Basi di Dati 1

Marco Casu

1 Introduzione

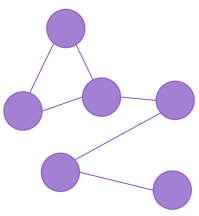
L'informazione memorizzata nei sistemi elettronici può essere di due tipi, **strutturata** e **non strutturata**. In questo corso ci occuperemo dell'informazion strutturata, ossia composta da oggetti matematici ben definiti. Un **sistema informativo** connette e contiene le informazioni, alla quale si può accedere da diversi componenti. Si prenda come esempio di sistema informativo l'archivio fisico dei documenti (ossia i dati) di una azienda, ad esso, possono accedere i vari reparti, come la sezione commercio o le risorse umane, ognuno ha a disposizione l'accesso ad un determinato sotto-insieme di dati (permessi). Prima di adoperare i sistemi informativi, ogni reparto aveva il suo personale archivio, ciò faceva si che molti dei dati fossero duplicati e presenti per più reparti, causando un elevata **ridondanza** di dati. Inoltre, due dati da 2 archivi diversi potrebbero dipendere tra loro. È quindi importante mantenere l'informazione **centralizzata**. I dati vanno organizzati, gestiti, e regolamentati da permessi di accesso a secondo dell'utente che vuole accedervi.

Un sistema informativo è composto dai seguenti componenti :

- Database (DB)
- Database Managment System (DBSM) - Ossia il software per il mantenimento dei dati
- Application Software
- Computer Hardware - La memoria nella quale è contenuto

È importante mantenere una visione astratta del modello di sistema informativo, che sia indipendente dall'hardware in uso. Fissiamo un modello che utilizzi tipi di dati che non possano variare nel tempo, strutturati in maniera completa, è necessario pensare ad un metodo formale per organizzare i dati. Vogliamo farlo in un ambiente condiviso, preoccupandoci quindi di quali e quanti dati vengono condivisi condivisi con i singoli utenti, che avranno permessi diversi e potranno accedere a dati diversi (ad esempio, uno studente può accedere ai suoi esami registrati, ma non a quelli degl altri). Inolte bisogna anche amministrare i metodi con cui si accede ai dati, preoccupandoci della concorrenza (accedere allo stesso dato nello stesso momento). I dati vengono organizzati in maniera omogenea, esiste un tipo di dato per descrivere un entità (ad esempio uno studente), e tutte le volte che si vuole immagazinare nel sistema uno studente, bisogna ricorrere allo stesso tipo di dato. L'informazione viene rappresentata da un aggregamento di più dati *grezzi*, ossia, rispettivamente la stringa "*Maurizio Ernesti*" e l'intero "3761523746", insieme rappresentano l'informazione di un professore (nome, cognome e numero di telefono).

Il modello è il modo in cui decidiamo di organizzare e collegare i dati, esistono modelli **logici**, indipendenti dalla struttura fisica, come il **modello relazionale**, ed esistono modelli **concettuali**, ossia rappresentazioni ancora più astratte indipendenti dal modello logico, i primi modelli sono stati introdotti negli anni 60.



un esempio è il modello *Mesh*, rappresentato con un grafo, dove i nodi sono i dati (i record), e gli archi le loro relazioni.

In questo modello i collegamenti sono esplicitati fisicamente (possiamo immaginare con dei puntatori), i modelli relazionali, diversamente, hanno relazioni rappresentate implicitamente dai valori stessi che contengono.

Studenti			
Matricola	Cognome	Nome	Compleanno
276545	Gialli	Lucia	25/11/1980
176515	Rossi	Mario	13/09/1982
348191	Verdi	Andrea	04/07/1981

Esami		
Studente	Voto	Corso
348191	25	01
176515	18	02
176515	27	02

La relazione tra gli esami e gli studenti (ogni studente ha n esami registrati) è data implicitamente dalla presenza del campo *Studente* nella tabella *Esami*, che equivale al campo *Matricola* della tabella *Studenti*, il modello relazionale è basato su oggetti, classi ed attributi.
Ad esempio, se volessi sapere che voto ha preso *Mario Rossi* al corso numero 02, mi basterebbe consultare la tabella *Esame*, e controllare il campo che ha la matricola equivalente a quella di *Mario Rossi*.
Lo schema logico descrive la presenza di tutte le entità con i loro rispettivi attributi, esistono poi gli schemi esterni, ossia sotto-insiemi degli schemi, destinati a determinati tipi di utenti che ne hanno l'accesso. Anche lo schema logico completo può essere schema esterno, il super-amministratore di un sistema informativo, come schema esterno, avrà l'intero schema logico, avendo accesso a tutte le entità.
Come già detto, lo schema logico rappresenta la struttura degli oggetti/entità, ed è invariata nel tempo, ma la sua istanza, ossia gli effettivi campi delle tabelle, possono variare, ed anche rapidamente. Per gestire gli schemi si utilizzano dei veri e propri linguaggi.

