SAPIENZA
UNIVERSITÀ DI ROMA

UNIVERSITÀ "SAPIENZA" DI ROMA
FACOLTÀ DI INFORMATICA

Basi di Dati

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

Author
Simone Bianco

10 gennaio 2023

Indice

Capitolo 0

Introduzione

Il seguente corso mira all'apprendimento dei principali argomenti teorici dietro allo sviluppo di un database ottimale:

- **Introduzione ai sistemi di gestione di basi di dati**: cenni storici, aspetti caratterizzanti dei sistemi di gestione di basi di dati, evoluzione di modelli e sistemi
- **Modello relazionale**: dominio, attributo, relazione, n-upla, schema, linguaggi di interrogazione, algebra relazionale, chiave di una relazione
- **Teoria della normalizzazione**: dipendenze funzionali, Terza Forma Normale (3NF), assiomi di Armstrong, chiusura di un insieme di dipendenze, chiusura di un insieme di attributi, copertura minimale di un insieme di dipendenze, scomposizioni che hanno un join senza perdita, scomposizioni che preservano le dipendenze.
- **Organizzazione fisica dei dati**: memoria secondaria, record fisici e record logici, puntatori, blocchi, file heap, file hash, file con indice (indici densi e indici sparsi), B-tree

Capitolo 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 **dadi 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 per tanto 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 (**commited**) 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	
	T2	Ricerca C1
Cambia bilancio in: Bilancio+1000	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.

Capitolo 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:

Definition 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.

Definition 2. Prodotto cartesiano

Siano $D_1, D_2, ..., D_k$ 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:

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

Definition 3. Relazione

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

$$r \subseteq D_1 \times \ldots \times D_k$$

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

Definition 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.

Observation 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 $D_1 : \{\texttt{white, black}\}, D_2 : \{0, 1, 2\}$. Il loro prodotto cartesiano sarà:

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

- La seguente relazione

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

  è una relazione di grado 2 e cardinalità 3

- La seguente relazione

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

  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 $\texttt{String} \times \texttt{String} \times \texttt{Integer} \times \texttt{Real}$:

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

  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:

Definition 5. Elemento di una tupla

Data una relazione $r \subseteq D_1 \times ... \times D_k$ 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:

Definition 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.

Definition 7. Schema relazionale

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

$$R(A_1, A_2, ..., A_k)$$

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

Definition 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 $t : R \to dom(A) : A \mapsto t(A)$ 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:

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

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

Observation 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.

Definition 9. Schema di un database e Database relazionale

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

$$(R_1, R_2, ..., R_n)$$

Definiamo come **database relazionale** uno schema di un database $(R_1, R_2, ..., R_n)$ su cui sono definite le istanze $(r_1, r_2, ..., r_n)$, 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

Observation 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:

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

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

Definition 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 $Y = \{\texttt{City, Population}\}$, allora:

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

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[`Student`] fa riferimento al campo `Matricola` dell'ultima tupla della tabella `Students`, mentre t[`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**

Definition 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.

Proposition 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

Definition 12. Chiave relazionale

Un **insieme X di attributi** appartenenti ad una relazione R sono una **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 $Y \subseteq X$ che soddisfa la prima condizione

All'interno di una relazione possono essere definite più chiavi alternative.

Definition 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**)

Observation 4

Poiché non possono esserci tuple identiche in una relazione, **vi è sempre almeno una chiave** all'interno di ogni istanza.

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.

2.3.2 Chiavi esterne

Definition 14. Vincolo di chiave esterna (o di riferimento)

Siano R_1, R_2 due relazioni, dove $X \in R_1$ è una chiave primaria di R_1. Definiamo come **vincolo di chiave esterna** (o di riferimento) l'obbligo per un insieme di attributi $Y \in R_2$ 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:
  – `Code` è chiave primaria di `Road Violations`
  – `Id` è chiave primaria di `Officers`
  – `(Prov, Plate)` è chiave primaria di `Cars`

- 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` siano 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:

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

$$\nexists t \in c \mid t_4[\texttt{Prov, Plate}] = t[\texttt{Prov, Plate}]$$

  - In altre parole, non esiste una tupla nell'istanza di `Cars` avente `(PR, 839548)` come valori di `(Prov, Plate)`

Capitolo 3

Algebra relazionale

Definition 15. 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

Definition 16. 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 $\pi_{\texttt{Name}}(\texttt{Customers})$, il cui output sarà:

$\pi_{\texttt{Name}}(\texttt{Customers})$
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 contente il nome "Rossi", nonostante nella relazione iniziale ve ne fossero tre.
- Per prevenire tale perdita di informazione, proviamo ad effettuare la proiezione $\pi_{\texttt{Name, Town}}(\texttt{Customers})$, il cui output sarà:

$\pi_{\texttt{Name, Town}}(\texttt{Customers})$	
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 $\pi_{\texttt{Name, C\#}}(\texttt{Customers})$, il cui output sarà:

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

Definition 17. 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:

- $A, B \in R(X) \mid dom(A) = dom(B)$
- $A \in R(X), a \in dom(A)$
- 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 $\sigma_{\texttt{Town='Roma'}}(\texttt{Customers})$ per ottenere le tuple richieste:

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

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

3.2 Unione, Intersezione e Differenza

Definition 18. Compatibilità in unione

Data una relazione R_1 con schema $R_1(A_1, \ldots, A_k)$ ed una relazione R_2 con schema $R_2(a_1, \ldots, A_h)$, tali relazioni vengono dette **compatibili in unione** se e solo se:

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

Definition 19. 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.

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

Observation 5

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 ∪ 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`:

$\texttt{Teachers} \cup \pi_{\texttt{Name, AdminCode, Department}}(\texttt{Admins})$		
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 $dom(\texttt{Teachers.Department}) \neq dom(\texttt{Admins.ServiceYears})$. Tuttavia, possiamo effettuare una proiezione su entrambe le relazioni, per poi unirle:

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

Definition 20. 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.

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

Definition 21. 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.

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

Observation 6

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 ∩ Admins		
Name	**Code**	**Department**
Verdi	C4	English
Bianchi	C5	English

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

Teachers − Admins		
Name	**Code**	**Department**
Rossi	C1	Math
Rossi	C2	Italian

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

Admins − Teachers		
Name	**AdminCode**	**Department**
Esposito	C1	English
Riccio	C2	Math

3.3 Ridenominazione e Prodotto Cartesiano

Definition 22. 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.

