si ha che $A, B, H \in K$
 - Proviamo quindi a calcolare ABH^+ utilizzando l'algoritmo visto precedentemente:
 - Inizializziamo $Z = ABH$ e $S = CDE$
 - Alla prima iterazione abbiamo $Z = ABCDEH$ e $S = CDE$
 - Poiché alla seconda iterazione si avrebbe $S \subseteq Z$, allora $ABH^+ = ABCDEH$

- Difatti, otteniamo che $ABH^+ = ABCDEH = R$, implicando che ABH sia superchiave di R . Tuttavia, siccome per ogni K chiave di R si ha che $A, B, H \in K$, non possono esistere sottoinsiemi di ABH che siano chiave, implicando quindi che ABH sia chiave di R
 - Inoltre, siccome A, B, H sono in ogni chiave di R , ogni altro possibile insieme di attributi $X \subseteq R \mid X^+ = R$ corrisponderebbe ad una superchiave contenente ABH , dunque ABH è l'unica chiave di R
2. • 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 4.3: 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:

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

(dimostrazione omessa)

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 per 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 per 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 per 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

4.5 Decomposizione di uno schema

Definizione 4.12: Decomposizione di uno schema

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

$$R = \bigcup_{i=1}^k R_i$$

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

Osservazione 4.5

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.

Definizione 4.13: Buona decomposizione di uno schema

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

Definiamo ρ come una **buona decomposizione di R** se:

- Ogni sottoschema $R_1, \dots, R_k \in \rho$ è in **Terza Forma Normale**
- ρ 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**)
- ρ 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	
A	B
a_1	b_1
a_2	b_1

R_2	
A	C
a_1	c_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
2. • Consideriamo lo schema $R = ABC$ e l'insieme di dipendenze funzionali $F = \{A \rightarrow B, C \rightarrow B\}$
- In questo caso, notiamo facilmente che AC è l'unica chiave di R , implicando che R non sia in 3NF poiché né $A \rightarrow B$ né $C \rightarrow B$ rispettano la definizione di 3NF.
 - Consideriamo quindi la seguente istanza r di R :

r		
A	B	C
a_1	b_1	c_1
a_2	b_1	c_2

- Proviamo quindi a decomporre R in $\rho = \{AB, BC\}$, dove $F_1 = \{A \rightarrow B\}$ è l'insieme di dipendenze di $R_1 := AB$ mentre $F_2 = \{C \rightarrow B\}$ è l'insieme di dipendenze di $R_2 := BC$
- Le due sottoistanze ottenute corrispondono a:

R_1		R_2	
A	B	B	C
a_1	b_1	b_1	c_1
a_2	b_1	b_1	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_1	b_1	c_2
a_2	b_1	c_1
a_2	b_1	c_2

- Notiamo come nell'istanza ottenuta si siano andate a creare nuove tuple precedentemente non esistenti, implicando una **perdita di informazione** (controintuitivamente, la creazione di tuple prima non esistenti implica che sia stata persa l'informazione "tali tuple non esistevano")
- Dunque, tale decomposizione non risulta essere una buona decomposizione

Proposizione 4.6

Dato uno schema R con decomposizione $\rho = R_1, \dots, R_k$ 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.

4.5.1 Preservazione di F

Definizione 4.14: 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 $F^+ = G^+$

Lemma 4.2: 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:

- Denotiamo come $G \xrightarrow{A} F$ la possibilità di ottenere F partendo da G applicando una determinata quantità di assiomi di Armstrong
- Ricordando che G^A è l'insieme di tutte le dipendenze funzionali ottenibile applicando assiomi di Armstrong su G , allora:

$$G \xrightarrow{A} F \iff \forall X \rightarrow Y \in F \text{ si ha } X \rightarrow Y \in G^A = G^+ \iff F \subseteq G^+$$

- Siccome $F \subseteq G^+ \iff G \xrightarrow{A} F$, per definizione di $F^A = F^+$ si ha che

$$F \subseteq G^+ \implies G \xrightarrow{A} F \xrightarrow{A} F^A = F^+ \implies F^+ \subseteq G^+$$

- Viceversa, si ha che $F^+ \subseteq G^+ \implies F \subseteq F^+ \subseteq G^+$, quindi concludiamo che $F \subseteq G^+ \iff F^+ \subseteq G^+$

□

Teorema 4.4: Preservazione di F

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

Si ha che ρ **preserva** F se:

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

dove $\pi_{R_i}(F) = \{X \rightarrow Y \in F^+ \mid XY \subseteq R_i\}$ 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

Corollario 4.4

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

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

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

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

Dimostrazione:

- Poiché $\forall R_i \in \rho$ si ha che $\pi_{R_i}(F) \subseteq F^+$, per definizione stessa di G si ha sempre che $G \subseteq F^+$.
- Siccome $G \subseteq F^+ \iff G^+ \subseteq F^+$ e siccome $F \equiv G \iff F^+ = G^+$, allora è sufficiente verificare se $F \subseteq G^+$ affinché $F^+ = G^+$, poiché $F \subseteq G^+ \iff F^+ \subseteq G^+$

□

Algoritmo 4.2: 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
```

Osservazione 4.6

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 4.3: Calcolo di X_G^+ tramite F

Dato uno schema R con decomposizione $\rho = R_1, \dots, R_k$, 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 := 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 $S_2 = B \subseteq B = Z$, 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

Teorema 4.5: Correttezza dell'algoritmo X_G^+ _with_F

Sia R uno schema con decomposizione $\rho = R_1, \dots, R_k$ 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_G^+

Dimostrazione (solo un'implicazione):

- Siano $Z_0, Z_1, \dots, Z_i, \dots$ e $S_0, S_1, \dots, S_i, \dots$ gli insiemi calcolati ad ogni iterazione del ciclo while dell'algoritmo
- Osserviamo che $Z_i \subseteq Z_{i+1} \forall i \in \mathbb{N}$, dunque $Z_0, Z_1, \dots, Z_i, \dots$ è una sequenza monotona limitata da R , implicando che $\exists f \in \mathbb{N} \mid Z_f = Z_{f+1}$
- Siccome ciò può accadere solo se $S_f \subseteq Z_f$, ossia quando l'algoritmo termina, si ha che Z_f è l'output dell'algoritmo
- Dimostriamo quindi per induzione che $Z_f \subseteq X_G^+$:
 - **Caso base ($i = 0$):** Alla 0-esima iterazione del while (ossia prima di esso) si ha $Z_0 = X \subseteq X_G^+$
 - **Ipotesi induttiva:** Per ogni $i \in \mathbb{N}$ si ha che $Z_i \subseteq X_G^+$
 - **Passo induttivo ($i > 0$):** Dato $A \in Z_{i+1} := Z_i \cup S_i$, si ha che $A \in Z_i \vee A \in S_i$. Dunque, si possono verificare due casi:
 - * Se $A \in Z_i$, allora per ipotesi si ha che $A \in Z_i \subseteq X_G^+$
 - * Se $A \in S_i$, allora per definizione stessa di S_i si ha che $\exists j \leq k \mid A \in ((Z_i \cap R_j)_F^+ \cap R_j)$

A questo punto, si ha che $A \in ((Z_i \cap R_j)_F^+ \cap R_j) \iff A \in (Z_i \cap R_j)_F^+ \wedge A \in R_j$

da cui otteniamo che $A \in (Z_i \cap R_j)_F^+ \iff (Z_i \cap R_j) \rightarrow A \in F^A = F^+$

Dunque, siccome $(Z_i \cap R_j) \subseteq R_j$ e siccome $A \in R_j$, allora si ha che

$$(Z_i \cap R_j) \rightarrow A \in \pi_{R_j}(F) = \{X \rightarrow Y \in F^+ \mid XY \in R_j\}$$

da cui traiamo che $(Z_i \cap R_j) \rightarrow A \in \pi_{R_j}(F) \subseteq G \subseteq G^+ = G^A$

Inoltre, siccome $(Z_i \cap R_j) \subseteq Z_i$ e siccome per ipotesi induttiva $Z_i \subseteq X_G^+$, allora $(Z_i \cap R_j) \subseteq Z_i \subseteq X_G^+$, implicando quindi che $X \rightarrow (Z_i \cap R_j) \in G^A$

Infine, per transitività otteniamo che:

$$X \rightarrow (Z_i \cap R_j), (Z_i \cap R_j) \rightarrow A \in G^A \implies X \rightarrow A \in G^A \iff A \in X_G^+$$

- Dunque, siccome in entrambi i casi si ha che $A \in Z_f \implies A \in X_G^+$, allora $Z_f \subseteq X_G^+$

□

4.5.2 Join senza perdita

Teorema 4.6: 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:

$$r = m_\rho(r) := \pi_{R_1}(r) \bowtie \dots \bowtie \pi_{R_k}(r)$$

Proposizione 4.7

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

Posto $m_\rho(r) := \pi_{R_1}(r) \bowtie \dots \bowtie \pi_{R_k}(r)$, per ogni istanza legale r di R si ha che:

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

Dimostrazioni:

1. Data una qualsiasi tupla $t \in r$, si ha che:

$$t \in r \implies t \in \{t[R_1]\} \bowtie \dots \{t[R_k]\} \subseteq \pi_{R_1}(r) \bowtie \dots \bowtie \pi_{R_k}(r) = m_\rho(r)$$

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

2. Poiché $r \subseteq m_\rho(r)$, allora effettuando una proiezione con $R_i \in \rho$ su entrambe, ne segue che $\pi_{R_i}(r) \subseteq \pi_{R_i}(m_\rho(r))$

Inoltre, per definizione di proiezione, si ha che:

$$t_{R_i} \in \pi_{R_i}(m_\rho(r)) \implies \exists t' \in m_\rho(r) \mid t_{R_i} = t'[R_i]$$

Infine, per definizione stessa di $m_\rho(r)$, si ha che:

$$t' \in m_\rho(r) \implies \exists t_1, \dots, t_k \in r \mid \forall R_j \in \rho, t_j[R_j] = t'[R_j]$$

In particolare, quindi, otteniamo che:

$$t_{R_i} \in \pi_{R_1}(m_\rho(r)) \implies t_{R_i} = t'[R_i] = t_i[R_i] \in \pi_{R_i}(r) \implies \pi_{R_1}(m_\rho(r)) \subseteq \pi_{R_i}(r)$$

3. Siccome $\pi_{R_i}(r) = \pi_{R_1}(m_\rho(r))$, allora si ha che:

$$m_\rho(m_\rho(r)) = \pi_{R_1}(m_\rho(r)) \bowtie \dots \bowtie \pi_{R_k}(m_\rho(r)) = \pi_{R_1}(r) \bowtie \dots \bowtie \pi_{R_k}(r) = m_\rho(r)$$

□

Algoritmo 4.4: 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 ρ presenta un join senza perdita.

