Teoria di basi di dati

Umberto Domenico Ciccia

December 2023

Contents

Chapter 1

Progettazione di una base di dati

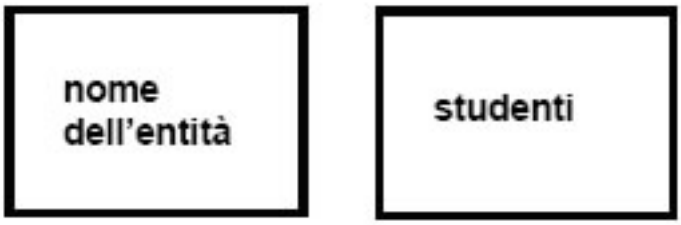
1.1 Progettazione concettuale

La progettazione concettuale è la fase di progettazione di una base di dati in cui prendiamo in **input** i **requisiti funzionali** dell'utente e restituiamo in **output** uno **schema** detto **entità/relazione** che rappresenta appunto entità e relazioni tra esse. Di seguito fornirò le informazioni riguardo i costrutti piu importanti di un **modello entità relazione**.

1.1.1 Entità

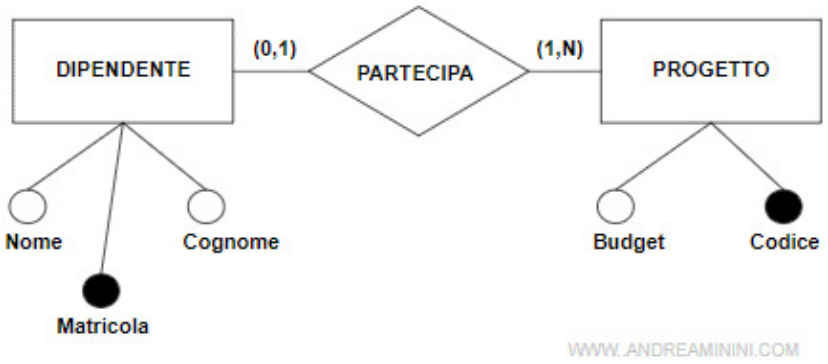
Un entità **rappresenta** un **concetto** a se stante. Ogni entità **deve avere** almeno una **chiave**. Può essere visto come il concetto di insieme.

esempio:



1.1.2 Relazione

Una relazione tra due o piu entità rappresenta il concetto matematico di **relazione** tra **due** o **piu insieme**. Una **relazione** tra due insiemi è **definita** dal **prodotto cartesiano tra** i **due**, ovvero l'insieme di tutte le possibili coppie di elemento e1 ed e2 tale che e1 appartenga al primo insieme ed e2 al secondo.



1.1.3 Attributi

Gli attributi rappresentano le **caratteristiche** di un'**entità o** di una **relazione**. **Non possono essere utilizzati** per **rappresentare** concetti come **liste**.

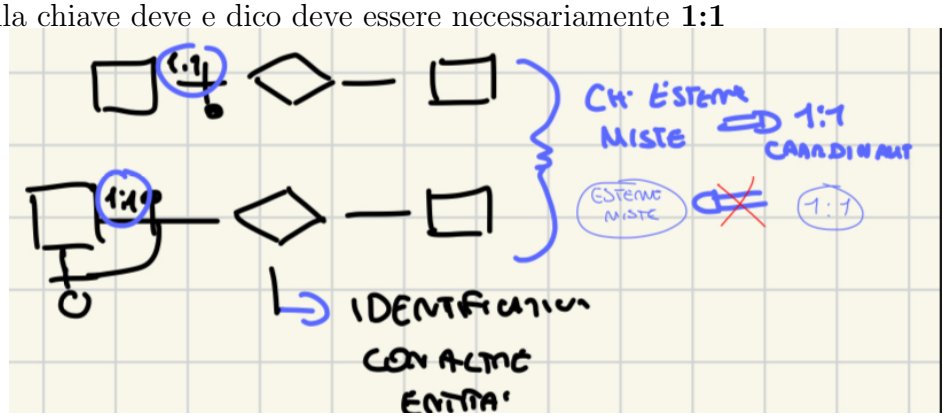
1.1.4 Cardinalità

La cardinalità è un concetto associa alle relazioni. Rappresenta il **numero** di **occorrenze** di **entità** che **possono partecipare** ad una **relazione**. Va sempre definita per ogni relazione ambo i lati. Guardando l'esempio superiore possiamo notare come un dipendente può partecipare ad un solo progetto mentre ad un progetto possono partecipare da 1 a n dipendenti.

1.1.5 Chiave

Il concetto di **chiave** è uno dei **concetti fondamentali** dello schema entità relazione. Una chiave può essere un attributo,un concetto, un insieme di

attributi o un insieme di concetti o entrambi. Una **chiave** è **definita** come un'**insieme** di **concetti** (attributi o entità) che **identificano** un **entità** ed è **minimale rispetto** alla **proprietà** di **essere chiave**. Essere minimale significa che non esiste un sottoinsieme di concetti della chiave che è anche essa chiave. In caso di **chiave mista** o **esterna** la **molteplicità** dal lato della chiave deve e dico deve essere necessariamente **1:1**



Chiave interna

Una chiave si definisce interna quando l'entità è identificata da uno o piu attributi interni all'entità.

Chiave esterna

Una chiave esterna quando l'entità è identificata dalla seconda, con la quale prima è in relazione.

