***01/03/2022: Introduzione alle basi di dati e concetti base/Capitolo 1***

Le *basi di dati* sono un insieme consistente, persistente, organizzato e condiviso dei dati utilizzati. Al di là di questo, si parla di dati, entità presente prima di ogni elaborazione e che devono essere interpretate sotto forma di informazioni. I dati stessi codificano le informazioni dando rappresentazioni più precise di informazione e conoscenza, magari per analisi o altri scopi.

Essi sono gestiti dai *DBMS (Database Management System)* garantendo privacy, affidabilità, efficienza ed efficacia. In generale i database possono contenere molti dati, ammettiamo per esempio terabyte di dati o miliardi di record (*grande*), indipendente dalla singola esecuzione di un programma (*persistente*), dando appunto ai vari utenti accesso in vari modi alle porzioni della base di dati acceduta (*condivisa*).

Naturalmente si deve garantire ridondanza, informazioni ripetute e che devono essere mantenute in maniera sicura per evitare incoerenze. A tale scopo, attenzione al controllo della concorrenza, magari anche con utilizzo di strutture come lock o semafori, assieme a meccanismi specifici di autenticazione.

L’uso dei DBMS cerca di garantire *l’affidabilità*, resistendo a malfunzionamenti hardware/software, gestendo le *transazioni*, quindi un insieme di operazioni che modificano lo stato di una base di dati. Prendiamo l’esempio di trasferimento di fondi, classico esempio del prelievo/versamento, quindi rimozione e/o aggiunta tra basi di dati, magari anche in contemporanea; qui si vede che l’ordine delle operazioni è fondamentale.

Naturalmente le risorse devono essere bastevoli tali che il sistema informativo stesso possa essere considerato efficiente. A noi interessa di più *l’efficacia* rispetto all’efficienza.

I dati sono organizzati normalmente secondo modelli di dati utilizzati per organizzare dati di interesse e quindi descriverne la dinamica, quindi con il modello relazionale, creando degli insiemi di record omogenei. Ogni riga è una *tupla* (prodotto cartesiano di n domini, matematicamente parlando), combinando tra di loro tutti i possibili elementi; in altri termini è un singolo elemento di un database relazionale caratterizzato da uno o più attributi. Un esempio è:

Immagine che contiene tavolo

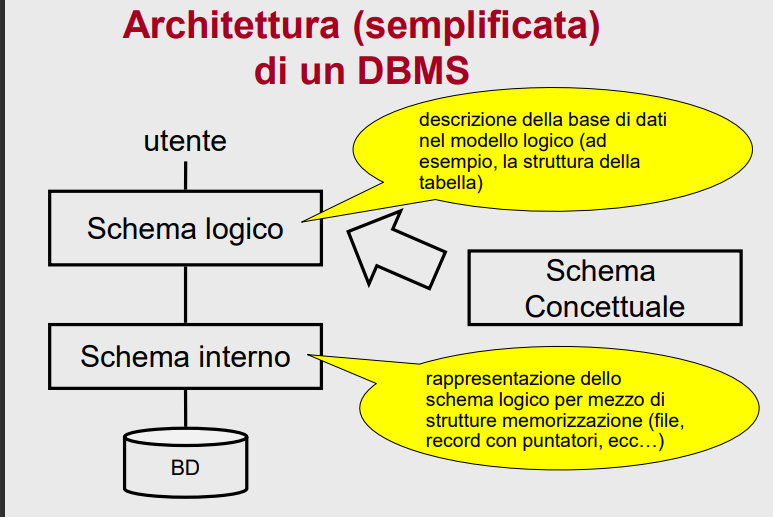
Descrizione generata automaticamente

In ogni base di dati vi è lo *schema,* che descrive la struttura di un database (aspetto intensionale, quindi che non cambia nel tempo, es. intestazioni delle tabelle) e l’*istanza*, con i valori attualmente utilizzati e che possono cambiare anche molto rapidamente (ad es. il corpo di ciascuna tabella).