Definition 23. 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#**	**Qnty**
O1	C1	A1	100
O2	C2	A2	200
O3	C3	A2	150
O4	C4	A3	200
O1	C1	A2	200
O1	C1	A3	100

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

$$\texttt{OrsersR} = \rho_{\texttt{C\#}\leftarrow\texttt{CC\#}}(\texttt{Orders}$$

- Successivamente, effettuando il prodotto cartesiano `Customers` × `OrdersR`, otteniamo:

Customers × OrdersR						
Name	**C#**	**Town**	**O#**	**CC#**	**A#**	**Qnty**
Rossi	C1	Roma	O1	C1	A1	100
Rossi	C1	Roma	O2	C2	A2	200
Rossi	C1	Roma	O3	C3	A2	150
Rossi	C1	Roma	O4	C4	A3	200
Rossi	C1	Roma	O1	C1	A2	200
Rossi	C2	Milano	O1	C1	A1	100
Rossi	C2	Milano	O2	C2	A2	200
⋮	⋮	⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮	⋮	⋮
Verdi	C4	Roma	O4	C4	A3	200
Verdi	C4	Roma	O1	C1	A2	200

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

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

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

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

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

3.4 Join Naturale e Theta Join

Definition 24. 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 $A_1, \ldots, A_k \mid \forall i \in [1,k], A_i \in X \cap Y$ si ha che:

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

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:

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

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

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

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

`Customers` $\bowtie$ `Orders`					
Name	**C#**	**Town**	**O#**	**A#**	**Qnty**
Rossi	C1	Roma	O1	A1	100
Rossi	C1	Roma	O1	A2	200
Rossi	C1	Roma	O1	A3	100
Rossi	C2	Milano	O2	A2	200
Bianchi	C3	Roma	O3	A2	150
Verdi	C4	Roma	O4	A3	200

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:

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

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

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

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

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

3. - Oltre alla soluzione appena vista, possiamo ottenere lo stesso risultato selezionando prima le tuple che rispettano le condizioni e i dati che ci interessano, per poi effettuare il join tra loro, rendendo così la query più efficiente, poiché la mole di dati su cui vengono applicati gli operatori è minore:

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

Proposition 2. 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:

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

Definition 25. 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 **join naturale** 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:

- $A \in R_1(X), B \in R_2(Y)$
- $dom(A) = dom(B)$
- 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

Observation 7

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:

$$\texttt{EmployeesC} = \texttt{Employees}$$

$$\texttt{EmpSup}_1 = \texttt{EmplyeesC} \bowtie_{\texttt{EmployeesC.Supervisor\# = Employees.C\#}} \texttt{Employees}$$

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

$$X = \{\texttt{Name, C\#, Section, Salary, Supervisor\#}\}$$
$$Y = \{\texttt{CName, CC\#, CSec, CSal, CSup\#}\}$$

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

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

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

oppure

$$\sigma_{\texttt{Salary > CSalary}}(\texttt{EmpSup}_2)$$

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

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

oppure

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

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

Observation 8

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:

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

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

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

- 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:

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

oppure, direttamente:

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

R	
Name	**Town**
Bianchi	Roma
Verdi	Roma

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

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

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

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

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

R	
Name	**C#**
Bianchi	C3

Capitolo 4

Teoria della normalizzazione

4.1 Dipendenze funzionali

Definition 26. 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 $X \to Y$ 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, $X \to Y$ 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 partità sempre allo stesso orario
     - C'è un solo volo con un certo pilota ad un certo orario in un certo giorno
     - C'è un solo pilota di un certo volo in un certo giorno
   - Dunque, imponiamo le seguenti dipendenze funzionali sullo schema:
     - `Code` $\to$ `Time`
     - `(Day, Pilot, Time)` $\to$ `Code`
     - `(Code, Day)` $\to$ `Pilot`

Definition 27. 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:

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

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 relazione su cui sono definite le seguenti dipendenze funzionali:

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

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 lo stesso valore in B

4.1.1 Chiusura di F

Definition 28. Chiusura di un insieme di dipendenze funzionali

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:

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

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 $A \to C$, che tuttavia non è in F. Dunque, si ha che $A \to B, B \to C, A \to C \in F^+$

Observation 9

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

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

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

Proposition 3

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

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

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

Dimostrazione:

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

$$\forall A_i \in Y, X \to A \in F^+ \iff \begin{cases} \forall t_1, t_2 \in r, t_1[X] = t_2[X] \implies t_1[A_1] = t_2[A_1] \\ \vdots \\ \forall t_1, t_2 \in r, t_1[X] = t_2[X] \implies t_1[A_k] = t_2[A_k] \end{cases} \iff$$

$$\iff \forall t_1, t_2 \in r, t_1[X] = t_2[X] \implies t_1[\{A_1, \dots, A_k\}] = t_2[\{A_1, \dots, A_k\}]$$

$$\iff \forall t_1, t_2 \in r, t_1[X] = t_2[X] \implies t_1[Y] = t_2[Y] \iff X \to Y \in F^+$$

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 $ABC \to ABC \in F^+$.
- Notiamo inoltre che tutte le tuple di tale istanza in cui A e B sono uguali anche A è uguale, dunque $AB \to A \in F^+$.
- Procedendo analogamente, in definitiva si ha che:

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

4.2 Assiomi di Armstrong

Definition 29. 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** ($F \subseteq F^A$):

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

- **Assioma di riflessività**:

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

- **Assioma di aumento:**

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

- **Assioma di transitività:**

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

Proposition 4. 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**:

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

- **Regola della decomposizione**:

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

- **Regola della pseudo-trasitività**:

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

Dimostrazione:

- **Regola dell'unione**:
  - Siano $X \to Y, X \to Z \in F^A$.
  - Per assioma di aumento, si ha che:
  $$X \subseteq R, X \to Y \in F^A \implies XX \to XY = X \to XY \in F^A$$
  - Analogamente, si ha che:
  $$Y \subseteq R, X \to Z \in F^A \implies XY \to ZY = XY \to YZ \in F^A$$
  - Infine, per assioma di transitività si ha che:
  $$X \to XY \in F^A \wedge XY \to YZ \in F^A \implies X \to YZ \in F^A$$
- **Regola della decomposizione:**
  - Sia $Z \subseteq Y \subseteq R$ e sia $X \to Y \in F^A$.
  - Per assioma di riflessività, si ha che:
  $$Z \subseteq Y \subseteq R \implies Y \to Z \in F^A$$
  - Infine, per assioma di transitività si ha che:
  $$X \to Y \in F^A \wedge Y \to Z \in F^A \implies X \to Z \in F^A$$
- **Regola della pseudo-transitività:**
  - Sia $X \to Y, YW \to Z \in F^A$.
  - Per assioma di aumento, si ha che:
  $$W \subseteq R, X \to Y \in F^A \implies XW \to YW \in F^A$$
  - Infine, per assioma di transitività si ha che:
  $$XW \to YW \in F^A \wedge YW \to Z \in F^A \implies XW \to Z \in F^A$$