- **Data Definition Language (DDL)** - Per la definizione degli schemi logici ed altre operazioni generali.
- **Data Manipulation Language (DML)** - Per interrogare lo schema logico, leggerne i valori ed eventualmente modificarli.

È largamente utilizzato il linguaggio **SQL (Structured Query Language)**, che funge sia da DDL che da DML.
Un base di dati deve essere :

- Manipolabile
- Modificabile
- Centralizzata
- Il minimo ridondante
- Sicura

I dati molto spesso devono soddisfare certi vincoli (ad esempio, ogni studente ha una sola residenza) , tali vincoli son chiamati **dipendenze funzionali**, e possono riguardare anche il dominio di certi attributi (ad esempio, il voto verbalizzato di un esame deve essere maggiore o uguale a 18 e minore o uguale a 30). I dati devono essere protetti da accessi non autorizzati, è necessaria la dichiarazione di regole di accesso.
Definiamo adesso uno specifico tipo di operazione sulle basi di dati, ossia le **transizioni**, che non sono altr che sequenze ordinate di operazioni che vanno obbligatoriamente eseguite

insieme in sequenza, con il divieto assoluto che ne venga eseguita solo una parte. Facciamo un esempio, si dia il caso che su una base di dati bancaria, si vogliano trasferire 1000 euro dal conto *C1* al conto *C2*, le seguenti operazioni definite informalmente sono :

- Cercare il conto *C1*
- Sottrarne al bilancio 1000
- Cercare il conto *C2*
- Aggiungerne al bilancio 1000

Se per errore si eseguono solo i primi 2 passi, ci si ritroveranno 1000 euro persi, sottratti al primo conto, ma non addizionati al secondo. Una transizione va quindi completamente eseguita, se non dovesse essere così, l'intera sequenza di operazioni va abortita. Inoltre, è importante notare che data la concorrenzialità, in una base di dati potrebbe accadere di accedere allo stesso dato nello stesso momento, ciò potrebbe causare errori, è quindi importante evitare di lavorare contemporaneamente su uno stesso specifico campo.

2 Il Modello Relazionale

Il modello relazionale è basato sulla relazione intesa in senso matematico, una relazione non è altro che un insieme di tuple, tutte della stessa lunghezza, con elementi appartenenti a diversi domini. Il dominio è l'insieme dei possibili valori che gli elementi delle tuple possono assumere. Se prendiamo una lista di k domini, il prodotto cartesiano di tutti i k domini è l'insieme di tuple di lunghezza k.

$$D_1 \times D_2 \times D_3.... \times D_k = \{(v_1, v_2, v_3..., v_k) | v_1 \in D_1, v_2 \in D_2, v_3 \in D_3..., v_k \in D_k\} \quad (1)$$

Per esempio, per $k = 2$ consideriamo i seguenti domini $D_1 = \{White, Black\}$ e $D_2 = \{0, 1, 2\}$, si ha che :

$$D_1 \times D_2 = \{(White, 0), (White, 1), (White, 2), (Black, 0), (Black, 1), (Black, 2)\} \quad (2)$$

È una relazione di grado 2, perchè ogni tupla ha 2 coordinate.

Teorema 1 *Il grado di una relazione è equivalente al numero di elementi di ogni tupla appartenente a tale relazione.*

In tale prodotto cartesiano è possibile costruire 2^6 possibili relazioni, ossia sotto-insiemi del prodotto cartesiano.

Teorema 2 *Una relazione è un qualsiasi sottoinsieme del prodotto cartesiano.*

2.1 Notazione con Indice

Sia r una relazione di grado k, data t una tupla appartenente alla relazione r, ed i un intero da 1 a k, con $t[i]$ si intende l'elemento alla cooridnata i-esima della tupla t.

Tabella	
Black	0
White	0

Se prendiamo t come la prima riga della tabella, si avrà che $t[1] =$ "*Black*" e $t[2] = 0$.
Ricordando la notazione tabellare, in basi di dati le intestazioni delle colonne ed il loro dominio è denominato **attributo** (ad Esempio *Name : string*). In una tabella, due attributi distinti non possono avere lo stesso nome.

2.2 Rappresentazione come Funzioni

Sia R un oggetto definito come insieme di attributi, una tupla su R, ossia un istanza di tale oggetto, può essere visto come una funzione definita su R che associa ad ogni attributo A, un valore presente nel dominio di A. Considerando ciò, presa t una tupla di R, ed A uno dei suoi attributi, indichiamo con $t(A)$, il valore (ossia l'istanza) di quell'attributo preso dalla funzione t sulla variabile A.
Ad esempio, la relazione R ha la tupla $t_1 = (Paolo, Rossi, 2, 26.5)$, considerando t_1 come una funzione, essa associa ad ogni attributo A, un elemento del suo dominio :

$$f : (Nome, Cognome, Esami, Media) \rightarrow string \cup string \cup int \cup real \tag{3}$$

Da qui si ha $\quad t_1(Nome) = Paolo \quad t_1(Cognome) = Rossi \quad t_1(Media) = 26.5$
Lo schema logico, è il dominio di tale funzione, l'istanza sono le tuple, le istanze sono insiemi di tuple, ossia una relazione.