In generale avremo due tipi di modelli:

1. *logici*, adottati dai DBMS organizzando i dati a livello logico, sono utilizzati dai programmi e sono indipendenti dalle strutture fisiche. Possiamo per esempio enunciare modelli di tipo relazionale, reticolare, gerarchico, ad oggetto, XML, ecc.
2. *concettuali*, rappresenta i dati in modo indipendente, cercando di descrivere i concetti reali e spesso usati nelle fasi preliminari di progettazione. Il modello in questo senso più diffuso è il cosiddetto *Entity-Relationship.*

A tale scopo definiamo così l’archittettura, intendendo come utente ciò che viene creato dalla macchina finale per gestire tutto ciò; più in generale segue lo schema ANSI/SPARC (considerato standard).



A noi interessa soprattutto il livello *logico* piuttosto che quello *fisico*, indipendenti l’uno dall’altro. Si accenna anche lo scherma *esterno* (schema logico e interno visibile da altre entità). Se io volessi verificare una serie di informazioni, esso viene fatto tramite una *vista*, vedendo una certa porzione di database in un certo momento con una query, ad esempio, e ciò viene fatto proprio per semplificarle e garantire privacy nel loro accesso.

Altro contributo all’efficacia delle basi di dati tramite linguaggi testuali interattivi (SQL), o anche comandi immersi in un linguaggio ospite (Pascal, Java, C, interagendo con SQL), tramite interfacce amichevoli (quindi non testuali).

Un esempio di interrogazione/query fatta per mezzo di SQL:



La nostra interazione grafica verrà fatta con POSTGRES. Nel caso pratico, anche cambiando sistemi di interazioni con le basi di dati, è possibile agire ugualmente sul sistema. La gestione è centralizzata e ha possibilità di standardizzazione purché si usino linguaggi simili al SQL standard. Questo funziona bene nelle grandi organizzazioni, ma in caso di pochi utenti vanno anche bene file semplici.

Le operazioni possibili su una base di dati avvengono solitamente tramite *DML (Data Manipulation Language)*, interrogando e/o aggiornando istanze delle basi di dati e *DDL (Data Definition Language)*, definendo schermi (logici, esterni, fisici) e altre operazioni generali (esempio, CREATE TABLE in SQL).

In generale i DBMS permettono ai dati di essere una risorsa comune, condivisa nell’organizzazione, centralizzata con possibilità di standardizzazione, riducendo inconsistenze/ridondanze e favorendo lo sviluppo indipendente delle applicazione, mantenendo i dati separati ma condivisi in un’organizzazione comune. Nel caso di sistemi informativi non particolarmente grandi non conviene adottare un DBMS, perché complesso e costoso; ciò è più consigliato nel caso di medio/grandi organizzazioni, purché si abbia una organizzazione concorrente, stabile ed adeguata.

***02/03/2022: Modello relazionale/Capitolo 2***

Il *modello relazionale* si basa sul concetto di *relazione*, quindi naturale rappresentazione per mezzo di tabelle. Abbiamo quindi n insiemi anche non distinti, descritti come D1…Dn da cui leghiamo i singoli elementi sotto forma di relazione matematica. Non viene violato il concetto di insieme, quindi nelle n-uple non ci sta un ordine prestabilito, le n-uple sono distinte e ogni n-upla è ordinata, cioè l’i-esimo valore proviene dall’i-esimo dominio; questo non è sempre vero, potendo adattare la struttura posizionale alle esigenze dei sistemi informativi.

Ogni dominio/colonnna è associato ad un nome unico (attributo) e, più in sintesi, riassumiamo questi fatti:

1. l’ordinamento tra le righe è irrilevante

2. l’ordinamento tra le colonne è irrilevante

3. le righe sono diverse fra loro

4. le intestazioni delle colonne sono diverse tra loro

5. i valori di ogni colonna sono definiti su domini omogenei

Ovviamente esistono dei legami tra i valori, in cui dati diversi vengono rappresentati per mezzo di domini di valori. Un esempio possibile è il collegamento per valore, con due colonne tra tabelle diverse che possiedono lo stesso valore e quindi hanno una correlazione logica. Essendo un dominio ciascuna tupla avrà una sua corrispondenza precisa e corrispondente.