Proposition 5

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

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

Dimostrazione:

- Siano $X, Y \subseteq R$, dove $Y = \{A_1, \ldots, A_k\}$.
- Per la regola dell'unione, si ha che:
$$\forall A_i \in Y, X \to A \in F^A \implies X \to \{A_1, \ldots, A_k\} = Y \in F^A$$
- Per la regola della decomposizione, invece si ha che:
$$X \to Y \in F^A \implies \forall A \in Y, X \to A \in F^A$$

4.2.1 Chiusura di X

Definition 30. 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:

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

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

Lemma 6

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

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

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

Dimostrazione:

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

$$X \to Y \in F^A \iff \forall A_i \in Y, X \to A_i \in F^A \iff Y = \{A_1, \dots, A_k\} \subseteq X^+$$

Corollary 1

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

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

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

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

4.2.2 $F^+ = F^A$

Theorem 7. $\mathbf{F^+ = F^A}$

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 $X \to Y \in F^A$ senza aver applicato alcun assioma di Armstrong, allora l'unica possibilità è che:

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

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

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

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

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

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

    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:

       $$X \to Y \in F^A \iff Y \subseteq X \subseteq R$$

       Ma se $Y \subseteq X \subseteq R$, allora $\forall r$ istanza legale di R si ha che:

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

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

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

       * $\exists V, W \subseteq R \mid \exists V \to W \in F^A$, ottenuta applicando $j \leq n$ assiomi di Armstrong
       * $\exists Z \subseteq R \mid X := VZ, Y := WZ$

       affinché si abbia che:

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

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

$$\begin{cases} V \to W \in F^+ \\ Z \to Z \in F^+ \end{cases} \implies \begin{cases} \forall t_1, t_2 \in r, t_1[V] = t_2[V] \implies t_1[W] = t_2[W] \\ \forall t_1, t_2 \in r, t_1[Z] = t_2[Z] \implies t_1[Z] = t_2[Z] \end{cases} \implies$$

$$\implies \forall t_1, t_2 \in r, t_1[VZ] = t_2[VZ] \implies t_1[WZ] = t_2[WZ]$$
$$\implies VZ \to WZ = X \to Y \in F^+$$

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

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

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

$$\begin{cases} X \to Z \in F^+ \\ Z \to Y \in F^+ \end{cases} \implies$$

$$\implies \forall t_1, t_2 \in r, t_1[X] = t_2[X] \implies t_1[Z] = t_2[Z] \implies t_1[Y] = t_2[Y] \implies$$
$$\implies X \to Y \in F^+$$

- 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	$\cdots$	A_i	A_j	$\cdots$	A_n
1	$\ldots$	1	1	$\ldots$	1
1	$\ldots$	1	0	$\ldots$	0

    dunque tale che $\forall t_1, t_2 \in r$ si ha:
    * $t_1[X^+] = (1, \ldots, 1) = t_2[X^+]$
    * $t_1[R - X^+] = (1, \ldots, 1) \neq (0, \ldots, 0) = t_2[R - X^+]$
  - Notiamo che $\forall V, W \subseteq R \mid V \to W \in F$ 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 $V \to W \in F$
    * Se invece $V \subseteq X^+$, per il lemma precedentemente visto si ha che

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

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

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

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

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

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

– Siccome in entrambi i casi r soddisfa $V \to W \in F$, allora r è legale, implicando che qualsiasi $X \to Y \in F^+$ è soddisfatta da r.

– Inoltre, siccome $X \subseteq X^+$ per il lemma precedentemente visto e siccome abbiamo mostrato che (in questo caso considerando $V = X$ e $W = Y$) per costruzione di r si ha che

$$X \subseteq X^+, X \to Y \implies Y \subseteq X^+$$

e dunque otteniamo che $Y \subseteq X^+ \iff X \to Y \in F^A$

Observation 10

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 $O(2^{nk})$: considerando anche solo l'assioma di riflessività, siccome ogni possibile sottoinsieme di R genera una dipendenza e siccome i sottoinsiemi possibili di R sono $2^{|R|}$, allora ne segue che $|F^+| >> 2^{|R|}$

4.3 Terza Forma Normale (3NF)

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

Definition 31. 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:

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

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

Esempio:

- Consideriamo lo schema `Student(Matr, LastName, FirstName, BirthD)`

- In questo caso, è ovvio imporre la seguente dipendenza funzionale in F:

$$\texttt{Matr} \to \{\texttt{LastName, FirstName, BirthD}\} \in F \subseteq F^+$$

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

- Siccome $\texttt{Matr} \subseteq \texttt{Matr}^+ \iff \texttt{Matr} \to \texttt{Matr} \in F^+$, per la regola dell'unione si ha che:

$$\texttt{Matr} \to \{\texttt{Matr, LastName, FirstName, BirthD}\} \in F^+$$

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`

Definition 32. 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:

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

Observation 11

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

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

Definition 33. 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 $\forall A \in R, X \subseteq R \mid X \to A \in F^+$ dove $A \notin X$, si ha che esiste una chiave $K \subseteq R$ tale che $K \subseteq X \lor A \in K$.

$$\forall X \to A \in F^+, A \notin X, \exists K \subseteq R \text{ chiave } \mid K \subseteq X \lor A \in K$$

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

Esempio:

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

– Per aumento:

$$BC \to ABD \in F^A \implies BCC = BC \implies ABCD = R \in F^A$$

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

- Deduciamo quindi che AC e BC siano chiave di R, mentre ABC è una superchiave di R
- Verifichiamo quindi se R sia in 3NF:
  - $AC \to B \in F^A = F^+$ rispetta la definizione di 3NF, poiché AC è chiave
  - $B \to AD \in F^A = F^+$ non va controllato, ma per decomposizione si ha che $B \to A, B \to D \in F^A = F^+$
  - $B \to A \in F^+$ rispetta la definizione di 3NF, poiché $A \in AC$ e dunque primo, mentre $B \to D \in F^+$ 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 $F = \{AB \to CD, BC \to A, D \to AC\}$ un insieme di dipendenze funzionali su R
- Applicando gli assiomi di Armstrong, si ha che AB, BC e BD sono chiavi di R
- Verifichiamo quindi se R sia in 3NF:
  - $AB \to CD \in F^+$ rispetta la definizione di 3NF, poiché AB è chiave
  - $BC \to A \in F^+$ rispetta la definizione di 3NF, poiché BC è chiave e A è primo
  - $D \to AC \in F^+ \implies D \to A, D \to C \in F^+$, i quali rispettano entrambi la definizione di 3NF, poiché A e C sono entrambi primi
  - Dunque, concludiamo che R sia in 3NF