```
def has_lossless_join(R: schema, F: set of dep., ρ: 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 ρ 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	a	b_1	a	b_1	b_1
ADE	a	b_2	b_2	a	a
CDE	b_3	b_3	a	a	a
AD	a	b_4	b_4	a	$b_{4,5}$
B	b_5	a	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 determinato 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 determinato B

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

Teorema 4.7: 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 `has_lossless_join(R, F, ρ)` determina che ρ presenta un join senza perdita se e solo se esiste una tupla in r contenente tutte " a " una volta terminato

Dimostrazione (solo un'implicazione):

- Supponiamo che ρ presenti un join senza perdita
- Siano r^0 e r^f rispettivamente lo stato iniziale e lo stato finale dell'istanza r
- Per costruzione stessa di r^0 , per ogni tupla $t_{R_i}^0 \in r^0$ si ha che $t_{R_i}^0[R_i] = ("a", \dots, "a")$
- Siccome l'algoritmo non modifica mai una "a" in una "b", allora si ha che $t_{R_i}^0[R_i] = t_{R_i}^f[R_i] = ("a", \dots, "a")$
- Sia quindi t^a la tupla contenente tutte le "a". Per costruzione di r , si ha quindi che:

$$t^a \in t_{R_1}^f[R_1] \bowtie \dots \bowtie t_{R_k}^f[R_k] \subseteq \pi_{R_1}(r^f) \bowtie \dots \bowtie \pi_{R_k}(r^f) = m_\rho(r^f)$$

- Siccome l'istanza r^f generata al termine dell'algoritmo è sempre legale, allora nel caso in cui ρ presenti un join senza perdita si ha $t^a \in m_\rho(r^f) = r^f$

□

4.5.3 Copertura minimale

Definizione 4.15: 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
- $\forall X \rightarrow A \in G, \nexists X' \subset X \mid G \equiv (G - \{X \rightarrow A\}) \cup \{X' \rightarrow A\}$, ossia non è possibile determinare funzionalmente A tramite un sottoinsieme proprio di X (rimozione determinanti ridondanti)
- $\nexists X \rightarrow A \in G \mid G \equiv G - \{X \rightarrow A\}$, ossia non è possibile determinare funzionalmente A tramite altre dipendenze (rimozione dipendenze ridondanti)

Algoritmo 4.5: 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 minimale 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 $\exists X' \subset X \mid X \rightarrow A \in F$ 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

Osservazione 4.7

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:

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

e conseguentemente che:

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

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

Osservazione 4.8

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**

Osservazione 4.9

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 $F_r \subseteq F \subseteq F^+ \iff F_r^+ \subseteq F^+$, 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 $\nexists Y \neq X \subseteq R \mid Y \neq A \in F$ 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.

