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

29 dicembre 2022

Indice

0 Introduzione **1**

1 Database e DBMS **2**
1.1 Database come sistema informativo 2
1.2 Modellazione dei dati 3
1.3 Linguaggi, Integrità e Transazioni 4

2 Modello relazionale **6**
2.1 Da relazioni a tabelle 8
2.2 Riferimenti e Valori nulli 10
2.3 Vincoli di integrità 11
2.3.1 Chiavi primarie 13
2.3.2 Chiavi esterne 14

3 Algebra relazionale **16**
3.1 Proiezione e Selezione 16
3.2 Unione, Intersezione e Differenza 19
3.3 Ridenominazione e Prodotto Cartesiano 23
3.4 Join Naturale e Theta Join 25
3.5 Quantificazione universale 30

4 Teoria relazionale **32**
4.1 Dipendenze funzionali 32
4.1.1 Chiusura di F 34

Capitolo 0

Introduzione

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:

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

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 `String` $\times$ `String` $\times$ `Integer` $\times$ `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**: $t[\texttt{Student}]$ fa riferimento al campo `Matricola` dell'ultima tupla della tabella `Students`, mentre $t[\texttt{Course}]$ 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 $A_1, \ldots, A_k$ un insieme di attributi. Una **proiezione** su R è un operatore unario che effettua una restrizione con $A_1, \ldots, A_k$ su tutte le tuple di R:

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

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

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:

$\sigma_{\texttt{Town='Roma'}}(\textbf{Customers})$		
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:

$\pi_{\texttt{Name, Code}}(\textbf{Teachers}) \cup \pi_{\texttt{Name, Code}}(\textbf{Admins})$	
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 in almeno una relazione tra R_1 e R_2.

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

Definition 21. Unione

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 almeno una relazione tra R_1 e 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 $R(A_1, \ldots, A_k)$. Una **ridenominazione** su R è un operatore unario che restituisce una nuova relazione R' con istanza r' e schema $R'(B_1, \ldots, B_k)$, dove le tuple di r' sono identiche alle tuple di r:

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

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` $\times$ `OrdersR`, otteniamo:

`Customers` $\times$ `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:

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

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

$\pi_{\texttt{Name, Town}}(\sigma_{\texttt{Qnty}>100 \wedge \texttt{Price}>2}(\textbf{CustOrdArt}))$	
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:

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

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

– 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 relazionale

4.1 Dipendenze funzionali

Definition 26. Dipendenza funzionale

Sia R una relazione con istanza r e schema $R(A_1, \ldots, A_k)$ e due sottoinsiemi di attributi $X, Y \subseteq \{A_1, \ldots, A_k\}$.

Una **dipendenza funzionale** tra X e Y, indicata come $X \to Y$ e letta "X determina Y", è 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

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

Definition 28. Chiave relazionale

Data una relazione con schema R e un insieme F di dipendenze funzionali su R, definiamo il sottoinsieme di attributi $K \subseteq R$ come **chiave** di R se:

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

Esempio:

- Consideriamo il seguente schema:

  `Student(Matr, LastName, FirstName, BirthD)`

- In questo caso, è facilmente individuabile la dipendenza funzionale

  $$\texttt{Matr} \rightarrow \{\texttt{LastName, FirstName, BirthD}\}$$

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

- Siccome non esiste alcun sottoinsieme di `Matr`, allora tale attributo sarà una chiave di `Student` (in questo caso sarà anche chiave primaria)

4.1.1 Chiusura di F

Definition 29. Chiusura di F

Data una relazione con schema R e un insieme F di dipendenze funzionali definite su R, definiamo come **chiusura di F**, indicata con F^+, l'insieme di **tutte** le dipendenze funzionali, incluse quelle non in F, soddisfatte da **ogni istanza** di R legale su F.

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

dove $L = \{r \text{ istanza di } R \mid r \text{ legale su } F\}$.

In generale, quindi, si ha che $F \subseteq F^+$.

Esempio:

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

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

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

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

Proposition 3. Dipendenza funzionale banale

Si consideri una relazione con schema R. Dato un sottoinsieme $X \subseteq R$, per ogni sottoinsieme $Y \subseteq X$ si ha che $X \to Y$ è soddisfacibile da ogni istanza di R.

Dunque, se F è un insieme di dipendenze su R, allora:

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

Tali dipendenze funzionali vengono dette **banali**, poiché vere in ogni contesto, dunque indipendenti

Dimostrazione:

- Siano $A_1, \ldots, A_k \subseteq Y$. Si vede facilmente che:

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

$$\iff \forall t_1, t_2 \in R, t_1[X] = t_2[X] \implies t_1[A_1 \cup \ldots \cup A_k] = t_2[A_1 \cup \ldots \cup A_k]$$

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

Esempio:

- Consideriamo la seguente relazione con schema $R = \{A, B, C, D\}$:

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, ad esempio, che tutte le tuple soddisfano la dipendenza $ABC \to ABC$.
- Analogamente, si nota facilmente che tutte le tuple in cui A e B sono uguali anche A è uguale, dunque $AB \to A$.
- In definitiva, si ha che:

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