Parliamo quindi di *relazione* (R(A1 ,..., An )) tra attributi, rappresentato sotto forma matematica, per poi raggruppare un insieme di relazione sotto forma di *schema di base di dati (*R = {R1(X1 ),.., Rk(Xk)}

Un esempio di istanza e tupla, concetti molto legati tra di loro, è il seguente:

Immagine che contiene testo

Descrizione generata automaticamente

L’insieme di tuple o elementi di una tabella è chiamata *istanza di una relazione*, mentre l’insieme delle tuple cambierà nel tempo e con esse anche le relazioni tra le tuple, conseguentemente parliamo di *istanza di base di dati* descrivendo l’insieme di relazioni. Non è possibile che un attributo, già di per sé tupla, contenga più di una riga. Si nota che le informazioni possono essere duplicate, che rappresenta di per sé uno spreco di spazio ed è un rischio forte di incoerenza.

Ecco quindi che l’idea migliore può essere di spezzettare le informazioni tra più tabelle, collegando per esempio tra di loro i singoli valori. Un’altra idea è l’aggiunta di attributi ulteriori solidificando la struttura (ad esempio, nel caso degli scontrini aggiungo il numero di riga oppure l’ora, giusto per dare un’idea). Non tutte le informazioni sono sempre subito disponibili; una soluzione può essere l’utilizzo di un valore nel dominio (“0”, “99”, stringa nulla, ecc.); tuttavia potrebbero esistere valori “non utilizzati” e quindi implementare controlli appositi, indebolendo il design di un database.

La soluzione, quindi, è l’utilizzo di un *valore nullo*, denotante l’assenza di un valore del dominio e di cui non ne fa parte; si possono anche aggiungere restrizioni eventualmente dicendo che non ci può essere un certo valore. Attenzione che i DBMS possono anche considerare valori nulli come valori sconosciuti, inesistenti oppure anche senza informazione.

Da questo punto di vista, *attenzione a non avere troppi valori nulli* (es. delle slide, tutti gli studenti devono avere una matricola, un voto non può essere dato ad uno studente sconosciuto senza corso, ecc.)

Oltre che *sintatticamente corretta* (cioè tutte le tuple sono diverse*)*, una istanza di una Base di dati può essere *semanticamente scorretta* (cioè dati impossibili per l’applicazione di interesse).

Per poter garantire entrambe queste proprietà, si usano i vincoli di integrità, funzioni booleane che restituiscono i valori vero/falso.

Abbiamo quindi due casi, quindi vincoli “supportati” dai DBMS nativamente (metti il caso in cui il DBMS “sappia già” che uno studente non può essere vuoto e che non esista il 27 e lode), rigettando tuple che violano i vincoli, oppure vincoli “non supportati” nativamente, in cui l’applicazione deve garantire il non inserimento di dati non conformi.

I vincoli che si riferiscono ad una sola relazione/tabella sono i *vincoli intrarelazionali*, oppure possono essere *interrelazionali*, coinvolgenti invece più tabelle. Altri esempi di vincoli sono i *vincoli di dominio*, che esprimono condizioni sui valori di ciascuna tupla, indipendentemente dalle altre, i *vincoli di tupla* che riguardano le singole righe/tuple (esempio banale la somma del netto e delle ritenute fiscali che deve ammontare al lordo), i *vincoli di chiave*, dove la chiave deve essere sempre univoca e con valori diversi su tutti gli altri attributi non chiave.

Non ci sono due tuple con lo stesso valore di chiave naturalmente (per esempio due matricole magari); esiste anche il concetto di *superchiave*, cioè se in una relazione non ci sono due tuple con gli stessi valori per tutti gli attributi dell’insieme considerato. Una superchiave identifica le tuple di una relazione; se tolgo un attributo, una superchiave rimane tale.