Osservazione 4.10

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:

1. • Vogliamo trovare la copertura minimale 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 $E \notin C_F^+ \implies C \rightarrow E \notin F^+$), 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

4.5.4 Algoritmo di decomposizione

Algoritmo 4.6: 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, dunque in $O(n^k)$, 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,  $\rho$  :=  $\emptyset$ 
    for  $A \in R \mid \nexists X \rightarrow B \in F, A \in X \vee A = B$ :
        S := S  $\cup$  A
    if S  $\neq \emptyset$ :
        R := R - S
         $\rho$  :=  $\rho \cup \{S\}$ 
    if  $\exists X \rightarrow A \in F \mid X \cup A = R$ :
         $\rho$  :=  $\rho \cup \{R\}$ 
    else:
        for  $X \rightarrow A \in F$ :
             $\rho$  :=  $\rho \cup \{XA\}$ 
    return  $\rho$ 
```

Teorema 4.8: 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 omessa)

Proposizione 4.8: Ottenere 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:

- 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:

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

5

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:**

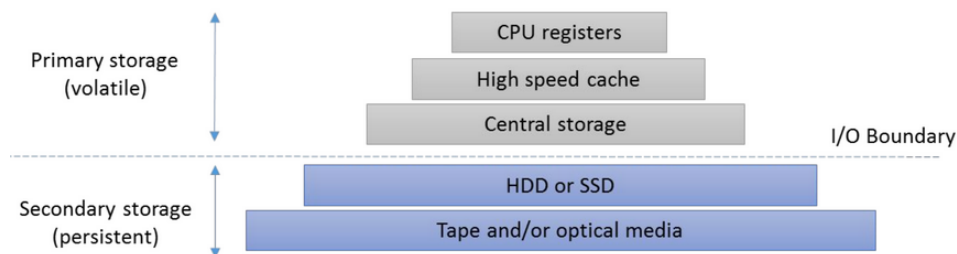
- Organizzazione con file heap
- Organizzazione sequenziale dei file
- Organizzazione casuale dei file (Hashing)
- Organizzazione indicizzata sequenziale dei file
- Organizzazione dei dati in lista
- Indici secondari e File invertiti
- B-trees e B⁺-trees

5.1 Hardware di archiviazione e progettazione fisica

Definizione 5.1: Gerarchia dell'archiviazione

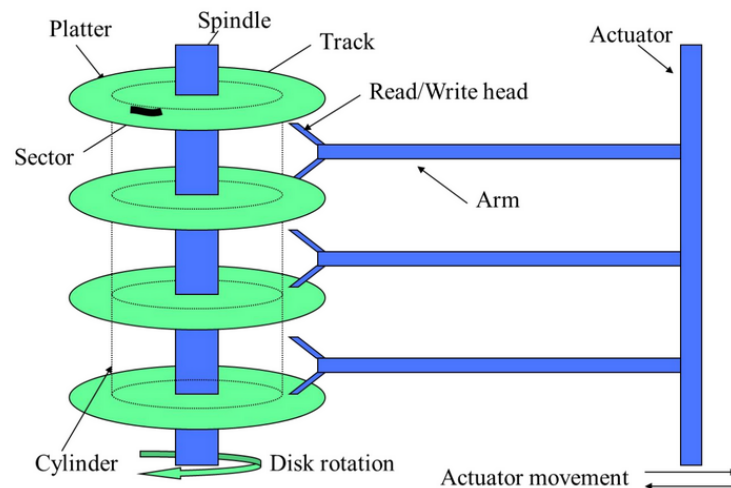
L'**Archiviazione primaria** di un DBMS contiene i buffer del database e i codici in esecuzione delle applicazioni e del DBMS. Essa è composta da CPU, Cache, Memoria principale (composta a sua volta da vari banchi di RAM).

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.



Osservazione 5.1

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

Definizione 5.2: Random Block Access e Sequential Block Access

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

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

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

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