Teorema 3 *L'istanza è un sotto-insieme di tuple*

Ogni riga della tabella è una tupla distinta, ed ogni colonna corrisponde al dominio. Possiamo quindi rappresentare un oggetto della base di dati con la seguente notazione :

$$R(A_1, A_2, A_3..., A_k) \tag{4}$$

Qui R è una relazione dello schema, quindi uno schema di basi di dati non è altro che un insieme di relazioni $(R_1, R_2, R_3..., R_k)$ (invarianti nel tempo) per le quali, ognuna di esse possiede un istanza (variante nel tempo). Da qui in poi, utilizzeremo R per denominare le relazioni, ed r per le loro istanze.
Riprendendo la notazione con indice vista in precedenza 2.1, si pò utilizzare puttosto che un indice intero, l'intestazione dell'attributo per il quale si voglia leggere l'istanza.

Luoghi		
Città	Regione	Popolozione
Roma	Lazio	3000000
Milano	Lombardia	1500000
Genova	Liguria	150000

Data t_1 la prima riga dell'istanza, si ha t_1[Città]="*Roma*". È possibile anche farlo con sottoinsiemi di attributi, ossia t_1[Regione,Popolazione]=("*Lazio*",3000000). Non è importante l'ordine degli attributi come argomenti, come "risultato" riceviamo una sotto-tupla della tupla t_1, detta *restrizione*. Abbiamo visto come le istanze delle tabelle non sono altro che insiemi di tuple, esistono tante possibili istanze quanto la cardinalità del prodotto delle cardinalità dei domini.
Può succedere in certi casi, che nell'istanza di una relazione, sia presente una riga in cui un attributo è **sconosciuto**, per rappresentare tale campo nelle basi di dati, si utilizzi il valore polimorfico[1] $NULL$, utilizzato per riempire gli spazi vuoti, ad esempio in una tabella contente

[1]Appartente a tutti i domini

i dati degli utenti iscritti ad un sito, è possibile che alcuni utenti abbiano omesso il numero di telefono, per loro il campo avrà valore $NULL$, ossia sconosciuto. Si ricordi che $NULL$ è diverso da 0.

Luoghi		
Matricola	Nome	Cellulare
1039	Luca	3475746371
4316	Giorgio	$NULL$
1499	Sandro	3857482845

2.3 Integrità dei Dati

La presenza di un valore $NULL$ può causare alcuni errori, vedremo come sono presenti alcuni attributi, le quali istanze devono per forza essere dichiarate e non sconosciute. Esistono però diversi tipi di errori, come dei campi identificativi duplicati o valori fuori dominio.

Studenti		
Matricola	Nome	Media
1039	Luca	28
4316	Giorgio	33
1039	Sandro	22

- Errore 1 - Nella prima e nella terza riga sono presenti due studenti con la stessa matricola. Il campo matricola identifica ogni singolo e distinto studente, e non può essere duplicato.
- Errore 2 - Uno studente ha come media dei voti 33, è impossibile dato che i voti sono compresi tra 18 e 30, è quindi un valore fuori dominio.

Tali errori vengono definiti problemi di **integrità dei dati**, per mantenere tale integrità è necessario che le istanze delle relazioni soddisfino delle determinate proprietà dette **vincoli**. Vedremo che esistono :

- Vincoli di chiave
- Vincoli di dominio
- Vincoli funzionali
- Vincoli di esistenza

Impiegati					
Codice Impiegato	Nome	Cognome	Ruolo	Assunzione	Dipartimento
01	Luca	Rossi	Analista	1785	01
02	Mario	Verdi	Amministratore	1980	02
02	Giorgio	Neri	Ricercatore	1985	05

Dipartimenti	
Numero	Nome
01	Managment
02	Amministrazione

Vediamo come nello schema logico appena mostrato ci sono diversi vincoli da definire che non sono rispettati. Ad esempio, va definito il vincolo di dominio per cui il valore "Assunzione"≥ 1980, ossia tale valore deve essere strettamente maggiore di una certa data (possibile data di nascita dell'azienda). Si noti che non si sta rispettando un vincolo di chiave, dato che il "Codice Impiegato" presente nella seconda e nella terza riga lo stesso valore, essendo esso l'attributo identificativo, non deve essere duplicato ("Codice Impiegato" $UNIQUE$). Un altro errore meno evidente, è che alla terza riga della tabella "Impiegati", è presente un campo dipartimento con codice 05, tale campo dovrebbe collegare quella riga con il suo rispettivo dipartimento presente su un altra tabella, ma notiamo che nella tabella "Dipartimenti", non è presente alcuna riga con "Numero" identificativo 05. Un altro vincolo noto è il vincolo di esistenza, che impone ad un certo attributo di non accettare valore $NULL$.

