Basi di dati

Luca Barra

Anno accademico 2022/2023

Indice

Capitolo 1

Teoria 1

1.1 Introduzione

Le basi di dati sono un insieme di dati utilizzati per il supporto allo svolgimento di attività.

Per esempio, in ambito universitario, bisogna tenere memorizzate grandi quantità di dati relativi a ogni studente. Per fare ciò non si possono usare array, alberi, liste, etc. perchè la memoria principale non può gestire velocemente tanti dati.

All'interno delle basi di dati i dati sono integri, flessibili e con una ridondanza controllata, ma hanno costantemente bisogno di manutenzione.

1.1.1 Sistemi informativi

I sistemi informativi gestiscono le informazioni di interesse. Ogni organizzazione ha un sistema informativo, esplicito o meno ed è indipendente dall'automazione (es. I banchieri fiorentini nel '500). Esso si occupa di:

- raccolta e acquisizione dei dati;
- archiviazione e conservazione;
- elaborazione, trasformazione e produzione;
- distribuzione, comunicazione e scambio.

Un dato è qualcosa di immediatamente percepibile, mentre un'informazione è costituita da dati uniti tramite delle interpretazioni.

1.2 DBMS

Le basi di dati devono essere gestite da un DBMS[1] che è esterno alle applicazioni. Alcuni esempi sono MySQL[2], MariaDB[3] e PostgreSQL. I DBMS sono persistenti poichè la loro vita è indipendente dalle applicazioni che li utilizzano, sono condivisibili in modo da ridurre eventuali ridondanze (informazioni ripetute) e incoerenze (allineamento scorretto dei dati). Tuttavia la condivisibilità delle basi di dati produce concorrenza (accesso contemporaneo a stessi dati) che deve essere regolamentata al fine di salvaguardare l'integrità. I DBMS garantiscono anche la privatezza mediante vari tipi di autorizzazione.

Inoltre sono affidabili, ovvero resistenti a malfunzionamenti, tramite la gestione delle transazioni. Una transazione[4] è un'operazione atomica (indivisibile) i cui risultati sono permanenti. Se si verifica un errore *durante* lo svolgimento di una transazione i risultati vengono anullati.

[1] *Database management system*
[2] Inizialmente libero, ma poi acquisito da Oracle
[3] Versione opensource di MySQL
[4] vedi 11.1

1.3 Modello di dati

Un modello di dati serve a organizzare e descrivere i dati. In ogni base di dati esistono:

- schema: descrive la struttura;
- istanza: i valori effettivi in un determinato istante.

Un modello concettuale serve per progettare una base di dati a un livello astratto. Di solito si usa il modello *entity-relationship (ER)*[5].

I modelli logici sono utilizzati dai programmi. In questo corso useremo il modello *relazionale*. Per rappresentare il modello logico si usano:

- schema logico: la descrizione base di dati nel modello logico;
- schema esterno: descrizione di parte della base di dati in un modello logico;
- schema interno (o fisico): rappresenta lo schema logico tramite strutture di memorizzazione (es. record di puntatori).

Il livello logico è indipendente da quello fisico. In questo corso si vedrà solo il livello logico.

L'accesso ai dati avviene solo tramite il livello esterno (che può coincidere con quello logico), generando due forme di indipendenza:

- logica: Il livello esterno è indipendente da quello logico;
- fisica: Il livello logico e quello esterno sono indipendenti da quello fisico.

1.4 Linguaggi per basi di dati

Si hanno diversi linguaggi:

- interfacce grafiche: senza un linguaggio testuale;
- SQL: linguaggio testuale interattivo;
- comandi SQL: si possono usare in vari linguaggi;
- SQL in linguaggi ad hoc: linguaggi propri di un DBMS.

Oltre a questo i linguaggi si dividono in due macrocategorie:

- DML[6]: servono per interrogare e aggiornare le istanze del DBMS;
- DDL[7]: definiscono schemi ed eseguono operazioni generali.

1.5 Persone che interagiscono con un DBMS

Breve elenco di alcune categorie che interagiscono con i DBMS:

- Progettisti e realizzatori di DBMS;
- DBA[8];
- Progettisti della base di dati;
- Progettisti e programmatori di applicazioni;
- Utenti finali: categoria di utenti che eseguono transazioni;
- Utenti casuali: categoria di utenti che eseguono operazioni diverse da transazioni.

[5] *Modello entità-associazione*
[6] *Data manipulation language*
[7] *Data definition language*
[8] Amministratori della base di dati

Capitolo 2

Teoria 2

2.1 Modelli logici

Ci sono principalmente tre tipi di modelli logici tradizionali: gerarchici, reticolari e relazionali. I modelli gerarchici e reticolari utilizzano puntatori tra record. Il modello relazionale (l'unico che vedremo in questo corso) è basato su valori.

2.1.1 Il modello relazionale

Il modello relazionale fu proposto da E. F. Codd nel 1970, ma divento disponibile nei DBMS commerciali solamente nel 1981. Esso si basa sul concetto di relazione. Ci sono tre possibili accezioni di relazione: quella matematica, quella del modello relazionale e quella del modello ER (spesso tradotta come *associazione*). Nelle relazioni matematiche la struttura è posizionale[1], mentre la relazione del modello relazionale usa una struttura non posizionale.

A ogni dominio viene associato un nome univoco chiamato attributo. Inoltre le sequenze di n elementi sono chiamate tuple. In una tabella che rappresenta una relazione l'ordine delle righe e delle colonne è irrilevante. Una tabella rappresenta una relazione se:

- le righe sono diverse tra di loro;
- le intestazioni delle colonne sono diverse tra di loro;
- ogni colonna ha valori omogenei (sempre dello stesso tipo).

I riferimenti tra i dati in relazioni diverse sono rappresentati con i valori dei domini delle tuple.

2.2 Strutture basate su valori

Una struttura basata su valori è indipendente dalle strutture fisiche e rappresenta solo ciò che è rilevante dal punto di vista dell'applicazione. I dati sono facilmente portabili da un sitema a un altro.

Definizioni importanti:

- Schema di relazione: un nome R con un insieme di attributi $A_1, ..., A_n \rightarrow R(A_1, ..., A_n)$. Es. Studenti(matricola, nome, cognome);
- Schema di base di dati: insieme degli schemi di relazione

  R = $\{R_1(X_1), ..., R_k(X_k)\}$.

  Es.{Studenti(matricola, nome, cognome), Esami(studente, voto, corso), Corsi(codice, titolo, docente)};
- Tupla: su un insieme di attributi $(A_1, ..., A_n)$ associa a ciascun attributo A_i un valore del dominio di A_i. t[A_i] è il valore della tupla t sull'attributo A_i. Es. Esami(studente, voto, corso), t=('3456',30,'04'), t[voto] = 30;

[1] I ruoli di ogni dominio sono distinti dalla posizione

- Istanza di relazione: su uno schema di relazione R(X) dove X sono attributi è l'insieme r di tuple su X;
- Istanza di base di dati: su uno schema di base di dati $\{R_1(X_1), ..., R_h(X_h)\}$ è l'insieme di relazioni $\{r_1, .., r_h\}$.

Le tuple costringono a usare determinati formati, ma in alcuni casi i valori potrebbero non essere disponibili. Es. Non tutti hanno un secondo nome. In questo caso si introduce un valore extra: il valore nullo (NULL).

2.2.1 Tipi di valore nullo

Ci sono vari casi di valore nullo, ma i DBMS non fanno distinzioni:

- valore sconosciuto: il valore esiste, ma non è noto. Es. il numero di telefono;
- valore inesistente: il valore non esiste. Es. permesso di soggiorno;
- valore senza informazione: il valore può essere sconosciuto o inesistente. Es. numero passaporto.

2.3 Strutture nidificate

Le strutture nidificate sono strutture che contengono altre strutture. Nei modelli relazionali i valori devono essere semplici e non relazioni, per cui non sono ammesse (*prima forma normale*). Per rappresentare strutture nidificate si usano più tabelle per cui si possono rappresentare informazioni più precise.

2.4 Vincoli

Un vincolo è un predicato logico che associa a ogni istanza un valore di verità.

2.4.1 Vincoli di integrità

I vincoli di integrità servono a garantire la correttezza dei dati. Se, dopo una modifica, il vincolo non risulta vero il DBMS rifiuta il cambiamento. Alcuni tipi di vincoli sono:

- intrarelazionali: che riguardano una sola relazione;
- interrelazionali: che riguardano più relazioni.

I vincoli di tupla sono intrarelazionali ed esprimono condizioni sul valore di ciascuna tupla. Es. voto $\geqslant 18$ AND voto $\leqslant 30$.

2.4.2 Vincoli di integrità referenziale

I vincoli di integrità referenziale sono detti anche vincoli di foreign key e servono per garantire la correttezza dei riferimenti tra tabelle. Questi vincoli hanno sempre un verso (es. X[tabella1] referenzia X[tabella2]). Se ci sono valori nulli i vincoli possono essere resi meno restrittivi modificando la base di dati. Alcune modifiche delle istanze possono essere gestite in questi modi:

- eliminazione a cascata: se viene eliminata una tupla referenziata si eliminano anche le tuple che la referenziano;
- introduzione di valori nulli: se viene eliminata una tupla referenziata si settano a NULL tutti i valori che la referenziano.

2.4.3 Vincoli di chiave

Una superchiave è un insieme di attributi usati per identificare univocamente le tuple di una relazione. L'insieme di tutti gli attributi è sempre una superchiave. Una superchiave minimale k è una superchiave a cui non è possibile rimuovere nessun attributo, altrimenti non sarebbe più una superchiave. k è una chiave (candidata) se e solo se è una superchiave minimale. Tutte le chiavi sono superchiavi. Una chiave primaria è una particolare chiave scelta come modo preferito per identificare le tuple. Ogni relazione ha una e una sola chiave primaria, ma può avere più superchiavi.

L'esistenza delle chiavi permette di accedere a ciascun dato. In presenza di valori nulli[2] i valori della chiave non permettono di identificare le tuple. La chiave primaria non può assumere valori nulli.

[2] vedi 2.2.1

Capitolo 3

Teoria 3

3.1 Algebra relazionale

L'algebra relazionale manipola relazioni per ottimizzare le *query*[1]. Ogni passo è definito attraverso gli operatori algebrici: selezione, proiezione, unione, intersezione, differenza, prodotto cartesiano, ridenominazione, join, quoziente. Ogni operatore riceve in input un argomento (una relazione) e produce in output una relazione virtuale. La cardinalità di una relazione virtuale è definita su un intervallo ed è il numero di tuple che la relazione contiene.

3.2 Operatori algebrici

3.2.1 Selezione

In una relazione r su uno schema A, $\sigma_p(r(A))$, l'operatore di selezione produce come risultato:

- schema: A;
- istanza: le tuple della relazione r che soddisfano il predicato p.

  Esempio: seleziona i pazienti con la residenza a Torino
  $(\sigma_{residenza=Torino}(\text{Pazienti}))$.
  La cardinalità della relazione virtuale dell'operatore di selezione è:
  $0 \leqslant |\sigma_p(r(A))| \leqslant |r(A)|$
  0 se il predicato è falso per tutte le tuple, $|r(A)|$ se il predicato è vero per tutte le tuple.

Proprietà:

- distributiva rispetto a proiezione, unione, intersezione, differenza, join, prodotto cartesiano;
- selezione multipla;
- sostituzione degli operatori.

3.2.2 Proiezione

In una relazione r su uno schema A, $\pi_{A_i,A_j,...,A_k}(r(A))$, l'operatore di proiezione produce come risultato:

- schema: $\{A_i, A_j, ..., A_k\}$;
- istanza: tutte le tuple della relazione argomento, ma solo rispetto agli attributi $A_i, A_j, ..., A_k$.