- 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

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

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:

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

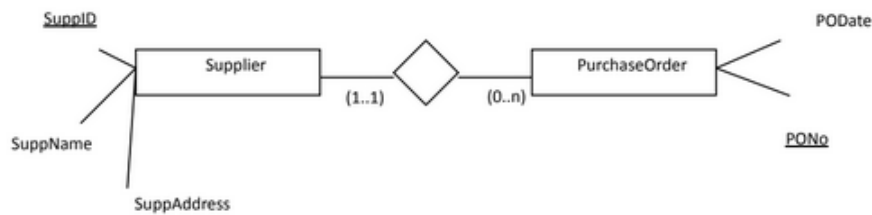
$$T_{SBA} \approx \frac{8.33}{2} \text{ ms} + \frac{4096 \text{ B}}{150 \cdot 2^{20} \text{ Bps}} \text{ s} \approx 4.165 \text{ ms} + 2.6 \cdot 10^{-2} \text{ ms} \approx 4.19 \text{ ms}$$

5.1.1 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:

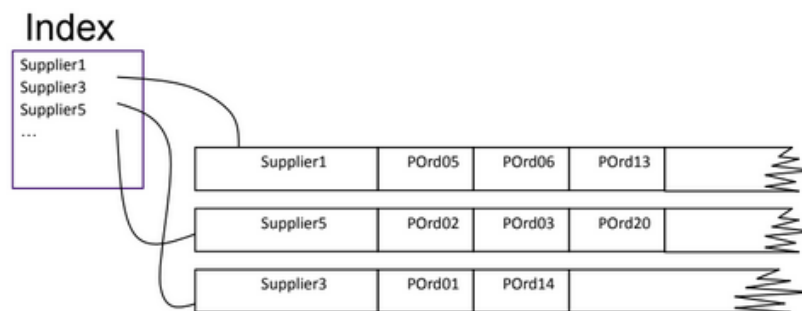


Modello logico:

Supplier(SuppID, SuppName, SuppAddress

PurchaseOrder(PONo, PODate, SuppID

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

5.2 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.

5.2.1 Organizzazione con file heap e file sequenziali

Metodo 5.1: 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 $\lceil \frac{N}{2} \rceil$ 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 5.2: 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 $\lceil \log_2(N) \rceil$ 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 $\lceil \frac{N}{2} \rceil$ 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:

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

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

$$\text{NB} = \left\lceil \frac{\text{NR}}{\text{RpB}} \right\rceil = \left\lceil \frac{30000}{20} \right\rceil = 1500$$

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

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

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

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

- 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

5.2.2 Organizzazione con file hash**Metodo 5.3: 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

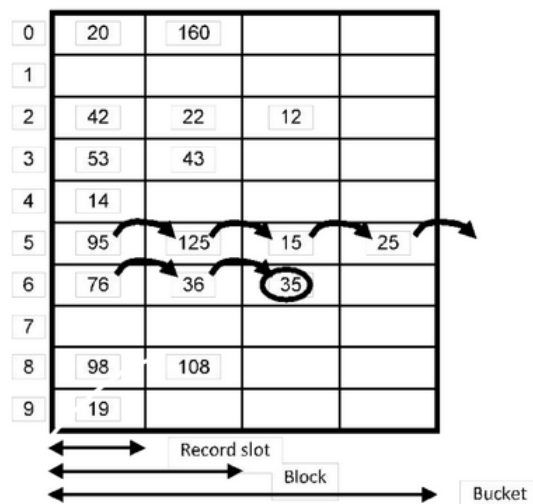
Osservazione 5.2

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:

$$PpB = \left\lfloor \frac{BS}{PS} \right\rfloor = \left\lfloor \frac{4096}{5} \right\rfloor = 819$$

- 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:

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

- 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:

$$NBpBk = \left\lceil \frac{RpBk}{RpB} \right\rceil = \left\lceil \frac{6190}{15} \right\rceil = 413$$

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