Chiave mista

Una chiave si definisce mista quando l'entità è identificata da uno o piu attributi e da una relazione esterna.

1.1.6 Generalizzazione

Una **generalizzazione** è un **legame** fra **due entità**, una **genitore** ed una **figlio**. Tutto **ciò** che **appartiene** al **padre** viene **ereditata** dal **figlio**. Inoltre **ogni occorrenza** del **figlio** è anche **occorrenza** del **genitore**. Le caratteristiche di una generalizzazione sono essenzialmente due:

- Totale/Parziale
  - **Totale**: **Ogni occorrenza** del **genitore** deve essere **necessariamente occorrenza** di **uno** dei suoi due **figli**. Supponi di avere una generalizzazione persona padre, studente e lavoratore figli. Ciò significa che una persona deve essere necessariamente studente o lavoratore.
  - Parziale: In questo caso invece seguendo l'esempio precedente una persona non deve necessariamente essere o studente o lavoratore
- Esclusiva/Inclusiva
  - Esclusiva: Ogni **occorrenza** del **genitore deve essere occorrenza** di **un solo figlio**. Uno persona o lavora o è studente ma non entrambi.
  - Inclusiva: Ogni occorrenza del genitore può essere occorrenza di piu figli contemporaneamente. Uno persona o lavora o è studente o entrambi.

1.2 Progettazione logica

La fase di progettazione logica prende in **input** il **modello e/r** e restituisce in **output** un **modello** detto **relazionale**.

1.2.1 Schema di relazione

Uno **schema** di **relazione o relazione** è **definito** dal **nome** della **relazione** e dall'**insieme** dei suoi **attributi**. L'attributo sottolineato identifica l'attributo chiave. R(attributo1,attributo2,...,attributon).

1.2.2 Schema di database

Uno **schema** di **database** è definito come l'**insieme** di **tutti** gli **schemi** di **relazione**.

1.2.3 Istanza di relazione

Un'**istanza** di **relazione** è **definita** come l'**insieme** di **tutte** le **tuple** di uno **schema** di **relazione**. Una **tupla** è una **funzione** che **associa** ad **ogni attributo** di una **relazione** un **valore appartenente** al **dominio** dell'**attributo**.

1.2.4 Istanza di database

Un **istanza** di **database** è definita come l'**insieme** di **tutte** le **istanze** di **relazione** di tutte le relazioni **facenti parte** dello **schema** di **database**.

1.2.5 Vincolo di tupla

Vincola i valori possibili di una tupla ad una certa condizione logica. Esempio data la seguente relazione Studente(Matricola, voto, nome). Voto ≥ 18 and Voto ≤ 30

1.2.6 Chiave

Definizione formale di chiave di una relazione: Un'**insieme** di **attributi K** di una **relazione** si **definisce chiave** se:

- L'**insieme K** è **super chiave**.
- L'**insieme K** è **minimale rispetto** alla **proprietà** di **essere super chiave**.

Definizione di **super chiave**: Un **insieme K** di **attributi** è **super chiave se non esistono due tuple ti** e **tj** dell'**istanza** di **relazione** che **abbiano** lo **stesso valore** per questo **insieme** di **attributi K**. Quindi $ti[k] \neq tj[k]$ $\forall\ ti, tj\ in\ R$

1.2.7 Chiave esterna

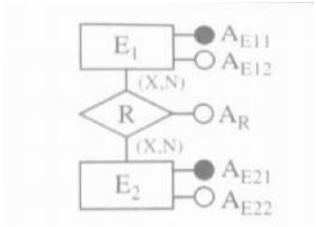
Un **vincolo** di **chiave esterna** è un **vincolo** tra un **insieme X** di **attributi** di una relazione **R1** con una **relazione R2**. **Per ogni istanza** di **R1** il **valore** dei **valori** presenti in **X** in **ogni tupla deve essere presente** come **valore chiave primaria** nella **istanza** della seconda relazione **R2**. Nel caso

in cui gli attributi in cui è definita la chiave esterna siano piu di uno bisogna definire un' ordinamento nell'insieme X e nel secondo insieme Y di R2. Ad ogni valore del primo insieme deve corrisponderne in ordine uno nel secondo.

1.2.8 Traduzione di relazioni

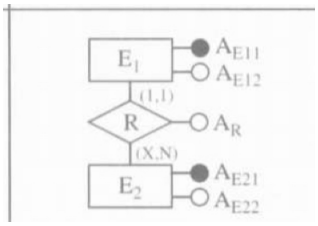
Le chiavi esterne verranno omesse. * significa che puo essere null.