[1] Elenco dei passi da eseguire per rispondere a un'interrogazione

Esempio: i cognomi di tutti i pazienti
($\pi_{cognome}$(Pazienti)).
La cardinalità della relazione virtuale dell'operatore di proiezione è:
$0 \leqslant |\pi_{A_i,A_j,...,A_k}(r(A))| \leqslant |r(A)|$
Se gli attributi proiettati $A_i, A_j, ..., A_k$ formano una superchiave della relazione argomento allora

$$|\pi_{A_i,A_j,...,A_k}(r(A))| = |r(A)|$$

Proprietà:

- distributiva rispetto a unione, join, prodotto cartesiano;
- proiezione multipla.

3.2.3 Unione, intersezione e differenza

Gli operatori insiemistici (unione, intersezione e differenza) richiedono che gli schemi dei loro argomenti siano uguali. Il risultato dell'operatore insiemistico sulle relazioni argomento r_1(A) e r_2(A) è una relazione che ha:

- schema: lo stesso schema A delle relazioni argomento;
- istanza unione: r_1(A) U r_2(A);
- istanza intersezione: $r_1 \cap r_2$
- istanza differenza: r_1(A) - r_2(A).

La cardinalità della relazione virtuale dell'operatore di unione è:
$max|r_1(A)|, |r_2(A)| \leqslant |r_1(A) U r_2(A)| \leqslant |r_1(A)| + |r_2(A)|$
La cardinalità della relazione virtuale dell'operatore di intersezione è:
$0 \leqslant |r_1(A) \cap r_2(A)| \leqslant min|r_1(A)|, |r_2(A)|$
La cardinalità della relazione virtuale dell'operatore di differenza è:
$0 \leqslant |r_1(A) - r_2(A)| \leqslant |r_1(A)|$
L'intersezione può essere ricavata dalla differenza: $r_1(A) \cap r_2(A) := r_1(A) - (r_1(A) - r_2(A))$

3.2.4 Prodotto cartesiano

Date due relazioni r_1(A) e r_2(B) con $A \cap B = \emptyset$ (i due schemi non hanno attributi in comune), il prodotto cartesiano r_1(A) X r_2(B) produce come risultato una relazione r' con:

- schema: R' composto dall'unione degli schemi A U B;
- istanza: combinazione di tutte le tuple di r_1(A) con tutte le tuple di r_2(B).

La cardinalità della relazione virtuale dell'operatore di prodotto cartesiano è:
$0 \leqslant |r_1(A) X r_2(B)| \leqslant |r_1(A)| \cdot |r_2(B)|$
Il prodotto cartesiano non ha utilità pratica di per sè, ma viene usato per definire altri operatori.

Proprietà:

- associativa;
- commutativa.

3.2.5 Ridenominazione

Il compito dell'operatore di ridenominazione è quello di cambiare il nome di alcuni o tutti gli attributi della relazione argomento.

$\sigma_{B_i,B_j,...} \leftarrow_{A_i,A_j,...} (r)$

- schema: A' = $A_1, ..., B_i, B_j, ..., A_n$;
- istanza: le tuple non vengono modificate.

3.3 Join interni

3.3.1 Theta-join

Date due relazioni r_1(A) e r_2(B) con $A \cap B = \emptyset$ (i due schemi non hanno attributi in comune) e una condizione (predicato) θ di join (tipicamente θ è una formula proposizionale con confronti tra attributi del tipo $A_i \phi B_j$ o confronti tra attributi e valori $A_i \phi$ costante dove ϕ è un simbolo di confronto) il θ-join (theta-join) è definito come una selezione in base al prodotto cartesiano:

$r_1(A) \bowtie_\theta r_2(B) := \sigma_\theta(r_1(A) X r_2(B))$

La cardinalità della relazione virtuale dell'operatore di theta-join è:

$0 \leqslant |r_1(A) \bowtie_\theta r_2(A)| \leqslant |r_1(A)| \cdot |r_2(B)|$

Può essere utile eseguire una proiezione per rimuovere eventuali attributi derivati in eccesso. Se si vuole fare un join su schemi non disgiunti occorre ridenominare alcuni attributi.

Proprietà:

- associativa ristretta;
- commutativa.

3.3.2 Equi-join

L'equi-join (θ_e) è un caso particolare del theta-join in cui i confronti sono solo uguaglianze.

La cardinalità della relazione virtuale dell'operatore di equi-join è:

$0 \leqslant |r_1(A) \bowtie_{\theta_e} r_2(A)| \leqslant |r_1(A)|$

3.3.3 Natural-join

Il natural-join serve a confrontare attributi con lo stesso nome in tabelle diverse. Esso è un equi-join che è sempre vero.

3.3.4 Semi-join

Il semi-join è un filtro sulla prima relazione usando la seconda, una proiezione che prende in considerazione gli attributi della prima relazione dopo aver fatto un join tra le due.

3.4 Composizione di operatori

L'algebra relazionale è composizionale, quindi si possono costruire espressioni complesse componendo operatori.

Esempio: $\pi_{Cod,Nome}(\sigma_{Residenza='TO' \vee Residenza='VC'}(Pazienti)$

3.5 Dot notation

La dot notation serve per evitare di scrivere operazioni molto lunghe. Essa è un'operazione di ridenominazione sottintesa. Nella parte prima del punto va indicata la relazione e in quella dopo il punto va indicato l'attributo.

$\sigma_{MEDICI.Cognome,MEDICI.Nome,MEDICI.Residenza,MEDICI.Reparto} \leftarrow$
$\leftarrow_{Cognome,Nome,Residenza,Reparto} (Medici)$

Capitolo 4

Teoria 4

4.1 Interrogazioni con negazione

Nelle basi di dati si assume un mondo chiuso[1]. Per cui gli unici fatti veri sono quelli presenti nella base di dati. Se un fatto non è descritto allora è falso.

Schema generale:

1. si definisce l'universo del discorso (U);
2. si risponde all'interrogazione in modo positivo (P);
3. si trova la risposta all'interrogazione originale con la differenza tra U e P[2].

4.2 Interrogazioni con quantificazione universale

L'algebra relazionale non è in grado di gestire direttamente la quantificazione universale per cui ci deve ricondurre alla quantificazione esistenziale.

Schema generale:

1. si definisce l'universo del discorso (U);
2. si ricavano tutte le combinazioni possibili;
3. si trova la differenza;
4. si proietta sugli attributi di interesse;
5. si sottrae il risultato del punto 4 all'universo del punto 1.

   Questo ragionamento corrisponde all'operatore quoziente.

4.3 Quoziente

Il quoziente è un operatore derivato: $r(A,B) \div s(B) := \pi_a(r) - \pi_a((\pi_a(r)Xs) - r)$. Serve per ricavare tutte le tuple di r che compaiono in ogni combinazione. Non è implementato in SQL per cui si devono utilizzare sotto-query.

La cardinalità della relazione virtuale dell'operatore quoziente è:
$0 \leqslant |r(A,B) \div s(B)| \leqslant |\pi_A(r)| \leqslant |r|$

[1] Closed-world assumption

[2] U e P devono avere lo stesso schema, inoltre ciò vale solo in un mondo chiuso

4.4 Join esterni

4.4.1 Left join

Il left join contiene tutte le tuple della relazione a sinistra in join con la relazione di destra. Nel caso una tupla della relazione destra non faccia join con nessuna tupla della relazione di sinistra si inserisce il valore NULL per gli attributi della seconda relazione.

4.4.2 Right join

Il right join contiene tutte le tuple della relazione a destra in join con la relazione di sinistra. Nel caso una tupla della relazione sinistra non faccia join con nessuna tupla della relazione di destra si inserisce il valore NULL per gli attributi della prima relazione.

4.4.3 Full join

Il full join è l'unione di left join e right join.

4.5 Semantica di Codd del valore nullo

La semantica di Codd del valore nullo è il comportamento di un'interrogazione con tuple che contengono valori nulli. Per fare ciò, Codd propose di passare da una logica a due valori (True, False) a una logica a tre valori (True, False, Unknown).

4.5.1 Logica a tre valori

Tre valori:

- True (T, vale 2);
- False (F, vale 0);
- Unknown (U, vale 1).

In questa logica:

- AND si calcola con il minimo;
- OR si calcola con il massimo;
- NOT si calcola con 2 - p.

Per cercare un valore nullo si usa il predicato ISNULL, per cercarne uno non nullo si usa ISNOTNULL. In questa logica non valgono nè il principio di non contraddizione, nè il principio del terzo escluso.

Capitolo 5

Teoria 5

5.1 Ottimizzazione delle interrogazioni

Ottimizzare significa rendere più efficiente un'implementazione.

Poichè i DBMS sono molto grandi sono memorizzati in memoria secondaria e, quindi, se un'applicazione richiede una tupla bisogna portare in memoria la pagina in cui è memorizzata. Per ottimizzare il tempo si deve minimizzare il numero di pagine da trasportare in memoria primaria (*buffer*). L'ottimizzazione si svolge in due passaggi:

- Ottimizzazione logica;
- Ottimizzazione fisica.

5.1.1 Ottimizzazione fisica

L'ottimizzatore fisico entra nei nodi (*operatori*) dell'albero sintattico, li esamina e in base alle strutture fisiche sceglie l'algoritmo ottimale per eseguire ogni nodo. Si vedrà più in dettaglio nella magistrale.

5.1.2 Ottimizzazione logica

L'ottimizzazione logica prende in input l'albero sintattico dell'interrogazione e lo trasforma sfruttando le proprietà dell'algebra relazionale.

Il principio su cui si basa l'ottimizzatore logico è: ridurre il numero di tuple coinvolte dall'interrogazione, mantenendo lo stesso risultato. In quasi tutte le operazioni di un sistema informativo si lavora su poche tuple per volta, compiendo una selezione[1].

Per ottimizzare si utilizza la proprietà distributiva della selezione. I predicati p della selezione sono in forma congiuntiva (si può ricondurre qualsiasi predicato a una congiunzione di disgiunzioni):

1. Si decompongono gli **AND**: $\sigma_p(r(A) \bowtie_\theta s(B)) \rightarrow \sigma_p(r(A)) \bowtie_\theta s(B)$, che è valida solo se gli attributi coinvolti da p sono contenuti solo in A;
2. Si trasferiscono le selezioni verso le foglie finché è possibile con le proprietà distributive della selezione;
3. Si traferiscono le proiezioni verso le foglie finché è possibile con le proprietà distributive della proiezione;
4. L'ottimizzatore ricompone insieme le selezioni multiple, applicando una sola selezione con una congiunzione di predicati;
5. Si traforma la selezione + prodotto cartesiano in un $\bowtie_\theta$ (perchè è meglio nell'ottimizzazione fisica);
6. Ricondurre a un'unica proiezione le proiezioni multiple;
7. Si esaminano le varianti dell'albero sintattico dovute alle proprietà associative scegliendo la variante di costo minimo.

[1] vedi 3.2.1

5.2 Aspetti quantitativi nelle interrogazioni

I DBMS hanno un dizionario di dati in cui vengono memorizzare varie informazioni:

- **CARD(r)** $= |r|$: è la cardinalità della relazione;
- **SIZE(t)** : ampiezza della tupla in byte;
- **VAL**(A_i**,r**) : numero di valori distinti che appaiono nella colonna A_i all'interno della tabella r;
- **MIN**(A_i**,r**) : il valore minimo di A_i contenuto in r;
- **MAX**(A_i**,r**) : il valore massimo di A_i contenuto in r;
- **NPAGE(r)** : il numero di pagine occupate da r.

  NPAGE(r) $= \frac{\textbf{CARD(r)}}{\textit{fattore di bloccaggio}}$. Il *fattore di bloccaggio* è il numero massimo di tuple contenute in una pagina.

5.3 Analisi dei costi delle interrogazioni

L'analisi quantitativa dell'interrogazione permette di predire a priori il risultato della cardinalità della relazione, senza eseguirla.

5.3.1 Stima del costo della selezione

Data la selezione $\sigma_p(r)$, conoscendo l'intervallo di variabilità della selezione $\sigma_p(r)$: $0 \leqslant |\sigma_p(r)| \leqslant |r|$, si può modellare la cardinalità della selezione $\sigma_p(r)$ con un *fattore di selettività* f_p per la cardinalità di r $|\sigma_p(r)| = f_p X |r|$. Il fattore di selettività f_p è legato al solo predicato p di selezione e varia tra 0 e 1.