Un altra importante distinzione da fare tra vincoli è di suddividerli in :

- **Intra-relazionali** - Vincoli definiti e da soddisfare all'interno della singola tabella (Ad esempio, un vincolo di dominio per il quale un valore deve essere sufficentemente grande).
- **Inter-relazionali** - Vincoli soddisfatti dai collegamenti di più tabelle (Un chiaro esempio è il sopra-citato errore sulla tabella del dipartimento).

2.4 Le Chiavi

In un istanza r di una relazione R, per ogni tupla è necessario che vi sia un attributo (o un insieme di attributi) X che la identifichi e distingua dalle altre tuple (che quindi non sia mai duplicato). Tale attributo/insieme di attributi è detto **chiave**, un chiaro esempio può essere il campo "Matricola" all'interno di un'ipotetica tabella studenti. Vediamo una definizione più formale.

Teorema 4
Punto 1 *- Per ogni istanza della relazione R, non esistono due tuple t_1, t_2 che hanno gli stessi valori per tutti i singoli attributi, preso un insieme di attribtui X, vale sempre $t_1[X] \neq t_2[X]$.*

$$sia\ X \in R(A_1, A_2..., A_k) | \forall t_1, t_2 \in r\ se\ t_1[X] = t_2[X] \implies t_1 = t_2 \tag{5}$$

Punto 2 *- Inoltre, non esistono sotto-insiemi di X che soddisfino la condizione sopra-citata.*

$$\forall X' \subseteq X, \exists t_1, t_2 \in r\ tale\ che\ t_1[X'] = t_2[X'] \wedge t_1 \neq t_2 \tag{6}$$

Approfondendo il punto 2, se esiste una relazione R con chiave $X = (A_1, A_2, A_3)$, è chiaro che, nella sua istanza r, non esisteranno due righe con gli stessi valori assegnati X, quindi, preso il sotto-insieme $X' = (A_1, A_2)$, se esistono due tuple t_1, t_2 tali che $t_1[X'] = t_2[X']$, per forza di cose esse non saranno la stessa tupla in quanto, per il punto 1, differiranno per l'attributo A_3. È importante per una relazione che abbia una chiave significativa basata sull'informazione che rappresenta (Una relazione che rappresenta degli studenti, non può avere come chiave il campo del nome, dato che potrebbero esserci 2 studenti con lo stesso nome, bensì si prediliga la matricola), si sceglie quindi una **chiave primaria**, ovviamente con vincolo di esistenza.

2.4.1 La Chiave Esterna

Può esistere inoltre un attributto nelle relazioni detto **chiave esterna** o **foreign key**, essa identifica all'interno di una relazione, un attributo associato ad un altra relazione (un altra tabella), identificandone la chiave primaria.

Studenti			
Matricola	Cognome	Nome	Compleanno
276545	Gialli	Lucia	25/11/1980
176515	Rossi	Mario	13/09/1982
348191	Verdi	Andrea	04/07/1981

Esami		
Studente	Voto	Corso
348191	25	01
176515	18	02
176515	27	02

Nella tabella "*Esami*", l'attributo "*Studente*" è una chiave esterna che identifica e collega tale tabella con la relazione "*Studenti*", tramite la sua chiave primaria "*Matricola*", Dato che ad ogni esame è associato uno studente che l'ha sostenuto. Il vincolo di integrità inter-relazionale precedentemente citato impone che per ogni valore presente su un attributo di una chiave esterna, esista il suo corrispettivo campo con tale valore nella relazione alla quale fa riferimento. Si ricordi che tale vincolo non è violato dalla presenza di un valore $NULL$.

Multe			
Codice	Data	Ufficiale	Targa
4312	01/12/1988	001	AA123AA
1351	12/04/1989	*NULL*	KA194AR
9572	10/10/1990	002	ND193MF

Ufficiali		
Codice	Nome	Cognome
001	Giancarlo	Pozzi
002	Sara	Tua
003	Nicola	Canti

Può quindi esistere un istanza dove la chiave esterna ha valore $NULL$, conseguentemente non avrà nessun riferimento nella relazione alla quale è collegata.

2.5 Le Dipendenze Funzionali

Come si possono definire facilmente i vincoli di integrità dei dati2.3 visti nei paragrafi precedenti? Nelle basi di dati si utilizzano le note **dipendenze funzionali**, ossia un insieme di attributi che dipende da un altro insieme di attributi all'interno dello stesso schema. Tale definizione può sembrare poco esplicativa, ma è di vitale importanza in questo paragrafo che lo studente abbia ben saldo in mente il concetto di dipendenza funzionale, in quanto centrale nel corso di *Basi di Dati*. Formalmente, una dipendenza funzionale stabilisce un collegamento semantico tra due distinti insiemi di attributi X e Y appartenenti allo stesso schema. Si scrive:

$$X \to Y \tag{7}$$

e si legge : "X **determina** Y", stabilendone dei vincoli di integrità.