Molti a molti



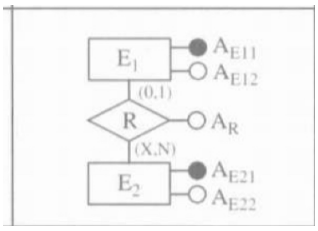
E1(<u>Ae11</u>,Ae12)
E2(<u>Ae21</u>,Ae22)
R(<u>Ae11,Ae21</u>,Ar)

Uno a molti obbligatorio



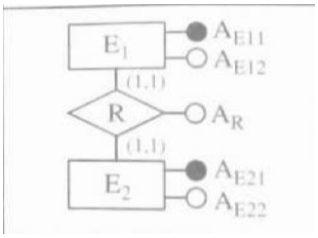
E1(<u>Ae11</u>,Ae12,Ae21,Ar)
E2(<u>Ae21</u>,Ae22)

Uno a molti opzionale



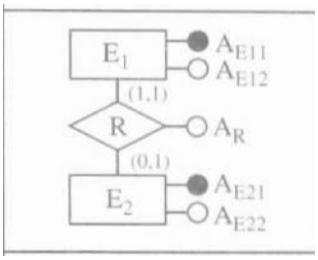
E1(<u>Ae11</u>,Ae12,Ae21*,Ar*)
E2(<u>Ae21</u>,Ae22)

Uno a uno obbligatorio



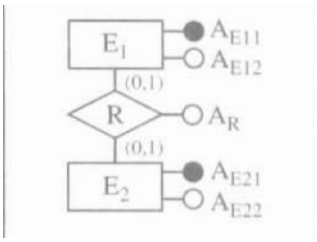
E1(<u>Ae11</u>,Ae12,Ae21,Ar)
E2(<u>Ae21</u>,Ae22)
o accorpando a e2

Uno a uno opzionale singolo



E1(<u>Ae11</u>,Ae12,Ae21,Ar)
E2(<u>Ae21</u>,Ae22)
Sempre accorpare dal lato 1:1

Uno a uno opzionale doppio



E1(Ae11,Ae12,Ae21*,Ar*)
E2(Ae21,Ae22)
o viceversa

1.2.9 Traduzione di generalizzazioni

Per tradurre le generalizzazioni abbiamo tre metodi possibili.

Accorpamento figli nel genitore

Accorpando i figli nel genitore, ad esso andrà aggiunto un attributo che identifica di quale istanza figlio era il genitore. Gli attributi dei figli potranno essere null nella nuova relazione

Accorpamento genitore nei figli

Accorpando i genitori nei figli tutto ciò che aveva il padre va ai figli.

Sostituzione generalizzazione con associazione

Questo ultimo metodo consiste nel trasformare la generalizzazione in una relazione in cui abbiamo 0:1 dal lato del padre e 1:1 con chiave totalmente esterna dal lato del figlio.

1.3 Progettazione fisica

La progettazione fisica prende in input lo schema di database e da in output lo schema fisico. La progettazione fisica verrà trattata nel Capitolo 4

Chapter 2

Algebra relazionale

In questo capitolo verrà trattata l'algebra relazionale e i suoi operatori principali.

2.1 Proprietà degli operatori

Di seguito sono riportate le due principali proprietà degli operatori.

- Monatico: Operatore con un solo argomento
- Rientrante: Operatore che restituisce un risultato che è dello stesso tipo dell'input. Prende relazione restituisce relazione

2.2 Selezione

L'operatore di Selezione dato come argomento una relazione e una funzione f restituisce una nuova relazione in cui tutte le tuple della nuova relazione soffisfano la funzione f.
Selezione: $\sigma_{\text{eta}>21}(\text{Persona})$

2.3 Proiezione

L'operatore di proiezione dato come argomento una relazione e una funzione f restituisce una nuova relazione proiettata solo negli argomenti della funzione

f.
Proiezione: $\pi_{\text{Nome, Eta}}$(Persona)

2.4 Join

L'operatore di join filtra il prodotto cartesiano fra due relazione secondo una condizione data in input. Join: $R \bowtie_{R.id=D.id} S$

2.5 Ridenominazione

Ridenominazione: $\rho_{\text{OriginalName- NewName}}$(Relation)

2.6 Operazioni insiemistiche

Le operazioni insiemistiche di unione differenza intersezione e prodotto cartesiano sono supportate essendo intrinsecamente le relazioni degli insiemi.

Chapter 3

MySql

3.1 Definire schema di database

```
CREATE DATABASE nomeDatabase;
```

3.2 Definire una relazione

```
CREATE TABLE nome as (
  attributo1 datatype vincolo,
  attributo2 datatype vincolo
);
```

3.3 Inserire tuple

Listing 3.1: Inserire tuple con valori in colonne specifiche

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

Listing 3.2: Inserire tuple con tutti i valori

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

3.4 Eliminare tupla

```
DELETE FROM table_name WHERE condition;
```

3.5 Select

L'operatore select lavora allo stesso modo dell'operatore di proiezione. Nella from si specifica da quale tabella prendere il risultato.

```
SELECT CustomerName, City
FROM Customers;
```

Con select distinct elimino i duplicati.

```
SELECT DISTINCT CustomerName, City
FROM Customers;
```

3.6 Where

L'operatore where estrapola le tuple dalla tabella inserita nella from in cui vale la condizione inserita nella where.

```
SELECT C1.CustomerName, C1.City
FROM Customers
WHERE Customers.City="Roma";
```

3.7 Join

L'operatore lavora allo stesso modo della join dell'algebra relazionale

```
SELECT C1.CustomerName
FROM Customers C1, City Ci
WHERE C1.city=C2.cod;
```

3.8 Viste

Le viste permettono di assegnare a un risultato di una query intermedia ad una tabella.

```
CREATE VIEW view_name (column1, column2, ...)AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

3.9 In Not in

L'operatore **In** permette di **verificare** se gli **attributi** presenti in una **colonna sono** in un'**insieme** di **attributi dati** in **input** o nel **risultato** di una **sotto query data** in **input**.