Definition 34. Dipendenza parziale

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

Definiamo $X \to A \in F^+$, dove $A \notin X$, come **dipendenza parziale** su R se A non è primo e se $X \subset K$ (quindi in particolare $X \neq K$), dove K è una chiave

$$A \notin X, X \to A \in F^+ \text{ dip. parziale} \iff A \in K, K \text{ chiave di } R \land X \subset K$$

Definition 35. Dipendenza transitiva

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

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

- A non è primo
- $\forall K \subseteq R$ chiave di R si ha che $X \subset K$ (quindi in particolare $X \neq K$) e $K - X \neq \emptyset$

Corollary 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.

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

Definition 36. 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

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

Observation 12

Uno schema in forma normale di Boyce-Codd è anche uno schema in terza forma normale, poiché la BCNF è una versione più restrittiva della 3NF, tuttavia, a differenza della 3NF, non è sempre possibile decomporre uno schema in BCNF in più sottoschemi.

4.4 Calcolare X^+

Method 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 $F = \{AB \to C, B \to D, AD \to E, CE \to H\}$ 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:
     - $AB \subseteq AB \wedge AB \to C \implies C \in S$
     - $B \subseteq AB \wedge B \to D \implies D \in S$

     Notiamo come tramite l'algoritmo stiamo implicitamente utilizzando gli assiomi di Armstrong per aggiungere C e D a $Z = AB$:
     - $A, B \in AB, AB \to A, AB \to B \in F^A$
     - $AB \to C, B \to D \in F \implies AB \to C, B \to D \in F^A$
     - $B \subseteq AB \implies AB \to B \in F^A$
     - $AB \to B, B \to D \in F^A \implies AB \to D \in F^A$
     - $AB \to A, AB \to B, AB \to C, AB \to D \in F^A \iff A, B, C, D \in AB^+$

  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:

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

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

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

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

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

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

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

   In questo caso, quindi, S rimane inalterato
5. Infine, siccome $S \subseteq Z$, l'output dell'algoritmo sarà $AB^+ = ABCDEH$

Theorem 8. 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 avrebbe che $A \in Z_i \subseteq X^+$
    * Se $A \in S^i$, allora $\exists Y \to V \in F \mid A \in V \subseteq R, Y \subseteq Z_i$.

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

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

      Dunque, avrebbe 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	$\cdots$	A_i	A_j	$\cdots$	A_n
1	...	1	1	...	1
1	...	1	0	...	0

  dunque tale che $\forall t_1, t_2 \in r$ 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 $\forall V, W \subseteq R \mid V \to W \in F$ 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 $V \to W \in F$
  - Se invece $V \subseteq Z_f$, allora $W \subseteq S_f$, poiché, per come viene calcolato S_f, si ha che:

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

    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

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

    e quindi r soddisfa ogni $V \to W \in F$
- Siccome in entrambi i casi r soddisfa $V \to W \in F$, allora r è legale.
- Dato $A \in X^+$ si ha che $X \to A \in F^A = F^+$, dunque deve essere soddisfatta da qualsiasi istanza legale, inclusa r, dunque si ha che

$$\forall t_1, t2 \in r, t_1[X^+] = t_2[X^+] \implies t_1[A] = t_2[A]$$

- Tuttavia, per costruzione di r si ha che $t_1[A] = t_2[A] \iff A \in Z_f$, dunque concludiamo che $X^+ \subseteq Z_f$

4.4.1 Trovare le chiavi di uno schema

Proposition 9

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 $X \subseteq R$. Per le regole della decomposizione e dell'unione e per , si ha che:

$$X \to R \in F^+ \iff \forall i \in [1, k], X \to A_i \in F^+ = F^A \iff$$

$$\iff \forall i \in [1, k], A_i \in X^+ \iff X^+ = \{A_i, \dots, A_k\} = R$$

- Se $X \to R \in F^+$, le uniche possibilità sono:
  - $\exists Y \subseteq X \mid Y$ chiave di $R \implies X$ superchiave di R

  - $\nexists Y \subset X \mid Y \to R \in F^+ \implies X$ chiave di $R \implies X$ superchiave di R

- Dunque, possiamo concludere che:

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

Corollary 3

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

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

Proposition 10

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

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

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 $\nexists X \to Y \in F \mid A \in Y$.
- 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 $K \to A_j \in F^A$.
- Tuttavia, si verifica che:
  - $K \to A_j \in F^A$ non può essere ottenuta tramite l'inclusione di F, poiché $K \to A_j \notin F$, siccome A_j non è determinato da alcuna dipendenza in F
  - $K \to A_j \in F^A$ non può essere ottenuta tramite riflessività, poiché implicherebbe necessariamente che $A_j \in K$
  - $K \to A_j \in F^A$ non può essere ottenuta tramite aumento, poiché implicherebbe necessariamente che $A_j \in K$
  - L'unica possibilità, quindi, è che $K \to A_j \in F^A$ sia ottenuta tramite transitività, implicando l'esistenza di $Y \subseteq R$ tale che

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

- Affinché $Y \to A_j \in F^A$, si ha necessariamente che $\exists V \to W \in F \mid A_j \in V \vee A_j \in W$, poiché altrimenti non sarebbe possibile ricavare $Y \to A_j \in F^A$ 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 $F = \{AB \to CD, C \to E, AB \to E, ABC \to D\}$, 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 $ABC^+ = ABCDEH$
   - 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 $F = \{AB \to D, G \to A, G \to B, H \to E, H \to G, D \to H\}$
   - 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

Theorem 11. Test dell'unicità

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

Posto:

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

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

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

     - Siccome $ABH^+ = R$, allora ABH è l'unica chiave di R
   - Per verificare che R sia in 3NF, ci basta vedere che:

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

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

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

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

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

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

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

- 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 $B \to E \in F$ è una dipendenza parziale, allora R non è in 3NF

4.5 Decomposizione di uno schema

Definition 37. Decomposizione di uno schema

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

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

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

Observation 13

Decomporre uno schema R in più sottoschemi $R_1, \ldots, R_k$ 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.

