Appunti di Basi di Dati

4 ottobre 2022

Rosso Carlo

Contents

1 introduzione

Def. 1.1 (base di dati) *insieme organizzato, persistente e condiviso di dati utilizzati per lo svolgimento delle attività automatizzate di un'organizzazione.*

Def. 1.2 (dato) *ciò che è immediatamente presente alla conoscenza, prima di ogni elaborazione.*

Def. 1.3 (informazione) *notizia, dato o elemento che consente di avere conoscenze più o meno esatte di fatti, situazioni, modi di essere.*

Elaborazione dei dati permette di estrarre delle informazioni dai dati.

Def. 1.4 (DataBaes Management System (DBMS)) *Le basi di dati sono gestite da un DBMS. Una base di dati è grande, persistenete e condivisa. Ovvero, grandi: il limite deve essere solo quello fisico dei dispositivi; persistenti: indipendente dalle singole esecuzioni dei programmi; condivise: applicazioni e utenti diversi accedono ad una porzione (sovrapponibile) delle basi di dati).*
Un DBMS garantisce alla base di dati: privacy, affidabilità, efficienza, efficacia.

La proprietà della condivisione che caratterizza i database ha alcuni problemi che devono essere risolti:

- ridondanza: dati duplicati;
- incoerenza: le versioni possono non coincidere;
- privacy: i fruitore delle basi di dati devono accedere ai dati per cui sono autorizzati.

Le soluzioni sono le seguenti:

- indipendenza delle operazioni: sono permesse attività diverse solo su dati diversi; dunque, sono previsti meccanismi di autorizzazione, per garantire la sequenzialità delle operazioni;
- accessi di più utenti a dati condivisi: meccanismi di controllo della concorrenza e talvolta è impiegato l'uso di lock e semafori.

I DBMS garantiscono affidabilità sfruttando le transazioni.

Def. 1.5 (transazioni) *insieme di operazioni, da considerare indivisibile, corrette anche in presenza di concorrenza e con effeti definitivi.*

Def. 1.6 (schema) *lo schema è quasi un'invariante nel tempo. Si tratta della descrizione delle tabelle contenute nel database.*

Def. 1.7 (istanza) *è la rappresentazione dei dati in un dato momento, nei modi indicati attraverso lo schema. L'istanza è soggetta a variazioni nel tempo.*

Un contributo all'efficacia dei DBMS è la disponibilità di vari linguaggi e interfacce. Per esempio il linguaggio testuale interattivo SQL, che si impegna a rappresentare l'algebra relazionale. In realtà i comandi SQL possono essere immersi in un linguaggio ospite, come ad esempio C++. Oppure si può gestire un DBMS attraverso un'interfaccia amichevole.
SQL svolge una doppia funzione: è un data manipulation language (DML), per cui, permette di interrogare e di aggiornare istanze di basi di dati; è un data definition language (DDL), per cui, permette di definire e modificare gli schemi.

2 Algebra relazionale

Def. 2.1 (relazione) *una relazione è un insieme di tuple (attributi), ciascuna delle quali è un insieme di valori, uno per ogni attributo. Gli attributi non hanno ordine, gli elementi dell'insieme non hanno ordine. Per cui (in un database), l'ordinamento tra le righe è irrilevante; l'ordinamento tra le colonne è irrilevante; le righe sono diverse tra loro; le intestazioni delle colonne sono diverse tra loro; i valori di ogni colonna sono definiti su domini omogenei.*

Pro	Contro
condividere i dati in un'organizzazione	ne vale la pena solo per grandi organizzazioni
gestione centralizzata, easy to scale	non utilizzare senza tanti utenti
servizi integrati	non utilizzare senza accessi concorrenti e stabili
riduzione di ridondanze e inconsistenze	
indipendenza dei dati per la manutenzione e lo sviluppo	

Table 1: Pro e contro dei DBMS

Talvolta, l'attributo di un'istanza potrebbe non avere alcun valore. In questo caso si ricorre all'utilizzo di valori speciali, che indicheranno l'assenza di valore. Questo permette di imporre restrizioni sulla presenza di valori nulli. Il valore nullo può avere tre origini, nasconde uno di tre significati: il valore è sconosciuto; il valore non esiste; il valore non ha informazione intrinseca.