```
SELECT *
FROM Customers
WHERE Country IN ('Germany', 'France', 'UK');;
```

```
SELECT *
FROM Customers
WHERE CustomerID IN (SELECT Orders.CustomerID
                     FROM Orders);
```

L'operatore Not in invece verifica se non sono presenti.

3.10 Exists Not exists

L'**operatore EXISTS restituisce true** se **esiste almeno** una **tupla** nella **sotto query** data come **argomento**. L'operatore **NOT EXISTS** restituisce **true se** non **esiste** una **tupla** nella **sotto query** data come **argomento**.

```
SELECT SupplierName
FROM Suppliers
WHERE EXISTS (SELECT ProductName
              FROM Products
              WHERE Products.SupplierID =
                 Suppliers.supplierID AND Price = 22);
```

3.11 Aggregati

Gli **operatori aggregati** permettono di **eseguire operazioni** come ricerca del **massimo** il **minimo** il **medio** eccetera. Ricorda se in una **select inserisci aggregati** la **select** potrà **contenere solo aggregati**.

Avg

Restituisce la media di tutti i valori presenti nella colonna data come argomento.

```
1 SELECT AVG(Price)
2 FROM Products;
```

Max

Restituisce il massimo tra tutti i valori presenti nella colonna data come argomento.

```
1 SELECT Max(Price)
2 FROM Products;
```

Min

Restituisce il minimo tra tutti i valori presenti nella colonna data come argomento.

```
1 SELECT MIN(Price)
2 FROM Products;
```

Sum

Restituisce la somma di tutti i valori presenti nella colonna data come argomento.

```
1 SELECT SUM(Price)
2 FROM Products;
```

Count

Restituisce il numero di valori presenti nella colonna data come argomento. Usa count (DISTINCT attributo) se non vuoi considerare le colonne che hanno gli stessi valori.

```
SELECT Count(Price)
FROM Products;
```

3.12 Group by

La group by **raggruppa** assieme le **colonne** che hanno gli **stessi valori** nella **colonna** passata come **argomento**. Sono spesso associate per raggruppare risultati di un aggregato. Se nella **select inserisco attributi** questi **devono** far **parte** della **group by**

```
SELECT SUM(CustomerID), Country
FROM Customers
GROUP BY Country;
```

3.13 Having

Utilizzata come una **where** però **per** gli **aggregati**. Va sempre in coppia con la group by.

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
```

3.14 Order by

Clausola utilizzata per **ordinare** il **risultato** di una query in **ordine crescente** o **decrescente**. ASC crescente DESC discendente.

```
SELECT *
FROM Products
ORDER BY Price ASC;
```

Chapter 4

Organizzazione delle chiavi in memoria secondaria

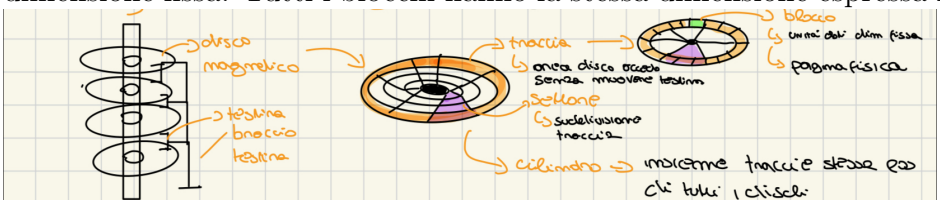
In questo capitolo si tratterà la progettazione fisica delle basi dati. Scopriremo infatti come le tuple sono effettivamente memorizzate sulla memoria secondaria per garantire cosi la loro non volatilità. **Organizzare** le **tuple** in memoria per **chiave** significa **associare** tramite una **funzione** la **chiave** della **tupla** alla **pagina** che la **contiene**. L'organizzazione può essere:

- **Primaria**: impone un criterio per la memorizzazione dei dati (esempio tupla t1 va nella pagina p1).
- **Secondaria**: semplicemente permette il recupero facile di essa (esempio t1 si trova in p1).
- **Statica**: l'organizzazione "rimane sempre la stessa"
- **Dinamica**: l'organizzazione se si adegua al numero di tuple da memorizzare.

4.1 Dischi magnetici

Una memoria di massa a disco magnetico è formata da un insieme di dischi magnetizzati su entrambe le superfici. Per accedere ad ogni posizione di un qualsiasi disco si utilizza un braccio al quale è collegata una testina di lettura scrittura. Un **disco** è **suddiviso logicamente** in **settori** e **tracce**. La **coppia ¡settore,traccia¿ identifica** un **blocco** di memoria del disco a

dimensione fissa. Tutti i blocchi hanno la stessa dimensione espressa in byte.



4.2 Organizzazione primaria in memoria per chiave

Vediamo ora i principali metodi di organizzazione per chiave. Organizzazione per chiave primaria può essere:

- **Procedurale**: Utilizza una procedura detta funzione hash che associa ad ogni chiave di una tupla la pagina che la contiene.
- **Albero**: Utilizza una struttura dati ad albero.

