Basi di Dati

Carmelo Gugliotta 213477

Anno 2021/2022

Contents

1 Lezione n1 27/09/2021

1.1 Differenza tra Informazioni e Dati

Informazioni e Dati non sono sinonimi.

- Un **Dato** è una rappresentazione oggettiva (concreta) di un concetto tramite una codifica.

  N.B Esempio codifica a basso livello - BIT

- **L'informazione**, che non può essere definita a partire dal dato (Abbiamo definito già il dato a partire dall'informazione), è un concetto primitivo difficile da definire. Possiamo però definire quest'ultima come un **Entità Astratta** che sta alla base della comunicazione.

  Difatti in basi di dati si parla di "Misura di una informazione"

  Misura di una informazione: Quanta è l'informazione di una cosa, più propriamente quanto viene alterato lo stato di chi la riceve rispetto al momento antecedente della ricezione.

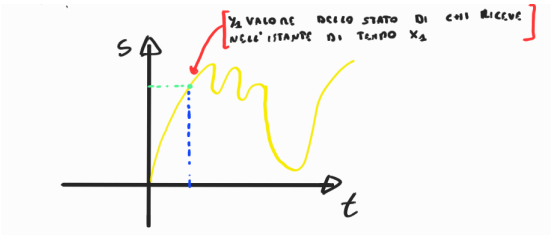


Figure 1: Esempio Grafico Misura Informazione

2 Lezione n2 30/09/2021

Durante l'implementazione di un sistema di basi di dati, è necessario, per una corretta elaborazione e gestione dello stesso, porsi le seguenti domande.

1. Come creare una base di dati?
2. Come scelgo quali informazioni rappresentare?
3. Come scelgo le strutture più adatte per rappresentare tali informazioni?
4. Come manipolare il contenuto di una base di dati esistente?

Per rispondere correttamente a tali domande dobbiamo intraprendere il ciclo di vita di un sistema di basi di dati rappresentato dal sistema a cascata o meglio **Waterfall Model**.

2.1 Waterfall Model

Si hanno le seguenti fasi:

1. **Studio di fattibilità**
   Raccogliere informazioni sufficienti a capire se le richieste effettuate sono soddisfacibili o meno rispettando anche eventuali vincoli economici o di altro tipo.
2. **Analisi dei requisiti**
   Raccolta delle specifiche funzionali e non funzionali che viene effettuata tramite *Interviste*
3. **Progettazione concettuale**
   Viene effettuata l'elencazione delle informazioni da rappresentare (**ES:** info fatture; info prodotti; info anagrafiche cliente) Alla fine di tale fase si ha come prodotto un vero e proprio progetto elaborato sulla base di una sintassi particolare **ModelloER** (Entità-relazione).
   Si noti che tale fase non è necessariamente svolta da ruoli competenti di informatica.
4. **Progettazione Fisica/Logica**
   Definire la struttura logica dei dati più adatta per rappresentare determinati dati, e determinare successivamente la realizzazione fisica della struttura (Scelta di quale tipo di implementazione usare).
   Ma attenzione nei sistemi di basi di dati la struttura dati è fissata (Tabella) e l'obbiettivo è trovare il modo migliore di rappresentare i dati tramite tabelle.
   In tale fase si effettua una visione della realtà a diversi livelli di astrazione, e inoltre è importante capire che non si parla più di informazioni ma di dati.
   Il risultato di tale fase è un progetto logico/fisico. Il progetto logico consiste in un modello relazionale.

5. Messa in Opera

L'insieme delle fasi concretizza il sistema a cascata (WaterFall Model).

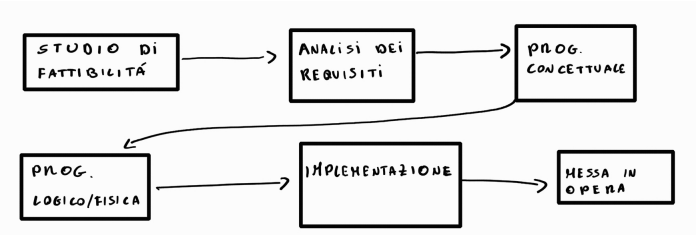


Figure 2: WaterFall Model

2.2 Modello Entità-Relazione

Il Modello ER è un modello teorico per la rappresentazione concettuale e grafica delle informazioni a un alto livello di astrazione, utilizzato nella terza fase della progettazione di basi di dati.

– Ogni concetto, derivante dall'analisi dei requisiti, ha un nome che lo identifica univocamente e viene rappresentato all'interno del diagramma tramite un rettangolo che prende il nome di **Entità** o meglio **Entity Set** .

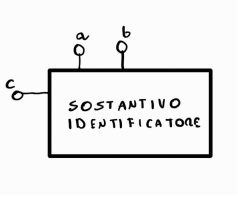


Figure 3: Entity Set

– Il legame tra due o più entità viene espresso tramite le **relazioni**.
Identificata tramite un set di coppie e rappresentata graficamente da un rombo contente il nome della relazione.
Il concetto di relazione segue il concetto matematico di relazione.
In matematica una relazione tra due o più insiemi è un sottoinsieme del prodotto cartesiano degli stessi.

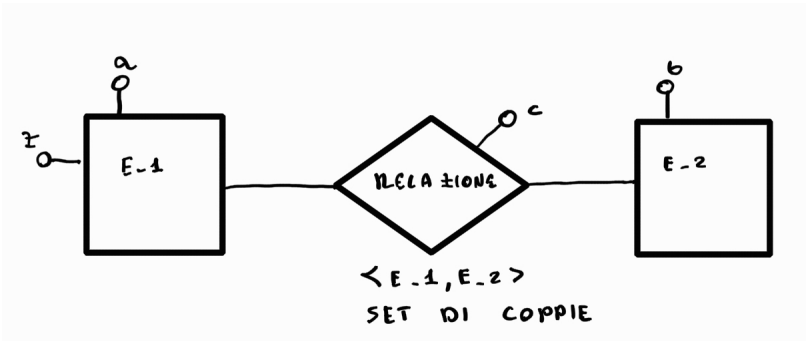


Figure 4: Relazione

– Le entità e le relazioni possono essere descritte usando una serie di **attributi** caratterizzanti/identificanti, rappresentati graficamente con un pallino collegato all'entità/relazione a cui è associato il nome.
Attenzione: Non sono istanze di Entity Set. Vedi Figura 3 e 4

– La **chiave** è un insieme di attributi identificanti e permette di implementare un meccanismo di identificazione dell'entità ed è rappresentato graficamente da un taglio degli attributi coinvolti.
Formalmente si ha che la **chiave** di un'entità è un insieme di attributi identificanti dell'entità che risulta minimale rispetto alle proprietà di essere identificante.

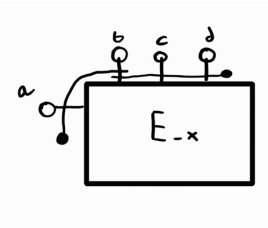


Figure 5: Esempio Chiave

Attenzione :

* Una Entità può avere più chiavi candidate. Osserva Figura 5
* Ogni Entità deve essere caratterizzata da almeno una chiave
* Per ogni Relazione **non** deve essere indicata alcuna chiave poiché la sua identificazione deriva dal concetto stesso di Relazione.
* **Minimale rispetto a una proprietà P**
Dato un insieme A che soddisfa P si dice minimale se non esiste $A_* \subset A$ che soddisfa P.

3 Lezione N3 01/10/2021

3.1 Vincoli di cardinalità e approfondimento su attributi e chiavi

Analizziamo il seguente schema :

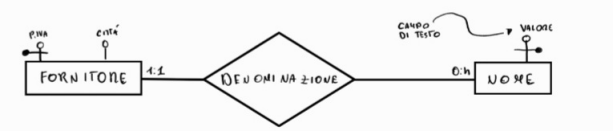


Figure 6: Esempio

Così facendo indico che nella mia realtà vi è un Entity set (un insieme) che contiene tutti i nomi della lingua italiana o inglese ecc. E indico inoltre che c'è una relazione tra fornitore e nome per cui a ogni fornitore viene associato un nome.
Si presti molta attenzione alla presenza della specifica della cardinalità (1:1 e 0:n) che permette di risolvere un problema legato alla definizione matematica di relazione. Senza la specifica infatti avremo un set di coppie, che costituisce la Relazione, pari al prodotto cartesiano tra Fornitore e Nome. Con la specifica invece indichiamo che un fornitore può apparire una e una sola volta all'interno del set di coppie e che il nome invece può apparire un numero di volte che va da 0 a qualunque.

Def : I **Vincoli di cardinalità** permettono quindi di arricchire i dettagli della relazione definendo come ogni entità partecipa alla relazione.

Attenzione :

- Quando c'è un 1:1 significa che tutte le coppie presenti nella relazione non solo sono identificati dalla coppia $< E_1, E_2 >$ ma anche solo dall'Entità con vincolo 1:1
- Utilizzare i vincoli di cardinalità potrebbe portare a una difficoltà di Data Entry quindi è consigliato utilizzarli solamente se necessario.
- Dal punto di vista dei concetti quando si dichiara un Entity Set, si sta istanziando un concetto assestante che poi dovrà trovare una proprio rappresentazione separata dal resto.
- Se si utilizzano gli attributi si esprime una caratteristica libera, quindi chi popola la basi di dati avrà la libertà massima di esprimere quello che vuole.

Introduciamo ora chiavi candidate di tipo Mista e Esterna:

- **Chiave Mista**
  È una chiave candidata che non solo coinvolge attributi ma anche Entità. Le Entità che coinvolge, per il concetto stesso di chiave candidata, deve essere caratterizzate da un vincolo di cardinalità pari a 1:1.
- **Chiave Esterna**
  Tutti i membri che costituiscono la chiave non sono attributi dell'Entità.

La Chiave Mista viene utilizzata per tutte le casistiche e molte volte a causa di ciò, Mista ed Esterna sono utilizzati come sinonimi.

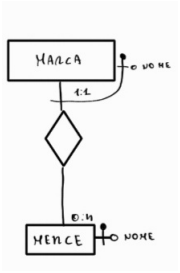


Figure 7: Esempio Chiave Mista

3.2 Approfondimenti sul Modello ER

In alcuni casi complessi si ha la necessità, di includere nel meccanismo di identificazione anche attributi facente parte delle relazioni. Ma per sintassi, **NON** può e non deve essere mai effettuata un operazione di questo tipo, perché per ogni relazione non deve essere indicata alcuna chiave. Il problema può essere risolto convertendo la relazione in una Entità.