Teorema 5 *Sia r un istanza della relazione R, la dipendenza funzionale $X \to Y$ è soddisfatta se :*

- *Sia X che Y sono due sotto-insieme distinti di R.*
- *Le tuple di r che sono identiche per X, sono anche identiche per Y.*

$$\forall t_1, t_2 \in R \text{ se } t_1[X] = t_2[X] \implies t_1[Y] = t_2[Y] \tag{8}$$

3 Algebra relazionale

Tramite **l'algebra relazionale** possiamo fare interrogazioni alla nostra base di dati per ottenere informazioni su una porzione di essa, le *query* sono scritte in linguaggio SQL, e vengono tradotte nel linguaggio formale e procedurale dell'algera relazionale. L'algebra relazionale fornisce degli operatori che lavorano sulle istanze delle nostre relazioni, vi sono 4 tipi di operatori :

- Operatori di rimozione sulle singole relazioni
- Operatori insiemistici
- Operatori che combinano tuple da relazioni diverse
- Operatore di rinomina

Vediamo nel dettaglio tutti gli operatori che abbiamo a disposizione :

3.0.1 Proiezione

L'operatore di proiezione, indicato con π, esegue un *taglio verticale* su una tabella, selezionando un sotto-insieme degli attributi, creando una tabella come quella iniziale, ma con esclusivamente le istanze degli attributi selezionati.

$$\pi_{A_1,A_2,..A_k}(R)$$

Esempio :

Sia :

Customer		
Code	Name	Town
001	Giancarlo	Roma
002	Sara	Cagliari
003	Nicola	Roma

ho che $\pi_{Name}(Customer) =$

Roma
Cagliari

3.0.2 Selezione

La selezione, indicata con σ, esegue un *taglio orizzontale* sulla tabella, ossia, seleziona tutte le righe che soddisfano un determinato vincolo C, tale vincolo è un sepressione booleana della forma $A\Theta B$, dove $\Theta \in \{<, >, =, \leq, ge\}$. La condizione ovviamente per esser valida, ha bisogno che A e B abbiano lo stesso dominio (non posso comparare un intero con una stringa).

$$\sigma_C(R)$$

Esempio :

Sia :

Customer		
Code	Name	Town
001	Giancarlo	Roma
002	Sara	Cagliari
003	Nicola	Roma

ho che :

$\sigma_{Town="Roma"}(Customer) =$

Code	Name	Town
001	Giancarlo	Roma
003	Nicola	Roma

Ovviamente posso utilizzare gli operatori logici per eseguire richieste multiple :

$\sigma_{Town=\text{"}Roma\text{"} \wedge Code=001}(Customer) =$

Code	Name	Town
001	Giancarlo	Roma

Vale la proprietà : $\sigma_C(\sigma_{C'}(R)) = \sigma_{C \wedge C'}(R)$

3.0.3 Unione

L'operazione di unione fra due relazioni, denominata con il simbolo $\cup$, come nel senso insiemistico, crea una nuova istanza della relazione, contenente le tuple da entrambe le relazioni. Non si possono unire due relazioni qualsiasi, è necessario che esse siano **union-compatibili**, ossia che abbiano lo stesso numero di attributi, e che gli attributi corrispondenti abbiano lo stesso dominio.

Siano :

Insegnanti	
Code	Name
001	Giancarlo
002	Sara
003	Nicola

Admin	
Code	Name
001	Luca
004	Andrea
005	Carlo

Si ha che :

$Insegnanti \cup Admin =$

Code	Name
001	Giancarlo
002	Sara
003	Nicola
004	Andrea
005	Carlo

Si noti come la riga | 001 | Luca | non appare nell'unione, dato che condivide la stessa chiave con la riga | 001 | Giancarlo | , quindi, quando si uniscono due relazioni, bisogna fare attenzione che due righe da esse non condividano la stessa chiave, altrimenti si avrà una perdita di informazioni. Si noti che, se una delle due tabelle avesse presentato un attributo di troppo, esso sarebbe potuto essere stato rimosso con una proiezione, rendendole comunque union-compatibili.

3.0.4 Differenza

Come per l'unione, è necessario che le due relazioni sulla quale voglio applicare la differenza siano union-compatibili. Si indica con il simbolo $-$, ed il risultato fra due relazioni, conterrà le tuple del primo operando, che non stanno anche nel secondo operando.

Siano :

Insegnanti	
Code	Name
001	Giancarlo
002	Sara

Admin	
Code	Name
002	Sara

Si ha che :

$Insegnanti - Admin =$

Code	Name
001	Giancarlo

La differenza non è commutativa! Infatti, in questo caso $Admin - Insegnanti = \emptyset$

3.0.5 Intersezione

Anch'essa richiede che i due operandi siano union-compatibili. Si indica con $\cap$, ed il risultato conterrà esclusivamente le tuple che fanno parte sia della prima che della seconda relazione. (si noti che $A \cap B = A - (A - B)$).

Siano :

Insegnanti	
Code	Name
001	Giancarlo
002	Sara

Admin	
Code	Name
002	Sara
004	Andrea

Si ha che :

$Insegnanti \cap Admin =$

Code	Name
002	Sara

Vediamo adesso gli operatori della terza categoria, che creano **relazioni multiple** combinando tabelle eterogenee.