4.2.1 Hashing

La tecnica dell'**hashing** è una tecnica di **organizzazione** per **chiave procedurale statica**.
Definiamo **funzione hash** una **funzione** che **associa** ad ogni **chiave** la **pagina** che la contiene. A differenza di ciò che avviene con le tecniche di hashing viste in precedenza una **pagina potrà contenere** anche **piu** di **una chiave fino** a un **massimo dato** dalla **dimensione** della **pagina**. La funzione spesso utilizzata è la seguente $H(k) = k \ mod \ M$ dove M è il numero di pagine totali.
Definiamo **collisione** la situazione in cui **vogliamo assegnare** ad una **chiave** una **pagina** che è già **piena**. La gestione delle collisioni può essere di due tipi:

- Lista di trabocco
- Indirizzamento aperto

In particolare la gestione delle collisioni ad indirizzamento aperto può essere:

- Scansione Lineare

- Scansione Quadratica
- Doppia funzione hash

La tecnica a **scansione lineare** ha il problema dell'**agglomerazione primaria**. Si formeranno agglomerati di chiavi vicine. La tecnica a **scansione quadratica** invece dell'**agglomerazione secondaria**: Chiavi uguali finiranno nella stessa posizione.

Svantaggi

- Impossibilità ricerca tra valori
- Allocazione statica

4.2.2 Hashing Virtuale

La tecnica dell'hashing virtuale è una **tecnica** di **organizzazione** per **chiave procedurale dinamica**.
La tecnica si sviluppa nel seguente modo.

1. **Alloco M pagine** di **capacita C** con **funzione hash** $H_r(k) = K \; mod \; M(2^r \; M)$ dove r è il numero di raddoppi effettuati fino a questo momento.
2. **Alloco** un **vettore** β della stessa dimensione di M **binario** in cui assegno di default 0 ad ogni posizione. $\beta[i] = 1$ solo quando la pagina p_i contiene una chiave
3. **Inserisco** le **chiavi** con la **funzione hash** discussa precedentemente. All'inserimento di una chiave in una pagine la posizione nel vettore β corrispondente dovra modificare il suo valore in 1.

Gestione delle collisioni

Le collisioni verranno gestite nel seguente modo.

1. Si **raddoppia** la **dimensione** del **vettore** β e del **vettore** delle **chiavi**.
2. **Cambia** la **funzione hash incrementando r** che inizialmente era uguale a 0 di uno. $r = r + 1$

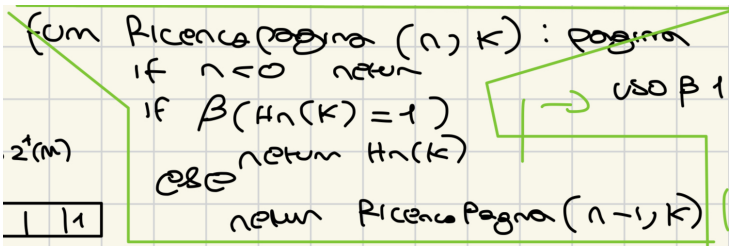
3. **Ridistribuisco** le **chiavi** della pagina **traboccata** e la **chiave** da **aggiungere** con la **nuova funzione hash**.

Svantaggi

- Bisogna **raddoppiare** sempre il **vettore** delle chiavi e β
- Le **chiavi traboccate finiranno sempre** o nella **vecchia posizione** o **M posizioni avanti** con la nuova funzione hash. Abbiamo quindi uno spreco di pagine.

Vantaggi

- **Un** solo **accesso** alla **memoria** per il **recupero** della **pagina** contenente la chiave grazie al vettore β



4.2.3 Hashing Estendibile

La tecnica dell'hahshing estensibile è una tencnica di **organizzazione** per **chiave procedurale dinamica senza struttura ausiliare**.
La tecnica si sviluppa nel seguente modo:

1. Al primo step si **alloca** un **vettore** β contenente due celle. Questo vettore a differenza dell'hashing virtuale, che indicava se la pagina corrispondente conteneva una chiave, **contiene** per **ogni posizione** un **puntatore** all'**area dati** che **contiene** la **chiave**. $B[pseudochiave] =$ Area dati contiene la chiave. La pseudo chiave viene scelta attraverso una funzione hash $H(k) =$ pseudo chiave.

2. il vettore conterrà anche una variabile p detta prefisso globale mentre l'area che contiene le chiavi avrà per ogni pagina una variabile detta prefisso locale p'.

3. Procedo a inserire le chiavi grazie alla funzione hash.

Gestione collisioni

Le collisioni vengono gestite nel seguente modo.

- $p = p'$ **raddoppio** β **clonandone** i **vecchi puntatori**. **Aumento** $p = p + 1$
- $p' < p$ **aggiungono** una nuova **pagina**. **Aumento** $p' = p' + 1$ sia per la **vecchia** che per la **nuova pagina**.
- **Ridistribuisco** le **pagine traboccate** tra la **vecchia** e la **nuova pagina aggiunta** in **base** ai **bit** $p' + 1esimi$
- **Modifico** i **puntatori** di β precedentemente **clonati facendoli puntare** alla **nuova pagina**

Vantaggi

Recupero della pagina con due accessi alla memoria

Svantaggi

No ricerca per intervallo

4.2.4 Hashing lineare

La tecnica dell'hashing lineare è una tecnica procedurale dinamica.
Funziona nel seguente modo.

1. **Allochiamo M pagine** con capacita C
2. **Inizializziamo** una **variabile** $p = 0$ che conta il **numero** di **trabocchi** avvenuti fino ad adesso.
3. **Inseriamo** le **chiavi** nella pagina corrispondente data dalla **funzione hash** $H(K) = K\ modM$.