Si osservi il seguente esempio

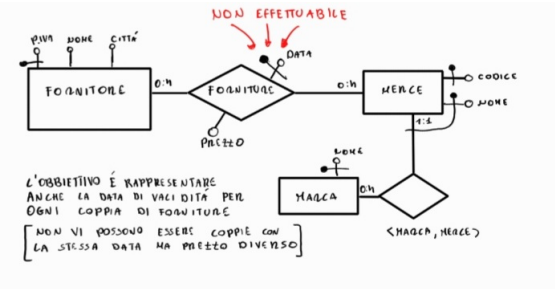


Figure 8: Problema

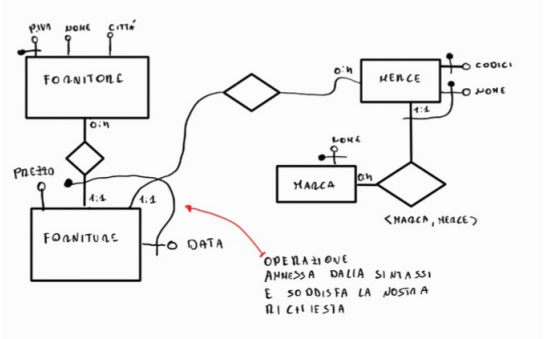


Figure 9: Soluzione

3.3 Relazioni Ternarie

È possibile avere una relazione ternaria, ma ciò significa rappresentare, nella banca dati, terne di oggetti appartenenti a ciascun Entità coinvolta. Questo tipo di operazione, pur essendo ammessa, non è mai utilizzata poiché crea molta confusione sull'interpretazione dei vincoli.

Ecco perché se strettamente necessario viene utilizzata la seguente soluzione:

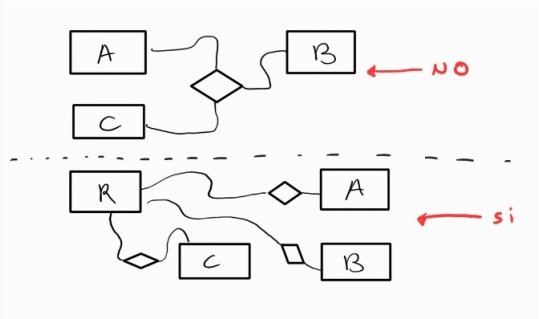


Figure 10: Problema e Soluzione Relazione Ternaria

In cui l'Entità R è rappresentata da una terna di Entità attraverso una relazione binaria con ciascuna di esse.

4 Lezione N4 04 10 2021

4.1 Relazione Di Generalizzazione

Rappresentano legami logici tra un'entità madre-padre, e una o più entità $E_1, ..., E_n$, dette entità figlie.
l'entità madre-padre è generalizzazione delle entità figlie.
l'entità figlie sono specializzazione dell'entità madre-padre.
Tra le entità coinvolte in una generalizzazione valgono le seguenti proprietà:

- Ogni occorrenza di un'entità figlia è anche un occorrenza dell'entità madre-padre.
- **Ereditarietà :**
  Ogni proprietà dell'entità genitore è anche una proprietà delle entità figlie. Ci si limita a inserire attributi,relazioni o chiavi candidate supplementari,solo se necessari.

Le generalizzazioni vengono rappresentate graficamente mediante delle frecce che congiungono le entità figlie con l'entità genitore.
Quando viene rappresentata una generalizzazione deve essere specificato il tipo di legame logico.

- **Totale o Parziale**
  Totale se ogni occorrenza dell'entità genitore è appartenente ad almeno una delle entità figlie, altrimenti è **parziale**.
  Nello specifico se esiste un elemento della entità più generale non appartenente ad almeno a una entità del tipo particolare allora è parziale.
- **Esclusiva o Inclusiva**
  Esclusiva se ogni occorrenza dell'entità madre-padre è al più un'occorrenza di una delle entità figlie, altrimenti inclusiva.

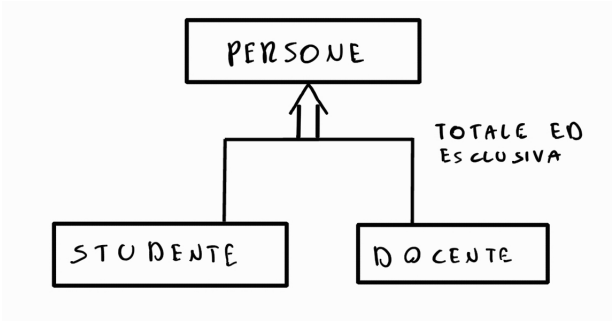


Figure 11: Esempio Relazione di Generalizzazione

- Se il tipo di legame logico in una Generalizzazione non viene specificato allora il legame è di tipo Totale ed Esclusivo. -
- In termini insiemistici definire una Generalizzazione Totale ed Esclusiva corrisponde a definire una Partizione. -

4.1.1 IS - A

La relazione "IS-A" è quella relazione che si ha quando si lavora con generalizzazioni, costituite da una sola entità figlia, basate su un legame logico di tipo parziale.

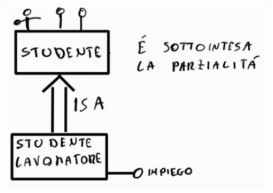


Figure 12: Esempio Generalizzazione IS - A

La sola Entità figlia è un sottoinsieme e specializzazione dell'Entità genitore.

4.2 Supportare La tracciabilità di informazioni

La tracciabilità delle informazioni (Es: lista di num.telefon,di materie insegnate,di indirzzi), **NON** può avvenire con l'aggiunta di attributi, poiché gli attributi sono informazioni di tipo **semplice**, sono di libera interpretazione (Vulnerabili a una probabile inconsistenza delle informazioni) e non possono essere entità.

Difatti per implementare la tracciabilità delle informazioni possono essere utilizzate le relazioni:

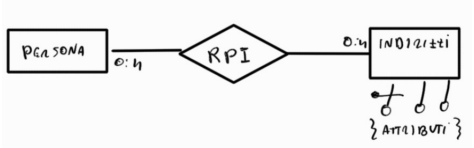


Figure 13: Soluzione tracciabilità delle informazioni

4.3 Discussioni finali sul Modello ER

4.3.1 Espressività limitata del modello ER

Si ipotizzi di voler registrare informazioni relative ai voti di un esame universitario, acquisiti da uno studente:

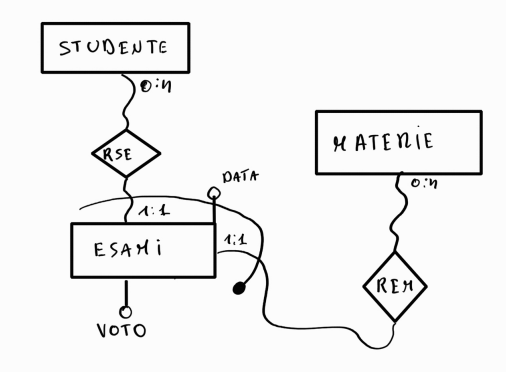


Figure 14: Limiti Modello ER

Agendo in tal modo si esprime che uno studente, può effettuare al più un esame di una specifica materia al giorno. Ma ovviamente dobbiamo effettuare delle modifiche affinché uno studente non possa accettare più esami associati alla stessa materia in date diverse.
Bene questo non è possibile per **limiti di espressività** che caratterizzano il **Modello ER** Per esprimere determinate specifiche durante la fase di progettazione concettuale, quest'ultime devono essere annotate nei **vincoli non espressi**.

4.3.2 Necessità di Relazioni di Generalizzazione

Possiamo commutare in alcune casistiche la relazione di Generalizzazione in relazioni $1 : 1 - -0 : n$? In nessun caso, poiché la commutazione della Generalizzazione in una Relazione, porterebbe alla perdita della specifica di Totalità e dell'Esclusività (Esempio Persone,Docenti,Studenti : Esistono obbligatoriamente persone che non sono né docenti né studenti, e Persone che possono essere Studenti e Docenti contemporaneamente). Nelle casistiche di tipo parziale-inclusiva, sarebbe errato perché si effettuerebbe un errore dal punto di vista sintattico. Difatti la Relazione non esprime la particolarità di un determinato concetto a partire da un concetto genitore più generale. La Generalizzazione implementa nel modello ER, una maggiore espressività, che non può essere sostituita dalla Relazione.

5 Lezione n5 7 10 2021

5.1 Progettazione Logica

Precedentemente abbiamo organizzato le informazioni tramite il modello **ER** ora tali informazioni devono essere rappresentate tramite strutture dati. In particolare si passa da un livello di astrazione più alto a un livello di astrazione più basso, che si basa sulla rappresentazione delle informazione tramite la scelta di opportune tabelle (non immediato quanto sembra). Tale fase prende il nome di **Progettazione logica**.

Si analizzi il seguente schema ER:

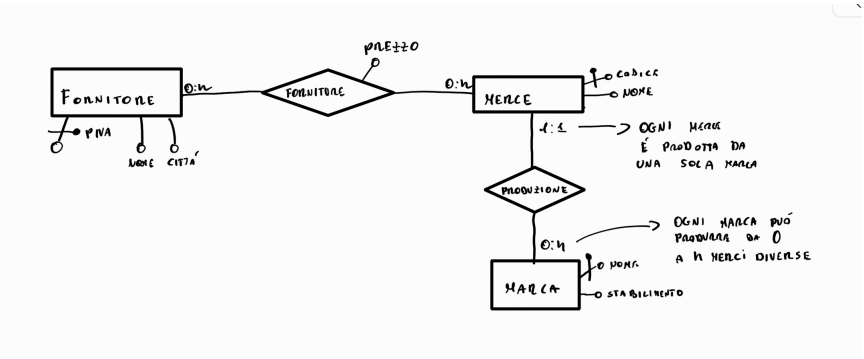


Figure 15: Esempio

Per produrre uno schema logico si ha la necessità di elencare le tabelle (Una o più di una) che si ritengono opportune, indicando il nome della tabella seguito da parentesi dalla lista degli elementi,ognuno dei quali identifica una colonna.

$$NomeTabella(elm_1, elm_2, ..., elm_n)$$

Ovviamente creare una sola tabella non è una scelta opportuna, poiché diviene difficile gestire e utilizzare il concetto di chiave, e inoltre si ha che l'informazione potrebbe essere rappresentata più volte e questo porterebbe a una inconsistenza dei dati.
Invece una metodologia più elegante è l'utilizzo di più tabelle. Si osservi il seguente esempio:

Fornitore(**P.IVA**,NOME,CITTÀ)
Forniture(**FORNITORE**,**MERCE**,PREZZO)
Merce(**CODICE**,NOME,MARCA)
Marca(**NOME**,STABILIMENTO)

P.IVA	NOME	CITTÀ
76	M1	Firenze

Table 1: Fornitore

FORNITORE	MERCE	PREZZO
76	45268	47euro

Table 2: Forniture

CODICE	NOME	MARCA
45268	Pasta	Barilla

Table 3: Merce

NOME	STABILIMENTO
Barilla	Crotone

Table 4: Marca

Si osservi che per indicare il meccanismo di identificazione si sottolinea l'attributo interessato (anche qui la chiave deve essere minimale) e inoltre per ogni attributo va specificato il tipo. Ad esempio se si vuole utilizzare un insieme di caratteri di lunghezza variabile (Stringa) allora deve essere utilizzato VarChar, se invece si conosce a priori la lunghezza massima allora char(n).