Il fattore di selettività f_p può essere interpretato come la probabilità che una tupla in r soddisfi il predicato di selezione p, ovvero la stima della percentuale di tuple che soddisfano il predicato di selezione. Per stimare f_p si assumono una *distribuzione uniforme* e *un'assenza di correlazione* tra attributi diversi.

5.3.2 Stima della cardinalità del join

Per stimare $|r(A) \bowtie_{A_i=B_j} s(B)|$, consideriamo prima una singola tupla $t' \in r$ per cui $t'[A_i] = v$ (v è una costante):

- la probabilità che una singola tupla $t'' \in s$ sia tale che $t''[B_j] = v$ è $\frac{1}{\textbf{VAL}(B_j,s)}$;
- considerando tutta la relazione s, ci saranno $\frac{1}{\textbf{VAL}(B_j,s)} \times \textbf{CARD(s)}$ tuple t" per cui t"$[B_j] = v$.

Se non esiste un vincolo di integrità referenziale, sappiamo che la stima precedente è ottimistica, perché potrebbe non esserci nessuna tupla in s tale che t"$[B_j] = v$.

Si possono fare due stime, entrambe ottimistiche, quindi si deve prendere quella minore: $|r(A) \bowtie_{A_i=B_j} s(B)|$ = min $\{\frac{1}{\textbf{VAL}(A_i,r)}, \frac{1}{\textbf{VAL}(B_j,r)}\} \times \textbf{CARD}(\text{r}) \times \textbf{CARD}(\text{s})$.

Capitolo 6

Teoria 6

6.1 Calcolo relazionale sulle tuple con dichiarazione di range

Il Calcolo relazionale sulle tuple con dichiarazione di range è la base teorica di SQL. In questo calcolo le variabili denotano tuple e bisogna specificare un *range* di valori possibili. Le sue interrogazioni sono composte da tre parti:

$\{T|L|F|\}$

- *Target (T)*: specifica gli attributi che compaiono nel risultato;
- *Range list (L)*: specifica il dominio delle variabili non quantificate in F;
- *Formula (F)*: specifica un'espressione logica che deve essere soddisfatta dal risultato.

  Il risultato è dato da:

- l'insieme dei valori degli attributi T;
- presi dalle tuple nelle variabili in L;
- che rispettano la formula in F.

6.1.1 Target list

La target list è l'elenco delle informazioni che si vogliono avere in uscita. Le variabili usate nella target list devono essere dichiarate nella Range list. Ci sono diverse sintassi possibili:

- variabile.Attributo1, variabile.attributo2,...;
- variabile.(Attributo1,Attributo2,...);
- variabile.*[1];
- Nome: variabile.attributo.

  Esempio: p.Nome, p.Cognome significa che nel risultato compariranno nomi e cognomi.

6.1.2 Range list

La range list è l'introduzione di variabili abbinate a relazioni di base.
nome_variabile(nome_relazione_di_base)
Esempio: p(Pazienti) significa che la variabile p assume valori nella relazione Pazienti. Ed è una qualunque tupla di pazienti.

[1] Restituisce tutti gli attributi

6.1.3 Formula

La formula è un predicato di logica del primo ordine che vincola le variabili della range list. Quindi si possono applicare i soliti operatori della logica proposizionale (AND, OR, NOT, etc.) e i quantificatori esitenziali e universali.

Esempio: p.Residenza='TO' significa che, data una tupla p, perché questa faccia parte del risultato, l'attributo Residenza di p deve valere 'TO'.

La formula è un predicato del primo ordine che può contenere sia variabili libere che quantificate (vincolate). Tutte le variabili libere presenti nella formula devono essere dichiarate nella range list.

$\exists$variabile(Relazione)(formula)

$\forall$variabile(Relazione)(formula)

6.1.4 Limitazioni

Nel calcolo relazionale sulle tuple con dichiarazione di range non è possibile esprimere l'unione. Infatti nella range list ogni variabile ha come dominio una sola relazione, mentre l'unione richiede che il risultato venga da una relazione o da un'altra. SQL (che è basato su questo calcolo) prevede un operatore esplicito di unione, ma non tutte le versioni prevedono intersezione e differenza. Inoltre manca il concetto di ricorsione, per cui non è possibile esprimere alcune query. Esempio: discendente(Persona1,Persona2) esprime tutte le discendenze e richiederebbe una chiusura transitiva[2].

6.2 Calcolo relazionale e SQL

Il calcolo relazionale è direttamente correlato alla sintassi di SQL.

- La target list corrisponde alla SELECT;
- La range list corrisponde alla FROM;
- La formula corrisponde alla WHERE.

Tuttavia in SQL non è presente il quantificatore universale per cui si ricorre al NOT e al quantificatore esistenziale.

[2]Le chiusure transitive sono implementate nelle ultime versioni di SQL

Capitolo 7

Teoria 7

7.1 Forme normali

Una forma normale è una proprietà di una base di dati relazionale che ne garantisce la qualità, cioè l'assenza di determinati difetti. Solitamente è definita nel modello relazionale. Se non è presente possono esserci anomalie durante operazioni di inserimento, cancellazione o modifica e ridondanze.

A una forma normale può essere associato un algoritmo di normalizzazione, che specifica come decomporre uno schema relazionale per renderlo in forma normale. In questo corso verranno trattate solo le 2 più usate: Boyce-Codd Normal Form (BCNF) e Terza forma normale (3NF).

7.2 Normalizzazione

La normalizzazione consiste nella decomposizione di uno schema di relazione in modo da ottenere più schemi che rispettino una forma normale e minimizzino le anomalie. Può essere usata come tecnica di verifica dei risultati della progettazione di una base di dati. Nelle sezioni successive si userà il seguente esempio:

ESAMI(MATR, NomeS, IndirizzoS, CAPS, CodiceFiscaleS, DataNascitaS, Corso, Voto, Lode, DataEsame, CodProf, NomeProf, Qualifica, TipoUfficio)

Supponiamo che ESAMI sia l'unica relazione che descrive il sistema informativo.

7.2.1 Anomalie di inserimento

Un nuovo studente deve immatricolarsi, dato che non ha superato nessun esame non può essere inserito perchè Corso non può essere NULL, dato che è chiave primaria[1].

Lo stesso errore si verifica con l'inserimento di un nuovo docente.

7.2.2 Anomalie di cancellazione

Si vogliono cancellare gli esami degli studenti laureati. Se un professore non ha esami di studenti che devono ancora laurearsi, viene cancellato anche ogni riferimento al professore anche se è ancora in servizio.

7.2.3 Anomalie di aggiornamento

Se uno studente cambia indirizzo bisogna cambiarlo in ogni sua tupla e ciò causa una criticità di efficienza.

7.2.4 Dipendenze funzionali

Presa una coppia di tuple della relazione ESAMI, se queste tuple coincidono sul valore della matricola, anche tutti gli altri attributi relativi allo studente devono coincidere.

Dati una relazione r(A) e due sottoinsiemi X e Y di attributi di A (X,Y $\subseteq$ A), il vincolo di dipendenza funzionale

[1] vedi 2.4.3

$$X \rightarrow Y \text{ (X determina Y)}$$

è soddisfatto solo se $\forall t_1, t_2 \in r(t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y])$

Le dipendenze funzionali vengono raccolte attraverso un'analisi attenta della realtà e non esiste un modo univoco per rappresentarle. Per ogni relazione r abbiamo un insieme F di dipendenze funzionali. Due basi di dati, una progettata con i vincoli F' e un'altra progettata con i vincoli F", se F' e F" sono equivalenti, evolvono nello stesso modo.

7.3 Teoria di Armstrong

La teoria di Armstrong fornisce una serie di regole (assiomi) per gestire le dipendenze funzionali. Usare gli assiomi di Armstrong equivale a ragionare con le definizioni di dipendenza funzionale.

7.3.1 Correttezza e completezza

La teoria di Armstrong è:

- corretta, perchè dato un insieme di dipendenze funzionali F, se è possibile dedurre $X \rightarrow Y$ tramite gli assiomi di Armstrong, allora è possibile ricavare $X \rightarrow Y$ tramite la definizione di dipendenza funzionale;
- completa, perchè dato un insieme di dipendenze funzionali F, se è possibile ricavare $X \rightarrow Y$ tramite la definizione di dipendenza funzionale, allora è possibile dedurre $X \rightarrow Y$ tramite gli assiomi di Armstrong.

7.3.2 Assiomi di Armstrong

Non sono propriamente assiomi, in quanto sono dimostrabili.

Riflessività: se $Y \subseteq X$, allora $X \rightarrow Y$.

Unione: se $Y \rightarrow Y$ e $Y \rightarrow Z$ allora $X \rightarrow YZ$, dove YZ = Y U Z.

Transitività: se $X \rightarrow Y$ e $Y \rightarrow Z$, allora $X \rightarrow Z$.

7.3.3 Regole aggiuntive

Queste regole si possono ricavare dagli assiomi.

Espansione: dati una dipendenza funzionale $X \rightarrow Y$ e un insieme di attributi W, allora $WX \rightarrow WY$.

Decomposizione: se $X \rightarrow YZ$, allora $X \rightarrow Y$ e $X \rightarrow Z$.

Pseudo-transitività: se $X \rightarrow Y$ e $WY \rightarrow Z$, allora $WX \rightarrow Z$.

Prodotto: date le dipendenze funzionali $X \rightarrow Y$ e $W \rightarrow Z$, allora vale $XW \rightarrow YZ$.

7.4 Chiusure

7.4.1 Chiusura di un insieme F

Si possono utilizzare le regole della teoria di Armstrong per dimostrare che due insiemi di dipendenze funzionali siano equivalenti. La chiusura di un insieme di dipendenze funzionali F è l'insieme F^+ di tutte le dipendenze funzionali derivabili da F. Due insiemi di dipendenze funzionali sono equivalenti solo se l'insieme di tutte le dipendenze funzionali derivabili sono uguali. Tuttavia, il tempo per questo calcolo è esponenziale.

7.4.2 Chiusura di un insieme di attributi

Dato un insieme di attributi R su cui è definito l'insieme di dipendenze funzionali F, dato un sottoinsieme $X \subseteq R$, la chiusura X_F^+ di X (o semplicemente X^+ se non ci sono ambiguità) è definita come:

$$X_F^+ = \{A | X \rightarrow A \in F^+\}$$

Questo algoritmo è sia completo che corretto. Il tempo per questo calcolo è polinomiale.

7.5 Nuova definizione di superchiave

Dato uno schema di relazione R(A) con un insieme di dipendenze funzionali F, un insieme di attributi K $\subseteq$ A è superchiave se e solo se A = K_F^+ (cioè se e solo se in F^+ si trova il vincolo di dipendenza funzionale K $\rightarrow$ A).

Proprietà: Se due progettisti identificano, su una medesima base di dati, due insiemi di dipendenze funzionali F e G equivalenti, i due progettisti arriveranno alle stesse identiche chiavi candidate.

Capitolo 8

Teoria 8

8.1 Decomposizione senza perdita

Si possono decomporre relazioni complesse in relazioni più semplici, mantenendo almeno un elemento in comune (per collegare le relazioni derivate).

Per esempio: S(Matr, NomeS, Voto, Corso, CodC, Titolare) si può decomporre in S1(Matr, NomeS, Voto, Corso) e S2(Corso, CodC, Titolare).

8.1.1 Tuple spurie

Le tuple spurie sono una perdita di informazioni quando si ricompongono le relazione. Ci sono delle tuple di troppo, scorrette che risultano indistinguibili dalle tuple corrette.

Decomposizione senza perdita di informazioni: Dato uno schema di relazione R(A), dati due sottoinsiemi di attributi $A_1 \in A$ e $A_2 \in A$, con $A_1 U A_2 = \text{A}$, $\{R_1(A_1), R_2(A_2)\}$ è una decomposizione senza perdita di informazione se e solo se per ogni istanza r(A) di R(A) vale

$$\text{r(A)} = r_1(A_1) \bowtie r_2(A_2)$$

dove

- $r_1(A_1) = \pi_{A_1}(r(A))$;
- $r_2(A_2) = \pi_{A_2}(r(A))$.

