Database

Riassunto corso di base di dati unipr del professore Flavio Bertini.

Nota, esami di Bertini richiedono conoscienze molto dettagliate, dunque affiancare questi appunti(intuitivi) a slide e/o libro.
Le basi di dati fanno parte del sistema informatico.

è opportuno distinguere il dato dall'informazione: il **dato** da solo non ha alcun significato, ma una volta interpretato e correllato opportunamente, diventa **informazione**, che permette di arricchire la nostra conoscienza del mondo.

Un **Dbms** è un software in grado di gestire collezioni di dati che siano grandi,condivise,persistenti, assicurando la loro affidabilità e privatezza. Un **Database** è una collezione di dati gestita da un dbms.

Alla mente del genio, sorge spontanea una domanda: "Concettualmente, come posso descrivere questi dati e quale struttura devono avere per essere elaborati da una macchina?" Abbiamo diversi modi per descrivere i dati, ognuno con livello di astrazione differente. Il livello concettuale è la rappresentazione più vicina alla logica umana ed è indipendente da ogni sistema, per la progettazione concettuale il modello più utilizzato è quello $E - R$.

Ragionando in maniera meno astratta, la maniera in cui rappresentare i dati in maniera comprensibile ad un macchina, in maniera indipendente dalla rappresentazione fisica,è descritto da il **Modello logico dei dati**. Tra i più comuni abbiamo:
1. Modello relazionale.
2. Modello gerarchico(basato su alberi,che sarebbero comunque grafi).
3. Modello reticolare(basato su grafi).
4. Modello a oggetti.
5. Modello XML.
6. Modello semi strutturato e flesibile(e.g NoSql).

I dbms seguono l'architettura a tre livelli ANSI-SPARC, i dati sono descritti secondo tre differenti livelli di astrazione, per ciascun livello esiste un'opportuno schema.
1. Livello esterno: esso si occupa della interlocuzione con gli applicativi, e quindi con gli utenti finali, infatti rappresenta la visione del database da parte dell'utente e quindi descrive quella parte del database che è importante per il singolo utente o per un gruppo di utenti.
2. Livello logico: Precedentemente descritto.
3. Livello fisico come vengono effettivamente memorizzati i dati dalla macchina(e.g. strutture dati utilizzate).

Questa architettura permette di modificare ognuno di questi livelli senza dover modificare gli altri 2.

Per comunicare con un dbms abbiamo 2 tipi di linguaggi:
1. DDL(data definition language) utilizzati per definire schemi logici, esterni, fisici e le autorizzazioni.

esempio:

```
CREATE TABLE hours (
    course CHAR(20),
    teacher CHAR(20),
    room CHAR(4),
    hour CHAR(5)
)
```

1. DML(data manipulation language) utilizzati per le query e aggiornamento di istanze di basi di dati.

esempio:

```
SELECT * from T;
```

Modello relazionale

In ogni base di dati esistono dati che rimangono invariati nel tempo, detti **schemi** e altri che variano detti **istanze**. Nel caso di un modello Relazionale, lo schema è costituito dalla sua intestazione,ovvero nome della relazione R seguito dai suoi attributi, che corrispondono alle colonne. $R(A_1, A_2, ..., A_n)$

L'**istanza** invece è costituita dall'insieme, variante nel tempo delle sue righe,ognuna di esse fa riferimento allo schema e solo attraverso di esso i dati possono essere interpretati.

Il modello relazionale si basa su il concetto di relazione matematica e tabella. In maniera intuitiva, possiamo dire che tra queste 2, esiste un isomorfismo, la correlazione di questi concetti, uno formale e l'altro intuitivo è uno dei motivi che ha permesso il successo di questo modello.

Una relazione è un sottoionsieme del prodotto cartesiano di n insiemi, chiamati domini della relazione. $S \subseteq D_1 \times D_2 \times ... \times D_n$

Esempio di relazione logica: $\text{Matches} \subseteq \text{string} \times \text{string} \times \text{int} \times \text{int}$

Barca	Bayern	3	1
Bayern	Barca	2	0
Barca	Psg	0	2
Psg	Real	0	1

In quanto il prodotto cartesiano è un insieme, inferiamo facilmente che
1. Non è definito un ordinamento tra le n-uple.
2. Le n-uple sono tutte distinte.

Allo stesso tempo però, è definito un ordinamento fra i domini, senza il quale non riusciremmo ad interpretare correttamente il significato della relazione, ad esempio in questo caso il primo dominio corrisponde alle squadre in casa ed il secondo a quelle in trasferta. Noi vogliamo una struttura che non sia posizionale. Ciò che vogliamo noi è poter far riferimento ad un valore in maniera simbolica, dunque indipendente dalla posizione.

Per ovviare a questo problema, andiamo ad associare ad ogni attributo un dominio. esempio:

SquadraDiCasa	**SquadraOspitata**	**RetiCasa**	**RetiOspitata**
Barca	Bayern	3	1
Bayern	Barca	2	0
Barca	Psg	0	2
Psg	Real	0	1

Spesso viene utile associare più relazioni, nel caso del modello releazionale ciò avviene tramite valori, il che rispetto all'utilizzo di puntatori(altro approccio) risulta più astratto e più indipendente dal contesto.