3.0.6 Prodotto Cartesiano

Il prodotto cartesiano è indicato con il simbolo $\times$, e rappresenta tutte le possibili combinazioni di righe fra gli elementi del primo operando con gli elementi del secondo, tale operazione risulta parecchio dispendiosa, e potrebbe associare elementi che non hanno nessuna correlazione sensata, ad esempio :

Siano :

Customer	
Code	Name
001	Giancarlo
002	Sara

Order	
CO	Customer
AX00	001
AX01	002

Si ha che :

$Customer \times Order =$

Code	Name	CO	Customer
001	Giancarlo	AX00	001
001	Giancarlo	AX01	002
002	Sara	AX00	001
002	Sara	AX01	002

Logicamente però, stiamo associando a dei clienti, degli ordini che non gli appartengono (il campo *Customer* di *Order* è la foreign key che identifica il campo *Code* della tabella *Customer*). Quindi, se volessimo una tabella con ogni cliente e gli ordini ad esso associati, combineremo anche una selezione del tipo :

$\sigma_{Customer.Code=Order.Customer}(Customer \times Order) =$

Code	Name	CO	Customer
001	Giancarlo	AX00	001
002	Sara	AX01	002

È chiaro che, per tabelle con numerosi elementi, il prodotto cartesiano da computare risulta parecchio dispendioso, dato che, se R_1 ha n elementi, ed R_2 ha m elementi, il prodotto cartesiano $R_1 \times R_2$ avrà $n \cdot m$ elementi, e quando si opera su basi di dati con milioni di righe, tale operazione deve essere evitata quando possibile, per questo esiste un'operazione simile al prodotto cartesiano, che seleziona automaticamente le righe da combinare secondo parametri ben precisi.

3.0.7 Join

L'operatore join, indicato dal simbolo $\bowtie$, seleziona le tuple del prodotto cartesiano che soddisfano una precisa condizione C, ossia che gli attributi con lo stesso nome, debbano avere anche lo stesso valore, è quindi importante che si denominino in maniera precisa gli attributi di relazioni che si vogliono combinare con il join.

$$R_1 \bowtie R_2 = \sigma_C(R_1 \times R_2) \tag{9}$$

$$C = R_1.A_1 = R_2.A_1 \wedge R_1.A_2 = R_2.A_2 \wedge ... \wedge R_1.A_k = R_2.A_k \tag{10}$$

dove $A_1, A_2..., A_k$ sono gli attributi condivisi con lo stesso nome fra R_1 e R_2. La tabella risultante, non presenterà due volte gli attributi da R_1 e R_2 con lo stesso nome, ma li unirà in un attributo solo.
Se dovesse capitare che le due relazioni non hanno alcun attributo in comune, il risultato sarà il prodotto cartesiano.

Siano :

Customer	
Code	Name
001	Giancarlo
002	Sara
003	Lucia

Order		
CO	Code	Item
AX00	001	Glue
AX01	002	Bricks
AX02	001	Shoes

Si ha che :

$Customer \bowtie Order =$

Customer.Code	Name	CO	Item
001	Giancarlo	AX00	Glue
001	Giancarlo	AX02	Shoes
002	Sara	AX01	Bricks

3.0.8 Θ-Join

Il Θ-Join (Che si pronuncia "theta join"), selezione le tuple risultanti dal prodotto cartesiano, che soddisfino la condizione $A\Theta B$ dove $\Theta \in \{<, >, =, \leq, ge\}$ ed A, B sono attributi rispettivamente della prima, e seconda relazione (il dominio di A deve essere lo stesso del dominio di B).

$$R_1 \bowtie_{A\Theta B} R_2 = \sigma_{A\Theta B}(R_1 \times R_2) \tag{11}$$

3.0.9 Condizioni Negative

È possibile utilizzare il simbolo $\neg C$ per intendere che si vogliono selezionare tutte le tuple che non soddisfino la condizione C.

Sia :

Customer		
Code	Name	Town
001	Giancarlo	Roma
002	Sara	Cagliari
003	Nicola	Roma
004	Gianluca	Venezia

ho che :

$\sigma_{\neg(Town="Roma")}(Customer) =$

002	Sara	Cagliari
004	Gianluca	Venezia

3.1 Quantificazione Universale

Per ora tutti gli operatori che abbiamo visto, implicano una quantificazione esistenziale, quando seleziono da una relazione le righe che soddisfano una condizione C, seleziona le righe anche se vi è un altro elemento che non soddisfa tale condizione.