Teorema della decomposizione senza perdita: Sia R(A) uno schema con dipendenze funzionali F decomposto in $\{R_1(A_1), R_2(A_2)\}$ dove $A_1 U A_2 = \text{A}$. La decomposizione di R(A) in $\{R_1(A_1), R_2(A_2)\}$ è senza perdita di informazione per ogni istanza che soddisfa le dipendenze funzionali F se e solo se:

$$A_1 \subseteq (A_1 \cup A_2)^+_F \vee A_2 \subseteq (A_1 \cup A_2)^+_F$$

8.2 Decomposizione che conserva le dipendenze

Data una relazione R(A) con dipendenze funzionali F, decomposta in $R_1(A_1)$ con la restrizione F_1 e $R_2(A_2)$ con la restrizione F_2, la decomposizione $\{R_1, R_2\}$ conserva le dipendenze quando $F_1 U F_2 \Rightarrow F$.

8.3 BCNF

La BCNF[1] prende il nome da Boyce (uno degli inventori di SQL) e Codd (che ha definito il modello relazionale). Esiste un algoritmo per la BCNF (con complessità esponenziale) ma non verrà visto in questo corso.

Data una relazione R(A) in 1NF e un insieme di dipendenze funzionali F, la relazione è in BCNF se e solo se per ogni $X \rightarrow Y \in F$ si verifica almeno una delle seguenti condizioni:

[1] Boyce-Codd normal form

- $Y \subseteq X$;
- X è superchiave di R.

La BCNF evita le ridondanze perchè ogni antecedente di una dipendenza funzionale è superchiave. La BCNF evita le anomalie. Esistono schemi che vìolano la BCNF e per cui non esiste alcuna decomposizione in BCNF che conservi le dipendenze.

Capitolo 9

Teoria 9

9.1 3NF

Se una relazione è in 3NF allora è anche in BCNF. La 3NF è sempre raggiungibile mantenendo le dipendenze funzionali, ma non elimina tutte le anomalie.

Una relazione (R(A),F) è in 3NF (terza forma normale) se per ogni ogni $X \rightarrow Y \in F$ si verifica almeno una delle seguenti condizioni:

- $Y \subseteq X$;
- X è superchiave di R;
- Y sono attributi primi.

Data una relazione R(A), gli attributi $Y \subseteq A$ sono detti attributi primi se e solo se $Y \subseteq K$, dove K è una chiave di R(A).

Partendo da un insieme di dipendenze funzionali, si deve trovare un altro insieme di dipendenze funzionali equivalente e minimale. Per fare ciò si introducono i concetti di attributo estraneo e dipendenza ridondante.

Attributo estraneo: un attributo in una dipendenza funzionale in F è estraneo se e solo se possiamo rimuovere l'attributo dalla dipendenza funzionale continuando ad avere un insieme di dipendenze funzionali equivalente.

Dipendenza ridondante: Una dipendenza funzionale è ridondante in un insieme di dipendenze funzionali F se e solo se possiamo rimuoverla da F continuando un insieme di dipendenze funzionali equivalente.

9.2 Insieme di copertura minimale

Un insieme F' di dipendenze funzionali è un insieme di copertura minimale rispetto a F quando:

- $F' \equiv F$ (equivalenza, indica che è minimale);
- in ogni $X \rightarrow Y \in F'Y$ è un attributo singolo (si dice che è in forma canonica, non è un requisito necessario, ma semplifica la trattazione);
- ogni $X \rightarrow Y \in F'$ è priva di attributi estranei;
- ogni $X \rightarrow Y \in F'$ non è ridondante.

9.2.1 Algoritmo per il calcolo di una copertura minimale

1. per ogni d. f. $X \rightarrow A_1...A_n \in F'$ si sostituisce in F' la d. f. $X \rightarrow A_1...A_n$ con $X \rightarrow A_1, ..., X \rightarrow A_n$;
2. per ogni d. f. $X \rightarrow A_i \in F'$ (per ogni $B_j \in X$ (se $A_i \in (X - B_j)^+_{F'}$ allora cancella B_j da X e aggiorna F'));
3. per ogni d. f. $X \rightarrow A_i \in F'$ (F* := F' - $(X \rightarrow A_i)$ se $A_i \in X^+_{F*}$ allora F' := F*) return F'.

Il passo 1 decompone le d. f. per trasformarle in d. f. di un attributo singolo. Il passo 2 serve per eliminare gli attributi estranei e il passo 3 serve per eliminare le dipendenze ridondanti. Bisogna sempre prima eliminare tutti gli attributi estranei e poi eliminare le dipendenze funzionali ridondanti, altrimenti si rischia di non riconoscere tutte le dipendenze funzionali ridondanti.

La copertura minimale F' non è unica, ma tutte le coperture minimali che si ottengono eseguendo l'algoritmo sono equivalenti. La complessità dell'algoritmo per il calcolo dell'insieme di copertura minimale è polinomiale.

9.3 Normalizzazione in 3NF

1. calcola la copertura minimale F' di F;
2. per ogni insieme di d. f. $\{X \rightarrow A_1, ..., X \rightarrow A_n\} \subseteq F'$ che contiene tutte le d. f. che hanno a sinistra gli stessi attributi X (crea la relazione $(R_X(\underline{X}A_1...A_n), \{X \rightarrow A_1...A_n\})$);
3. per ogni coppia di relazioni $R_X(\underline{X}Y), R_{X'}(\underline{X'}Y')$ in cui $X'Y' \subseteq XY$ (elimina la relazione $R_{X'}$ e aggiungi le d. f. di $R_{X'}$ a quelle di R_X;
4. se nessuna relazione contiene una chiave K qualsiasi di R(A) (trova K tale che $K^+ = A$ e crea una nuova relazione $R_K(K)$.

9.3.1 Proprietà della normalizzazione 3NF

- nelle relazioni $R_i(XA_1...A_n)$ generate dalle d. f. $X \rightarrow A_i$, X è chiave di R_i;
- genera relazioni in 3NF;
- ha complessità polinomiale;
- conserva le dipendenze, infatti troviamo ogni dipendenza di F' all'interno della relazione corrispondente;
- garantisce la decomposizione con join senza perdita.

Tutte le dimostrazioni di queste proprietà sono spiegate in dettaglio nelle slide del corso.

Capitolo 10

Teoria 10

10.1 Rapporto tra normalizzazione e schemi ER

ER produce relazioni in BCNF, soffrendo delle sue limitazioni, per cui si può decidere di passare alla 3NF. Si può utilizzare la normalizzazione per verificare la qualità di uno schema ER, verificando se rispetta le d. f.. Nelle slide del corso sono presenti più costrutti, ma non sono stati trattati nel corso della lezione. Per gli esempi sulla verifica delle qualità fare riferimento alle slide sulla Normalizzazione IV.

Entità. Un entità E con identificatore I e attributo A diventa $R_E(\underline{I}, A)$. L'entità E rappresenta la d. f. $I \to A$.

Associazione molti a molti. Da E_1 si ha $I_1 \to I_2$ e dà E2 si ha $I_2 \to A_2$. Dall'associazione A si ha che ogni sua occorrenza (e del suo attributo B) è individuata dalla coppia di occorrenze di E_1 e E_2, per cui $I_1 I_2 \to B$.
La traduzione è: $R_{E_1}(\underline{I_1}, A_1), R_{E_2}(\underline{I_2}, A_2), R_A(\underline{I_1}, \underline{I_2}, B)$.

Associazione uno a molti. Da E_1 si ha $I_1 \to I_2$ e dà E2 si ha $I_2 \to A_2$. Dall'associazione A si ha che ogni sua occorrenza (e del suo attributo B) è individuata dalla coppia di occorrenze di E_1 e E_2, per cui $I_1 I_2 \to B$. A causa della cardinalità dell'associazione uno a molti (1,1) si ha $I_1 \to B I_2$.
La traduzione è: $R_{E_1}(\underline{I_1}, A_1, B, I_2), R_{E_2}(\underline{I_2}, A_2)$.

Identificazione esterna. Da E_1 si ha $I_1 \to I_2$ e da E2 si ha $I_2 \to A_2$. Da E_3 abbiamo che, considerando l'identificazione esterna, preso un determinato valore di I_3 abbinato a una coppia di occorrenze di E_1 e di E_2, troviamo un determinato valore di A_3. Quindi $I_1 I_2 I_3 \to A_3$.
La traduzione è: $R_{E_1}(\underline{I_1}, A_1), R_{E_2}(\underline{I_2}, A_2), R_{E_3}(\underline{I_1}, \underline{I_2}, \underline{I_3}, A_3)$.

Generalizzazione. Non è considerata dato che viene eliminata nella ristrutturazione dello schema ER.

Capitolo 11

Teoria 11

11.1 Transazioni

In una base di dati si possono effettuare operazioni di inserimento (INSERT), cancellazione (DELETE) e aggiornamento (UPDATE). Il DBMS accetta modifiche solo se rispettano i vincoli [1], altrimenti le rifiuta.

I vincoli possono bloccare istruzioni singole. La soluzione è consentire al DBMS di eseguire quelle istruzioni e attendere l'arrivo di altre istruzioni che pongano rimedio alla violazione dei vincoli. Questo ragionamento è alla base del concetto di transazione.

Una transazione è:

- un'unità di programma che inizia con ***begin transaction*** che comunica al DBMS la richiesta di interazione da parte dell'applicazione;
- il DBMS identifica l'inizio della transazione T_i e la abbina in modo univoco con l'utente/applicazionec che ne ha fatto richiesta;
- il DBMS riceve dei comandi DML in sequenza e li abbina alla transazione;
- solitamente breve (buona pratica);
- se la transazione va a buon fine si termina con il comando ***commit work***;
- se la transazione fallisce si termina con il comando ***rollback work***.

11.2 Ciclo di vita di una transazione

Con stato 1 si indica la transazione attiva, con stato 2 si indica la transazione fallita e con stato 3 si indica la transazione parzialmente terminata. I due stati finali sono aborted e committed. Questa simbologia serve per una rappresentazione mediante un DFA[2] del ciclo di vita di una transazione.

1. La transazione ha inizio (Stato 1);
2. Finchè la transazione riceve comandi di inserimento, cancellazione e modifica rimane in questo stato (Stato 1). Se fallisce va al punto 3 (Stato 2), se va a buon fine va al punto 4 (Stato 3);
3. Dopo il rollback la transazione può solo raggiungere lo stato finale di **aborted**;
4. Dopo il commit se la transazione rispetta i vincoli passa allo stato finale di **committed**, altrimenti passa al punto 3 (Stato 2);

[1] vedi 2.4
[2] come visto nel corso di LFT

11.3 Proprietà delle transazioni

Una transazione (in una base di dati relazionale) ha le seguenti proprietà (dette ACID):

- atomica: una transazione avviene o per intero o non avviene affatto. La BD non può essere lasciata in uno stato intermedio. Per garantire questa proprietà si usano comandi di UNDO o REDO, più file di log;
- consistente (coerente): l'esecuzione di una transazione non deve violare i vincoli;
- isolata: per garantire l'esecuzione concorrente come se ogni operazione venisse eseguita singolarmente;
- durabile (persistente): tutte le modifiche devono rimanere, anche se sono presenti guasti.

Capitolo 12

Teoria 12

12.1 Architettura dei DBMS

Gestore delle interrogazioni: riceve i comandi SQL delle interrogazioni, li rende più efficienti ed elabora un piano per la loro esecuzione.

Gestore delle transazioni: invia i comandi DML, per gestire il ciclo di vita delle transazioni, alle componenti che ne fanno richiesta. Garantisce parte della proprietà di consistenza.

Serializzatore: garantisce la proprietà di isolamento e parte della proprietà di consistenza.

Gestore del ripristino: gestisce ripristini in seguito a guasti ed è responsabile delle proprietà di atomicità e di integrità.