Gestione delle collisioni

1. **Aggiungiamo** una **pagina** a **indirizzo** $p+M$ e **inseriamo** una **lista** di **trabocco** nella **cella traboccata**.
2. Ridistribuiamo le chiavi nella posizione p con la funzione hash $h(k) = K\ mod 2M$
3. In fine **aumentiamo** $p = p + 1$

Vantaggi

Evitiamo il problema dello spreco dell'hashing virtuale.

Svantaggi

No ricerca per intervalli.

4.2.5 Albero

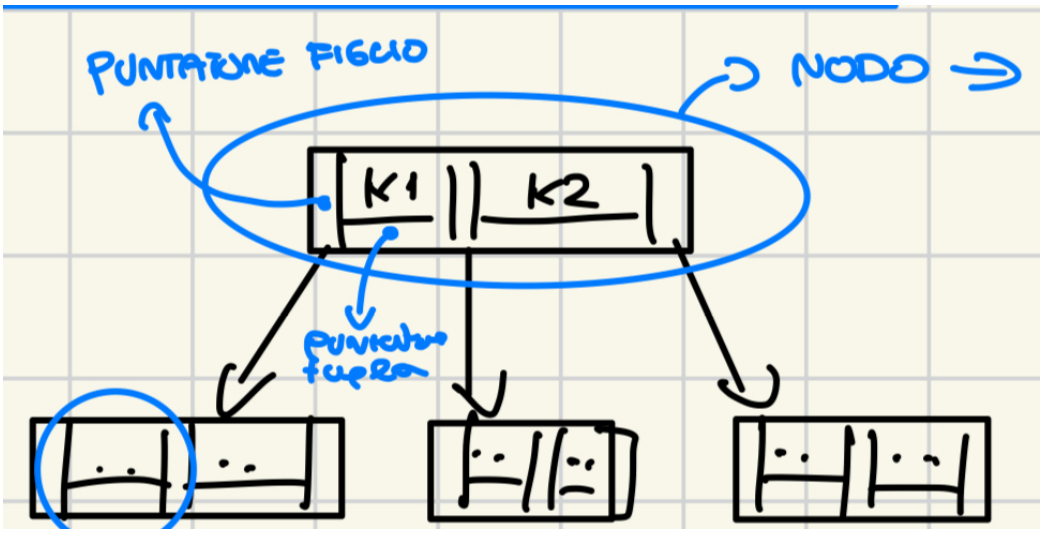
Vediamo ora invece le **tecniche** di **organizzazione primaria** per **chiave** in memoria ad **albero**.

B-tree

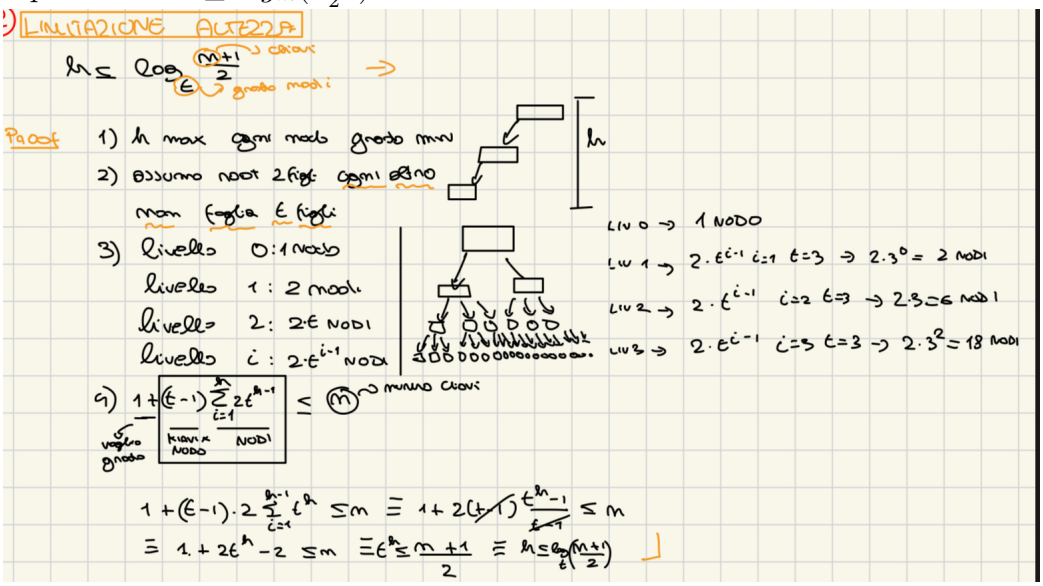
Un b-tree è un albero n-ario che rispetta precisamente un certo numero di proprietà:

- Un nodo corrisponde ad una pagina
- Ogni nodo contiene un intestazione di un certo numero di byte
- Ogni cella di un nodo contiene i puntatori ai suoi figli e il puntatore alla tupla associata alla sua chiave
- Ogni nodo può avere massimo M figli
- Tutte le foglie si trovano allo stesso livello
- Le chiavi sono memorizzate in un nodo da mantenere l'ordinamento e ciò vale ricorsivamente ai figli. $k_1 < k2 < ... < k_m$.