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

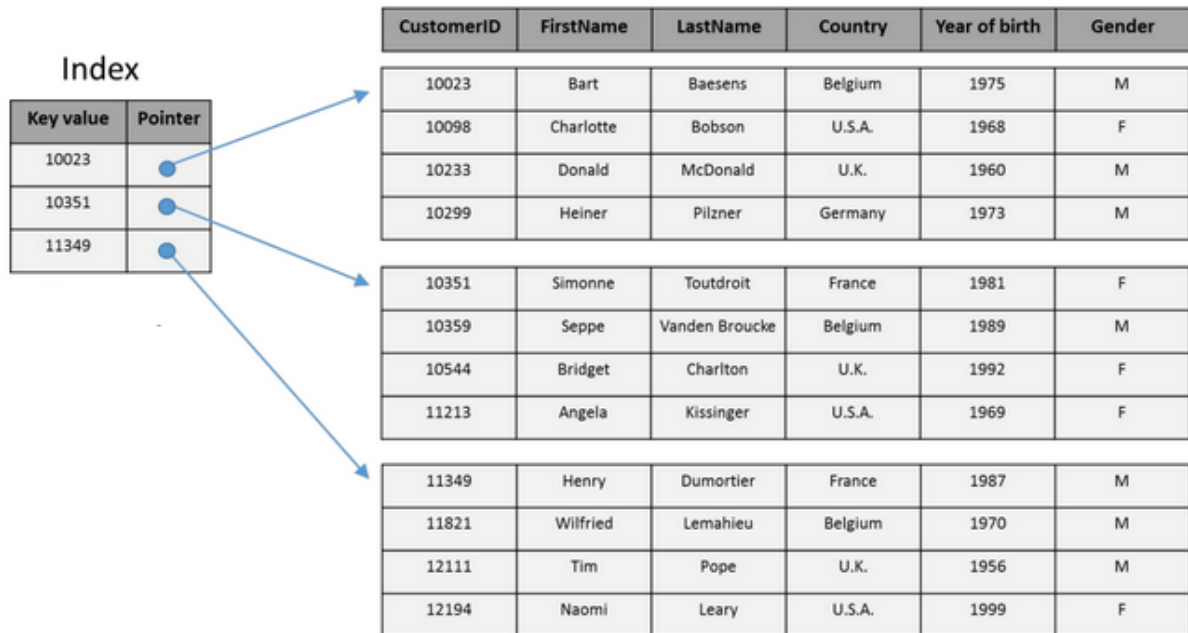
- Il numero medio di accessi per effettuare una ricerca sarà:

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

5.2.3 Organizzazione con file indicizzati sequenziali (ISAM)

Metodo 5.4: 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



Osservazione 5.3

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

Osservazione 5.4

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:

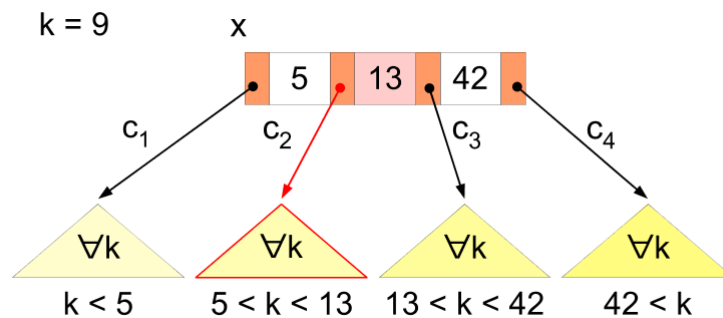
$$NA = \lceil \log_2(NBpFI) \rceil + 1 = \lceil \log_2(2288) + 1 \rceil + 1 = 12 + 1 = 13$$

5.2.4 Organizzazione con B-Tree

Definizione 5.3: 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' di un nodo possono assumere solo valori nell'intervallo aperto (k, k')



Definizione 5.4: 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**

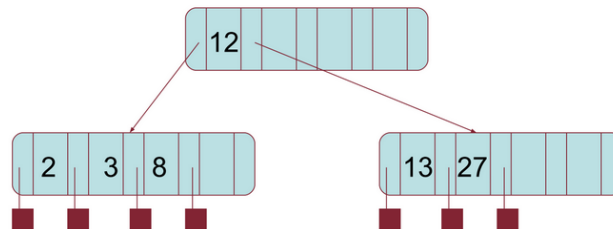
Metodo 5.5: Organizzazione con B-Tree

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

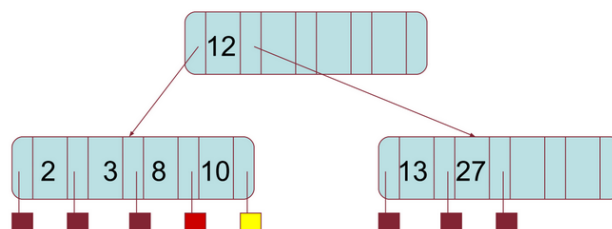
- Ogni nodo corrisponde ad un **blocco** e, per via dei vincoli imposti dalle proprietà, essi sono sempre **carichi almeno a metà** (ossia almeno metà della loro capienza è occupata), implicando che esistano almeno d nodi figli
- Ogni nodo contiene:
 - Un **insieme di chiavi di ricerca**
 - Un **insieme 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 vengono archiviati separatamente, dunque essi non fanno parte del B-Tree)

Esempio:

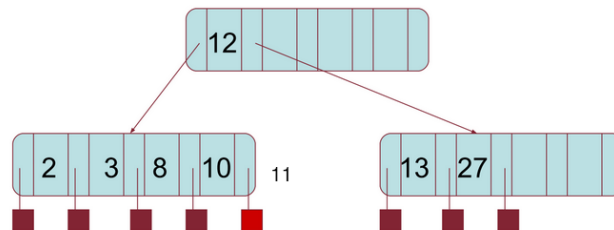
- Consideriamo un B-Tree a 5 vie, implicando che il numero massimo di nodi figli per ogni nodo sarà 5, mentre il numero minimo di nodi figli per ogni nodo sia $\lceil \frac{5}{2} \rceil = 3$ (fatta eccezione per la radice, la quale avrà minimo 2 figli, a meno che essa non sia una foglia)



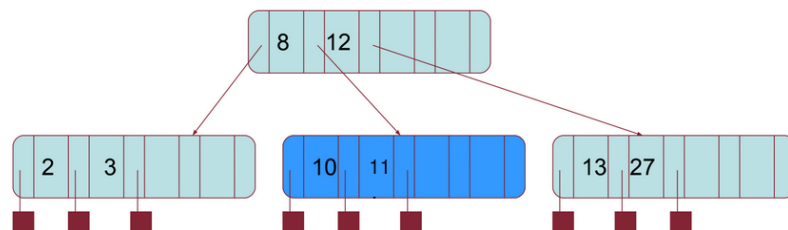
- Inserendo la chiave 10, procediamo come un normale inserimento di un albero binario di ricerca, aggiungendo quindi 10 al figlio sinistro della radice



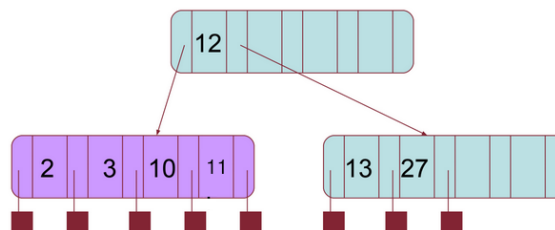
- Nel caso in cui volessimo inserire la chiave 11, essa verrebbe posta all'interno del figlio sinistro, il quale tuttavia può avere una capienza massima pari a 5, dunque l'inserimento della chiave 11 porterebbe il nodo ad essere sovraccarico