Gestore del buffer: garantisce la proprietà di durabilità.

12.2 Gestore del buffer

Il DBMS usa alcune funzionalità del *file system* creando una propria astrazione dei file che consente di garantire efficienza e transazionalità. La struttura dei file è gestita direttamente dal DBMS.

12.2.1 Richiami di architettura degli elaboratori

I programmi possono fare riferimento solo a dati in memoria principale. I dati in memoria secondaria possono essere utilizzati solo se prima trasferiti in memoria principale. Le basi di dati sono troppo grandi per essere memorizzate interamente in memoria principale, inoltre la persistenza richiede esplicitamente che i dati siano salvati in memoria secondaria.

12.2.2 Richiami di sistemi operativi

Tutti i dati del database (tabelle, indici, log, dump) e i record (la realizzazione fisica del concetto di tupla) sono organizzati in pagine. La dimensione delle pagine è variabile e dipende dal sistema. Se la transazione ha bisogno di lavorare su un determinato record , il gestore del buffer cerca una pagina *pid* (page id) che contiene il record desiderato. Il gestore del buffer, riceve una richiesta chiamata fix pid, che significa: metti a disposizione della transazione la pagina pid.

Quando il gestore del buffer riceve la richiesta di una pagina essa viene presa dalla memoria secondaria trasferendola in cache. Dalla cache la pagina viene portata in memoria principale. Dopo di chè il gestore del buffer risponde alla transazione, che aveva richiesto la pagina, con l'indirizzo della memoria in cui trovare la pagina. Il costo temporale grava soprattutto sulla movimentazione delle pagine da memoria secondaria a memoria principale.

I dispositivi di memoria secondaria sono organizzati in blocchi di lunghezza fissa (grandi alcuni KB). Le uniche operazioni sui dispositivi sono la lettura e la scrittura di una pagina, cioè dei dati di un blocco[1]. Se la dimensione di un blocco è superiore alla dimensione media di un record allora un blocco può contenere più record e lo spazio residuo può essere usato per record spanned (altre relazioni) o non usato.

Un record ID è composto da due parti:

- un pid che punta all'intera pagina;
- un offset che punta al record preciso di quella pagina.

Nelle pagine sono anche presenti un PinCount (che conta le transazioni che hanno avuto l'okay di accesso alla pagina), un dirty bit (che è a 1 se la pagina è stata modificata rispetto alla versione presente in memoria secondaria) e uno stack dei record (contenente i record della pagina).

12.3 Strutture primarie per l'organizzazione dei file

Le strutture primarie per l'organizzazione dei file specificano come sono organizzati record e pagine:

- file di record a heap (struttura seriale);
- file di record ordinati (struttura sequenziale);
- struttura a hash[2].

12.3.1 Record a heap

Questa struttura è la più usata dai DBMS, spesso associato a strutture di accesso secondarie. I record vengono inseriti nelle pagine in ordine di arrivo (in coda o al posto di record cancellati). La cancellazione lascia spazio inutilizzato, per cui i blocchi vanno ricompattati periodicamente. Il costo di un'interrogazione va valutato a priori: una ricerca con insuccesso prevede la lettura di tutte le pagine, una con successo prevede, in media, di leggere metà delle pagine (perchè il record ha la stessa probabilità di trovarsi in ogni pagina).

12.3.2 Struttura ordinata

L'inserimento è costoso perché richiede di spostare pagine. La cancellazione si può attuare contrassegnando un record come cancellato e attuando una riorganizzazione periodica. La ricerca con insuccesso sull'attributo chiave (usato per ordinare i record) ha lo stesso costo della ricerca con successo perché non si ha bisogno di scorrere tutte le pagine (ci si può arrestare quando si supera il valore richiesto). Se si fa una ricerca su un attributo non chiave, in caso di insuccesso, si devono scorrere tutte le pagine (come nel caso dei record a heap).

Si possono migliorare i tempi di accesso delle strutture seriali e sequenziali aggiungendo strutture secondarie.

12.4 Strutture secondarie (indici)

Un indice è memorizzato in una sezione diversa dalle aree che contengono le pagine. Gli indici contengono puntatori (RID) ai record memorizzati in strutture primarie.

Un DB administrator crea un indice quando il carico di lavoro in termini di interrogazioni è tale per cui il costo di accesso all'indice riduce notevolmente i tempi di risposta rispetto alla scansione sequenziale. Tuttavia ciò appesantisce inserimenti, cancellazioni e modifiche (gli indici sono utili se il DB non viene modificato spesso).

Gli indici possono essere:

- primari: se definiti sullo stesso attributo chiave relazionale su cui sono ordinati i record;
- clusterizzati: se definiti su un attributo non chiave relazionale su cui sono ordinati i record;
- secondari: se definiti su un attributo qualunque sui cui i record non sono ordinati.

[1] In questo corso pagina e blocco verrano considerati come sinonimi
[2] Che non è argomento di questo corso

12.4.1 B-tree

Un B-tree è una generalizzazione degli alberi binari di ricerca[3] in cui ogni nodo può avere m figli (branching factor):

- ogni sottoalbero di sinistra di una chiave k ha chiavi di ricerca strettamente inferiori alla chiave k;
- ogni sottoalbero di destra di una chiave k ha chiavi di ricerca strettamente superiori alla chiave k.

I B-tree sono bilanciati.
Dato un B-tree con branching factor m, il B-tree ha le seguenti proprietà:

- ogni nodo ha al massimo m-1 chiavi;
- ogni nodo (tranne la radice) ha almeno $\frac{m}{2} - 1$ chiavi (è mezzo pieno);
- se non è vuoto la radice ha almeno una chiave;
- tutte le foglie sono allo stesso livello;
- un nodo non-foglia che ha k chiavi ha k+1 figli.

Il dato quantitativo importante per i DBMS è il numero di livelli (la stima del numero minimo e del numero massimo di livelli è presente sulle slide). Si può approssimare il tutto a $N \simeq m^L$ (dove N è il numero di chiavi, m è il branching factor e L è il numero di livelli).

Nei sistemi DBMS, gli indici reggono bene un carico del 70% della capacità massima (con 3 livelli si reggono bene 700 mila chiavi).

12.4.2 B+-tree

Si cerca di mantenere in memoria principale i nodi più interni (perchè attraversati più spesso). Le chiavi di ricerca vengono duplicate nelle foglie, per alleggerire i nodi interni. Inoltre le foglie sono linkate le une alle altre.

Tipi diversi di data entry K*:

- <k,tupla> il data entry è il record stesso (B+-tree è usato come struttura di memorizzazione primaria);
- <k,RID>: il data entry è puntatore al record nell'area primaria;
- <k,lista_di_RID>: il data entry è una lista di puntatori con la stessa chiave di k, serve quando si vuole fare riferimento a più record.

12.4.3 Regole generali per l'utilizzo degli indici

- Evitare gli indici su tabelle di poche pagine;
- Evitare indici su attributi volatili;
- Evitare indici su chiavi poco selettive;
- Evitare indici su chiavi con valori sbilanciati;
- Limitare il numero di indici;
- Definire indici su chiavi relazionali ed esterne;
- Gli indici velocizzano le scansioni ordinate;
- Conoscere a fondo il DBMS.

[3] visti nel corso di Programmazione II

Capitolo 13

Teoria 13

13.1 Gestione delle transazioni

Supponiamo che il DBMS riceva contemporaneamente le transazioni T_1 e T_2. Queste due transazioni devono godere della proprietà di isolamento. Per cui l'esecuzione completa di T1 (isolata) seguita dall'esecuzione completa di T2 (isolata) ha la proprietà di lasciare la BD in una situazione di consistenza. Se il DBMS riceve le transazioni contemporaneamente, la scelta di eseguire la sequenza T_1T_2 oppure T_2T_1 è irrilevante. In base al sistema informativo le due transazioni possono essere eseguite in parallelo oppure prima una e poi l'altra (ma questo caso è inefficiente per via dei tempi morti).

L'esecuzione in parallelo di due transazioni richiede di interfogliare (interleave) le attività delle transazioni.

Inconsistenze dell'interfogliamento: l'interfogliamento può causare problemi interferendo con le transazioni.

13.2 Serializzabilità

La schedulazione (o storia) è uno specifico interfogliamento. Una storia è la sequenza di azioni eseguite dal DBMS per far fronte alle transazioni.

Dato che si assume che l'esecuzione seriale delle transazioni sia consistente, sono consistenti tutti gli interfogliamenti equivalenti alle storie seriali. Il criterio di serializzabilità dice che una storia S è corretta se è equivalente a una qualsiasi storia seriale delle transazioni coinvolte da S. Date n transazioni ci possono essere n! storie seriali.

13.2.1 Meccanismo dei lock

Il meccanismo dei lock è un protocollo che evita a priori la non serializzabilità.

Le transazioni possiedono alcuni comandi per richiedere l'autorizzazione a compiere azioni sull'oggetto X[1]:

- LS(X): lock shared sull'oggetto X, da richiedere prima della lettura;
- LX(X): lock exclusive sull'oggetto X, da richiedere prima della scrittura;
- UN(X): unlock sull'oggetto X.

Di solito LS, LX e UN vengono invocati implicitamente dal gestore della concorrenza.

Lock shared: quando una transazione T_i vuole eseguire un'azione di lettura $r_i(X)$, prima di effettuare la lettura deve avere acquisito almeno il permesso per leggere l'oggetto X (lock shared). Il lock shared (condiviso) si chiama così perché transazioni diverse possono acquisire un lock condiviso sul medesimo oggetto.

Lock exclusive: la richiesta di lock exclusive è fatta da una transazione per modificare un oggetto X: questa richiesta deve sempre precedere una $w_i(X)$. Quando T_i ha acquisito l'autorizzazione esclusiva a scrivere l'oggetto X, nessuna altra transazione può acquisire lock (né LS né LX) sullo stesso oggetto.

[1] Assimilabile a una tupla

Unlock: quando una transazione non ha più bisogno di leggere o scrivere l'oggetto X, può rilasciare il lock (shared o exclusive) attraverso l'unlock.

Se il DBMS non può concedere il lock la transazione che lo ha richiesto entra in stato di wait. Tutti i lock sono memorizzati in una tabella dei lock.

Esiste una tabella di compatibilità che descrive il funzionamento del dbms in presenza di più lock.

Possesso/richiesta	**LS**	**LX**
LS	Concede	Nega
LX	Nega	Nega

È possibile aggiornare il lock da LS a LX se una stessa transazione ne fa richiesta ed è l'unica che possiede il LS.

Tuttavia il meccanismo dei lock, da solo, non è sufficiente a risolvere la concorrenza.

13.2.2 2PL

Il two-phase lock (2PL) divide il meccanismo dei lock in due fasi:

- fase di acquisizione dei lock;
- fase di rilascio dei lock.

Questo sistema garantisce la serializzabilità[2], però il protocollo del lock a due fasi è piuttosto rigido, infatti esistono storie serializzabili che non sono possibili con il lock a due fasi

13.2.3 Granularità del lock

L'attributo X può essere:

- una tupla (soluzione più diffusa);
- un attributo (aumenta il parallelismo, ma è più difficile da gestire);
- una pagina (facile da gestire, ma riduce il parallelismo);
- un file (usato nella gestione degli indici[3]).

[2] vedere le slide per esempi
[3] vedi 12.4

Capitolo 14

Teoria 14

14.1 Gestore del ripristino

Il gestore del ripristino si occupa dell'affidabilità dei dati (tranne nei casi di guasti hard) a seguito di guasti o anomalie.

Guasti soft (più comuni):

- Crash di sistema: guasto di natura hardware, software o di rete durante una transazione;
- Errori di programma o di sistema: una transazione viene interrotta da un utente o fallisce a causa di un errore o di un bug;
- Eccezioni gestite dalla transazione: non sono errori, perchè previsti dallo sviluppatore (esempio: saldo insufficiente in un conto corrente);
- Rollback di transazioni forzate dal gestore della transazione e dal serializzatore: per uscire da un deadlock.