Il modello relazionale permette di modellare anche realtà complicate, nonostante spesso avvenga in modo indiretto e rigido.

t Per indicare l'assenza di un valore si può utilizzare il valore null.(Ci sono diversi contesti che meriterebbero tipi più specifici,esempio unknown,inexistant,uninformative)

Definizioni

- Sia X l'insieme degli attributi e D l'insieme dei domini, specifichiamo la corrispondenza fra attributi e domini per mezzo di una funzione dom : $X \to D$ che $\forall A \in X$ associa un valore del dominio $\text{dom}(A) \in D$.
- Una **tupla** su un insieme di attributi X è una funzione t che $\forall A \in X$ associa un valore del dominio $\text{dom}(A)$.

La nostra definizione di relazione verrà rivisitata per renderla non posizionale.

- Una relazione su X è un insieme di tuple su X.

Notazione: $T[A]$ dove $A \in X$ indicherà il valore di A su t.

esempio: se t è la tupla rappresentata dalla prima riga della tabella, $t[\text{SquadraOspitata}] = \text{Lazio}$.

- Uno **schema di Relazione** è costituito da un simbolo R detto nome della relazione e da un insieme di attributi $X = \{A_1, A_2, ..., A_n\}$, lo schema viene indicato come $R(X)$. Ovviamente ad ogni attributo viene associato un dominio come precedentemente descritto.
- Uno **schema di database** è un insieme di schemi di relazione $R = \left\{R_1(X_1), R_2(X_2), ..., R_{n(X_n)}\right\}$.

Esempio: nell'immagine precedente possiamo dire che

$R = \{\text{Student}\{\text{Number}, \text{Surname}, \text{Name}, \text{BirthD}\},$

$\text{Exam}\{\text{Student}, \text{Grade}, \text{Lecture}\},$

$\text{Lecture}\{\text{Code}, \text{Name}, \text{Lecturer}\}\}$

- Un'**istanza di relazione** o **relazione su uno schema** $R(X)$ è un insieme r di tuple su X.
- Un'**istanza di base di dati** o **base di dati** su uno schema $R = \left\{R_1(X_1), R_2(X_2), ..., R_{n(X_n)}\right\}$ è un insieme di relazioni $r = \{r_1, r_2, ..., r_n\}$ dove ogni r_i per $1 \leq i \leq n$, è una relazione sullo schema $R_i(X_i)$.

Vincoli di integrità

I vincoli di integrità sono delle proprietà (che possono essere interpreate come un predicato(v-f)) che devono essere verificate per rappresentare le informazioni in maniera accurata,corretta e di qualità. Nel caso il dbms non supporti tutti i vincoli il programmatore può programmare il costraint fuori dal dbms.

Abbiamo diversi tipi di vincoli:

- Vincolo intrarelazionale, ovvero il suo soddisfacimento è definito rispetto ad una singola relazione. esempio vincolo su una tupla(una tupla deve soddisfare una proprietà) o su un valore(detto anche vincolo di dominio).
- Vincolo interrelazionale, ovvero coinvonge più relazioni.

Abbiamo anche i vincoli di integrità referenziale, prima di parlare di essi però è necessario definire il concetto di chiave e superchiave. Concettualmente una **chiave** sarebbe un insieme di attributi che identificano univocamente una tupla in una relazione.

- Un insieme K di attributi è **superchiave** di r se r non contiene due tuple distinte $t_1, t_2 : r[t_1] = r[t_2]$
- K è **chiave** se è una **superchiave minimale** di r,ovvero non contiene un'altra superchiave.

Notare che ogni relazione contiene una superchiave che è l'insieme di tutti gli attributi e dunque almeno 1 chiave. Le chiavi permettono di correllare tuple su relazioni differenti,ovviamente mi aspetto che queste correlazioni siano coerenti.

Più formalmente, un **vincolo di integrità referenziale**, chiamato anche **foreign key** tra gli attributi X di una relazione R_1 e una relazione R_2 impone che i valori su X in R_1 siano chiave primaria di R_2.

Un attributo A in R_1 è una chiave esterna che si riferisce ad una relazione R_2 se:
- L'attributo A ha lo stesso dominio dell'attributo che è chiave primaria di R_2.
- Il valore dell'attributo FK in una tupla t_1 in R_1 è null oppure è il valore di una tupla t_2 in R_2 per cui $t_1[\text{FK}] = t_2[\text{PK}]$.

Nel caso un vincolo referenziale venisse violato,ad esempio una tupla contenente la chiave primaria venisse rimossa, abbiamo diversi approcci:
- Rimozione a cascata(viene rimossa anche tupla contenente FK).
- Introduzione di valore null nella FK oppure valore di default.

Nota, nel caso di costraint su più attributi l'ordine degli attributi è importante.

Algebra relazionale
Vedere slide, inutile riassumere.