Def. 2.2 (sintatticamente corrette) *tutte le tuple sono diverse tra loro.*

Def. 2.3 (semanticamente corrette) *tutte le tuple rispettano le restrizioni imposte. I valori assunti hanno senso per l'applicazione di interesse.*

2.1 vincoli intrarelazione

Def. 2.4 (non nullo) *Data r di $R = A_1, \ldots, A_n$, e $A_i \in R$, allora si dice che A_i è non nullo in r se e solo se per ogni $t \in r, t[A_i]$ è definita.*

Def. 2.5 (unicità) *Sia $\bar{R} = A_{i_1}, \ldots, A_{i_k} \subset R$ un sottoinsieme non vuoto ($k \geq 1$). Allora si dice che $\bar{R}$ è unico in r se e solo se per ogni $t_1, t_2 \in r, t_1[\bar{R}], t_2[\bar{R}]$ definiti abbiamo $t_1[\bar{R}] = t_2[\bar{R}] \Rightarrow t_1 = t_2$.*

Def. 2.6 (superchiave) *$\bar{R}$ si un'instanza r di R è un sottoinsieme $\bar{R} \subset R$ che soddisfa il vincolo di unicità su r.*

Def. 2.7 (chiave) *$\bar{R}$ di un'istanza di R è una superchiave $\bar{R} \subset R$ per r tale che $\forall \tilde{R} \subset \bar{R}, \tilde{R}$ non è una superchiave per R (minimalità).*

Def. 2.8 (chive primaria (primary key, pk)) *$\bar{R}$ di r su R è una chiave di r per cui vale non nulla su ogni attributo di $\bar{R}$ in r.*

Def. 2.9 (schema di relazione) *Lo schema di una relazione R è costituito da una relazione R e un insieme di vincoli $V_i, \ldots, V_c$ intrarelazionali su R.*

Def. 2.10 (istanza (valida) di uno schema) *r di R è un'istanza (valida) di R se e solo se r soddisfa $V_1, \ldots, V_c$.*

2.2 vincoli interrelazionali

Def. 2.11 (chiave esterna (foreign key)) *Date 2 relazioni $R_s = A_1, \ldots, A_{m_a}$, $R_t = B_1, \ldots, B_{m_b}$, una chiave esterna da R_s verso R_t è un insieme finito di coppie $FK = \{(A_{i_1}, B_{j_1}), \ldots, (A_{i_k}, B_{j_k})\}$, dove B_{j_i}, tale che $x \in [1, \ldots, k]$, sono tutti distinti e $Dom(A_{i_x}) = Dom(B_{j_x})$ per ogni $x \in [1, \ldots, k]$.*

Def. 2.12 (vincolo di chiave esterna) *$R_s = A_1, \ldots, A_{m_a}, R_t = B_1, \ldots, B_{m_b}, FK = \{(A_{i_1}, B_{j_1}), \ldots, (A_{i_k}, B_{j_k})\}$, da R_s verso R_t, r di R_s, $\bar{r}$ di R_t, allora si dice che r soddisfa il vincolo di chiave esterna FK se e solo se:*

1. *$B_{j_1}, \ldots, B_{j_k}$ è pk in $\bar{r}$;*
2. *sia $\forall t \in r$, tale che $t[A_{i_1}], \ldots, t[A_{i_k}]$ definito, allora $\exists \bar{t} \in \bar{r}$ tale che $t[A_{i_1}], \ldots, t[A_{i_k}] = \bar{t}[B_{j_1}], \ldots, \bar{t}[B_{j_k}]$.*

2.3 operatori su relazioni

L'insieme degli operatori su relazioni producono relazioni e possono essere composti. Operatori insiemistici, le relazioni sono insiemi, il risultato sono altre relazioni è possibile applicare unione, intersezione e differenza solo a relazioni definite sugli stessi attributi.

ridenominazione [ρ] modifica lo schema lasciamo inalterata l'istanza dell'operando. La ridenominazione è un operatore unario.

selezione [σ] il codominio è un sottoinsieme del dominio. La selezione è un operatore unario. Sono tolte delle tuple.

proiezione [π] il codominmio è un sottoinsieme del dominio. La proiezione è un operatore unario. Sono tolti degli attributi. Notiamo che se X è una superchiave di R, allora $\pi_X(R)$ ha cardinalità uguale a quella di R.

join [$\bowtie$] operatore binario su due relazioni A e B. Il risultato è l'unione degli attributi degli operandi; in particolare, il risultato sono le tuple costruite com e$A \times B$ mantenendo quelle con valori uguali su attributi uguali.