Una *chiave* è quindi una superchiave minimale (formata da un solo attributo) e una superchiave è chiave se non ha altre superchiavi al suo interno. Una chiave non può essere sottoinsieme di un’altra chiave e non esiste una chiave che coinvolga tutti gli attributi; ciò può accadere invece per la superchiave. *Ogni tabella ha sempre almeno una superchiave*, in quanto non ci devono essere attributi duplicati e l’insieme stesso degli attributi della tabella è considerabile superchiave; ovviamente, *una tabella deve avere anche almeno una chiave*.

Nell’esempio sotto, la chiave è il campo Matricola nel caso di studenti universitari, in quanto è superchiave ed è minimale. Analogamente, Nome/Cognome/Natricola è superchiave ed è anch’essa minimale.

Immagine che contiene testo, tabellonesegnapunti, blu, screenshot

Descrizione generata automaticamente

Si nota quindi come i vincoli devono corrispondere alla realtà di interesse; in questo caso specifico, vista l’implementazione, un’altra possibile chiave e corretta è la coppia Cognome/Corso, chiave “per caso” perché solo in questo contesto funziona.

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Già dagli esempi sopra si nota che non esiste una soluzione univoca, in quanto deve corrispondere alla logica utilizzata; in particolare una soluzione va bene se non viola i vincoli stabiliti di dominio.

Le *chiavi* quindi identificano le tuple di una relazione e sono utilizzate per *referenziare le altre tabelle*. Diciamo anche che una *chiave* è tale *se non ci sono sottoinsiemi considerabili superchiave*. Tuttavia le chiavi possono avere valori *nulli*. In presenza di valori nulli, naturalmente, la chiave non identifica tuple e quindi non realizza collegamenti con altre relazioni.

Una chiave che non ha valori nulli perché non ammessi è la *chiave primaria*. Si parla di *integrità referenziale* per correlare tra di loro i singoli valori con relazioni diverse mantenendo una coerenza tra questi e gli attributi hanno almeno una chiave primaria di seconda relazione, cosiddetta *chiave esterna*. Si intende quindi che questa integrità viene mantenuta tra più tabelle, infatti riguarda la situazione dei vincoli interrelazionali.

È l’esempio della tabella *Infrazioni*, collegata univocamente con la tabella *Vigili* per mezzo della chiave esterna *Matricola*. Analogamente, un’infrazione è collegata ad un auto; da questo si ha il collegamento con una ipotetica tabella *Auto*, avendo come superchiave l’insieme *Stato/Numero*.

Un vincolo di integrità referenziali impone ai valori su X nella relazione R1 di comparire come valori nella chiave primaria di R2; tradotto significa che si collega ad un campo chiave primaria.

***03/03/2022: Modello referenziale/Capitolo 2 e Algebra Relazionale***

Un esempio di vincolo di integrità referenziale non rispettato è la corrispondenza di una chiave esterna su più tuple (nelle slide, si sfrutta il discorso appena descritto Infrazioni/Auto); ciò non può essere situazione valida. I vincoli quindi agiscono come meccanismi di gestione delle singole azioni, prevenendo violazioni.

Immagine che contiene testo

Descrizione generata automaticamente

***Esercizio: Individuare tabelle e vincoli***

Idea iniziale di implementazione è la creazione di una tabella

Legenda🡪 chiave esterna – chiave primaria

Treni:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Numero treno* | *Destinazione* | *Categoria* | *Orario* | *Ferm. intermedie* |
| 1740 | Roma | ES | 11 | Ferrara |
| “” | Firenze | “” | 11 | “” |
| “” | Bologna | “” | 11 | “” |

Problema: è tutto ridondante, si noti la ripetizione di tutti gli orari, ad esempio.

Soluzione: Attuo la *normalizzazione*, evitando di mantenere un’informazione ripetuta; separo tutto e creo tabelle separate per Treni e Fermate, ammettiamo.