Esempio:

- Consideriamo lo schema $R = ABC$ e l'insieme di dipendenze funzionali $F = \{A \to B, B \to C\}$
- In questo caso, notiamo facilmente che A è l'unica chiave di R, implicando che R non sia in 3NF.
- Possiamo quindi provare a decomporre R in due modi:
  - Decomponiamo R in $R_1 = AB$ con $F_1 = \{A \to B\}$ e $R_2 = BC$ con $F_2 = \{B \to C\}$, i quali risultano essere entrambi in 3NF
  - Decomponiamo R in $R_1' = AB$ con $F_1' = \{A \to B\}$ e $R_2' = AC$ con $F_2' = \{A \to C\}$, i quali risultano essere entrambi in 3NF
- Entrambi gli schemi, quindi, sono in 3NF. Tuttavia, la seconda soluzione non è corretta:
  - 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 $F = \{A \to B, B \to C\}$, notiamo come $B \to C$ non **preservata dalla decomposizione**, ossia non sia più soddisfatta da tale istanza di R, rendendola quindi illegale.

– Dunque, l'unica decomposizione che preserva le dipendenze di F è la prima

Definition 38. 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 ricostruire preservando F, ossia mantenendo 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 nelle 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**)

Observation 14

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:

$$r_k = \pi_{R_i}(r_1)$$

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

Definition 39. 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 12. Inclusione delle chiusure

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

$$F \subseteq G^+ \iff G \xrightarrow{A} F \iff F^+ \subseteq G^+$$

Dove $G \xrightarrow{A} F$ indica che F è ottenibile da G utilizzando assiomi di Armstrong

Dimostrazione:

- Ricordando che $G^+ = G^A$, dunque è l'insieme di tutte le dipendenze funzionali ottenibile applicando assiomi di Armstrong su G, allora:

$$G \xrightarrow{A} F \implies \forall X \to Y \in F, X \to Y \in G^A = G^+ \implies F \subseteq G^+$$

- Analogamente, se $F \subseteq G^+ = G^A$, allora F sarà una parte di tutte le dipendenze funzionali ottenibili applicando assiomi di Armstrong su G, dunque si ha che:

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

  dunque concludiamo che:

$$G \xrightarrow{A} F \iff F \subseteq G^+$$

- Siccome $F \subseteq G^+ \iff G \xrightarrow{A} F$ e siccome per definizione di F^+ si ha sempre che $F \xrightarrow{A} F^+$, allora concludiamo che

$$F \subseteq G^+ \iff G \xrightarrow{A} F \xrightarrow{A} F^+ \iff F^+ \subseteq G^+$$

  poiché $G \xrightarrow{A} F^+ \iff F^+ \subseteq G^+$

Definition 40

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

Dato un sottoschema $R_i \in \rho$, definiamo come **proiezione di F su R_i** l'insieme di tutte le dipendenze di $X \to Y \in F$ tali che X ed Y sono insiemi di attributi di R_i:

$$\pi_{R_i}(F) = \{X \to Y \in F^+ \mid X, Y \subseteq R_i\}$$

Theorem 13. 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:

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

Corollary 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:

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

Method 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 dependencies, F₂: set of dependencies):
    for X → Y ∈ F₁:
        if Y ⊈ X⁺_F₂:
            return False
    return True
```

Tale algoritmo viene eseguito in $O(k \cdot T(X'^+_{F_2}))$, dove $|F| = k$ e dove $T(X^+_{F_2})$ è il costo computazionale del calcolo di $X^+_{F_2}$

Dimostrazione:

- Dato $X \to Y \in F_1$, se $Y \subseteq X^+_{F_2} \iff X \to Y \in F_2^A = F_2^+$, allora

$$X \to Y \in F_1, Y \subseteq X^+_{F_2} \implies X \to Y \in F_2^+ \implies F_1 \subseteq F_2^+$$

Observation 15

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.

Method 3. Calcolo di X_G^+ tramite F

Dato uno schema R con decomposizione $\rho = R_1, \ldots, R_k$, 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
```

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

Esempi:

1.
   - Dato lo schema $R = ABC$ e l'insieme di dipendenze funzionali $F = \{A \to B, B \to C\}$, vogliamo vedere se la decomposizione $\rho = \{AB, AC\}$ preserva F.
   - Per il corollario precedentemente visto, sappiamo che è sufficiente utilizzare l'algoritmo di verifica se $F \subseteq G^+$, richiamante a sua volta l'algoritmo del calcolo di X_G^+ tramite F

- Verifichiamo quindi se $A \to B \in F$ sia anche in G^+:
  - Inizializzando l'algoritmo, dunque ponendo $Z := A$ otteniamo che:
    - $S_1 = S_0 \cup ((A \cap AB)^+_F \cap AB) = \emptyset \cup (A^+_F \cap AB) = \emptyset \cup (R \cap AB) = AB$
    - $S_2 = S_1 \cup ((A \cap AC)^+_F \cap AC) = AB \cup (A^+_F \cap AC) = AB \cup (R \cap AC) = ABC = R$
  - Siccome $S_2 = R \not\subseteq AB = Z$, allora entriamo nel ciclo while ponendo $Z := Z \cup S_2 = R$. A questo punto, all'iterazione successiva avremmo che:
    - $S_3 = S_2 \cup ((R \cap AB)^+_F \cap AB) = R \cup (AB^+_F \cap AB) = R$
    - $S_4 = S_3 \cup ((R \cap AC)^+_F \cap AC) = R \cup (AC^+_F \cap AC) = R$
  - Siccome $S_4 = R \subseteq R = Z$, allora l'algoritmo termina con $A^+_G = Z = R$, implicando a sua volta che $B \subseteq A^+_G = R \iff A \to B \in G^+$
- Verifichiamo quindi se $B \to C \in F$ sia anche in G^+:
  - Inizializzando l'algoritmo, dunque ponendo $Z := B$ otteniamo che:
    - $S_1 = S_0 \cup ((B \cap AB)^+_F \cap AB) = \emptyset \cup (B^+_F \cap AB) = \emptyset \cup (B \cap AB) = B$
    - $S_2 = S_1 \cup ((B \cap AC)^+_F \cap AC) = B \cup ((\emptyset)^+_F \cap AC) = B$
  - Siccome $S_2 = B \subseteq B = Z$, allora l'algoritmo termina con $B^+_G = Z = B$, implicando a sua volta che $B \not\subseteq A^+_G = R \iff A \to B \notin G^+$
- Dunque, concludiamo che $F \not\subseteq G^+$ e dunque che la decomposizione non preserva F