- 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



Osservazione 5.5: 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 Pd_i
- 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

Proposizione 5.1: 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 $d := \lceil \frac{m}{2} \rceil$ il numero minimo di figli che un nodo possa avere, l'**altezza** h del B-Tree è compresa tra:

$$\lceil \log_m(N + 1) \rceil \leq h \leq \left\lfloor \log_d \left(\frac{N + 1}{2} \right) \right\rfloor + 1$$

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:

- Osserviamo che, per definizione stessa di B-Tree, ogni nodo possiede un numero di figli compreso tra $[d, m]$ e un numero di chiavi compreso tra $[d - 1, m - 1]$
- Sia N il numero di chiavi nel B-Tree e sia b il numero di nodi figli presenti nel B-Tree
- Osserviamo che il numero massimo di nodi dell'albero è raggiunto quando tutti i nodi hanno capienza massima, dunque quando ognuno di essi possiede m figli (eccetto le foglie) e $m - 1$ chiavi

Per la natura ricorsiva dell'albero, quindi, si ha che:

$$b_{max} = \sum_{i=0}^{h-1} m^i = \frac{m^h - 1}{m - 1}$$

Dunque, siccome ogni nodo possiede massimo m figli, si ha che:

$$N_{max} = (m - 1) \cdot b_{max} = (m - 1) \cdot \frac{m^h - 1}{m - 1} = m^h - 1$$

- Analogamente, il numero minimo di nodi dell'albero è raggiunto quando tutti i nodi hanno capienza minima, dunque quando la radice possiede due soli figli (dunque una sola chiave) mentre ogni nodo interno possiede d figli e $d - 1$ chiavi

Per la natura ricorsiva dell'albero, quindi, si ha che:

$$b_{min} = 1 + 2 \sum_{i=0}^{h-2} d^i = 1 + 2 \frac{d^{h-1} - 1}{d - 1}$$

Dunque, siccome la radice possiede una chiave ed ogni nodo figlio possiede minimo d chiavi, si ha che:

$$N_{min} = 1 + (d - 1)(b_{min} - 1) = 1 + (d - 1) \left(1 + 2 \frac{d^{h-1} - 1}{d - 1} - 1 \right) = 2d^{h-1} - 1$$

- In definitiva, quindi, concludiamo che

$$2d^{h-1} - 1 \leq N \leq m^h - 1$$

da cui ricaviamo che:

$$\lceil \log_m(N + 1) \rceil \leq h \leq \left\lfloor \log_d \left(\frac{N + 1}{2} \right) \right\rfloor + 1$$

□

Osservazione 5.6: Performance di un B-Tree

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

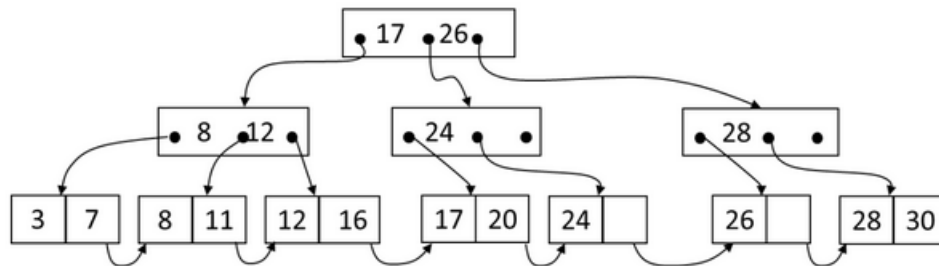
- L'operazione **get(k)** impiega massimo h accessi al disco
- L'operazione **put(k)** impiega massimo $3h + 1$ accessi al disco