Si presti molta attenzione ai vincoli di cardinalità, non solo per la scelta dei meccanismi di identificazione (Soprattutto nel rappresentare una probabile relazione), ma anche per evitare possibili inconsistenze.
In particolare

- **Vincoli 1:1**
  Viene effettuato **l'accorpamento**, cioè l'informazione data da una specifica relazione, viene inserita nella tabella che rappresenta l'entità.
- **Vincoli 0:1**
  Si hanno due possibilità, o si effettua l'accorpamento, e ove non ci sono occorrenze si indica tramite il sostantivo **null** oppure si effettua l'aggiunta di una tabella assestante. Utilizzare il sostantivo null,non è molto opportuno. Infatti può essere problematico dal punto di vista delle strutture dati,perché si ha come conseguenza righe di dimensione diversa.

  Per inserire riferimenti in altre tabelle già create bisogna scrivere:

  Forniture[**fornitore**] **C_fk fornitore**[**P.IVA**] ; **fk=foreign key**
- **Vincoli 0:n**
  Si realizza una tabella assestante.

DEFINIZIONI IMPORTANTI:

- **Vincolo di chiave**:
  Definire un insieme di attributi che identificano le touple, che popolano la tabella, che viene scelta sulla base di minimalità rispetto alla proprietà di identificazione.
- **Vincolo di chiave esterna**:
  Applicato su uno o più attributi,significa che quest'ultimi non possono assumere valori "liberi", ma acquisiscono i valori esplicitati dalla chiave di una tabella (relazione) esterna.

6 Lezione N6 14 10 2021

6.1 Traduzione Modello ER a Modello Relazionale

1. Per ogni Entità si effettua lo sviluppo di una tabella.
2. Si ricorda che per ogni tabella, deve esistere necessariamente , **una chiava primaria**, scelta sulla base del meccanismo di identificazione più appropriato per effettuare eventuali ricerche. Se vi sono più meccanismi di identificazione questi vanno indicati, al fine di non violare la coinsistenza dei dati, tramite i vincoli di unicità **UNIQUE**.
3. Come già visto in precedenza (Scorsa Lezione) si deve prestare molta attenzione ai vincoli di cardinalità,per rappresentare correttamente una relazione. In particolare si hanno possibilità di effettuare eventuali **accorpamenti** (Vincolo 1:1 e 0:1) oppure la creazione di **tabelle** (relazioni) **assestanti**(Vincolo 0:1 e 0:n).
   Si noti però che per effettuare l'accorpamento o la creazione di tabelle(per relazione Molti a Molti), si ha bisogno di uno strumento già descritto precedentemente, ovvero il vincolo di chiave esterna, e dal punto di vista sintattico l'uso corretto di tale strumento è il seguente:

Esempio Rappresentazione relazione Molti a Molti

Tabella: NomeRelazione(**E_1**,**E_2**,Att1,Att2)
FK_1: NomeRelazione[E_1] $\subseteq_{Fk}$ E_1[Chiave_Primaria]
FK_2: NomeRelazione[E_2] $\subseteq_{Fk}$ E_2[Chiave_Primaria]

SI RICORDA che tale sintassi non esplica che E1 o E2 sono chiavi primarie di NomeRelazioni ma che l'attributo E1 e l'attributo E2 nell'istanza di NomeRelazione dovranno essere riempiti con valori presenti in corrispondenza dell'attributo chiave dell'istanza di E1 e di E2. Inoltre L'attributo che è a sinistra (Della FK) può essere NULL ma non può esserlo l'attributo a destra, che deve essere una Chiave Primaria.
OVVIAMENTE questo non ci vieta di sottolineare E1 ed E2, per indicare, che sono gli attributi alla base del meccanismo di identificazione della Relazione(Tabella) di NomeRelazione (ChiavePrimaria), a me no che quest'ultimi possano essere **NULL**. .

Tabella e Relazione nella Progettazione Logica sono sinonimi

Altro vincolo a cui prestare molta attenzione,è il **vincolo di cardinalità 1:n**, difficile da tradurre e affrontati solamente in casi critici (Solitamente il vincolo viene rilassato a 0:n). Se non si effettua il rilassamento del vincolo, allora si procede all'accorpamento dell'attributo(per cui deve esisterne almeno 1,quindi deve già esistere al momento del data entry) all'interno della tabella che descrive l'entità con vincolo 1:n, e inoltre si istanzia una tabella assestante, che include tale attributo.(Situazione Complessa quasi mai incontrata) (Si osservino gli appunti presenti in Samsung Notes).

L'ultimo tipo di vincolo a cui prestare attenzione, è il **vincolo x1:x2**, ove x1 e x2 possono assumere qualsiasi valore positivo (2:2 5:5 100:100 0:2 ecc). Non ci sono modi puliti di gestire tali situazioni, ma si ha la possibilità, nel caso in cui il valore di x1 e x2 siano molto elevati, di rilassare il vincolo a 0:n. Se non è possibile allora la risoluzione dipende dal problema che si sta affrontando (Si osservino gli appunti presenti in Samsung Notes).

4. La traduzione delle Generalizzazione, all'interno della progettazione logica, non può avvenire in maniera fedele, quindi si opera al variare della situazione affrontata al fine di creare meno problemi possibili.

   (a) **Soluzione**:
   Non posso gestire, proprio per limite di sintassi, la generalizzazione totale esclusiva,allora trasformo il seguente schema utilizzando due relazione, perdendo totalità ed esclusività, ma acquisendo la possibilità di distinguere le due Sotto-Entità.

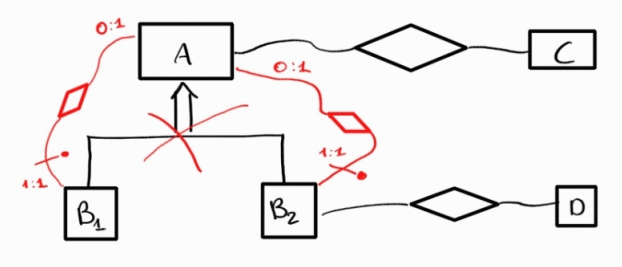


Figure 16: Soluzione 1

   (b) **Soluzione**:
   Si sceglie di eliminare le Sotto-Entità, e di accorpare gli attributi di quest'ultime nell'entità genitore (Ovviamente anche eventuali relazioni)

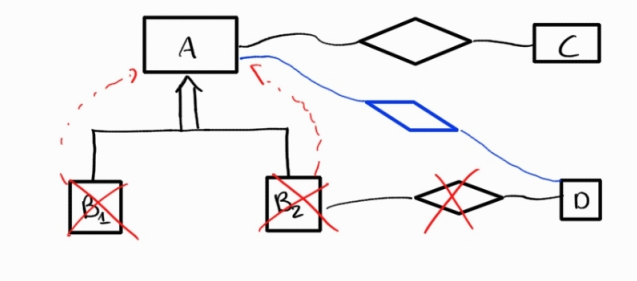


Figure 17: Soluzione 2

   (c) **Soluzione**:
   Nel caso non vi siano molte relazioni con l'entità genitore, si sceglie di eliminare quest'ultima.

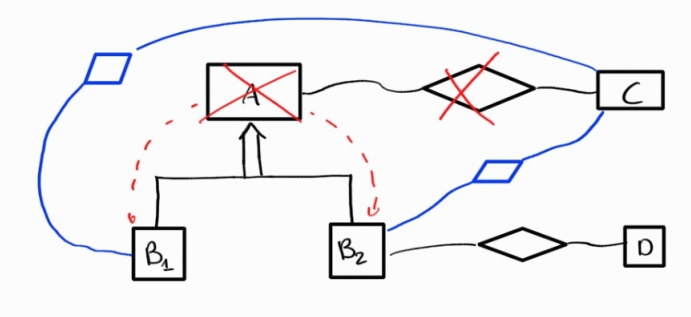


Figure 18: Soluzione 3

5. È rimasta da tradurre la Generalizzazione (IS-A). Nel caso in cui l'Entità figlia sia caratterizzata da eventuali relazione, nel modello ER, allora è conveniente utilizzare tabelle assestanti per Entità genitore, Entità Figlia,e Relazioni (Rombi). Se invece l'Entità Figlia ha pochi attributi, e non è legata ad altre entità, allora è conveniente accorpare l'Entità Figlia nell'Entità Genitore, e indicare con booleano o intero la specializzazione.

7 Lezione n7 18 10 2021

7.1 Algebra-Relazionale

Linguaggio, non utilizzabile "attivamente" su database, che tramite un insieme di operatori di cui è caratterizzata, permette di specificare le informazioni da estrarre da una tabella/relazione.
In particolare permette di esprimere, condizioni in base alle quali esprimere le tuple di interesse e presenta in seguenti operatori:

- **Operatore di Selezione** $\sigma_{Condition}$:
  Operatore "Monadico" (un solo argomento come input) che riceve come argomento una tabella, e restituisce un'altra tabella che presenta **solamente** tuple che rispettano le condizioni esplicitate al pedice dell'operatore.
  Il risultato quindi è una tabella/relazione con medesima struttura alla tabella in input, ma le quali righe sono state ottenute filtrando quelle presenti nella tabella originaria. SI noti inoltre che le condizioni possono essere rappresentate anche da una formula booleana parentesizzata complessa,utilizzando le virgolett eper distinguere ciò che è un attributo e ciò che è un valore di un attributo.

$$(Locazione = \text{'CS'}) \vee (PIVA \neq \text{'F1'} \wedge Nome \neq \text{'C'})$$

- **Operatore di Proiezione** Π_{Lista}:
  Operatore "Monadico", che riceve come argomento una tabella, e restituisce un'altra tabella che presenta **solamente** le colonne esplicitate nella lista presenta al pedice dell'operatore.
  La proiezione, dunque, modifica la **Arità** di una tabella, e in alcuni casi anche la sua cardinalità.

  Arità: Con tale termine, indichiamo, il numero di colonne che costituiscono una relazione, differente dalla Cardinalità che indica il numero di tuple.