2.
- Dato lo schema $R = ABCD$ e l'insieme di dipendenze funzionali $F = \{AB \to C, D \to C, D \to B, D \to A, C \to B\}$, vogliamo vedere se la decomposizione $\rho = \{ABC, ABD\}$ preserva F.
- Siccome $AB \subseteq ABC$ e $AB \subseteq ABD$, ne segue che la dipendenza $AB \to C \in F$ venga proiettata su entrambi i sottoschemi, dunque essa sarà ovviamente preservata.
- Difatti, provando a verificare se $AB \to C \in F$ sia anche in G^+, ossia verificando se $C \subseteq AB^+_G$, abbiamo che:
  - Inizializzando l'algoritmo, dunque ponendo $Z := AB$ otteniamo che:
    - $S_1 = S_0 \cup ((AB \cap ABC)^+_F \cap ABC) = \emptyset \cup (AB^+_F \cap ABC) = \emptyset \cup (ABC \cap ABC) = ABC$
    - $S_2 = S_1 \cup ((AB \cap ABD)^+_F \cap ABD) = AB \cup (AB^+_F \cap ABD) = ABC \cup (ABC \cap ABD) = ABC$
  - Siccome $S_2 = ABC \not\subseteq AB = Z$, allora entriamo nel ciclo while ponendo $Z := Z \cup S_2 = ABC$ e ripetendo il procedimento:
    - $S_3 = S_2 \cup ((ABC \cap ABC)^+_F \cap ABC) = ABC \cup (ABC^+_F \cap ABC) = ABC \cup (ABC \cap ABC) = ABC$

    * $S_4 = S_3 \cup ((ABC \cap ABD)^+_F \cap ABD) = ABC \cup (AB^+_F \cap ABD) = AB \cup (ABC \cap ABD) = ABC$

  - Siccome $S_4 = ABC \subseteq ABC = Z$, allora l'algoritmo termina con $AB^+_G = Z = ABC$, implicando a sua volta che $C \subseteq AB^+_G = ABC \iff AB \to C \in G^+$

- Procediamo quindi verificando se $D \to A, D \to B, D \to C \in F$ siano anche in G^+, ossia verificando se $A, B, C \subseteq D^+_G$. 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 $D^+_G = Z = ABCD$, implicando a sua volta che $A, B, C \subseteq D^+_G = ABCD \iff D \to A, D \to B, D \to C \in G^+$

- Infine, verifichiamo se $C \to D \in F$ sia anche in G^+, ossia verificando se $B \subseteq C^+_G$. Calcoliamo quindi D^+_G:
  - Inizializzando l'algoritmo, dunque ponendo $Z := C$ otteniamo che:
    * $S_1 = S_0 \cup ((C \cap ABC)^+_F \cap ABC) = \emptyset \cup (C^+_F \cap ABC) = \emptyset \cup (BC \cap ABC) = BC$
    * $S_2 = S_1 \cup ((C \cap ABD)^+_F \cap ABD) = BC \cup ((\emptyset)^+_F \cap ABC) = BC$
  - Siccome $S_2 = BC \not\subseteq C$, allora entriamo nel ciclo while ponendo $Z := Z \cup S_2 = BC$ e ripetendo il procedimento:
    * $S_3 = S_2 \cup ((BC \cap ABC)^+_F \cap ABC) = BC \cup (BC^+_F \cap ABC) = \emptyset \cup (BC \cap ABC) = BC$
    * $S_4 = S_3 \cup ((BC \cap ABD)^+_F \cap ABD) = BC \cup (B^+_F \cap ABC) = BC \cup (B \cap ABC) = BC$

  - Siccome $S_4 = BC \subseteq BC = Z$, allora l'algoritmo termina con $C_G^+ = Z = BC$, implicando a sua volta che $B \subseteq C_G^+ = BC \iff C \to B \in G^+$
- Dunque, siccome tutte le dipendenze di F sono in G^+, l'algoritmo terminerà concludendo che $F \subseteq G^+$, implicando che $F \equiv G$ e quindi F venga preservato

Theorem 14. Correttezza dell'algoritmo `X`$_G^+$`_with_F`

Sia R uno schema con decomposizione $\rho = R_1, \ldots, R_k$ 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 $Z_0, Z_1, \ldots, Z_i, \ldots$ e $S_0, S_1, \ldots, S_i, \ldots$ 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 $Z_0, Z_1, \ldots, Z_i, \ldots$ 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 avrebbe 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 $A \in (Z_i \cap R_j)_F^+ \iff (Z_i \cap R_j) \to A \in F^A = F^+$

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

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

Quindi, per definizione stessa si ha che $(Z_i \cap R_j) \to A \in \pi_{R_j}(F) \subseteq G \subseteq G^+ = G^A$

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 $X \to (Z_i \cap R_j) \in G^A$

Infine, per transitività otteniamo che:

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

– Dunque, siccome in entrambi i casi si ha che $A \in X_G^+ \implies Z_i \subseteq X_G^+$

4.5.2 Join senza perdita

Theorem 15. 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.

Si verifica che ρ **presenta un join senza perdita** se per ogni istanza legale r di R si ha che:

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

Proposition 16

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

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

1. $r \subseteq m_\rho(r)$
2. $\forall i \in [1, k] \pi_{R_i}(m_\rho(r)) = \pi_{R_i}(r)$
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

$$r \subseteq m_\rho(r) \implies \pi_{R_i}(r) \subseteq \pi_{R_i}(m_\rho(r))$$

   Inoltre, per definizione di proiezione si ha che:

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

e di conseguenza 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:

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

da cui concludiamo che $\pi_{R_i}(r) = \pi_{R_1}(m_\rho(r))$

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

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

Method 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 $r = \{(r_{1,1}, \dots, r_{1,n}), \dots$ $\dots, (r_{k,1}, \dots, r_{k,n})\}$ di R, dove $\forall i \in [1,k], \forall j \in [1,n]$ si ha:

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

il seguente algoritmo determina se ρ presenta un join senza perdita:

```
def has_lossless_join(R: schema, F: set of dependencies, ρ: decomposition):
    r := costruisci_r(R,ρ)
    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] == aj:
                            t2[Aj] := t1[Aj]
                        else:
                            t1[Aj] := t2[Aj]
    if ∃ t ∈ r | t[A1] == . . . == t[An] == "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 $F = \{C \to D, AB \to E, D \to B\}$, 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_1	$b_{1,2}$	a_3	$b_{1,4}$	$b_{1,5}$
ADE	a_1	$b_{2,2}$	$b_{2,3}$	a_4	a_5
CDE	$b_{3,1}$	$b_{3,2}$	a_3	a_4	a_5
AD	a_1	$b_{4,2}$	$b_{4,3}$	a_4	$b_{4,5}$
B	$b_{5,1}$	a_2	$b_{5,3}$	$b_{5,4}$	$b_{5,5}$

- Effettuiamo quindi la prima iterazione del ciclo:
  - Considerando $C \to D \in F$, notiamo che la prima e la terza tupla sono uguali nel determinante C, dunque modifichiamo $b_{1,4} \to a_4$ affinché esse siano uguali anche nel determinato D
  - Considerando $AB \to E \in F$, notiamo che tale dipendenza è già soddisfatta, poiché non ci sono tuple uguali nel determinante AB

  - Considerando $D \to B \in F$, notiamo che la prima (poiché abbiamo già modificato $b_{1,4} \to a_4$), la seconda, la terza e la quarta tupla sono uguali nel determinante D, dunque modifichiamo $b_{2,2} \to b_{1,2}, b_{3,2} \to b_{1,2}$ e $b_{4,2} \to b_{1,2}$ affinché esse siano uguali anche nel determinato B

	A	**B**	**C**	**D**	**E**
AC	a_1	$b_{1,2}$	a_3	$b_{1,4} \to a_4$	$b_{1,5}$
ADE	a_1	$b_{2,2} \to b_{1,2}$	$b_{2,3}$	a_4	a_5
CDE	$b_{3,1}$	$b_{3,2} \to b_{1,2}$	a_3	a_4	a_5
AD	a_1	$b_{4,2} \to b_{1,2}$	$b_{4,3}$	a_4	$b_{4,5}$
B	$b_{5,1}$	a_2	$b_{5,3}$	$b_{5,4}$	$b_{5,5}$

- Siccome la tabella è stata modificata, allora effettuiamo un'altra iterazione del ciclo:
  - Considerando $C \to D \in F$, notiamo che tale dipendenza è già soddisfatta
  - Considerando $AB \to E \in F$, notiamo la prima, la seconda e la terza tupla sono uguali nel determinante AB, dunque modifichiamo $b_{1,5} \to a_5$ e $b_{4,5} \to a_5$ in modo che siano uguali nel determinato E
  - Considerando $D \to B \in F$, notiamo che tale dipendenza è già soddisfatta

	A	**B**	**C**	**D**	**E**
AC	a_1	$b_{1,2}$	a_3	a_4	$b_{1,5} \to a_5$
ADE	a_1	$b_{1,2}$	$b_{2,3}$	a_4	a_5
CDE	$b_{3,1}$	$b_{1,2}$	a_3	a_4	a_5
AD	a_1	$b_{1,2}$	$b_{4,3}$	a_4	$b_{4,5} \to a_5$
B	$b_{5,1}$	a_2	$b_{5,3}$	$b_{5,4}$	$b_{5,5}$

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

	A	**B**	**C**	**D**	**E**
AC	a_1	$b_{1,2}$	a_3	a_4	a_5
ADE	a_1	$b_{1,2}$	$b_{2,3}$	a_4	a_5
CDE	$b_{3,1}$	$b_{1,2}$	a_3	a_4	a_5
AD	a_1	$b_{1,2}$	$b_{4,3}$	a_4	a_5
B	$b_{5,1}$	a_2	$b_{5,3}$	$b_{5,4}$	$b_{5,5}$

- 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"

Theorem 17. 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 contente 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 $t_i^0 \in r^0$ si ha che $t_i^0[R_i]$ contiene tutte "a"
- Siccome l'algoritmo non modifica mai una "a" in una "b", allora si ha che $t_i^0[R_i] = t_i^f[R_i]$, dunque $t_i^f[R_i]$ contiene tutte "a"
- Sia quindi t^a la tupla contenente tutte le "a". Per costruzione di r, si ha quindi che:

$$t^a \in t_1^f[R_1] \bowtie \dots \bowtie t_k^f[R_k] \subseteq \pi_{R_1}(r^f) \bowtie \dots \bowtie \pi_{R_k}(r^f) = m_\rho(r^f)$$

- Siccome r^f è l'istanza generata al termine dell'algoritmo, il quale ricordiamo si basa sul rendere legale r, allora r^f è un'istanza legale di R, implicando quindi che $r^f = m_\rho(r^f)$ e dunque che $t^a \in m_\rho(r^f) = r^f$

4.5.3 Copertura minimale

Definition 41. 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:

- $G \equiv F$
- $\forall X \to A \in G, A \in G$, ossia il determinato di ogni dipendenza è un attributo
- $\forall X \to A \in G, \nexists X' \subset X \mid G \equiv (G - \{X \to A\}) \cup \{X' \to A\})$, ossia non è possibile determinare funzionalmente A tramite un sottoinsieme proprio di X, in modo che ogni determinante non sia ridondante
- $\nexists X \to A \in G \mid G \equiv G - \{X \to A\}$, ossia non è possibile determinate funzionalmente A tramite altre dipendenze, in modo che ogni dipendenza non sia ridondante

Method 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 $X \to A \in F$ dove $\exists X' \subset X \mid X \to A \in F$ tale che

$$G \equiv (G - \{X \to A\}) \cup \{X' \to A\})$$

   viene rimpiazzata direttamente con $X' \to A$, ripetendo tale processo ricorsivamente nel caso in cui esistano ancora dipendenze rispettanti tale condizione.

   Nel caso in cui la nuova dipendenza fosse già in F, allora la dipendenza originale viene semplicemente scartata
3. Ogni dipendenza $X \to A$ tale $G \equiv G - \{X \to A\}$ viene scartata

Observation 16

Durante lo step 2), chiamiamo F l'insieme di dipendenze originale, dunque contenente $X \to A$, mentre chiamiamo G l'insieme ridotto, dunque contenente $X' \to A$.

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

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

$$X' \subset X \implies X \to X' \in G^A = G^+$$

e conseguentemente che:

$$X \to X', X' \to A \in G^A = G^+ \implies X \to A \in G^A = G^+$$

Dunque, affinché $F \equiv G$ è sufficiente verificare che $X' \to A \in F^+$

Observation 17

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

Observation 18

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