Fermate:

|  |  |  |
| --- | --- | --- |
| *Numero treno (anche parte della superchiave)* | *Stazione* | *Orario fermata* |
| 1740 | Firenze | 13:15 |
| 1740 | Bologna | 12:00 |
|  |  |  |

Treni:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Numero treno* | *Destinazione* | *Categoria* | *Orario* | *Partenza* |
| 1740 | Roma | ES | 11 | Venezia |
|  |  |  |  |  |
|  |  |  |  |  |

Altra idea: piuttosto che avre una tupla per ogni fermata intermedia posso supporre di avere il numero del treno, l’identificativo della fermata di andata/ritorno e gli orari delle fermate andata/ritorno.

La soluzione è buona concettualmente ma occuperebbe troppo spazio da un punta di vista pratico.

Altra possibile soluzione (slide)

Immagine che contiene testo

Descrizione generata automaticamente

Il modello basato sui valori deve implementare dei meccanismi per prevenire violazioni e prevedere possibili valori nulli alle chiavi, in particolar modo sulle chiavi esterne.

Ad esempio, posso imporre come vincolo di tupla che un certo campo non sia nullo.

Supponendo di avere un’istanza di una base di dati e avendo una tupla, che succederebbe se provassi a togliere una tupla? Ciò potrebbe provocare una violazione.

Ci sono delle possibilità:

* *rifiutare* l’operazione in tronco;
* *generare un’eliminazione a cascata* (ad es. se cancello un treno, controllo tutte le tabelle relazionate cancellando a cascata tutte le fermate associate);
* *introdurre dei valori nulli nella tabella usata*, se possibile, indicando come soluzione ragionevole a livello logico che non utilizzerò o che quel campo è stato cancellato e non mi servirà.

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene tavolo

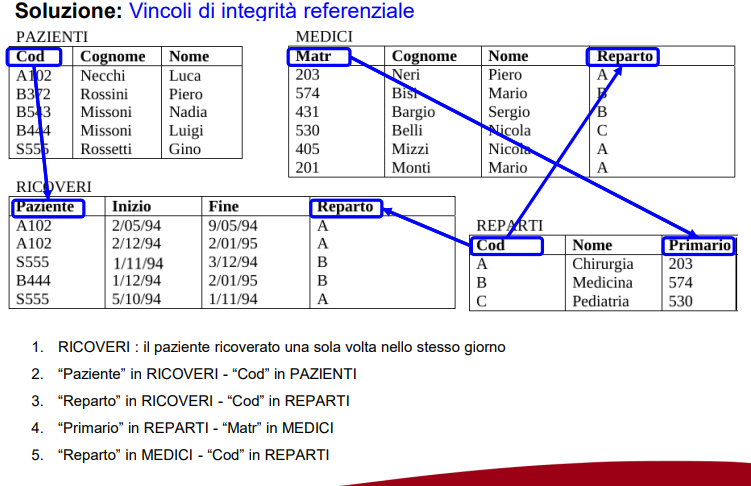
Descrizione generata automaticamente

Immagine che contiene tavolo

Descrizione generata automaticamente

Immagine che contiene tavolo

Descrizione generata automaticamente



Per le interrogazioni, operazioni ed interrogazioni/query parte del DML si focalizza sulla *algebra relazionale*, usando un linguaggio dichiarativo come SQL. Le operazioni procedurali specificano la generazione del risultato e adotta un insieme di operatori che producono relazioni e possono essere composti. Le relazioni sono insiemi, i risultati debbono essere relazioni e ad essi è possibili applicare le operazioni di *unione, intersezione, differenza* solo se sono definite sugli stessi attributi.