Guasti hard (meno comuni):

- Guasti delle periferiche di storage;
- Eventi catastrofici: furtii, incendi, allagamenti, etc..

Il gestore del ripristino si occupa anche di atomicità e durabilità.

14.2 Log

Il file di log è un diario di bordo in cui il DBMS tiene traccia delle operazioni effettuate sulla BD tramite transazioni. Un file di log è sequenziale append-only[1] memorizzato su storage, quindi non può essere affetto dai guasti soft, ma solo dai guasti hard. Periodicamente ne viene fatto il backup. Esso rende possibile UNDO e REDO.

Quando si ha un guasto si annullano tutte le operazioni non committed e si ripristinano quelle committed.

14.2.1 Struttura del file di log

Per ogni transazione si tiene traccia dei comandi start transaction / commit transaction / abort transaction. Per ogni operazione (insert/delete/update) effettuata sul DB $< T_i, X, BS(X), AS(X) >$, che contiene:

- l'identificativo T_i della transazione;
- l'oggetto X su cui è stata eseguita l'operazione;
- before state (BS(X)), lo stato precedente a X, tranne in caso di insert;
- after state (AS(X)), lo stato successivo a X, tranne in caso di delete.

[1] Si va solo avanti con la scrittura

14.2.2 Undo e Redo

Le operazioni di Undo e Redo sono idempotenti[2]

Undo di una azione su x:

- update, delete: si scrive nell'oggetto il valore di BS;
- insert: si elimina X.

Redo di una azione su x:

- update, insert: si scrive nell'oggetto il valore di AS;
- delete: si elimina X.

Il processo di ripristino può fallire e in quel caso deve essere rieseguito.

Tutte le modifiche al file di log avvengono in memoria primaria e successivamente vengono portate in memoria secondaria. Questo causa alcuni problemi (esempio: una insert(X) viene scritta su storage ma avviene un crash prima che il log venga scritto su storage).

14.2.3 Regole fondamentali

- Write-Ahead Log: BS dei record di log deve essere scritto prima dei corrispondenti record della base di dati (garantisce UNDO);
- Commit-Precedenza: AS dei record di log deve essere scritto prima di effettuare il commit (garantisce REDO).

Scrittura dei record di commit: quando una transazione richiede il commit, il DBMS sceglie in modo atomico e indivisibile tra abort e commit e scrive sul log in modo sincrono/atomico (*primitiva force*) il record di commit. Se il guasto avviene prima del commit si procede con l'UNDO, se avviene dopo con il REDO.

Se T_i richiede un rollback, il DBMS impone un abort o la transazione richiede un commit e almeno un vincolo non è soddisfatto: il DBMS esegue UNDO(T_i) che annulla tutte le azioni di T_i. Al termine di questo processo viene memorizzato nel log il record < T_i, abort> seguito da **FORCE LOG** per forzare la scrittura del log su storage.

14.3 Ripristino

Per fare un REDO si esplora il file di log in avanti. Per fare un UNDO si esplora il file di log a ritroso.

14.3.1 Algoritmo di ripristino

Quando avviene un crash e il sistema riparte senza transazioni si esegue in ripristino:

1. AT := insieme delle transazioni non terminate;
2. CT := insieme delle transazioni che hanno raggiunto il commit;
3. UNDO(AT);
4. REDO(CT);

Nei grossi sistemi informativi, il file di log può contenere centinaia di migliaia di record e, se avvenisse un crash di sistema, il ripristino risulterebbe estremamente costoso. I crash possono avvenire in qualsiasi momento, anche in pieno orario di attività ed è necessario ridurre al minimo i tempi necessari al ripristino, perchè il ripristino richiede il blocco dell'attività transazionale.

[2] undo(undo(X)) = undo(X) e redo(redo(X)) = redo(X)

14.3.2 Checkpoint

Periodicamente avviene un processo di checkpoint:

1. si sospendono le transazioni;
2. si costruisce il record di checkpoint contenente l'elenco delle transazioni che in quel momento sono attive col relativo puntatore alla posizione dello start della transazione nel file di log;
3. si esegue un FORCE LOG;
4. si esegue un FORCE delle pagine delle transazioni committed;
5. si aggiunge un flag di OK nel record di checkpoint e si esegue un nuovo FORCE LOG;
6. si riavviano le transazioni.

Hot restart (ripresa a caldo):

1. trova l'ultimo checkpoint;
2. recupera la lista AT dellee transazioni ancora attive durante il crash;
3. recupera la lista CT delle transazioni che hanno raggiunto il commit dopo l'ultimo checkpoint;
4. UNDO(AT);
5. REDO(CT).

14.3.3 Ripristino dopo eventi catastrofici

In questo caso si deve fare riferimento a memorie secondarie stabili (esenti da guasti, ma sono impossibili). Per approssimare una memoria secondaria a una memoria stabile si può fare un backup con informazioni di dimensioni contenute tenuto al sicuro.

Il dump è una copia completa della BD salvata in memoria stabile. Quando si fa il dump si svuota il file di log e si aggiunge un record di dump.

Cold restart (ripresa a freddo):

1. si ripristinano i dati a partire dai backup (restore);
2. si legge tutto il log e si effettua il REDO(CT);
3. si effettua la hot restart.

Capitolo 15

Teoria 15

Questa lezione non fa parte del programma d'esame, ma può essere spunto per approfondimenti.

15.1 Database non relazionali

Negli ultimi anni si sono diffusi approcci NoSQL[1] (database non relazionali): database distribuiti (condivisi tra più computers) con dati semi-strutturati ad alte prestazioni, scalabili, disponibili e replicabili.

NoSQL ha applicazioni nel campo dei "big data": social, web link, post, tweet, email, etc.

15.1.1 Scalabilità

La scalabilità è la capacita di una macchina di crescere.

Scalabilità verticale: aumenta la potenza di calcolo di una macchina migliorandola (più RAM, CPU, storage).

Scalabilità orizzontale: aumenta la potenza di calcolo aggiungendo nuove macchine (i database relazionali hanno una limitata scalabilità orizzontale).

Commodity machines: sono computers poco potenti e identici che servono a favorire la scalabilità orizzontale (esempio: server di google).

15.2 Caratteristiche di NoSQL

Si evita:

- il costo delle proprietà ACID;
- la complessità delle query in SQL;
- la progettazione a priori dello schema;
- le transazioni.

Vengono permessi:

- cambiamenti facili e frequenti alla BD;
- uno sviluppo veloce;
- la gestione di grandi quantità di dati;
- dei database senza schema.

[1] Not only SQL

Quindi No SQL è adatto per log di dati e dati temporanei, ma non per dati finanziari o aziendali.
I database non relazionali si possono raggruppare in base al modello di dati utilizzato:

- chiave-valore: associano ogni chiave a un valore come fanno gli array associativi o le tabelle hash dei linguaggi di programmazione. Sono utili per fare cache in memoria principale per applicazioni web;
- documento: gestiscono enormi volumi di dati su server diversi (nodi). Hanno una grande scalabilità orizzontale;
- a colonna: un documento consiste in un ID associato a valori di vari tipi come hash. Possono contenere strutture annidate;
- a grafo: i data sono altamente interconnessi e nodi e archi possono avere proprietà.

15.3 Mongo

mongoDB è un database NoSQL orientato ai documenti. Il nome Mongo deriva da humongous (gigantesco). In mongo:

- i dati sono memorizzati come documenti, in formato JSON/BSON;
- i documenti non devono aderire ad uno schema standard, ma possono contenere qualsiasi campo;
- per le operazioni di ricerca, si recuperano i documenti basandosi sul valore di un determinato campo;
- il DB è scalabile orizzontalmente, supportando partizionamento (sharding) dei dati in sistemi distribuiti;
- esistono funzionalità per aggregazione e analisi dei dati.

15.4 Quando conviene usare NoSQL?

NoSQL è un'alternativa ai database relazionali.

Vantaggi:

- alte prestazioni;
- riduzione dei tempi di sviluppo;
- supporto alla scalabilità orizzontale.

Svantaggi:

- nessun supporto per join e transazioni;
- grande ridondanza dei dati;
- mancanza di un linguaggio standard;
- mancanza di vincoli di integrità.

Capitolo 16

Laboratorio 1

16.1 Introduzione alla progettazione

Il ciclo di vita di una base di dati è diviso in fasi: studio di fattibilità, raccolta e analisi dei requisiti, progettazione, implementazione, validazione e collaudo, funzionamento e manutenzione. In un progetto si devono minimizzare i costi e i tempi e massimizzare la qualità e la funzionalità. Solitamente, nel mondo reale, ci si deve accontentare di un compromesso.

La progettazione di un sistema informativo[1] riguarda due parti:

- dati : la parte stabile;
- funzionalità : la parte meno stabile.

16.2 Modello entity-relationship

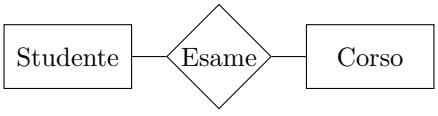
Il modello entity-relationship è il modello concettuale più diffuso per progettare database. Esso ***non*** modella il comportamento del sistema, ma modella i dati.

16.2.1 Entità

Le entità sono aspetti del mondo reale autonomi. Esse sono l'insieme di tutte le occorrenze dei dati, perciò. Per esempio "impiegato" è l'insieme di tutti gli impiegati. Le entità sono rappresentate da un rettangolo.

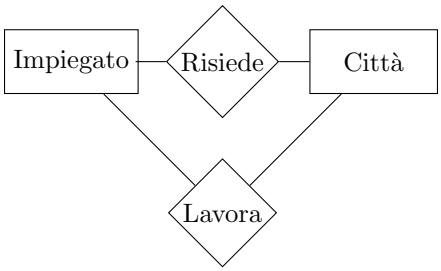
16.3 Associazioni

Le associazioni rappresentano legami logici tra due o più entità. Esse non hanno un verso di lettura e sono rappresentate da rombi.

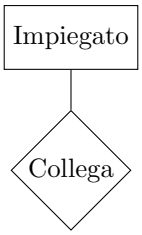


Si possono avere associazioni diverse tra stesse entità.

[1] vedi 1.1.1



Si possono avere associazioni ricorsive.



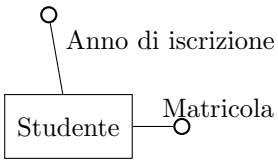
Le occorrenze di un'associazione sono le coppie o le triple delle occorrenze delle entità. La semantica delle associazioni ***non*** permette ripetizioni.

16.3.1 Cardinalità delle associazioni

La cardinalità delle associazioni descrive il numero minimo e massimo di possibili occorrenze dell'associazione a cui le occorrenze delle entità partecipano.

16.4 Attributi

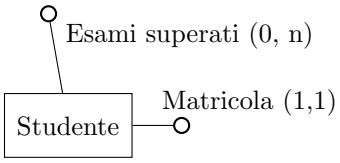
Gli attributi descrivono proprietà di entità o associazioni. Ogni attributo è caratterizzato da un dominio che comprende i valori ammissibili. Si possono avere attributi con lo stesso nome, ma devono essere legati a entità/associazioni diverse. Gli attributi sono rappresentati da pallini vuoti.



16.4.1 Cardinalità degli attributi

Anche gli attributi possono avere una cardianlità:

- 0, l'attributo è opzionale;
- 1, l'attributo è obbligatorio;
- n, l'attributo è multivalore.



16.4.2 Attributi composti

Gli attributi composti raggruppano attributi simili. Per esempio indirizzo può raggruppare via e numero civico.

Capitolo 17

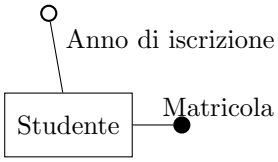
Laboratorio 2

17.1 Identificatori delle entità

Gli identificatori delle entità servono per identificarle univocamente. Possono essere:

- interni: sono costituiti dagli attributi delle entità;
- esterni: sono costituiti da attributi delle entità più entità esterne, tramite associazioni.

Gli identificatori sono rappresentati come pallini pieni.