Se seleziono tutti i clienti che hanno ordini con prezzo superiore a 100 euro, selezionerà tutti i clieni che hanno **almeno** un ordine superiore a 100 euro, non solo quelli che hanno **esclusivamente** ordini superiori a 100 euro.
Vogliamo scrivere query che implichino la quantificazione universale e non esistenziale, ossia, che tale condizione valga "**per ogni**", e ciò è equivalente a "**non c'e ne sono tali che**":

- I clienti che hanno solo ordini superiori a 100 euro = I clienti che non hanno ordini inferiori o uguali a 100 euro

Siano :

Customer	
Code	Name
001	Giancarlo
002	Sara
003	Lucia

Order		
CO	Code	Price
AX00	001	90
AX01	002	120
AX02	001	200

Vogliamo elencare nome e codice dei clienti che hanno fatto ordini esclusivamente con prezzo superiore a 100 euro, quindi, *Giancarlo* non sarà incluso, dato che ha fatto si un ordine da 200 euro, ma anche uno da 90 euro. La nostra query Q sarà :

$$Q = Customer - \pi_{Code,Name}(\sigma_{Price \leq 100}(Customer \bowtie Order)) \tag{12}$$

Quindi, prendere tutti coloro che hanno fatto solo ordini superiori a 100 euro, equivale a prendere il totale, e sottrarne coloro che hanno fatto almeno un ordine inferiore o uguale a 100 euro.

$Q =$

Code	Name
002	Sara

3.2 Esempi di Esercizi

3.2.1 Esempio 1

Ci sono situazioni in cui dobbiamo combinare una relazione con se stessa per ottenere paia di tuple della stessa tabella, vediamo un esempio, si osservino le seguenti relazioni :

Impiegati			
Name	Cod	Salario	CodSup
Rossi	C1	100	C3
Pirlo	C2	200	C3
Bianchi	C3	500	NULL
Verdi	C4	200	C2
Neri	C5	150	C1
Tosi	C6	100	C1

Dove *CodSup* identifica il *Cod* della riga che ogni impiegato ha come supervisore. Ad esempio, Rossi ha come supervisore Bianchi, e Neri ha come supervisore Rossi. Vogliamo trovare tutti gli impiegati che hanno lo stipendio superiore o uguale al loro supervisore. Per procedere possiamo combinare tramite il prodotto cartesiano la tabella con se stessa, prima però rinominando ogni attributo aggiungendoci una C davanti, per identificare una tabella che è copia dell'altra:

$$ImpiegatiC = \rho_{Name,Cod,Salario,CodSup \to CName,CCod,CSalario,CCodSup}(Impiegati) \tag{13}$$

Adesso procediamo col combinare tali tabelle, però selezionando esclusivamente le tuple che hanno il *CodSup* ed il *CCod* identici, in modo che avremo una lista dei dipendenti con a destra il loro superiore.

$$\sigma_{CodSup=CCod}(Impiegati \times ImpiegatiC) \tag{14}$$

Name	Cod	Salario	CodSup	CName	CCod	CSalario	CCodSup
Rossi	C1	100	C3	Bianchi	C3	500	NULL
Pirlo	C2	200	C3	Bianchi	C3	500	NULL
Verdi	C4	200	C2	Pirlo	C2	200	C3
Neri	C5	150	C1	Rossi	C1	100	C3
Tosi	C6	100	C1	Rossi	C1	100	C3

Fatto ciò adesso, ci basta selezionare quelli che hanno il salario superiore o uguale a quello del proprio supervisore, ossia $Salario \geq CSalario$:

$$\sigma_{(CodSup=CCod) \wedge (Salario \geq CSalario)}(Impiegati \times ImpiegatiC) \tag{15}$$

Name	Cod	Salario	CodSup	CName	CCod	CSalario	CCodSup
Verdi	C4	200	C2	Pirlo	C2	200	C3
Neri	C5	150	C1	Rossi	C1	100	C3
Tosi	C6	100	C1	Rossi	C1	100	C3

3.2.2 Esempio 2

Vediamo adesso un esempio di un altro tipo, si consideri sempre la stessa relazione di prima :

Imp			
Name	Cod	Salario	CodSup
Rossi	C1	100	C3
Pirlo	C2	200	C3
Bianchi	C3	500	NULL
Verdi	C4	200	C2
Neri	C5	150	C1
Tosi	C6	100	C1

Adesso vogliamo trovare l'impiegato che il salario più alto, ma come possiamo fare? Procederemo con il comparare ogni impiegato con tutti gli altri, seleziondo esclusivamente quelli che hanno lo stipendio inferiore all'impiegato con la quale sono stati comparati (selezioniamo esclusivamente il codice impiegato):

$$Imp2 = Imp \tag{16}$$

$$\pi_{Imp.cod}(Imp \bowtie_{Imp.Salario < Imp2.Salario} Imp2) \tag{17}$$

Ottengo la tabella :

Impiegati.Cod
C1
C2
C4
C5
C6

Che rappresenta tutti gli impiegati che hanno qualcuno con lo stipendio superiore al loro. Quindi, chi non è presente in questa tabella, sarà l'impiegato con lo stipendio più alto. Prendiamo allora tutti gli impiegati e sottraiamo ad essi la nostra tabella.

$$\pi_{Cod}(Imp) - \pi_{Imp.cod}(Imp \bowtie_{Imp.Salario<Imp2.Salario} Imp2) =$$

$$= \begin{array}{|c|}\hline C1 \\ \hline C2 \\ \hline C3 \\ \hline C4 \\ \hline C5 \\ \hline C6 \\ \hline \end{array} - \begin{array}{|c|}\hline C1 \\ \hline C2 \\ \hline C4 \\ \hline C5 \\ \hline C6 \\ \hline \end{array} = \begin{array}{|c|}\hline C3 \\ \hline \end{array}$$

Quindi l'impiegato di codice $C3$, ossia | Bianchi | C3 | 500 | NULL | è colui con lo stipendio più alto.