Date due relazioni $R_1(X_1),\ R_2(X_2),\ R_1 \bowtie R_2$ è una relazione su X_1X_2, ovvero $X_1 \cup X_2$, del tipo

$$R_1 \bowtie R_2 = \{t \in X_1X_2 \mid t[X_1] \in R_1,\ t[X_2] \in R_2\}. \tag{1}$$

Θ-join si tratta del prodotto cartesiano tra le relazioni operando. Diventa utile se combinato alla selezione, infatti si impone una condizione che deve essere rispettata. In effetti Il Θ-join è un join a cui è applicata la selezione:

$$\sigma_{condizione}(R_1 \bowtie R_2) = R_1 \bowtie_{condizione} R_2. \tag{2}$$

In algebra, due espressioni sono equivalenti se producono lo stesso risultato qualunque sia l'istanza attuale delle relazioni della base di dati.

I DBMS usano regole di equivalenza per ottimizzare le interrogazioni. L'obiettivo delle ottimizzazioni è quello di minimizzare il numero di tuple e di attributi nelle relazioni intermedie.

In SQL esistono forme apposite per riferirsi ai valori nulli: `IS NULL` e `IS NOT NULL`.

2.4 viste

Le viste sono rappresentazioni diverse per gli stessi dati; ovvero, sono relazioni il cui contenuto è funzione del contenuto di altre relazioni.

Alle viste si contrappongono le relazioni di base, che hanno contenuto autonomo.

viste materializzate si tratta di relazioni derivate (in quanto si tratta di viste) memorizzate nella base di dati. Il pro delle viste materializzate è che non c'è bisogno di ricalcolare la vista ogni volta che viene richiesta. I contro sono che: le informazioni sono ridondanti; aggiornare il database diventa un processo più lento; raramente sono supportate dai DBMS.

viste virtuali (o viste) sono supportate da tutti i DBMS. Una interrogazione su una vista viene eseguita "ricalcolando" la vista, a meno di ottimizzazioni interne da parte del DBMS.

Le viste permettono di avere degli schemi esterni. Per cui, si mostra agli utenti solo ciò che sono interessati a conoscere. Inoltre, semplifica la scrittura di interrogazioni complesse. Diventa possibile riutilizzare programmi già esistenti su schemi ristrutturati. L'utilizzo di viste non influisce sull'efficienza delle interrogazioni, perché il DBMS ottimizza le interrogazioni.

3 Forme Normali

Def. 3.1 (forma normale) *Una forma normale è una proprietà di una base di dati relazionale che ne garantische la "qualità", cioè l'assenza di determinati difetti.*

NB quando una relazione non è normalizzata, presenta ridondanze; oppure si presta a comportamenti poco desiderabili durante gli aggiornamenti.

La normalizzazione è il processo che permette di trasformare uno schema non normalizzato in uno schema che soddisfa una forma normale. La normalizzazione è usata come tecnica di controllo dei risultati della progettazione di una base di dati.

Def. 3.2 (dipendenza funzionale) *Condizioni:*

- *sia R una relazione su X;*
- *dati due insiemi di attributi $A, B \subset X$;*
- *$\forall t_1, t_2 \in R$, se $\pi_A(t_1) = \pi_A(t_2)$, allora $\pi_B(t_1) = \pi_B(t_2)$.*

Allora si dice che esiste in R una dipendenza funzionale $A \to B$.

Def. 3.3 (forma normale di Boyce-Codd (BCNF)) *Condizioni:*

- *R è una relazione su X;*
- *$K_1, \ldots, K_n$ sono chiavi di R;*
- *F è l'insieme delle dipendenze funzionali su R;*
- *$\forall f \in F$ tale che $f = X \to Y$, X è superchiave.*

Allora si dice che R è in forma normale di Boyce-Codd.