Ad esempio *l’unione* considera l’aggregazione delle tuple, operazione che può anche aumentare il numero di tuple risultanti. Esempio di unione sensata ma impossibile: tabelle *Paternità* e *Maternità* in cui, avendo campi diversi, l’unione non è logicamente possibile, perché operando su campi semantici diversi. Se cambiassi il nome del campo *Padre* con *Madre* la cosa sarebbe possibile, in quanto si opera su campi omogenei (hanno lo stesso nome e quindi operano sugli stessi dati logicamente).

Idea migliore: *ridenominazione* del campo, il *Padre* diventa *Genitore*, analogamente anche la *Madre* diventa *Genitore*; in questo modo l’unione diventa possibile.

È possibile applicare l’operatore di ridenominazione su un solo attributo oppure anche su più attributi.

Altra definizione è la *differenza*, restituendo tutte le tuple che non rispettano un vincolo (ad esempio nelle slide, tutti i laureati che non sono specialisti).

***08/03/2022: Algebra relazionale: selezione, proiezioni, join e tipi di join***

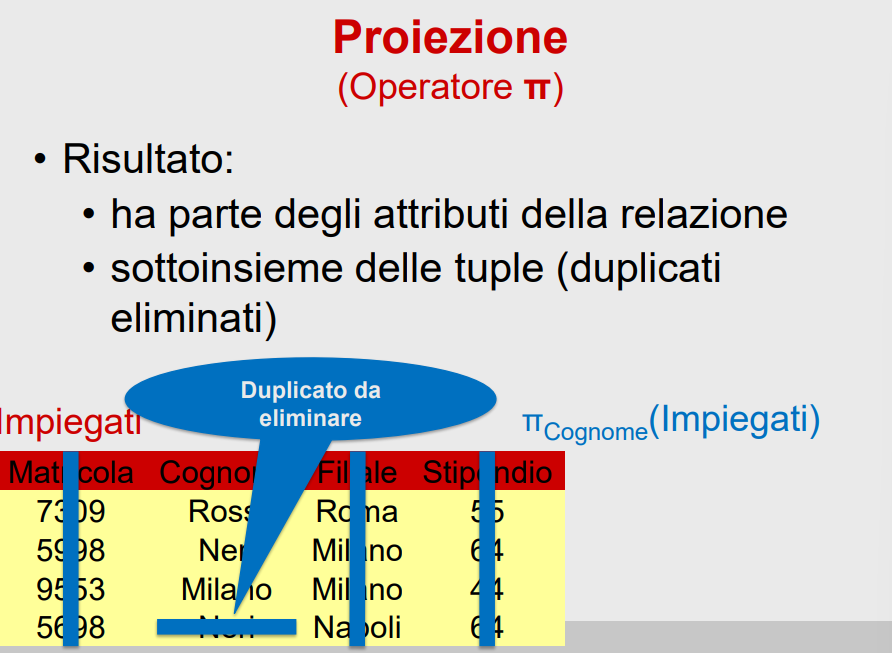
Come detto la volta scorsa, una volta ridenominati correttamente degli attributi è quindi possibile stabilire relazioni logiche sensate. Si può usare la *selezione (operatore σ)* avendo come risultato lo stesso schema ed un sottoinsieme delle tuple (decomposizione orizzontale) che soddisfano una certa condizione o un vincolo, eliminando quindi quelle che non soddisfano la condizione.

Si possono anche combinare le singole condizioni, ad esempio:

Immagine che contiene tavolo

Descrizione generata automaticamente

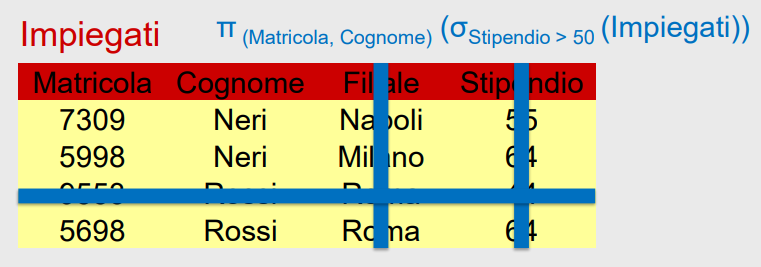
Altro operatore è quello di *proiezione (operatore π)*, quindi mantengo un sottoinsieme questa volta delle colonne, eliminandone alcune (decomposizione verticale). Possiede quindi parte degli attributi della relazione e dunque un sottoinsieme di tuple, quindi non sono ammessi duplicati (nota: i duplicati vengono eliminati senza uno specifico criterio, perché si ha a che fare con un insieme).