- **Operatore di Ridenominazione** $\rho_{\$C \to x}$:
  Operatore "Monadico", che riceve in input una tabella, e restituisce la medesima ma con diversa intestazione della colonna esplicitata (tramite forma posizionale) al pedice dell'operatore.
- **Unione,Intersezione,Prodotto Cartesiano,Negazione** $\cap; \cup; \times; /$:
  All'interno dell'Algebra-Relazionale possono essere utilizzati tali operatori, che assumono il medesimo significato espresso nell'insiemistica. Ovviamente gli argomenti in input (Le tabelle) devono contenere tuple per cui è ragionevole utilizzare il meccanismo desiderato, oltre a dover presentare la medesima struttura (Arità equivalente).
  Si noti che alcuni problemi legati all'intestazione delle colonne, possono essere risolti tramite l'operatore di redominazione.

È importante, mettere in evidenza, che **Tutti** gli operatori dell'Algebra-Relazionale sono **Rientranti**, ovvero, sono applicati a un dominio e restituiscono elementi dello stesso, e ciò permette la possibilità di effettuare innestazioni di diverse operazioni. Non ci si dovrà preoccupare dell'efficienza dell'operazione, poiché qualunque sia la forma utilizzata per scrivere la **Query** (Interrogazione), il database la riscrivirà a modo suo.
Importantissima è invece, la correttezza delle operazioni che si effettuano, poiché il risultato, dipende dall'ordine degli operatori.

8 Lezione n8 22 10 2021

8.1 Join

Altro operatore importantissimo nell'Algebra-Relazionale è l'operatore di **Join**, indicato tramite il simbolo $\bowtie$ che permette di combinare le tuple di due relazioni sulla base dell'uguaglianza dei valori degli attributi comuni alle due relazioni.
Infatti ogni tupla che compare nel risultato del join naturale di r1 e r2, è ottenuta come combinazione ("match") di una tupla di r1 con una tupla di r2 sulla base dell'uguaglianza dei valori degli attributi comuni.

Esempio : Le coppie 'nome-fornitore'e 'nome-merce' tale che il fornitore fornisce la merce

$$\Pi\$2, \$8[(Fornitore \bowtie_{\$1=\$1} Forniture) \bowtie_{\$5=\$1} Merce]$$

Vari Esercizi

9 Lezione n9 22 10 2021

1h CORREZIONE ESERCIZI

9.1 Richieste: Almeno N, Esattamente N, Al più N

Tali casistiche si possono, *grosso modo*, generalizzare; in particolare, per soddisfare le richieste espresse dalla parola chiave **almeno N** è sufficiente effettuare N **Join**, quindi N prolungamenti, della stessa tabella, con annesse le condizioni richieste. $N >= 2$.
Esempio: Utilizzando sempre lo stesso schema delle lezioni precedenti, si chiede di restituire una tabella di Fornitori che forniscono almeno 3 merci. (','=AND)

$$\Pi_{\$1}[(Forniture \bowtie_{\$1=\$1,\$2!=\$2} Forniture) \bowtie_{\$1=\$1,\$2!=\$2,\$5!=\$5} Forniture]$$

Per invece ' l'Esattamente N e Al più N' si deve effettuare un ragionamento un pò più complesso; dal punto di vista insiemistico, dato un insieme di elementi, il sotto-insieme che soddisfa almeno 2 volte una certa proprietà è incluso nel sotto-insieme che soddisfa quella stessa proprietà almeno 1 volta. Ovvero generalizzando si ha:

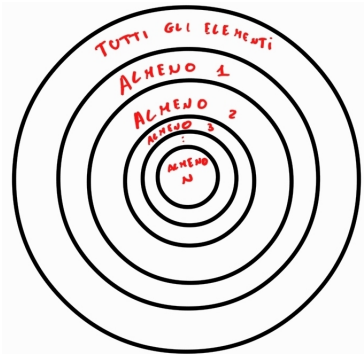


Figure 19: Insieme

È facile carpire che l'Esattamente N è l'intervallo intermedio tra 'Almeno N' e 'Almeno N+1', ottenibile quindi come

$$EsattamenteN = Almeno(N) - Almeno(N+1)$$

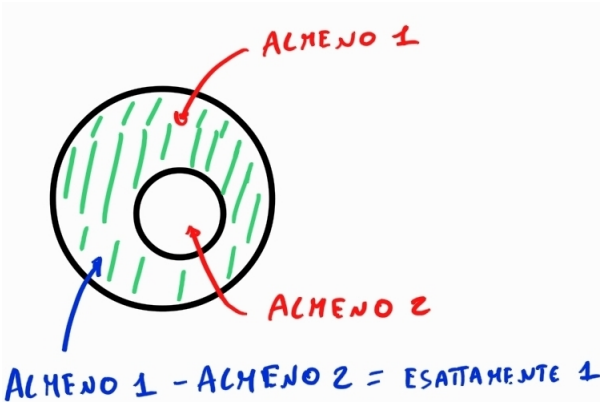


Figure 20: Esempio Esattamente 1

Invece 'Al più N' è l'intervallo intermedio tra 'Tutti gli elementi' e 'Almeno N+1', ottenibile quindi come:

$$AlPiùN = Insieme - Almeno(N+1)$$

9.2 Ricerca del minimo/massimo

Dato uno schema R(A,B) si richiede di effettuare la ricerca del minimo/massimo di B in R; Tale richiesta può essere soddisfatta effettuando una join di R su se stessa, in modo tale da trovare tutti quei valori che non sono il minimo/massimo. La condizione di join dovrà specificare che per ogni attributo B deve esistere almeno un suo minore \$2 > \$2 (ricerca minimo) o che per ogni attributo B deve esistere almeno un suo maggiore \$2 < \$2 (ricerca massimo). In tale maniera vengono tenute tutte le tuple tranne quella in cui l'attributo B assume il valore minore nel primo caso maggiore nel secondo.
Per ottenere il valore minimo/massimo, sarà sufficiente sottrarre dall'insieme iniziale, l'insieme delle tuple dove B non è il minimo/massimo.

9.3 Inoltre...

Quando si effettua l'elaborazione di una richiesta in una espressione di algebra-relazione, è utile tradurre quest'ultima utilizzando il **negato** e l'**esistenziale**; più formalmente è utile passare da un quantificatore universale a un quantificatore esistenziale negato:

$$\forall x \text{ vale } P(x) \quad \equiv \quad !\exists x \text{ tale che } !P(x)$$

Esempio: Nomi di fornitori che forniscono tutte le merci è equivalente a richiedere i nomi di fornitori tale che **non esista alcuna merce** che essi **non forniscono**.

Si noti che gli operatori di **Join** e di **Selezione** traducono il quantificatore Esistenziale, invece la **sottrazione** traduce il quantificatore universale.

10 Lezione n10 04/11/2021

10.1 Lezione di soli esercizi

11 Lezione n11 08 11 2021

11.1 DBMS e SQL

un **Database Management System** (abbreviato in DBMS o Sistema di gestione di basi di dati) è un sistema software progettato per consentire l'organizzazione dei dati all'interno del database. Invece SQL, ovvero **Structured Query Language**, è un linguaggio (a query language) non un database e consente operazioni di gestione dei dati sul database (creare, cancellare, modificare) e consiste in più differenti tipi di linguaggio, tra cui **DDL** e **DML**.

- **DDL**:
  Anche detto **Data Definition Language**, utilizzato per definire la struttura dei dati. Per esempio creare, modificare, cancellare tabelle.
  Generalmente si ha che definisce le colonne (attributi) delle tabelle.
- **DML**:
  Anche detto **Data Manipulation Language**, il quale è utilizzato per manipolare i dati. Per esempio inserire, cancellare, modificare, estrarre tuple.
  Generalmente si ha che aggiunge o aggiorna le righe delle tabelle.

11.1.1 Comandi SQL

- **CREATE**: Permette di creare database (lo spazio logico) o i suoi oggetti(come tabelle o views).

```
1
2     CREATE DATABASE MioDB; --Abbiamo creato lo spazio logico in cui
      stanziare le tabelle (Necessario solo la prima volta)
3
4     USE MioDB;
5
6     /*CHAR senza il prefisso VAR, indica che la stringa ha una lunghezza
      fissata, e non puo' essere ne' minore e ne' maggiore, i sostantivi
      PRIMARY KEY e UNIQUE indicano rispettivamente che l'attributo
      associato e' chiave primaria(1) e che l'attributo    identificativo
      (2)*/
7     CREATE TABLE FORNITORE AS
8         (PIVA, CHAR(20) PRIMARY KEY,
9         NOME VARCHAR(40) UNIQUE,
10        CITTA VARCHAR(20));
11
12    /* Per un codice piu' pulito si utilizza la seguente struttura di
      stanziamento*/
13    CREATE TABLE FORNITORE AS
14        (PIVA CHAR(20), NOME VARCHAR(20) NOT NULL,
15        CITTA VARCHAR(20),
16        PRIMARY KEY PIVA, UNIQUE NOME);
17
18
19
20
```

```
    CREATE TABLE MERCE AS(
        Codice NUMBER(10),
        Nome VARCHAR(40),
        Marca VARCHAR(40),
        PRIMARY KEY codice);

    --COME DEFINIRE UNA FOREIGN KEY
    CREATE TABLE FORNITURA(
        Fornitore CHAR(20),
        Merce NUMBER(10),
        Prezzo NUMBER(4,2) NOT NULL,
        PRIMARY KEY (Fornitore, Merce),
        FOREIGN KEY Fornitore REFERENCES FORNITORE(PIVA)
                    Merce REFERENCES MERCE(Codice)
        UNIQUE ...
        );
    --METODO ALTERNATIVO
    CREATE TABLE FORNITURA(
        Fornitore CHAR(20) REFERENCES FORNITORE(PIVA),
        Merce NUMBER(10) REFERENCES MERCE(Codice),
        Prezzo NUMBER(4,2) NOT NULL,
        PRIMARY KEY (Fornitore, Merce),
        UNIQUE ...
    );

```

Listing 1: CREATE Example

- **DROP**: Utilizzato per eliminare oggetti dal database.
- **ALTER**: Utilizzato per alterare/modificare le strutture del database.
- **INSERT**: Utilizzato per inserire dati nelle tabelle (popolamento del database)

```
        INSERT INTO NomeTabllea VALUES
            ('Att1', 'Att2', 'Att3');