Def. 3.4 (decomposizione senza perdita) *Condizioni:*

- *R è una relazione su X;*
- *$R_1, \ldots, R_n$ sono relazioni su X;*
- *$\cup_{R_i} = R$;*
- *$\pi_{R_1}(R) \bowtie \ldots \bowtie \pi_{R_n}(R) = R$.*

Allora si diche che la relazione R si decompone senza perdita su $R_1, \ldots, R_n$.

NB una decomposizione senza perdita è garantita se gli attributi in $R_i \cap R_j$, con $i \neq j$, sono una chiave per R_i o (non esclusivo) R_j.
Una decomposizione dovrebbe sempre preservare le dipendenze funzionali, ovvero i vincoli di integrità originari della relazione.

NB La forma normale di Boyce-Codd non è sempre raggiungibile, ecco perchè in genere si usa la terza forma normale.

Def. 3.5 (terza forma normale (3NF)) *Una relazione R è in terza forma normale se per ogni dipendenza funzionale non banale $X \to Y$, almeno una delle seguenti condizioni è vera:*

- *X è superchiave;*
- *ogni attributo in Y è contenuto in una chiave di R.*

La terza forma normale è una forma normale più debole della BCNF, ma ha il pro di essere sempre raggiungibile. Una possibile strategia è quella di decomporre in terza forma normale e poi si verifica se lo schema ottenuto è anche in BCNF.

Def. 3.6 (seconda forma normale (2NF)) *Una relazione è in seconda forma normale se la relazione non ha attributi che dipendono funzionalmente da una parte della chiave.*

Def. 3.7 (chiusura) *Condizioni:*

- *sia R una relazione su X;*
- *sia F un insieme di dipendenze funzionali definite su X;*
- *sia A un insieme di attributi di X.*
- *sia $A_F^+ = \{B \in X | \exists f \in F$ tale che $f = A \to B\}$.*

Allora si diche che A_F^+ è la chiusura di A rispetto a F, ovvero è l'insieme di tutti gli attributi che dipendono funzionalmente da X.

NB se $\{X_1, X_2\} \to Y$ allora $\{X_1, X_2, \dots\} \to Y$.

Sia F un insieme di dipendenze funzionali su R. Ecco alcune proprietà desiderabili:

- F sia **non ridondante**: non esiste dipendenza $f \in F$ tale che $F - \{f\} \Rightarrow f$;
- F sia **ridotto**: non esiste un insieme F' di dipendenze funzinali su R equivalente a F, ottenuto eliminando attributi dai primi membri delle dipendenze funzionali di F;

4 transazioni

Def. 4.1 (transazione) *Una transazione è un'unità di elaborazione che gode delle proprietà ACID: atomicità, consistenza, isolamento e durabilità. Una transazione è una parte del programma caratterizzata da:*

1. *inizio di transazione (begin-transaction);*
2. *corpo della transazione: una serie di insert, delete e update in SQL;*
3. *fine transazione (end-transaction): una transazione si può concludere:*
   - *commit: terminazione corretta, i cambiamenti sono definitivi;*
   - *rollback: la transazione è abortita, si torna allo stato antecedente l'inizio della transazione.*

Def. 4.2 (proprietà ACID)

***Atomicità**: una transazione deve essere eseguita completamente o non deve essere eseguita affatto;*

***Consistenza**: una transazione deve portare il database da uno stato consistente ad un altro stato consistente;*

***Isolamento**: una transazione non espone i suoi stati intermedi;*

***Durabilità**: una transazione che ha fatto commit non può essere persa.*

Una sistema transazione (OLTP) è in grado di definire ed eseguire transazioni per un certo numero di applicazioni concorrenti. C'è un log che permette di effettuare *undo/redo*. Si tratta di un archivio permanente delle operazioni svolte sul database. Le regole fondamentali del log sono:

- write-ahead-log: si scrive il log prima del database;
- commit-precedence: il log di un commit precede il commit stesso;