Una proiezione contiene quindi al più tante tuple quante l’operando, tuttavia può contenerne di meno. Se X, però, è superchiave, non posso avere due tuple con lo stesso valore (duplicati). Se facessi l’operazione di proiezione sulla superchiave e trovassi dei duplicati, naturalmente, essa non sarebbe superchiave. Anche se prendessi una parte della superchiave, comunque anch’essa superchiave (minimale se chiave invece), sarebbe garantita l’univocità.

Naturalmente possiamo combinare i due operatori. Fondamentale anche l’ordine con cui applico gli operatori, in quanto se invertiti potrebbe non essere più possibile una delle due operazioni (dipende dal contesto; esempio banale, se io volessi prendere Matricola/Cognome degli Impiegati come proiezione potrebbe non essermi possibile la selezione).

Un esempio di applicazione sia di selezione che di proiezione:



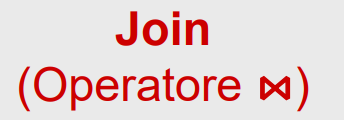
Operatore molto importante è quello di *join*, operatore binario su due relazioni A e B. Come risultato restituisce uno schema della relazione che è un’unione degli attributi degli operandi e come tuple il prodotto cartesiano le tuple A x B mantenendo quelle *con valori uguali su attributi uguali*.

Immagine che contiene tavolo

Descrizione generata automaticamente

A tuple uguali corrispondono tuple con la stessa logica: combinassi ad esempio le tuple con 1, andrei a prendere quella con il valore 1 sugli attributi comuni.

Più formalmente si definisce in questo modo:

Immagine che contiene testo

Descrizione generata automaticamente

Ogni singola tupla contribuisce ad un join nel caso del *join completo*; se parallelamente qualche tupla non contribuisce si ha invece un *join non completo*. Si noti che nel primo caso non ho perdita di informazioni mentre nel secondo caso potrebbe succedere e le tuple vengono “tagliate fuori” dal risultato.

Esempi di join (completo e non completo):

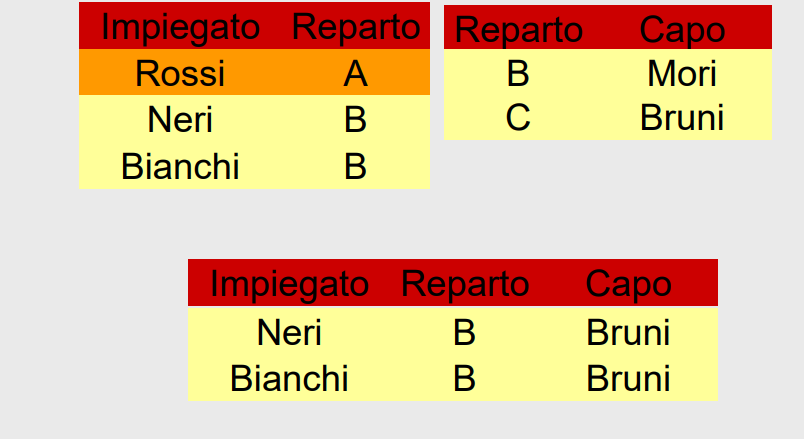


Immagine che contiene tavolo

Descrizione generata automaticamente

Può anche capitare un *join vuoto*, in cui si ha una sorta di prodotto cartesiano, quindi una combinazione di *n x m* tuple delle relazioni in atto. Nel caso sotto ad esempio non ci sono tutte le tuple corrispondenti e si genera la situazione del vuoto. Un join, in altri casi, potrebbe benissimo corrispondere invece ad un prodotto cartesiano delle tuple.