- Ogni nodo non foglia se contiene k chiavi deve avere k+1 figli.
- Ogni nodo diverso dalla radice deve avere un riempimento minimo $\lceil \frac{M}{2} \rceil - 1$



Limite altezza L'altezza di un b-tree è limitata superiormente dalla seguente espressione: $h \leq log_m(\frac{n+1}{2})$

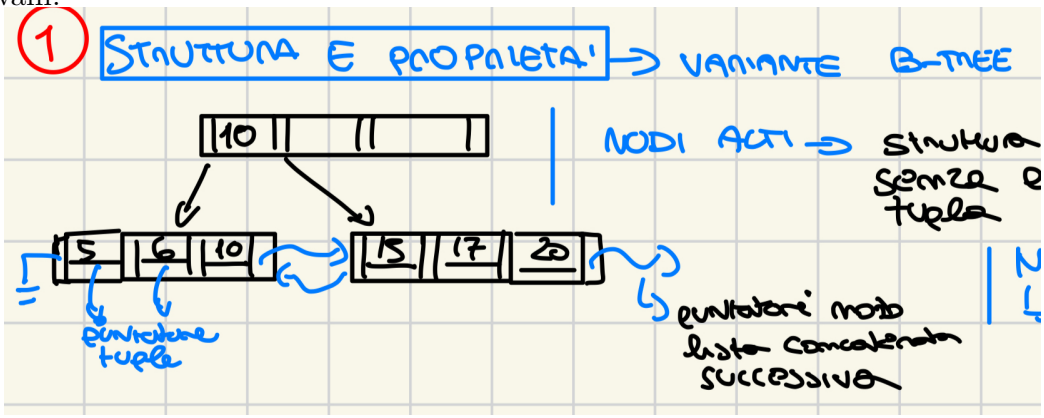


Vantaggi Operazioni con costi logaritmici

B+-tree

I b+-tree sono una variante dei b-tree. Le proprietà sono le stesse del b-tree. Il **cambiamento** principale sta nel fatto che questa volta **ogni cella mantiene** il **puntatore solo** ai suoi **figli** e non al record corrispondente. Saranno solo i **nodi foglia strutturati** a **lista concatenata** bidirezionale a **contenere** i **puntatori** alle **tuple data** ogni **chiave**.

Vantaggi Operazioni con costi logaritmici e possibilità di ricerca per intervalli.



4.3 Organizzazione secondaria in memoria per chiave

L'organizzazione secondaria in memoria per chiave avviene attraverso gli indici.

4.3.1 Indice

Un **indice** è una **struttura dati** che ad **ogni chiave associa** la **tupla** che la contiene.

4.3.2 Indice Clustered

Un **indice clustered** è un **indice** che **mantiene** sia **fisicamente** che **logicamente** all'interno dell'indice l'**ordinamento** delle **chiavi** di **ricerca**. Generalmente in ogni DBMS alla **chiave primaria** si **associa** un indice **clustered**.

4.3.3 Indice Non Clustered

Un indice non clustered è una struttura dati che associa ad ogni chiave di ricerca o gruppo di chiavi di ricerca le tuple che la contengono

4.3.4 Vantaggi

Velocizzano le ricerche

4.3.5 Svantaggi

Possono rallentare le operazioni di inserimento modifica e rimozione. Anche gli **indici** sono **memorizzati** su **disco** quindi **sprecano spazio**.

Chapter 5

Transazioni

Una transazione è un **insieme** di **istruzioni** che se **eseguite correttamente cambiano** lo **stato** della **base** di **dati**.

5.1 Proprietà

- A tomicità: Una transazione deve essere un'operazione atomica. O viene eseguita tutta o se si verifica per qualche motivo un interruzione le modifiche effettuate vanno annullate.
- C onsistenza: Una transazione sia se eseguita correttamente che non deve mantenere il database in uno stato consistente. Uno stato si definisce consistente quando tutti i vincoli sono soddisfatti.
- I solamento: Una transazione deve lavorare come se fosse isolata dalle altre.
- D uratura: L'effetto delle operazioni di una transazione eseguita correttamente devono rimanere nel tempo.

5.1.1 Garantire atomicità

Quando una transazione viene abortita (terminata con un fallimento) deve intervenire una operazione detta **Roolback** che ha il compito di **ripristinare** la **base** di **dati** allo **stato** che aveva **prima** dell'inizio della **transazione**. La roolback è possibile tramite un file di log che mantiene per ogni variabile

il valore prima e dopo l'esecuzione della transazione e che mantiene l'id della transazione che ha modificato le variabili.

5.1.2 Garantire durabilità

Per garantire la durabilità ci viene in aiuto una operazione detta **commit**. Una volta eseguita la commit tutto ciò che è stato eseguito da una transazione non può piu essere ripristinato.

5.1.3 Garantire isolamento

Per garantire l'**isolamento** ci vengono in aiuto i **lock** di cui parleremo a breve.

5.1.4 Garantire consistenza

Per garantire la consistenza bisogna eseguire le **transazioni** in **serie** per **evitare** situazioni di **racing condition**.

5.2 Scheduler e Schedule

Abbiamo precedentemente affermato che per mantenere la consistenza le transazioni devono essere eseguite in serie. Ciò però non porta vantaggi poiché **consentendo** l'**esecuzione contemporanea** delle **transazioni** avremmo **miglioramenti** sia in termini di **througput** che di **riduzione** dei **tempi** di attesa delle transazioni considerando anche che l'**algoritmo** di **assegnazione** dell'**elaborazione** alle **transazioni** lavora "alla **round robin**".
Per evitare errori ci viene in aiuto il concetto di serializzabilità che tratteremo a breve. E' necessario prima dare i concetti basilari di schedule e scheduler.
Uno **scheduler** è un **algoritmo** che da come **risultato** uno **schedule**.
Uno **schedule** è una **sequenza** di **esecuzione** delle **istruzioni** di **piu transazioni contemporanee**(ne definisce l'ordine di esecuzione delle istruzioni).