```

Listing 2: INSERT Example

- **DELETE**: Utilizzato per eliminare touple dalle tabelle che costituiscono il database.
- **UPDATE**: Utilizzato per aggiornare dati già esistenti all'interno delle tabelle del database.
- **SELECT**: Consente di acquisire i dati dal databse (Estrarre Tuple).

```
        SELECT columnX,columnt Y ...
        FROM table_name
        --columnX,ColumnY indicano i nomi dei campi
        --della tabella e table_name e' la tabella
        --dalla quale vogliamo estrarre le tuple.

        SELECT *
        FROM table_name
        --Permette di estrarre tutta la tabella
        --con tutti i campi di cui e' costituita.
```

```
18          SELECT *
19          FROM table_name
20          WHERE column1=X;
21          --Nelle Query in MySQL WHERE e' utilizzato
22          --come filtro delle righe attraverso
23          --specifiche condizioni.
24
25          --Inserire piu' tabelle in FROM equivale
26          --al prodotto cartesiano tra le due
27          --Se aggiungiamo anche WHERE allora
28          --l'operazione effettuata equivale
29          --alla join
30          SELECT F1.Nome, F2.Nome
31          FROM FORNITORE AS F1, AND FORNITORE AS F2
32          WHERE F1.Citta=F2.Citta
33              AND F1.PIVA!=F2.PIVA;
34          --Viene prima eseguita la FROM,
35          --successivamente la WHERE
36          --e come ultima viene eseguita la SELECT
37
38
```

Listing 3: SELECT Example

– **Except**: Implementa la sottrazione tra insiemi presente nell'Algebra-Relazione all'interno di MySQL, e per utilizzarla le tabelle devono essere caratterizzate dalla medesima arità, e inoltre ogni EXCEPT richiede di essere inserita tra due SELECT.

```
1           SELECT Fornitore
2           FROM FORNITURA
3               Except
4           SELECT Piva
5           FROM FORNITORE
6           WHERE  Nome='x';
7
```

Listing 4: EXCEPT Example

– **UNION e INTERSECT**: Implementano l'operazione di unione e l'operazione di intersezione (Anche se quest'ultima non viene mai utilizzata). Medesime regole di sintassi introdotte con la EXCEPT

– **View**: Una view è una tabella virtuale che è il risultato di operazioni in SQL, permettendo cosi' lo sviluppo di risultati parziali.

```
1           CREATE VIEW view_name(column1) AS
2           (SELECT column1
3           FROM table_name
4           WHERE condition);
5
```

Listing 5: CREATE View Example

Guardare Samsung Notes per esempi ed esercizi.

12 Lezione n12 11 11 2021

La formulazione delle Query utilizzata fino a questo momento, è detta **procedurale** (Sia in Algebra Relazionale che in MySQL), poiché dalla sua formulazione vengono definiti in maniera esplicita, non quello che viene richiesto, ma i passi da compiere per ottenere il risultato desiderato. Tramite l'introduzione degli Operatori **In**, **Not In**, **Exist**, **Not Exist** avremo sempre una formulazione procedurale che però raggiunge anche una formulazone dichiarativa.

12.1 In Not In Operators

L'operatore **IN/Not In** permette di verificare la presenza/assenza di una touple in un dato insieme specificato.

12.1.1 In Syntax

```
1         SELECT column_name
2         FROM table_name
3         WHERE column_name IN (SELECT column_name
4                               FROM table_name);
5
```

In particolare verrà presa ogni riga di table_name, tale per cui il valore colum_name è presente nell'insieme specificato e infine tramite la select viene effettuata la proiezione rispetto a column_name.

IN traduce il Test Di Appartenenza

Si noti che molti BDMS necessitano che il nome delle colonne (colum_name) debbano avere lo stesso nome, anche se su carta noi rilassiamo questa particolare situazione (Non è necessario effettuare ridenominazione al compito).

Esempi su Samsung Notes

12.2 Exist, Not Exist

L'operatore **Exist/Not Exist** permette di verificare se il risultato di una query innestata è vuoto (non contiene tuple) o no.

12.2.1 Exist Syntax

```
1         SELECT column_name
2         FROM table_name
3         WHERE EXISTS (SELECT column_name
4                       FROM table_name
5                       WHERE condition);
6
```

Per ogni riga di fornitore verrà verificato se esiste almeno un elemento all'interno dell'insieme specificato, e successivamente verrà effettuata la proiezione rispetto a column_name.

Esempi di Riepilogo

12.3 Funzioni di aggregazione

Le funzioni di aggregazione sono particolari funzioni che operano su più righe. In generale, non tutti i database supportano le stesse funzioni, per cui è bene controllare sempre il manuale. Le funzioni più comuni sono:

- **COUNT()**:Per effettuare conteggi sul gruppo; In particolare se tra le parentesi non è specificata un espressione, verrà effettuato il conteggio di tutte le righe presenti nel gruppo indipendentemente dai valori assunti, se invece è specificata, allora verranno conteggiate righe che hanno quell'espressione(column_name) non nulla. (Utilizzato principalmente con GROUP BY) ;
- **SUM()**: Per le somme;
- **MAX()** e **MIN()**;
- **AVG()** Per calcolare la media;

12.4 Group By

GROUP BY serve a specificare quali sono i campi su cui effettuare raggruppamenti; in particolare, il motore di query, esaminerà i campi di ogni riga e la inserirà nel gruppo corrispondente. Successivamente ad avere specificato una clausola in GROUP BY allora di conseguenza, deve essere specificata nella clausola SELECT o un campo specificato nella clausola GROUP BY oppure una funzione di aggregazione, questo perché quando vengono aggregate le righe il database deve sapere come comportarsi per ogni riga da restituire; inoltre **non è possibile** inserire nella where un'aggregazione perché non si è ancora visualizzato rispetto a cosa effettuare il raggruppamento.

VARI ESEMPI ED ESERCIZI PER CASA

13 Lezione n13 13 10 2021

13.1 Riepilogo regole clausola Group By

1. All'interno della stessa Query posso specificare più aggregati nella Select ma devono essere necessariamente coerenti con il raggruppamento effettuato.
2. Non è possibile inserire nella Where un'aggregazione.
3. Non è possibile innestare operatori di aggregazione dentro ulteriori operatori aggregazione. Non posso quindi coinvolgere elementi appartenenti a touple (righe) diverse.
4. Posso coinvolgere elementi di una stessa tupla (ES: sum(quantità * PrezzoUnitario))

13.2 HAVING

Nel linguaggio SQL la clausola Having mi permette di aggiungere una condizione di tipo aggregato sui gruppi creati con Group By.

```
1    SELECT ...
2    FROM table_name
3    GROUP BY att1,...,attn
4    HAVING cond
```

Listing 6: Having

L'interrogazione SELECT visualizza soltanto i dati aggregati che soddisfano la condizione indicata in HAVING. La condizione è un'espressione booleana coposta da costanti e operatori aggregati.

13.3 DISTINCT

La clausola **Distinct** è utilizzata per ritornare solamente valori distinti.

In particolare, si ha che in SQL ogni tabella non è un insieme di tuple (come in Algebra che è rientrante rispetto al concetto di tabella), ma vengono interpretate come multiinsieme, e quindi i risultati intermedi o risultati di query possono contenere valori duplicati.

```
1    SELECT DISTINCT column1, column2, ...
2    FROM table_name
```

Listing 7: SELECT

Eliminare i duplicati dal risultato di una query, ha un costo quadratico rispetto al numero di tuple e quindi per non incorrere in tali costi non è un operazione svolta di default (Ecco perhè SQL non è rientrante rispetto al concetto di tabella).

13.4 Definizione Relazione e Database

Quando parliamo di **Relazione** dobbiamo essere specifici, poiché potremmo riferirci a un' **Istanza** o a uno **Schema**:

– **Schema di Relazione**:
Indica il nome di relazione seguito da un insieme di attributi, ciascuno dei quali è associato ad un dominio, che rappresenta l'insieme dei possibili valori che può assumere che può assumere ciacuna tupla dell'istanza della relazione.

$$nomeRelazione(A_1 : D_1, A_2 : D_2, ..., A_n : D_n)$$

– **Istanza di relazione**:
Dato uno schema di relazione $R(A_1 : D_1, A_2 : D_2, ..., A_n : D_n)$ un' istanza r dello schema è un sottoinsieme del prodotto cartesiano

$$r \subseteq D_1 \times D_2 \times ... \times D_n$$

Osserviamo quindi che una sostanziale differenza tra **Schema** e **Istanza**, è che lo Schema non presenta tuple, invece l'istanza di una relazione si.

Anche quando parliamo di **Database** dobbiamo distinguere in:

– **Schema di Database**:
È l'insieme di schemi relazionali che definisce il nostro Database.
– **Istanza di Database**:
È l'insieme di istanze di schemi relazionali, una per ogni schema di relazione presente nello schema di Database.

13.5 Domande Esame

– Definizione di chiave nel modello relazionale;
– Definizione di chiave esterna; (nel database);
– Definizione dipendenza funzionale (Che vedremo nella prossima lezione)

13.6 Approfondimenti Join

In verità vi sono diversi tipi di Join:

– **Join Naturale**: Non presente condizioni esplicite, e implicitamente è una join che viene effettuata sulle chiavi esterne o sugli attributi omonimi. (Non viene utilizzata proprio perché non descrive in maniere esplicita l'operazione che si sta effettuando)
– **Equi-Join**: Join in cui tutte le condizioni sono solo uguaglianze.
– Θ **Join**: Join con condizioni generiche, quelle che utilizziamo.
– **Left outer Join**: Join che restituiscono anche le touple a sinistra che non hanno effettuato matching con le touple a destra. La toupla di partenza è prolungata con valori NULL.
– **Full outer join**:Join che restituisce le tuple di A (tabella a sinistra) che non hanno effettuato match con tuple di B(tabelle a destra) vengono inserite e prolungate con valori NULL e viceversa.
– Ultimo operatore ma non si è capito a che cazzo serve e si indica con il simbolo diviso : $\div$, ma ha detto di non utilizzarlo che crea effetti disastrosi.

14 Lezione n14 15 11 2021

14.1 Dipendenza Funzionale

Una **Dipendenza Funzionale** è un particolare **vincolo** per il modello relazionale che descrive legami di tipo funzionale tra gli attributi di una relazione.

Prima di proseguire con la definizione formale di dipendenza funzionale, soffermiamoci sulla definizione di funzione. Che cos'è una funzione? Una funzione esprime una relazione tra due insiemi, detti dominio e codominio, tale che ad ogni elemento del dominio viene associato uno e uno solo elemento del codominio.