In generale nei casi di join, consideriamo che: (si considerano due relazioni generiche R1 ed R2)

* primo punto, notando che si ha almeno un attributo comune, il numero di tuple varia da 0 al prodotto cartesiano del numero delle tuple.
* secondo punto, se il join viene realizzato su una delle due relazioni, il sottoinsieme è <= a una delle due relazioni
* terzo punto, in cui ogni tupla corrisponde ad una tupla dell’altra. Nel join ce ne saranno esattamente *n*, quindi la cardinalità di R1. Non avendo chiavi esterne, non è obbligatorio che tutte le tuple siano collegate tra di loro; se la tupla è nulla e non è chiave esterna non è nel risultato del join.

Immagine che contiene testo

Descrizione generata automaticamente

Il join *esterno* estende, con valori nulli, le tuple che verrebbero tagliate fuori da un join interno. Esiste in tre versioni, sinistro, destro, completo.

In questo caso,

* il *join sinistro/left* mantiene tutte le tuple del primo operando estendole con valori nulli, se necessario;
* il *join destro/right* mantiene tutte le tuple del secondo operando estendole con valori nulli, se necessario;
* il *join completo/full* mantiene tutte le tuple di entrambi gli operandi estendole con valori nulli, se necessario.

Esempio di join sinistro, destro e completo (in ordine)

Immagine che contiene tavolo

Descrizione generata automaticamenteImmagine che contiene tavolo

Descrizione generata automaticamente

Immagine che contiene tavolo

Descrizione generata automaticamente

Esiste inoltre il *semijoin*, operatore su due relazioni R1 ed R2, restituendo una relazione su *X1* con le tuple di *R1* che contribuiscono al join con *R2*.

Immagine che contiene tavolo

Descrizione generata automaticamenteAd esempio, se io eseguissi il join tra Impiegati e Reparti per poi eseguire la proiezione, avrei ottenuto lo stesso risultato del semijoin. Seguono esempi:

Immagine che contiene tavolo

Descrizione generata automaticamente

Il *natural join* (join naturale), considera come risultato la colonna con lo stesso nome rispetto a quello cercato e stessi tipi di dati presenti tra le tabelle su cui operare il join. Esso è simile ad un prodotto cartesiano, coincidendo proprio con questo quando non si hanno attributi in comune.

L’operazione viene chiamata *theta-join* (join condizionale) viene eseguita quando un prodotto cartesiano è seguito da selezione.

Essa ha questa sintassi:

Nell’esempio il reparto viene rinominato in codice e poi facendo il join con i reparti, ottenendo Impiegato, Codice e Capo. Altro caso, facciamo la join tra Reparti e Selezione e poi eseguendo un theta join rinominando Codice e Reparti; il risultato qui è diverso, dato che ho un attributo in più.

Eseguendo il prodotto cartesiano e poi la selezione, ecco il theta join (theta come lettera perché storicamente indicava il confronto).

*Morale: o faccio un join, oppure posso fare un join e poi una selezione.*

Usando l’operatore di uguaglianza nel theta-join, si applica un *equi-join.*

Immagine che contiene tavolo

Descrizione generata automaticamente

Esempi pratici (con relative soluzioni che si susseguono, incollati tutti perché il caro prof dice che vengono molto sbagliati agli esami. Si consideri inoltre che nell’esempio 3 viene volutamente fino alla parte 3 mostrata una soluzione sbagliata; quella corretta è nella parte 4).

Immagine che contiene tavolo

Descrizione generata automaticamente

Immagine che contiene tavolo

Descrizione generata automaticamente

Immagine che contiene tavolo

Descrizione generata automaticamente

Immagine che contiene tavolo

Descrizione generata automaticamente

Immagine che contiene tavolo

Descrizione generata automaticamente



Immagine che contiene tavolo

Descrizione generata automaticamente

***09/03/2022: Algebra relazionale***