Se sono necessari più attributi si rappresentano con una sbarra con pallino nero sopra gli attributi necessari.
Ogni entità deve avere almeno un identificatore e ogni attributo che fa parte di un identificatore deve avere cardinalità (1, 1).

17.2 Generalizzazione

La generalizzazione mette in relazione una o più entità E_1, E_2, ..., E_n con una entità E che le comprende come casi particolari:

- E è generalizzazione di E_1, E_2, ..., E_n;
- E_1, E_2, ..., E_n sono specializzazioni di E;

Ogni occorrenza di E_1, E_2, ..., E_n è anche occorrenza di E. Ogni proprietà di E è anche proprietà di E_1, E_2, ..., E_n.
Una generalizzazione può essere:

- totale se ogni occorrenza dell'entità genitore è occorrenza di almeno una delle entità figlie;
- parziale se non è totale;
- esclusiva se ogni occorrenza dell'entità genitore è occorrenza di al più una delle entità figlie;
- sovrapposta se non è esclusiva.

17.3 Documentazione schemi concettuali

- Descrizione di concetti: dizionari per entità e associazioni;
- Vincoli non esprimibili in ER: di integrità o di derivazione.

Capitolo 18

Laboratorio 3

18.1 Progettazione concettuale

L'analisi inizia con i primi requisiti raccolti (in linguaggio naturale). Le possibili fonti sono:

- utenti, attraverso documenti o interviste;
- documentazione già esistente, come normative o regolamenti interni;
- realizzazioni preesistenti.

18.1.1 Acquisizione tramite interviste

Utenti diversi possono fornire informazioni diverse (complementari o contradditorie). Gli utenti ad alto livello vedono il quadro generale, mentre gli utenti a basso livello vedono i dettagli.

18.1.2 Suggerimenti per la progettazione

Se un concetto ha proprietà significative e descrive oggetti con esistenza autonoma è un'entità.
Se un concetto è semplice e non ha proprietà è un attributo.
Se un concetto lega tra loro due o più concetti è un'associazione.
Se un concetto è un caso particolare di un altro concetto è una generalizzazione.

18.1.3 Requisiti (documentazione descrittiva)

Si deve scegliere il corretto livello di astrazione, standardizzare la struttura delle frasi ed evitare frasi contorte. Si devono unificare i termini eliminando omonimi[1] e sinonimi[2], rendendo espliciti i riferimenti tra i termini. Si deve costruire un glossario dei termini (tabella).

18.2 Pattern di progettazione

Sono soluzioni progettuali pronte per problemi comuni[3]:

- Reificazione di attributo di entità: è la trasformazione di un attributo in un'identità;
- Part-of: due casi, il primo nel quale una parte non può esistere senza l'intero e il secondo in cui la parte può esistere senza l'intero;
- Instance-of: rappresenta il concetto istanza-classe;
- Reificazione di un'associazione binaria: si trasforma l'associazione binaria in un'entità;

[1] Hanno lo stesso nome, ma si riferiscono a concetti diversi
[2] Hanno nomi diversi, ma si riferiscono allo stesso concetto
[3] Per la spiegazione in dettaglio si rimanda alle slide

- Reificazione di un'associazione ricorsiva: si trasforma l'associazione ricorsiva in un'entità;
- Reificazione di associazione ternaria: si trasforma l'associazione ternaria in un'entità;
- Reificazione di attributo di associazione;
- Caso particolare di un'entità: livelli diversi della generalizzazione partecipano ad associazioni diverse;
- Storicizzazione di un'entità: si usa la generalizzazione per rappresentare le informazioni correnti e contemporaneamente tenere traccia dello storico;
- Storicizzazione di un'associazione;
- Evoluzione di un concetto: si usa la generalizzazione per rappresentare l'evoluzione di un concetto mettendo nel genitore gli attributi e le associazioni comuni.

18.3 Strategie di progetto

Top-Down: si individuano i concetti più importanti e si procede per raffinamenti successivi. Essa è conveniente perchè permette di trascurare momentaneamente alcuni dettagli, ma la si può utilizzare solo quando si ha una visione generale del progetto.

Bottom-Up: le specifiche vengono divise in parti più semplici e poi unite alla fine. Questa strategia è adatta per i progetti di gruppo, ma l'integrazioni di varie parti può essere difficoltosa.

Inside-Out: è una variante della Bottom-Up in cui si parte dai concetti più importanti e ci si espande a macchia d'olio. Non richiede integrazione, ma è necessario rivisitare periodicamente i requisiti per essere certi di rappresentare tutti i concetti.

Mista: nella realtà si procede con una soluzione ibrida.

18.4 Qualità di uno schema concettuale

Correttezza: devono essere utilizzati propriamente i costrutti messi a disposizione dal modello concettuale di riferimento.

Completezza: deve modellare tutte le specifiche.

Leggibilità: deve poter essere compreso in maniera immediata.

Minimalità: le specifiche devono presentarsi una volta sola.

Capitolo 19

Laboratorio 4

19.1 Progettazione logica

La progettazione logica è la fase successiva alla progettazione concettuale.

Dati in ingresso:

- schema concettuale;
- informazioni sul carico applicativo;
- modello logico che si vuole adottare.

Dati in uscita:

- schema logico;
- vincoli di integrità;
- Documentazione associata.

Si può dividere in due sottofasi: la ristrutturazione dello schema concettuale (ER) e la traduzione verso il modello logico con relative ottimizzazioni.

19.2 Ristrutturazione dello schema ER

La ristrutturazione dello schema ER serve a semplificare la traduzione nel modello logico e a ottimizzare le prestazioni. Si usano due indicatori per tenere traccia delle prestazioni: il tempo (numero di occorrenze visitate per un'operazione nel DB) e lo spazio (quantità di memoria per rappresentare i dati). Per poter valutare questi parametri si ha bisogno di alcune informazioni: il volume dei dati (numero di occorrenze e dimensione degli attributi) e le caratteristiche delle operazioni (se interattive o di batch, la frequenza e il numero di entità/associazioni coinvolte). Queste informazioni vengono rappresentate su apposite tavole.

Per la ristrutturazione dello schema concettuale si possono seguire i seguenti passi.

19.2.1 Analisi delle ridondanze

Una ridondanza è un'informazione significativa, ma ricavabile da altre informazioni. Si deve decidere se eliminare le ridondanze o mantenerle, con un calcolo. Le ridondanze rendono più efficienti le operazioni di interrogazione/lettura dei dati, ma rendono meno efficiente l'inserimento e la modifica dei dati, inoltre occupano spazio in memoria.

19.2.2 Eliminazione delle generalizzazioni

Le generalizzazioni non sono rappresentabili nel modello relazionale per cui vanno sostituite con entità, associazioni o regole aziendali (*business rules*). In generale si hanno tre modi per eliminare una generalizzazione:

- si accorpano i figli della generalizzazione nel genitore ("uccidendo" le entità figlie);

- si accorpa il genitore della generalizzazione nei figli ("uccidendo" l'entità genitore);
- si sostituisce la generalizzazione con associazioni, aggiungendo eventuali business rules.

Si possono anche adottare soluzioni ibride.

19.2.3 Partizionamento di concetti

Si separano attributi di uno stesso concetto ai quali si accede in operazioni diverse e si accorpando attributi di concetti diversi a cui si accede con le medesime operazioni. Spesso è possibile rimandare questo problema alla fase di progettazione fisica (che non è argomento di questo corso).

19.2.4 Scelta degli identificatori principali

Si devono scegliere gli identificatori che diventeranno chiave primaria seguendo alcuni criteri:

- assenza di opzionalità;
- semplicità;
- utilizzo nelle operazioni più frequenti/importanti.

Se nessun identificatore rispetta questi criteri si devono introdurre nuovi attributi (per esempio codici) che serviranno da chiave primaria.

19.2.5 Attributi composti e attributi multivalore

Si possono trasformare gli attributi multivalore, reificando l'attributo e aggiungendo un'associazione. Gli attributi composti non sono rappresentabili direttamente in relazionale e devono essere trasformati.

19.3 Traduzione verso il modello relazionale

Le idee di base sono due:

- le entità diventano relazioni con gli stessi attributi delle entità;
- Le associazioni diventano relazioni con attributi delle associazioni + gli identificatori delle entità coinvolte

Si devono dare nomi più espressivi agli attributi della chiave della relazione che rappresenta l'associazione.
La traduzione non riesce a tener conto delle cardinalità minime delle associazioni molti a molti. La traduzione dell'associazione uno a molti riesce a rappresentare efficacemente la cardinalità minima della partecipazione che ha 1 come cardinalità massima (se è 0 ammette valori nulli, se è uno non ammette valori nulli). L'identificazione esterna è sempre su un'associazione uno a molti o un'associazione uno a uno.

Capitolo 20

Appendice - SQL

In questo capitolo verranno messi tutti i concetti della parte di laboratorio su SQL. Tutte le nozioni di seguito sono applicabili a PostgreSQL, ma non tutte sono applicabili ad altri fornitori (es. Oracle).

SQL come linguaggio dichiarativo. In SQL l'utente deve definire *cosa* vuole ottenere e il DBMS si occupa del *come* ottenerlo. Una query SQL viene analizzata da un ottimizzatore del DBMS e trasformata per essere eseguita in modo efficiente (ragion per cui non ci si preoccuperà troppo dell'efficienza della scrittura delle query).

20.1 Sintassi

La struttura essenziale è la seguente:

```
1 SELECT ListaAttributi
2
3 [FROM ListaTabelle]
4
5 [WHERE Condizione];
```

Listing 20.1: Struttura base di una query

Questo comando restituisce una tabella con gli attributi di *ListaAttributi* del prodotto cartesiano delle tabelle in *ListaTabelle* che rispettino la *Condizione* (che è opzionale). Nella clausola select si possono scrivere espressioni riguardo qualsiasi attributo. Di seguito alcuni esempi di comandi con relative spiegazioni.

Esempio semplice: Dà come risultato una tabella con tutti gli attributi di *Tabella* utilizzando le sue tuple.

```
1 SELECT *
2
3 FROM Tabella;
```

Listing 20.2: Semplice query

Esempio selezione: Dà come risultato una tabella con tutti gli attributi dopo aver applicato una selezione[1] in base all'attributo A3.

```
1 SELECT *
2
3 FROM Tabella
4
5 WHERE A3 <= 20;
```

Listing 20.3: Semplice selezione

[1] vedi 3.2.1

Esempio proiezione: Dà come risultato una tabella con le stesse tuple di *Tabella* a cui è stata apllicata una proiezione[2] su A1 e A2.

```
SELECT A1, A2

FROM Tabella;
```

Listing 20.4: Semplice proiezione

20.2 Definizione di Dati

Ci sono una serie di domini per definire i dati, inoltre è possibile definire dei propri domini (semplici, solamente a scopo di astrazione).

Stringhe.

- lunghezza fissa: char(length). Per esempio char(20) è una stringa lunga ***esattamente*** 20 caratteri;
- lunghezza variabile: varchar (max_length). varchar(20) è una stringa lunga ***al massimo*** 20 caratteri.

Numeri esatti.

- numeri decimali: decimal(precisione, scala)/numeric(precisione, scala). La precisione è il numero totale di cifre decimali, la scala è il numero di cifre decimali dopo la virgola (sono entrambe opzionali);
- numeri interi: smallint (almeno 16 bit, con segno)/integer (almeno 32 bit, con segno)/bigint (almeno 64 bit, con segno).

Numeri approssimati.

- float(precisione). La precisione è il numero di cifre binarie per la mantissa (opzionale);
- real/double precision.

Istanti temporali.

- date: si usa *date* che comprende year, month, day;
- orari: si usa *time*(precisione) che comprende hour, minute, second;
- Per specificare data e orario si usa *timestamp* (precisione) che comprende year, month, day, hour, minute, second.

Intervalli.

- durate in anni: interval year;
- durate in anni e mesi: interval year to month;
- durate in giorni, ore, minuti, secondi e centesimi di secondo: interval day to second (2),
- alcune durate in cui non è possibile passare in modo preciso non sono permesse (per esempio non si può passare da mesi a giorni perchè i mesi hanno durate diverse).