- **DEFINIZIONE**:
  Dato uno Schema Relazionale $R(A_1, ..., A_n)$ e un Istanza Relazionale r, si dice che esiste su r una dipendenza funzionale tra X e Y se per ogni coppia di tuple t_1 e t_2 di r aventi gli stessi valori sugli attributi X, t_1, t_2 hanno gli stessi valori anche sugli attributi Y.

$$X \text{ e } Y \subseteq \{A_1, ..., A_n\}$$

$$\forall t_1, t_2 \in r \text{ se } t_1[x_1] = t_2[x_1] \wedge .... \wedge t_1[x_n] = t_2[x_n] \Rightarrow t_1[y_1] = t_2[y_1] \wedge .... \wedge t_1[y_k] = t_2[y_k]$$

  Generalmente indicata con: $X \rightarrow Y$

Una dipendenza funzionale può anche essere formata da diversi attributi (sia nella parte destra che nella parte sinistra)

1. $att_1, att_2 \rightarrow att_3$:
   Significa che 2 tuple con valore di att1 e valore di att2 uguali, devono avere necessariamente lo stesso valore dell'att3.
2. $att_1 \rightarrow att_3; att_2 \rightarrow att_3$:
   Non ha lo stesso significato della prima, poichè in questo caso stiamo ammettendo che ci siano due tuple con att1 uguale e att2 diverso che hanno l'att3 uguale.

N.B. La parte sinistra di una dipendenza funzionale non può essere divisa per crearne 2 distinte. Invece, la parte destra di una dipendenza funzionale può sempre essere spezzata per dar luogo a 2 dipendenze funzionali.

La dipendenza funzionale è simile al vincolo di chiave, si supponga di avere la seguente relazione:

$$Nome\_tabella(att_1, att_2, att_3, att_4, att_5)$$

$$att_1, att_3 \rightarrow att_2, att_4, att_5$$

La formulazione sopra riportata esprime un vincolo di chiave tramite una dipendenza funzionale.

In **SQL** non si possono definire Dipendeze Funzionali in maniera diretta, ma al più posso scrivere delle procedure, ovvero parti di codice eseguite ogni volta che si effattua un INSERT (Popolamento)

```
1     CREATE PROCEDURE (...)
2     CREATE TRIGGER ... INSERT
```

Listing 8: Procedura

Tramite le dipendenze funzionali, posso notare se una determinata basi di dati è caratterizzata da una certa ridondanza o meno. Ricordiamo che la ridondanza nelle basi di dati lascia spazio ad eventuali errori.

In particolare se una dipendenza funzionale (parte sinistra) è caratterizzata da una chiave allora non è indice di ridondanza, in caso contrario si.

15 Lezione N15 19 11 2021

15.1 Ancora sulle Dipendenze Funzionali ...

Le dipendenze funzionali, permettono anche di verificare la "solidità" della struttura di una base di dati (DB); in particolare si ha, come detto in precedenza, che presa una Dipendenza Funzionale del tipo $X \to Y$, se sulla parte sinistra si hanno attributi che non sono chiavi, allora significa che gli attributi a destra dipendono da una porzione di dati che possono ripetersi più volte e ciò indica ridondanza (Più tuple coincidono negli attributi facente parte dell'insieme X, quindi potrei scrivere infinite volte tali attributi).

15.1.1 Forma normale di Boyce-Codd

Supponiamo uno **Schema relazionale** e un **Insieme di Dipendenze Funzionali** su tale schema.

$$R(A_1, ..., A_n)F \text{ Insieme di Dipendenze Funzionali} \tag{1}$$

Allora si dice che R è in **Forma normale di Boyce-Codd** (Abbreviato BCNF) se

$$\forall(X \to Y) \text{ Non triviale}$$

L'insieme X è un sovrainsieme di una chiave, o meglio, una **Superchiave** (Non è indice quindi di ridondanza).

15.1.2 Grafo delle dipendenze: Det. chiavi candidate di uno Schema R

Un insieme è una possibile chiave candidata, quando identifica le tuple dell'istanza dello schema relazionale associato. Per determinare ciò, a partire dalle dipendenze funzionali, si utilizza il **Grafo delle dipendenze**. La procedura è molto meccanica e possiamo riassumerla nei seguenti passi:

1. Ogni Attributo andrà ad essere rappresentato tramite un Nodo;
2. Per ogni Dipendenza Funzionale creo un arco multiplo che parte dagli attributi di sinistra e arriva agli attributi a destra.
3. Successivamente se tramite un nodo, o un gruppo di nodi, riesco a raggiungere tutti i nodi rimanenti, ho trovato una Superchiave.

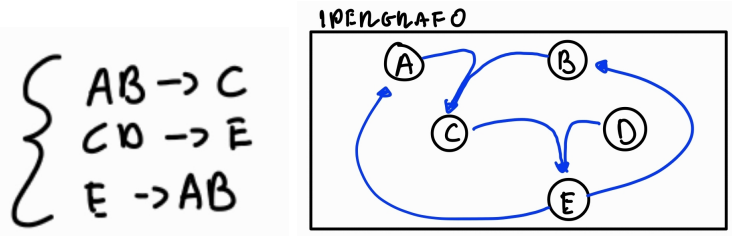


Figure 21: Esempio Grafo delle dipendenze

Tramite l'utilizzo di un **IperGrafo**, traduco il problema proposto, in un problema di raggiungibilità.

L'insieme di attributi in cui le tuple devono coincidere, è detta **Chiusura di** $(Att_1, ..., Att_n)$ e nel grafo corrisponde all'insieme di nodi, raggiungibili da $(Att_1, ..., Att_n)$.

Approccio non meccanico su Samsung Notes. Si basa sull'analizzare attentamente le implicazioni.

15.1.3 Dipendenze Funzionali e Copertura Minimale

La **Copertura Minimale** consiste in una riscrittura nella forma più compatta possibile, dell'insieme F in un insieme di Dipendenze Funzionali che descrive le medesime proprietà dell'insieme di partenza, dalla quale però non posso più togliere "pezzi".

PROCEDURA:

1. Si verifica se è possibili ridurre la parte sinistra delle Dipendenze Funzionali appartenenti al nostro insieme; possibile, solo nel caso in una porzione della parte sinistra della dipendenza funzionale analizzata, implica la porzione rimanente (È possibile ragionare in termini di chiusura).
2. Si verifica se è possibile ridurre la parte destra delle Dipendenze Funzionali appartenenti al nostro insieme; in particolare, si analizza una Dipendenza Funzionale, e considerando tutte, tranne quella analizzata, si "cerca" di ottenere il medesimo risultato. Se si ottiene, allora è superflua e può essere scartata, altrimenti no.

– **Esempio**:

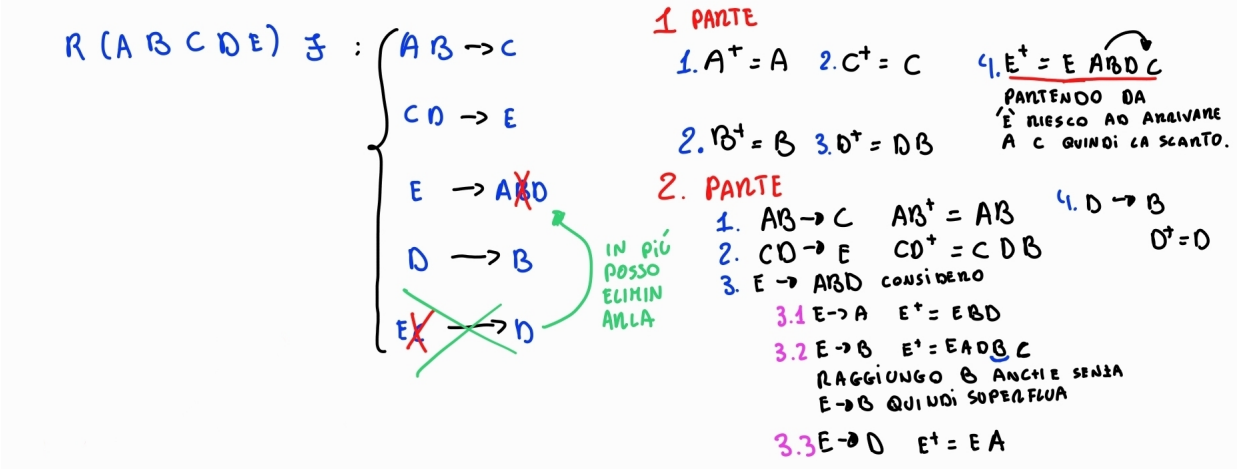


Figure 22: Esempio Copertura minimale

Effettuato il calcolo della copertura minimale di un insieme di Dipendenze Funzionali, può essere determinato l'insieme delle Chiavi Candidate.

16 Lezione n16 22 11 2021

Le Dipendenze Funzionali, permettono quindi:

- Esprimere relazioni di consisitenza non esprimibili con altri vincoli.
- Aiutano a sottolineare eventuali informazioni ridondanti nelle relazioni.

16.1 Decomposizione in BCNF

La **Decomposizione in BCNF** è un operazione che si basa sul partire dallo schema iniziale e ottenere un insieme di Schemi Relazionali definiti sugli attributi dello Schema Relazionale iniziale, tale per cui ciascuno degli schemi ottenuti è in BCNF e ciascuna dipendenza riportata nello schema è la proiezione delle dipendenze funzionali iniziali sullo schema. (O meglio, per ogni relazione scrivo tutti i vincoli implicati).

L'obiettivo è ottenere una decomposizione che rispetti 2 proprietà:

- Prese le nuove istanze si riesce ad ottenere quella originale tramite join. (Proprietà Lostless Join).
- Tutti i vincoli di integrità dello schema iniziale valgono anche per il nuovo schema.

In generale però è molto comune perdere le dipendenze funzionali (2 proprietà) quando si compie questo tipo di operazione, però il tutto è risolvibile effettuando controlli sulle entry (Quindi non è strettamente necessaria). Ecco perchè esiste un algoritmo per la decomposizione che permette di garantire la validità della prima proprietà ma non della seconda.

16.2 Algoritmo per la decomposizione

Dati i seguenti dati:

$$R(A, B, C, D, E, F)$$

$$X \subseteq \{A_1, ..., A_n\}; Y \subseteq (X^+ - X); Z = \{A_1, ...A_n\} - (X \cup Y);$$

Allora

$$\begin{cases} R_1(XY) \\ R_2(XZ) \end{cases}$$

È una decomposizione senza perdita di informazione. Nel caso in cui gli schemi appena ottenuti non dovessoro essere in BCNF, allora si riapplica il procedimento in maniera iterativa. Prima o poi gli schemi di relazione diventeranno in BCNF, poiché alla fine si otterranno al più schemi di relazione costituiti da due attributi (Se hanno due attributi allora sono sicuramente in BCNF). Bisogna tenere conto di quanto segue:

- X è l'insieme di attributi che stanno a sinistra di una dipendenza funzionale che viola la BCNF.
- Y viene definita come la chiusura di X privata di X.
- e in Z sono contenuti gli attributi rimanenti.