Siccome $G \subseteq F \subseteq F^+$, è sufficiente verificare che ogni dipendenza $X \to A$ scartata sia ancora in G^+, dunque che $X \to A \in G^+ \iff A \in X_G^+$, per affermare che $F \subseteq G^+$ e quindi conseguentemente che $F \equiv G$.

Inoltre, ogni dipendenza $X \to A \in F$ 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

Observation 19

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, dunque ad ogni riduzione è necessario ricalcolare la chiusura

Esempi:

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

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

   - Consideriamo quindi la dipendenza $AB \to C \in F$ e verifichiamo se $A \to C, B \to C \in F^+$, dunque se $C \in A_F^+$ e se $C \in B_F^+$:
     - $A_F^+ = A \implies C \notin A_F^+$
     - $C_F^+ = CE \implies C \notin B_F^+$

     dunque, non possiamo rimpiazzare la dipendenza $AB \to C$
   - Procedendo analogamente, verifichiamo che $A \to E, B \to E, A \to D, B \to D \notin F^+$, dunque ne traiamo che $AB \to E, AB \to D \in F$ non possano essere rimpiazzate
   - Infine, considerando $ABC \to D \in F$, sappiamo gia che $AB \to D \in F \subseteq F^+$, dunque tale dipendenza può essere rimpiazzata data $AB \to D$ (e di conseguenza rimossa, poiché già presente in F)
   - Al termine dello step 2), quindi, abbiamo che

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

- Applichiamo quindi lo step 3)
  - C e D sono determinati rispettivamente solo da $AB \to C$ e $AB \to D$, dunque tali dipendenze non possono essere rimosse
  - Considerando $C \to E \in F$ e l'insieme provvisorio $G = \{AB \to C, AB \to D, AB \to E\}$ in cui essa è stata rimossa, si ha che $C \to E \in G^+ \iff E \in C_G^+$. Tuttavia, siccome $E \notin C_G^+ = C$, ne segue che $C \to E$ non possa essere rimossa
  - Considerando $AB \to E \in F$ e l'insieme provvisorio $G = \{AB \to C, AB \to D, C \to E\}$ in cui essa è stata rimossa, si ha che $AB \to E \in G^+ \iff E \in AB_G^+$. Siccome $E \in AB_G^+ = ABCDE$, allora possiamo rimuovere la dipendenza $AB \to E$
- Infine, otteniamo che $F = \{AB \to C, AB \to D, C \to E\}$ è la copertura minimale di F

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

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

- 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:
  - $D \in B_F^+ \implies B \to D \in F^+$ e $D \in C_F^+ \implies C \to D \in F^+$, dunque $BC \to D$ può essere rimpiazzata sia da $B \to D$ sia da $C \to D$. Tuttavia, siccome sia $B \to D, C \to D \in F$, allora possiamo scartare direttamente $BC \to D$
  - $E \notin B_F^+ \implies B \to E \notin F^+$ (e anche $E \notin C_F^+ \implies C \to E \in F^+$), dunque $BC \to E$ non può essere rimpiazzata
  - $A \in B_F^+ \implies B \to A \in F^+$ e $A \in C_F^+ \implies C \to A \in F^+$, dunque $BC \to A$ può essere rimpiazzata sia da $B \to A$ sia da $C \to A$ (dunque, in base alla scelta, otterremo due coperture minimali diverse). Nel nostro caso, sceglieremo $B \to A$
  - $L \notin B_F^+ \implies B \to L \notin F^+$ (e anche $L \notin C_F^+ \implies C \to L \in F^+$), dunque $BC \to L$ non può essere rimpiazzata
- Al termine dello step 2), quindi, abbiamo che

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

- Procediamo quindi con lo step 3):
  - Siccome E è determinato solo da $BC \to E$, allora non possiamo scartare tale dipendenza
  - Considerando $C \to D$ e $G = \{BC \to E, B \to D, E \to L, D \to A, B \to A, BC \to L\}$, si ha che $D \notin C_G^+ = C \implies C \to D \notin G^+$, dunque $C \to D$ non può essere scartata

- Considerando $B \to D$ e $G = \{BC \to E, C \to D, E \to L, D \to A, B \to A, BC \to L\}$, si ha che $D \notin B_G^+ = BA \implies B \to D \notin G^+$, dunque $B \to D$ non può essere scartata
- Considerando $E \to L$ e $G = \{BC \to E, C \to D, B \to D, D \to A, B \to A, BC \to L\}$, si ha che $L \notin E_G^+ = E \implies E \to L \notin G^+$, dunque $E \to L$ non può essere scartata
- Considerando $D \to A$ e $G = \{BC \to E, C \to D, B \to D, E \to L, B \to A, BC \to L\}$, si ha che $L \notin E_G^+ = E \implies E \to L \notin G^+$, dunque $E \to L$ non può essere scartata
- Considerando $B \to A$ e $G = \{BC \to E, C \to D, B \to D, E \to L, D \to A, BC \to L\}$, si ha che $A \in B_G^+ = ABD \implies B \to A \notin G^+$, dunque $B \to A$ può essere scartata
- Considerando $BC \to L$ e $G = \{BC \to E, C \to D, B \to D, E \to L, D \to A, B \to A\}$, si ha che $L \in BC_G^+ = ABCDEL \implies BC \to L \notin G^+$, dunque $BC \to L$ può essere scartata

- Infine, otteniamo che $F = \{BC \to E, C \to D, B \to D, E \to L, D \to A\}$ è la copertura minimale

4.5.4 Algoritmo di decomposizione

Method 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, ρ := ∅
    for A ∈ R | ∄ X → B ∈ F, A ∈ X ∨ A = B:
        S := S ∪ {A}
    if S ≠ ∅:
        R := R - S
        ρ := ρ∪ {S}
    if ∃ X → A ∈ F | X ∪ A = R:
        ρ := ρ ∪ {R}
    else: for X → A ∈ F:
            ρ := ρ ∪ {XA}
    return ρ
```

Theorem 18. 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)

Proposition 19. Ottenere una buona decomposizione

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

Data una chiave K di R e la decomposizione $\rho =$ `decompose_R(R,F)`, la decomposizione $\rho \cup K$ è sempre una **buona decomposizione** (ossia preserva F, presenta un join senza perdita ed è composta da sottoschemi in 3NF)

Di conseguenza, è sempre possibile ottenere una buona decomposizione di uno schema.

(dimostrazione omessa)