- L'operazione **remove(k)** impiega massimo $3h$ accessi al disco

Definizione 5.5: 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

**Esempio:**

- 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 quando 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:

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

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

$$NB_{pFP} = \left\lceil \frac{NR}{R_{pB}} \right\rceil = \left\lceil \frac{1700000}{8} \right\rceil = 21250$$

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

$$NB_{pL0} = \left\lceil \frac{NB_{pFP}}{m} \right\rceil = \left\lceil \frac{21250}{42} \right\rceil = 506$$

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

$$NB_{pL1} = \left\lceil \frac{NB_{pL0}}{m} \right\rceil = \left\lceil \frac{506}{42} \right\rceil = 13$$

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

$$NB_{pL2} = \left\lceil \frac{NB_{pL1}}{m} \right\rceil = \left\lceil \frac{13}{42} \right\rceil = 1$$

- 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:

$$NB_{pFI} = NB_{pL0} + NB_{pL1} + NB_{pL2} = 21250 + 506 + 13 + 1 = 21770$$

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 quando 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:

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

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

$$NB_{pFP} = \left\lceil \frac{NR}{R_{pB}} \right\rceil = \left\lceil \frac{17000000}{4} \right\rceil = 42500$$

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

$$NB_{pL0} = \left\lceil \frac{NB_{pFP}}{m} \right\rceil = \left\lceil \frac{42500}{22} \right\rceil = 1932$$

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

$$NB_{pL1} = \left\lceil \frac{NB_{pL0}}{m} \right\rceil = \left\lceil \frac{1932}{22} \right\rceil = 88$$

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

$$NB_{pL2} = \left\lceil \frac{NB_{pL1}}{m} \right\rceil = \left\lceil \frac{88}{22} \right\rceil = 4$$

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

$$NB_{pL3} = \left\lceil \frac{NB_{pL2}}{m} \right\rceil = \left\lceil \frac{4}{22} \right\rceil = 1$$

- 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:

$$\begin{aligned} NB_{pFI} &= NB_{pL0} + NB_{pL1} + NB_{pL2} + NB_{pL3} = \\ &= 42500 + 1932 + 88 + 4 + 1 = 44525 \end{aligned}$$