5.3 Recuperabilità transazione

Se una transazione è stata abortita come già accennato bisogna ripristinare attraverso una roolback i cambiamenti. Per far si che ciò sia possibile uno schedule deve essere recuperabile.
Definizione di **schedule recuperabile**: Uno schedule si definisce recuperabile quando per ogni coppia di transazioni T_i e T_j se la transazione T_j legge i valori scritti da T_i allora la commit di T_i deve avvenire prima della commit di T_j. Questa informazione non basta poiché potrebbe portare all'inefficienza di roolback a cascata. Per evitare ciò ci viene in aiuto la definizione di schedule cascatless.
Definizione **schedule cascatless**: Uno schedule si definisce cascatless quando per ogni coppia di transazioni T_i e T_j se la transazione T_j legge i valori scritti da T_i allora la commit di T_i deve avvenire prima della lettura di T_j.
cascatless $\Rightarrow$ recuperabilità

5.4 Serializzabilità

Veniamo ora al concetto piu importante ovvero il concetto di Serializzabilità. Uno **schedule** S si **definisce serializzabile** se è **equivalente** ad uno **schedule** S2 **Seriale**.

5.4.1 Equivalenza

Due **schedule** sono **equivalenti** quando per **ogni istanza** di **database** lo **schedule s1 porta** il **database** nello stesso **stato** in cui lo **porterebbe s2**.

5.4.2 Conflict Serializable

Conflict Serializable è un sottoinsieme della serializzabilità. Per capirla dobbiamo quando le istruzioni si definiscono in conflitto.

Istruzioni in conflitto

Due istruzioni si definiscono in conflitto quando appartengono a due transazioni diverse, sono eseguite una dopo l'altra, operano sullo stesso dato, una delle due istruzioni è una operazione di scrittura.

Conflict equivalent

Due **schedule** si definiscono **conflict equivalent** quando lo schedule **s1** può essere **trasformato** nello schedulo **s2** mediante uno **scambio** di **operazioni non** in **conflitto**. $S1 \equiv_c S2$

Conflict serializable

Uno **schedule** si defnisce **conflict serializable** quando è **conflict equivalent** ad uno **schedule seriale**. $S1 \equiv_c S2$ S2 seriale

Determinare se uno schedule è CS

Il modo piu facile per verificare se uno schedule è conflict serializable è attraverso il **grafo** delle **precedenze** e l'**algoritmo topologico**.
Grafo delle precedenze: Grafo in cui i nodi sono le transazioni e gli archi le operazioni in conflitto.
Algoritmo topologico: Algoritmo che verifica se sono presenti cicli. L'**assenza** di **cicli implica** che lo **schedule** è **conflict serializable**.

5.4.3 View Serializable

La view serializable è una sottoclasse della serializable. La CS è un sottoinsieme a sua volta della VS.

View Equivalent

Due schedule S1 e S2 sono view equivalent $S1 \equiv_v S2$ se per ogni dato Q:

- Se la transazione t_i legge il valore iniziale di q in S1 allora t_i leggerà il valore finale di q anche in s2
- Se la transazione t_i scrive il valore iniziale di q in S1 allora t_i scrive il valore finale di q anche in s2
- Se la transazione t_i legge il valore di q in S1 scritto da t_j allora t_i leggerà il valore di q scritto da t_j anche in s2

View serializable

Uno **schedule** si definisce **view serializable** se è **view equivalent** ad uno **seriale**. $S1 \equiv_v S2$ S2 seriale

5.4.4 Gerarchie Serializzabilità

$CS \Rightarrow VS \Rightarrow S$

5.5 Lock

I lock sono il meccanismo principale con il quale si **mantiene** l'**isolabilità** delle **transazioni** e la mutua esclusione sui dati.

5.5.1 Lock-S

Permette la lettura del dato sul quale si è acquisito il lock. Permette piu letture contemporanee da piu transazioni ma blocca le scritture.

5.5.2 Lock-X

Permette la scrittura del dato sul quale si è acquisito il lock. Non permette ne letture ne scritture contemporanee sul dato da altre transazioni.

5.5.3 Protocolli 2PL

Il protocollo 2PL permette l'acquisizione e il rilascio dei lock solo attraverso queste due fasi.

- GrowingPhase: Una transazione acquisisce tutti i lock che necessita e non ne può rilasciare nessuno
- ShrinkingPhase: Una transazione rilascia tutti i lock non ne può più acquisire nessuno

Se le transazioni di uno schedule seguono questo protocollo allora lo schedule è conflict serializable e quindi serializable.
Questo protocollo non garantisce però la cascatless. **Soluzione S2PL**.

5.5.4 Protocolli S2PL

Variante della 2PL. Stesse regole ma i lock-X rilasciati solo dopo la commit.
S2PL ⇒ Cascatless

5.5.5 Protocolli R2PL

Variante della 2PL. Stesse regole ma tutti i lock rilasciati solo dopo la commit.
R2PL ⇒ Transazioni seriali nell'ordine commit

5.5.6 Dimostrazione 2PL implica CS