Views
Le viste sono un modo di rappresentare dati in diverse modi, molto utile nel livello esterno(ansi/sparc).

Abbiamo diversi tipi di views:
1. views materializzate, ovvero la tabella viene salvata fisicamente nel database, ciò con maggiore velocità nell'accesso ma a discapito della memoria utilizzata, inoltre non tutti db supportano.
2. releazione virtuale(o views), trivialmente la view query viene trasformata in una query che verrà eseguita ogni volta dal dbms.

Sono utili per astrazione, e per decidere a quali dati un utente è interessato.

Calcolo relazionale
Famiglia di linguaggi basata sulla logica del prim'ordine.
- Domain Relational Calculus Pro: dichiarativo, contro: espressioni senza senso,espressioni molto lunghe e verbose, inoltre è eq ad algebra relazionale.
- Tuple Relation Calculus with Range Declarations Permette di sorpassare limitazioni di DRC, ovvero fa si che una variabile sia associata ad una tupla, mentre prima 1 var per attributo. Inoltre tutti i dati provengono dallo stesso db.

Non può esprimere unione, però si possono fare query ricorsive.

SQL
Vedere slide.

Architettura dei DBMS
Un Database Management system è un software usato per creare e gestire database.

Ci sono 3 componenti principali
1. Processore delle query: Parsa,Preprocessa(trasforma in algebra) e ottimizza, L'engine di esecuzione esegue le query interagendo con buffer,scheduler,log manager...
2. Manager delle risorse. Solitamente i dati del database risiedono in memoria secondaria per garantirne la persistenza, questo gestore di occupa di di storare e accedere velocemente a queste risorse.

Include la struttura dati index.. il buffer manager che divide la memoria in blocchi in maniera che possano essere trasferiti in maniera efficiente e lo storage manager che ricorda le posizioni dei file sul disco e risponde alle richiest del buffer manager.
3. Manager delle transazioni, si occupa di gestire le transazioni(più info su esse dopo).

Contiene componenti come Il logger e il "recoverer",che per poter tornare allo stato precedente, nel caso fallisse la transazione salva su un disco tutti i cambiamenti che avvengono nel db,per poi in casi di fallimento ripristinare. Dunque prima di scrivere effettivamente sul disco che contiene i dati, questi verranno prima loggati.

Inoltre si occupa di controllare che non avvengano problemi di concorrenza, ciò tramite lock delle risorse. Ciò nonostante possono accadere situazioni di deadlock, ovvero in cui l'esecuzione non procedede a causa del fatto che due o più azioni si stanno bloccando a vicenda.

Sull'implementazione del JOIN
Esattamente ciò che ha spiegato bertini.

Si tratta dell'operazione più dispendiosa in termini di tempo, ci sono 4 tecniche differenti per eseguire una JOIN. Il fatto che una venga scelta rispetto all'altra dipende da situa.
1. Nested-loop join

Ovvero 2 cicli for annidati in cui tutti i valori della prima tabella(outer) vengono confrontati con il primo della seconda(inner) and so on. In questo caso, se entrambe le tabelle sono di piccole dimensioni possiamo salvarle entrambe nella memoria principale per evitare di andare ogni volta a leggere memoria secondaria. Se cosi non fosse si opta per tenere in ram la tabella più pesante, in maniera da minimizzare il numero delle letture del disco secondario.

2. Single loop join, in questo caso, abbiamo un index o (key di hashing) per uno degli attributi dei join. Non ci basta altro che iterare sull'altra tabella e usare la struttura di accesso per accedere agli elementi che matchano della prima.
3. Sort-merge Join, in breve si basa sul fatto che Le relazioni R e S siano ordinate in base ai valori di join rispetto all'attributo A e B rispettivamente. Ciò ci permette di risparmiare iterazioni.

Di solito viene scelto quando non si tratta di equi join(altre op, tipo <,>) oppure sorting serve comunque.
4. Hash-based Join

Si usa le stessa funzione di hashing sugli attributi da joinare, Verranno creati N bucket per gli attributi della prima relaz, ora non ci resta che confrontare valori di hash della seconda(dunque H(s) verrà confrontato con i valori nel bucket(ci possono essere collisioni)). Dunque nel disco salverò praticamente solo i bucket senza bisogno ogni volta di riscannare memoria.

Transazioni
Vedere slide

Design del database
Vedere Slide, la parte della generalizzazione è spiegata male, dunque per capire. Ci sono diversi tipi di relazione.

1. Totali, se ogni occorrenza dell'entità genitore è almeno una delle entità figlie, ad esempio data la generalizzazione Persona e le specializzazioni maschio,femmina, una persona può essere o maschio o femmina, dunque è totale(in altre parole unione delle specializzazione = generalizzazione)

2. Parziali otherwise.

1. Esclusive(disjoint) ogni occorrenza del genitore è al più occorrenza di uno dei figli, dunque intersezione tra specializzazione = empty set
2. Sovrapposte(overlap) altrimenti

Altro, vedere slide.