4.1 livello di isolamento

Il livello di isolamento può essere scelto per ogni transazione:

- read uncommitted;
- read committed;
- repeatable read;
- serializable;

Per capire la differenza tra i livelli di isolamento spieghiamo gli errori che si possono verificare:

- perdita di aggiornamento (loss update(?)): due transazioni scrivono lo stesso dato, ma solo l'ultimo viene mantenuto;
- lettura sporca (dirty write): una transazione legge un dato che è stato scritto da un'altra transazione che poi abortisce;
- letture inconsistenti (non-repeatable read): una transazione legge lo stesso dato due volte, ma il valore è cambiato;

- aggiornamento fantasma: un dato appare "improvvisamente" aggiornato.
- inserimento fantasma (phantom): una transazione legge un insieme di dati, poi una seconda transazione inserisce un nuovo dato che soddisfa la condizione della prima transazione, che però non lo vede.

Di seguito la tabella che riassume i livelli di isolamento:

Anomalie	read uncommitted	read committed	repeatable read	serializable
perdita di aggiornamento	✓	✓	✓	✓
lettura sporca	-	✓	✓	✓
letture inconsistenti	-	-	✓	✓
aggiornamento fantasma	-	-	✓	✓
inserimento fantasma	-	-	-	✓

5 schedule

Def. 5.1 (schedule) *Uno schedule è un'insieme di operazioni di input e output di transazioni concorrenti.*

Def. 5.2 (scheduler) *un sistema che accetta o rifiuta o riordina le operazioni richieste dalle transazioni.*

Def. 5.3 (serial schedule) *Uno schedule è seriale se le operazioni di ogni transazione sono eseguite in modo consecutivo, una per volta.*

Def. 5.4 (legge-da) *$r_i(x)$ legge da $w_j(x)$ in uno schedule S, se $w_j(x)$ precede $r_i(x)$ e non esiste $w_k(x)$ tale che $w_j(x)$ si trova tra $w_j(x)$ e $r_i(x)$ in S.*

Def. 5.5 (scrittura-finale) *$w_j(x)$ è una scrittura finale di x in uno schedule S se non esiste $w_k(x)$ tale che $w_j(x)$ precede $w_k(x)$ in S.*

Def. 5.6 (view-equivalente) *Se due schedule S_1, S_2 hanno la stessa relazione legge-da e le stesse scritture finali, allora si dice che sono view-equivalenti ($S_1 \equiv S_2$).*

Def. 5.7 (view-serializzabile (VSR)) *Uno schedule è serializzabile se è view-equivalente ad un qualche schedule seriale.*

NB la verifica della view-equivalenza di due dati schedule è lineare sulla lunghezza dello schedule. Decidere sulla view'serializzabilità di uno schedule S è un problema in NP, perché occorre provare tutti i possibili schedule seriali, ottenuti per permutazioni dell'ordine delle transazioni. Vuol dire che nella pratica non è utilizzabile.

Def. 5.8 (conflitto) *Siano a_i, a_j tali che $i \neq j$ due opearzioni che opearano sullo stesso oggetto. Se almeno una delle due è una scrittura, allora si dice che sono in conflitto.*

Def. 5.9 (schedule conflict-equivalenti) *Siano S_i, S_j due schedule, tali che includono le stesse operazioni e ogni coppia di opearzioni in conflitto compare nello stesso ordine in entrambi, allora si dice che sono schedule conflict-equivalenti.*

Def. 5.10 (schedule conflict-serializzabili (CSR)) *Uno schedule è conflict-serializzabile se è conflict-equivalente ad un schedule seriale.*

NB $CSR \subset VSR$.

Def. 5.11 (two phase locking (2PL)) *Tutte le letture per una risorsa x sono precedute da $r_{lock}(x)$ e sono seguite da unlock(x). Tutte le scritture per una risorsa x sono precedute da $w_{lock}(x)$ e sono seguite da unlock(x).*
Inoltre, una transazione dopo aver rilasciato un lock, non può acquisirne altri.

NB $2PL \subset CSR$.
Riassumento da capo abbiamo che: $schedule\_seriale \subset 2PL \subset CSR \subset VSR \subset schedule$.