Ulteriori domini.

- vero/falso: boolean;
- immagini, video, file: blob;
- lunghi file di testo: clob.

[2] vedi 3.2.2

20.3 Definizione di tabelle

La sintassi per la creazione di una tabella è la seguente;

```
CREATE TABLE NomeTabella (

    NomeAttributo1 Dominio1 [ValoreDefault1][Vincoli1]

    NomeAttributo2 Dominio2 [ValoreDefault2][Vincoli2]

    ...

    NomeAttributoN DominioN [ValoreDefaultN][VincoliN]

);
```

Listing 20.5: Creazione tabella

Il ValoreDefault è un valore opzionale che verrà usato, se nel momento dell'inserimento, di una tupla non viene specificato alcun valore. Se non viene scelto un ValoreDefault si utilizza NULL.

Alcuni vincoli sono particolari:

- **not null**: indica che il valore NULL non è ammesso (l'attributo è obbligatorio). Se l'attributo non viene specificato in fase di inserimento (e non è presente un ValoreDefault) l'operazione è annullata;
- **unique**: il valore di un attributo o di una superchiave deve essere unico. Eccezzione per il valore NULL. Sintassi su più attributi: unique (A1, A2,...,AN);
- **primary key**: un attributo o un insieme di attributi che fungono da chiave primaria. Questi attributi non possono essere NULL e può esserci solo un vincolo primary key per tabella;
- **Vincolo di integrità referenziale**: crea un vincolo tra i valori di uno (o più) attributi della tabella (interna) su cui è definito e uno (o più) attributi di un'altra tabella (esterna).

Sintassi del vincolo di integrità referenziale:

```
A1, Dominio1,

A2, Dominio2,

...

foreign key (A1, A2,...)

references NomeTabellaEsterna (B1, B2,...)
```

Listing 20.6: Integrità referenziale

Ci sono casi in cui viene violata l'integrità (inserimento, modifica, cancellazione). Alcuni casi rifiutano l'operazione (se cambia la tabella interna), altri casi prevedono varie possibilità di reazione:

- **cascade**: il nuovo valore dell'attributo della tabella esterna viene riportato in tutte le corrispondenti righe della tabella interna (in caso di modifica) o tutte le righe della tabella interna corrispondenti alla riga cancellata nella tabella esterna vengono cancellate (in caso di cancellazione);
- **set null**: all'attributo referente della tabella interna viene assegnato il valore NULL;
- **set default**: all'attributo referente della tabella interna viene assegnato il valore di default;
- **no action**: la modifica/cancellazione è rifiutata.

Sintassi per la modifica (subito dopo il vincolo): si usa update.

```
ON UPDATE Reazione
```

Sintassi per la cancellazione (subito dopo il vincolo): si usa delete

```
ON DELETE Reazione
```

Inoltre, si può dare il nome a un vincolo (per i dettagli sui vincoli vedere 2.4):

```
CONSTRAIN NomeVincolo DefinizioneVincolo
```

In SQL è possibile modificare definizioni di tabelle, domini e vincoli introdotti precedentemente usando i comandi **alter** (modifica), **drop** (cancellazione), **add** (aggiunta). Per inserire nuove righe in una tabella si usa il comando **insert**.

```
INSERT INTO Tabella(A1, ..., AN) values
        (ValoreAttributo1, ..., ValoreAttributoN);
```

Listing 20.7: Inserire nuove righe in una tabella

Se un attributo non viene inserito, non ha specificato un ValoreDefault e non è *nullable* l'operazione viene rifiutata. Se si specificano i valori per tutte le colonne, la lista di attributi può essere omessa. Per cancellare (con condizione) delle righe si usa il comando **delete**.

```
DELETE FROM Tabella WHERE Condizione
```

Listing 20.8: Cancellare righe in una tabella

La clausola where. La clausola where è costituita da un'espressione boolean in logica proposizionale.

Per confrontare stringhe è possibile usare l'operatore **like**, avendo accesso a caratteri speciali:

- _ (trattino basso): un carattere qualsiasi;
- % (percentuale): una sequenza di caratteri qualsiasi maggiore o uguale a zero;

Per esempio: _%@gmail._% può essere usato per identificare un indirizzo mail con almeno un carattere precedente @ e con un dominio di almeno 2 caratteri.

Ordinamento dei risultati. Per ordinare i risultati di una query si usa la clausola **order by**. Specificando più attributi, a parità di primo attributo, vengono ordinate con il secondo attributo e così via. Si possono aggiungere asc (crescente)e desc (decrescente). Esempio:

```
ORDER BY A1 [asc / desc],
A2 [asc / desc], ...
```

Listing 20.9: Ordinare i risultati in una query

Le righe con valore NULL sono messe tutte insieme (o all'inizio o alla fine).

Gestione dei valori nulli. Un attributo NULL è diverso da 0, stringa vuota o *blank*. Un confronto con il valore NULL dà sempre Unknow nella logica a tre valori[3]. Per cui si utilizzano i predicati **is null** e **is not null**. In alcuni casi è meglio trttare i valori NULL com 0, per cui si usa la funzione **coalesce**. Esempio:

```
SELECT A1, COALESCE (A2, 0)
FROM A;
```

Listing 20.10: I valori nulli

[3] vedi 4.5

Commenti. I commenti su una riga iniziano con – (doppio trattino). Per i commenti su più righe si usa /* commento */.

20.4 DML: tipi di Join

Per eseguire interrogazioni che riguardano più tabelle si deve usare il join[4]. La congiunzione avviene sui valori in comune tra le tabelle. Con il join, in SQL si possono avere righe duplicate (al contrario dell'algebra relazionale che prevede l'unicità delle tuple). Per evitare ciò si deve ricorrere alla parola chiave **distinct** subito dopo **select**. Il join inizia eseguendo il prodotto cartesiano sulle tabelle nella clausola **from** e vengono eliminate quelle che non rispettano la clausola **where**. Ci sono tre modi per usare un join in SQL di seguito elencati tramite esempi.

Join e clausola from.

```
SELECT A1, A2, B1, C1, C2

FROM A, B, C;
```

Join e clausola where.

```
SELECT A1, A2, B1, C1, C2

FROM A, B, C

WHERE (A1 = B1 OR A1 = C1) AND A2 <= C2;
```

Join come parola riservata.

```
SELECT *

FROM A [INNER] JOIN B ON A1 = B2

WHERE A3 = 'IT'  AND A2 = 25;
```

Listing 20.11: Join

La parola inner è opzionale, inoltre è possibile mettere in join più tabelle. Si possono anche effettuare join esterni mettendo prima di join left, right o full.

Si può effettuare il join di una tabella con sè stessa (self join), ricordandosi di specificare degli alias per gli attributi. Esempio:

```
\begin{center}
SELECT ...

FROM A JOIN A_1 ON A.A1 = A_1.A1

A.A2 <= A_1.A2 ...
```

Listing 20.12: Self-Join

[4] vedi 3.3 e 4.4

20.5 DML: funzioni aggregate

In SQL è possibile valutare proprietà e condizioni su gruppi di righe. Le funzioni aggregate prendono in considerazione gruppi di righe e danno come risultato un unico valore per gruppo. Non è possibile usare funzioni aggregate (che considerano gruppi di righe) direttamente nella clausola where (che viene valutata per ogni riga). Non è possibile combinare funzioni aggregate (che restituiscono un unico valore) con attributi non aggregati che possono assumere valori diversi nello stesso gruppo. Di solito si usano conteggio, somma, massimo e minimo.

count(*) conta il numero di righe (compresi valori NULL e valori duplicati). Si può specificare un nome per la colonna di questa interrogazione. Esempio:

```
SELECT COUNT(*) Nome

FROM A

WHERE A1 = 'UK';
```

Listing 20.13: Count

count(Attr) conta il numero di valori non NULL per l'attributo Attr (si può usare il distinct per eliminare i duplicati). PostgreSQL non supporta count(distinct *).

Altre funzioni.

- sum(Attr): somma dei valori nella colonna Attr;
- avg(Attr): media dei valori nella colonna Attr;
- max(Attr): massimo dei valori nella colonna Attr;
- min(Attr): minimo dei valori nella colonna Attr.

Query con raggruppamento. È possibile suddividere le righe di una tabella in più gruppi e applicare la funzione aggregata a ogni gruppo. È possibile considerare più di un attributo per formare raggruppamenti più specializzati. Per fare ciò si scrive al fondo **group by** e i nomi degli attributi, rispettando la seguente sintassi.

```
SELECT SottinsiemeAttributiDiscriminanti,

FunzioneAggregata(AltroAttributo)

FROM Tabelle

WHERE Condizione

GROUP BY AttributiDiscriminanti;
```

Listing 20.14: Query con raggruppamento

Clausola having. Nella clausola **having** vanno messe solo condizioni in cui compaiono funzioni aggregate (essa viene scritta dopo il **group by**. Esempio:

```
SELECT A1

FROM A

WHERE A2>=20

GROUP BY A1

HAVING COUNT(*) >= 2;
```

Listing 20.15: Having

20.6 DML: operatori insiemistici

SQL mette a disposizione gli operatori insiemistici union, intersect, except corrispondenti agli operatori di unione, intersezione e differenza dell'algebra relazionale[5]. Questi operatori rimuovono i duplicati. Se si vogliono mantenere si deve aggiungere la parola chiave **all**.

Sintassi unione.

```
SELECT EspressioneListaAttributi1

FROM ListaTabelle1

WHERE Condizioni1

UNION [ALL]

SELECT EspressioneListaAttributi2

FROM ListaTabelle2

WHERE Condizioni2;
```

Listing 20.16: Union

Sintassi intersezione.

```
SELECT EspressioneListaAttributi1

FROM ListaTabelle1

WHERE Condizioni1

INTERSECT [ALL]

SELECT EspressioneListaAttributi2

FROM ListaTabelle2

WHERE Condizioni2;
```

Listing 20.17: Intersect

Sintassi differenza.

```
SELECT EspressioneListaAttributi1

FROM ListaTabelle1

WHERE Condizioni1

EXCEPT [ALL]

SELECT EspressioneListaAttributi2

```

[5] vedi 3.2.3

```
11 FROM ListaTabelle2
12
13 WHERE Condizioni2;
```

Listing 20.18: Except

20.7 DML: query nidificate

In SQL si possono annidare sottoquery in query, ciò è utile per risolvere sottoproblemi. Esistono due tipologie di sottoquery:

- semplici (o stratificate): è possibile valutare prima l'interrogazione più interna (una volta per tutte), poi, sulla base del suo risultato, valutare l'interrogazione più esterna;
- correlate (o incrociate): se l'interrogazione più interna fa riferimento a una delle tabelle appartenenti all'interrogazione più esterna. Per ciascuna riga candidata alla selezione nell'interrogazione più esterna, è necessario valutare nuovamente la sottoquery.

Se la sottointerrogazione restituisce più di un valore, per effettuare il confronto è necessario utilizzare uno dei quantificatori seguenti:

- **any**: il predicato deve essere soddisfatto da almeno una riga restituita dalla sottointerrogazione;
- **all**: il predicato deve essere soddisfatto da tutte le righe restituite dalla sottointerrogazione, ma non viene quasi mai utilizzato perchè in presenza di valori NULL restituisce una query vuota.

IN e NOT IN. Per verificare che un valore sia presente o meno in un insieme si usano le condizioni **in** e **not in**.

Sottointerrogazioni correlate. SQL permette all'interrogazione annidata di fare riferimento al contesto relativo all'interrogazione più esterna (*passaggio di binding*), permettendo di valutare un'espressione di una riga esaminata dalla query più esterna. Bisogna che gli attributi in query distinte siano visibili (dichiarati in select). In queste sottointerrogazioni si possono usare i costrutti **exist** (la riga in esame nella query più esterna soddisfa il predicato exists se la query annidata non restituisce l'insieme vuoto) e **not exist** (la riga in esame nella query più esterna soddisfa il predicato not exists se la query annidata restituisce l'insieme vuoto).