Esempi su Appunti Samsung Notes

17 Lezione n17 26 11 2020

17.1 Definizione Decomposizione

Dato uno schema relazionale:

$$R(A_1, ..., A_n)$$

$$R_1(A_1^1, ..., A_{k1}^1) \quad R_2(A_1^2, ..., A_{k2}^2) \quad ... \quad R_n(A_1^n, ..., A_{kn}^n)$$

L'insieme di relazioni si dice decomposizione di R se è tale che l'unione dei vari attributi presenti nelle relazioni è pari all'insieme di partenza.

$$\cup_{i=1}^{n} \cup_{j=1}^{ki} \{A_j^i\} = \{A_1, ..., A_n\}$$

Come già riportato in precedenza una **Decomposizione** di R in $R1, ..., Rn$ è in **BCNF** se $\forall i \in [1, ..., n]$ R_i è in **BCNF**. Per determinare se R_i è in BCNF occorre riferirsi all'insieme di dipendenze funzionali F_i ottenute proiettando l'insieme di dipendenze funzionali F (definite su R) su R_i.

17.2 Definizione Third Normal Form

Dato il seguente schema relazionale e un insieme di dipendenze funzionali associato:

$$< R(A_1, ..., A_n), F >$$

R è in **3NF** se $\forall X \rightarrow Y \in F$ in forma canonica vale almeno una delle seguenti condizioni:

1. X è un chiave.
2. Y appartiene ad una chiave.

17.2.1 Corollario 1

La relazione R è in **BCNF**, allora è in **3NF**. Ma non è vero il viceversa.

17.3 Corollario 2

Esiste sempre una scomposizione in **3NF**, senza perdita di informazioni e senza perdita di dipendenze funzionali.

Supponiamo infatti di avere il seguente schema e le seguenti Dipendenze Funzionali:

$$R(ABCDEF)$$

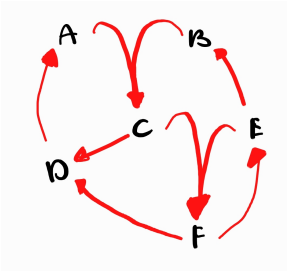


Figure 23: Grafo Dipendenze Funzionali

Per ogni dipendenza funzionale, si effettua l'elaborazione di uno schema con associato tutte le dipendenze funzionali e le proiezioni coinvolte. Alla quale viene applicata anche la copertura minimale.
Si ottiene quindi:

1. $R(ABC); \{AB \to C, [C \to A]\}$
2. $R_2(CEF); \{CE \to F, F \to E, [F \to EC]\}$
3. $R_3(DEF); \{F \to DE, [DE \to F]\}$
4. $R_4(CD); \{C \to D\}$
5. $R_5(DA); \{C \to D\}$
6. $R_6(EB); \{E \to B\}$

Nel caso non vi si almeno uno schema che presenta una chiave, allora sarà necessario, per garantire la **Lostless Join**, uno schema relazionale che presenta come membri, i membri di una chiave, e un insieme vuoto di dipendenze funzionali associato.

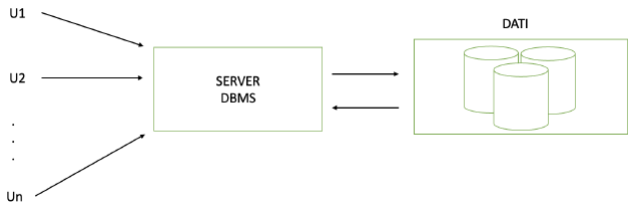
17.3.1 Decomposizione Minima

Alcune volte può essere richiesto di verificare l'esistenza di coppie di relazioni che possono essere accorpate senza perdere la condizione di **3NF**. Generalmente è conveniente effettuare gli accorpamenti su eventuali cicli.

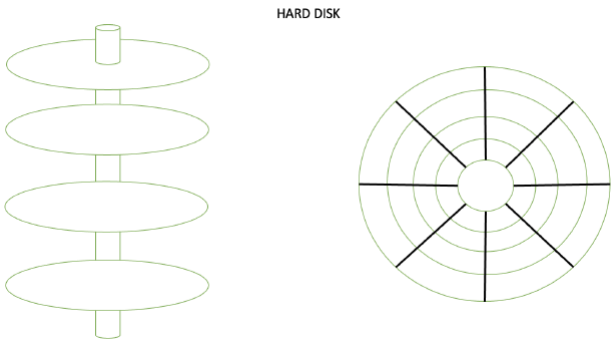
18 Lezione n18 29 11 2021

18.1 Sistema di indicizzazione

Al fine di indicizzare correttamente i dati vengono utilizzate strutture dati specifiche; un indice indica ove è possibile trovare una tupla.
I DMBS sono basati, generalmente su un'architettura di tipo *client-server*; in particolare il server è continuamente in ascolto, in ingresso riceve degli statement SQL sulla base dei quali sin interfaccia con un insieme di dati. (La codifica dei dati avviene in maniera proprietaria).



La gestione delle tuple, pervede che i dati che contengono sia **Durabili**, quindi è necessario, per garantire tale caratteristica, l'uso di memorie secondarie non volatili (hard disk).



Le strutture dati da utilizzare per l'organizzazione delle tuple, devono necessariamente avere un costo logaritmico, per la lentezza che caratterizza l'hard-disk e l'elevato quantitativo di utenti che ha la necessità di accedere alle informazioni (i costi lineari si accumulerebbero e questo porterebbe a temi di attesa molto elevati). Si osservi, che la testina che scorre sul disco magnetico, non preleva solamente una singola parola (come accade nella ram) ma un settore (blocco di dati–¿pagina) della traccia analizzata.

Quindi si avrà che una singola tabella sarà organizzata in una serie di pagine, e quindi il numero di accessi non varia non al variare delle tuple analizzate, ma al variare del numero di pagine.

$$N + x$$

Ove **N** indica l'intervallo di tempo necessario per accedere a n pagine e x che indica il tempo necessario a trovare l'elemento richiesto (A scorrere le tuple presenti nelle pagine) Ma lo scorrimento delle pagine e quindi la visione del contenuto della pagina ottenuta avviene in Memoria Centrale (Architettura 1000 volte più veloce della memoria non volatile) ed è quindi trascurabile.

18.1.1 Memorizzazione dei dati in maniera seriale

Il modo più facile da implementare per memorizzare le tuple, poiché quest'ultime vengono memorizzate in cascata, a seconda dell'ordine d'inserimento e quando una pagina si satura se ne crea un'altra. Analizziamo la complessità, trascurando eventuali tempi di controllo.

Inserimento	$\theta(1)$
Ricerca	$\theta(N)$

18.1.2 Memorizzazione dei dati in maniera sequenziale

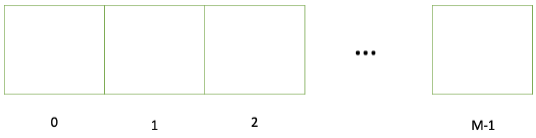
Alternativa al metodo precedente, si inseriscono le tuple ordinate rispetto ad un attributo di ricerca principale (chiave). Valutiamo ora la complessità trascurando eventuali tempi di frammentazione (Una pagina dopo sequenze di Rim/Ins potrebbe risultare vuota e quindi allocata ma non utilizzata).

Inserimento	$\theta(N)$
Ricerca	$\theta(N)$

Nonostante siano di facile implementazione, i costi lineari non sono compatibili con il nostro problema, quindi inutilizzabili.

18.2 Tabelle Hash

Tale struttura dati è organizzata in celle, ognuna delle quali rappresenta una determinata pagina (blocco dati).



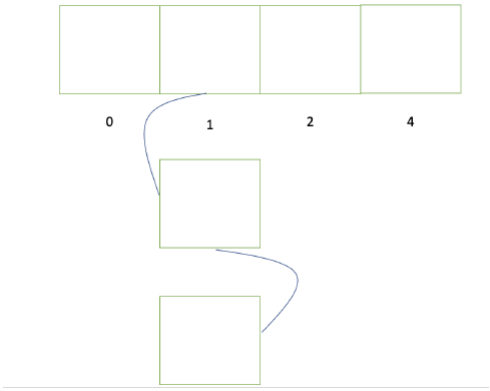
La struttura della funzione di hashing che è utilizzata per inserire le tuple in ingresso:

$$Hash(K) = K\%M$$

Ove K è la chiave corrispondente alla tupla in ingresso. Ogni pagina può contenere al suo interno un numero finito di tuple, e viene riempita finchè non avvengono trabocchi. Ad ogni inserimento, si effettua un controllo solo sulla pagina corrente, poiché data la funzione di hashing se la tupla inserita non è presente nella cella corrente non è presente da nessun'altra parte.

18.2.1 Liste di collisioni

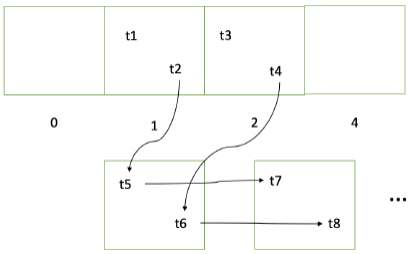
Una prima soluzione alla gestione dei trabocchi, avviene tramite le liste di collisioni; in particolare ad ogni trabocco aggiungo il puntatore alla prossima pagina di conseguenza **Accesso** e **Inserimento** diventano lineari rispetto alla lunghezza della lista di collisione.



Purtroppo con sequenza d'inserimenti e rimozioni potrei andare in contro a problemi di frammentazione.

18.2.2 Unica Lista di collisione

Un miglioramento del caso precedente, può essere ottenuto utilizzando un'unica lista di trabocco. Man a mano che si hanno collisioni si elaborano liste di trabocco logiche e si inseriscono i puntatori tra i record:



Questo comporta un numero minore di pagine di trabocco. I singoli record anziché puntare ad altri record possono puntare alla pagina dedicata.

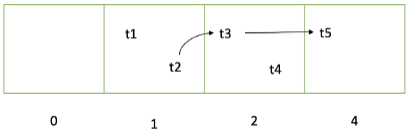
Il lato negativo e che se vi sono una moltitudine di trabocchi è richiesto un accesso a più pagine. Bisogna scegliere quindi un valore di M ragionevole; in particolare:

$$M \geq \frac{N_{totElementi}}{N_{totEleInPagina}}$$

Si noti che è sempre meglio spostarsi di un 10% rispetto al valore esatto.

18.2.3 Utilizzo Area primaria

Generalmente, invece di utilizzare una lista di trabocchi esterna (Che viene allocata ove c'è spazio disponibile) si può inserire la tupla nella prima pagina che presenta una posizione libera.