4 Design di un Database

L'obbiettivo è capire come creare uno schema in maniera corretta, immaginiamo di dover creare una base di dati per memorizzare le informazioni relative agli studenti di un corso di laurea triennale, ed i relativi esami sostenuti.

Matricola	SurN	Name	BirthD	City	Prov	ExCode	ExName	Doc	Date	Grade
01	Rossi	Mario	...	Roma	Roma	10	Physics	Pippo	...	28
02	Bianchi	Paolo	...	Tolfa	Roma	10	Physics	Pippo	...	26
01	Rossi	Mario	...	Roma	Roma	20	Chemistry	Pluto	...	27

È estremamente sbagliato salvare tutti i dati in una sola relazione, per ogni esame, devo salvare ogni volta tutti i dati di uno studente, creando ridondanza e spreco di memoria, inoltre se uno studente non ha sostenuto alcun esame, non apparirà nell'archivio, ed un esame che non è stato sostenuto da nessuno risulterà inesistente. È corretto suddividere tale relazione in 3 diverse tabelle :

Studenti					
Matricola	SurN	Name	BirthD	City	Prov
01	Rossi	Mario	...	Roma	Roma
02	Bianchi	Paolo	...	Tolfa	Roma
01	Rossi	Mario	...	Roma	Roma

Corso		
ExCode	ExName	Doc
10	Physics	Pippo
20	Chemistry	Pluto

Esami			
Matricola	ExCode	Data	Grade
01	10	...	28
01	20	...	27
02	10	...	26

4.1 Definizioni Formali

Uno **schema relazionale**, che si denota con R è un insieme di attributi del tipo:

$$R = \{A_1, A_2, A_3..., A_k\}$$

Solitamente un sotto insieme di attributi di R viene indicato con le lettre X oppure Y, e l'unione si può denotare $XY \equiv X \cup Y$. Data una relazione R, una **tupla** r è una funzione che associa ad ogni attributo, un valore appartenente al suo dominio :

$$R = \{nome, cognome\}\ r[nome] = marco, r[cognome] = rossi$$

Preso $X \subset R$, diciamo che due tuple r_1, r_2 **coincidono** su X se :

$$\forall A \in X, r_1[A] = r_2[A]$$

Corso		
Nome	Cognome	Voto
Paolo	Rossi	29
Mario	Rossi	29

Le due tuple coincidono su {Cognome,Voto}

Un **istanza** su uno schema relazionale R è l'insieme di tutte le tuple su R. Una **dipendenza funzionale** su R è una coppia ordinata X, Y di sotto-insiemi di R, ossia di attributi di R, che si denota con $X \to Y$, ed indica che :

$$\text{per ogni coppia di tuple } r_1, r_2, \text{ vale che } r_1[X] = r_2[X] \implies r_1[Y] = r_2[Y]$$

E si dice X *determina* Y, dove X è detto determinante ed Y dipendente. Le dipendenze funzionali esprimono dei *vincoli*. Uno schema R può avere un **insieme di dipendenze funzionali** denotato con F :

$$F = \{A \to B, D \to B..., G \to H\}$$

se un istanza soddisfa ogni dipendenza di F, ossia tutte le dipendenze funzionali, è detta istanza **legale**. Le dipendenze godono di una relazione di *transitività*, ossia, se $A \to B$ e $B \to C$, allora $A \to C$, non è però necessario aggiungere quest'ultima all'insieme F, dato che è implicita, e vogliamo mantenere F il più piccolo possibile. Ci sono quindi dipendenze funzionali che sono soddisfatte per ogni istanza della relazione, che però non è necessario includere in F, un altro esempio :

$$\text{se } taxCode \to name, surname$$
$$\text{è ovvio che } taxCode \to name \text{ e } taxCode \to surname$$

Ma queste due ultime non saranno incluse in F. Tali dipendenze sono dette **banali**, ad esempio, se $Y \subset X$, se $A \to X$ ovviamente $A \to Y$. Il numero delle dipendenze banali è *esponenziale* :

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

Tutte le dipendenze banali che non è necessario includere in F, si trovano in un insieme più grande detto **chiusura di** F, che si denota con F^+, contenente tutte le dipendenze funzionali soddisfatte da un'istanza, è chiaro che $F \subseteq F^+$.
Passiamo alla definizione di **chiave**. Un sotto-insieme di attributi K è detto chiave se :

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

Detto in maniera meno formale, non esisteranno due tuple di una relazione con gli stessi valori per le chiavi. Nei linguaggi come *SQL*, fra gli attributi della chiave se ne definisce uno in particolare detto **chiave primaria**,che non può assumere valore `NULL`.