Questo metodo è preferibile se M può essere scelto in maniera preventiva in modo tale da non creare pagine aggiuntive rispetto a quelle già allocate e ridurre il tempo di accesso alle liste di collisione. Inoltre le pagine adiacenti devo necessariamente non essere allocate in maniera contigua, così da dare il tempo necessario al puntatore per accedere totalmente alla pagina senza dover ri completare il giro.

18.2.4 Hashing Lineare a indirizzamento aperto

Questo sistema adotta lo stesso schema precedente ma evita i puntatori. Ogni volta che devo allocare una tupla che subisce collisione anziché assegnarle K come chiave le assegno K + i, dove i indica l'i-esima collisione.

$$H(k) = (K + i)\%M$$

Nonostante questo sistema sia più leggero del precedente perché non tiene conto dei puntatori soffre di **agglomerazione primaria**.
L'agglomerazione primaria tende ad aumentare la probabilità di una singola pagina di essere assegnata come luogo per mantenere la tupla traboccata. Una tabella hash per essere ottima deve avere, per ogni cella, o in questo caso pagina, una probabilità di inserimento di $\frac{1}{M}$. Rischiano quindi di esserci trabocchi a catena. Per risolvere questo problema, i viene scelto in maniera causale. Tutte le tecniche viste fino a questo momento sono dette di **hashing statico**. Ovvero durante l'inserimento di tuple la dimensione della tabella non varia.

19 Lezione n19 3/12/2021

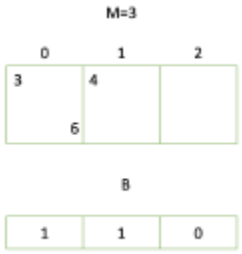
19.1 Hashing Dinamico

Precedentemente abbiamo visto tecniche di hashing statico, andiamo ad analizzare ora tecniche di **Hashing Dinamico** che comporta la modifica dell'area primaria nel momento in cui vengono inserite/rimosse tuple.

19.1.1 Hashing Virtuale

l'hashing virtuale funziona nella seguente maniera:

1. Inizialmente si alloca l'area dati di un certo M di pagine contigue.
2. Si introduce un vettore binario B con size pari al numero di pagine dell'area dati. Quando viene inserita una tupla in una pagina, il corrispondente elemento di B viene sovrascritto con un bit a 1.
3. Si utilizza una funzione di hashing H_0 che, applicata ad una chiave k, produce un indirizzo tra 0 e $M-1$.
$$H_0(k) = k \quad mod \quad M$$



4. Se si dovesse verificare un trabocco allora si eseguono le seguenti opeazioni:
   a. Si raddoppia l'area primaria
   b. Si sostituisce la precedente funzione di hashing con la seguente:
   $$H_1(k) = k \quad mod \quad 2M$$
   c. Si ridistribuiscono le tuple presenti nella pagina traboccata più la tupla che ha generato il trabocco tramite la nuova funzione di hashing. (**Caso sfortunato**: non ho risolto il trabocco deve raddoppiare nuovamente)

Generalizzando la formula di hashing ad ogni nuovo trabocco sarà:

$$H_i(k) = k \quad mod \quad 2^i M$$

ove i indica il numero di trabocchi incontrati

Il vettore di Bit B risulta utile per la ricerca che viene effettuata con i seguenti passi:

1. Si calcola la chiave k tramite l'ultima funzione di hashing alla quale sono arrivato, supponiamo $H_i(k)$ e si visualizza il bit associato al numero di pagina ottenuto, si hanno due situazioni possibili:

   bit 1 : Scorro le tuple presenti e se la trovo bene, se non è presente l'elemento interessato allora mi fermo poiché quest'ultimo non sarà presente da nessun'altra parte.

   bit 0 : La pagina è sicuramente vuot, non posso concludere niente sull'esistenza o meno della tupla nell'intera struttura. Ricalcolo allora il numero di pagina rispetto alla precedente funzione di hashing $H_{i-1}(k)$. E ripeto il procedimento in maniera ricorsiva.

   Ci fermiamo per quando incontriamo il primo bit a 1 o quando arriviamo alla funzione di hashing $H_0(k)$

Il vettore di bit è contenuto in memoria centrale, quindi ogni qual volta effettuo un controllo sul bit non effettuo accessi al disco. Effettuo accessi al disco quando trovo un bit pari ad 1, quindi effettuo un solo accesso.

Costo ricerca: 1 Accesso al disco

L'inserimento quando non vi è il trabocco costa 2 accessi al disco, poiché leggo la pagina e la riscrivo. Quando vi è trabocco si ha un costo di 3 accessi al disco, leggo la pagina traboccata, scrittura di alcune sulla medesima e scrittura di altre su una nuova pagina.

Si osservi che una proprietà dell'algebra modulare è che: Quindi a ogni trabocco ho la necessità

$$K \mod 2M = \begin{cases} K \mod M \\ (K \mod M) + M \end{cases}$$

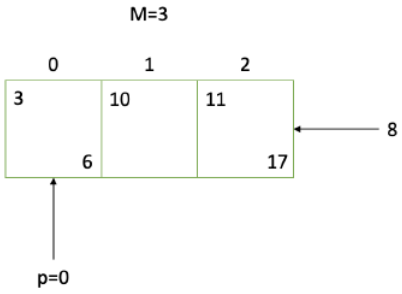
di raddoppiare l'area dati, e non posso in alcun modo aggiungere una sola pagina poiché non avrei nessuna relazione tra la precedente funzione di hashing e la corrente. Tuttavia questa tecnica è definita di hashing virtuale poiché non è sostenibile visto il costo esponenziale che la caratterizza.

20 trabocchi $\rightarrow 2^{20}$ volte size più grande
Se la size cresce esponenzialmente ho un altro problema, la struttura dati tende a essere sparsa.

19.1.2 Hashing lineare dinamico

Da non confondere con hashing lineare a indirizzamento aperto che è una tecnica statica.
Con L'**Hashing Lineare dinamico** l'idea di base delle organizzazioni dinamiche con espansione lineare è la divisione delle pagine, ma la pagina suddivisa non è quella che ha dato origine al trabocco ma la si sceglie secondo un ordine prefissato. Si introduce un parametro intero p che indica la pagina più a sinistra non ancora distribuita.

Nel caso di trabocchi si hanno 2 casi possibili:



a. p **non punta alla pagina che ha traboccato**, creo una lista nell'area secondaria per gestire i trabocchi (inserisco quindi la tupla che ha traboccato). Aggiungo una pagina alla fine dell'area primaria all'indirizzo $p + m$. Creo una nuova funzione di hashing:

$$H_1(k) = k \mod 2M$$

Ridistribuisco gli elementi della pagina puntata da p attraverso la nuova funzione di hashing (eventualmente anche la lista di trabocchi associata).

Si osservi che la pagina p non riscontra errori durante la ridistribuzione con la nuova funzione di hashing poichè le tuple andranno a finire al più nella pagina d'indice $p + M$ che esiste poiché l'abbiamo creata nei passi precedenti (abbiamo aumentato la size dell'area primaria di 1).

Successivamente alla ridistribuzione incremento p poiché deve puntare alla pagina più a sinistra che non ha subito riallocamenti.

b. p **punta alla pagina che ha traboccato**: Creo una pagina aggiuntiva. Rialloco le tuple, contenute nella pagina puntata, direttamente (senza creare la lista di trabocchi).

Si noti che l'aggiornamento della funzione di hashing con raddoppio di M si effettua solo al primo trabocco incontrato

Quando p diventa pari ad M allora vi è un reset della struttura: $p = 0$; $H_0(k) = k \mod M$; $M = 2M$.

Considrazioni:
Quindi ogni lista ha vita breve e se le liste non vengono eliminate significa che ci sono pochi trabocchi. Inoltre nel caso effettuassimo una **ricerca**, utilizzeremmo l'ultima funzione di hashing e se restituisse una pagina che non esiste (Non ho bisogno di accessi al disco per verificarlo)

$$N_{pagina} \leq p + M \text{ Vero: esiste; Falso: non esiste;}$$

Si utilizza la funzione di hashing precedente.

Domanda Esame:
Cosa faccio se, nel riallocare gli elementi usando, della pagina p, il metodo di Hashing lineare dinamico, la funzione di hashing mi restituisce il valore di una pagina che non esiste ? NIENTE! Non può accadere infatti che venga restituita una pagina con un valore superiore di $p + M$

19.1.3 Hashing estendibile

L'hash estendibile è un metodo che garantisce il reperimento di ogni record con non più di due accessi alla memori secondaria. Rispetto all'hash virtuale esso impiega un vettore B più complesso (ma non raddoppia l'area dati e non usa l'algebra modulare), che contiene i riferimenti alle pagine, detto **indice**; inoltre ogni celletta è associata a una stringa di bit;
A ogni pagina associo un numero intero $p^{'}$ (**Storia Dei Trabocchi**) che indica il numero di trabocchi che ha portato alla creazione di quella pagina.
Si tiene conto anche di un numero intero p che indica il **numero di volte che l'area dati dell'indice viene raddoppiata**.
Se $p^{'} < p$ allora l'indice è stato sufficientemente ridimensionato per tenere traccia di tutte le pagine in memoria principale. Quando $p^{'} = p$ viene raddoppiata la dimensione dell'indice.
Indice:
Si osservi che l'associazione di una tupla a una determinata pagina avviene tramite l'indice, o meglio, ogni celletta i-esima è associata a una stringa di bit e le tuple con chiave k codificata in binario, avente come ultimi bit tale stringa sono associate alla pagina $i - esima$.

L'inserimento avviene nel seguente modo:

- La chiave k associata a una tupla, viene codificata in binario.
- la **pseudochiave** ottenuta viene utilizzata per accedere al vettore B di dimensione 2^p; in particolare vengono presi in considerazione i p bit meno significativi della pseudochiave.

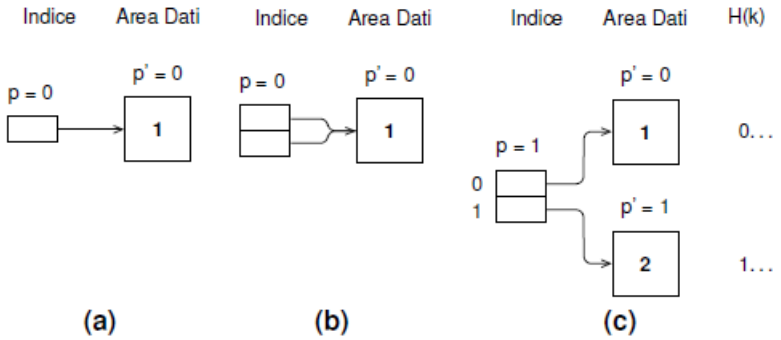


Figure 24: a) situazione iniziale; b) raddoppio del vettore dopo un trabocco c) modifica del vettore, sdoppiamento della pagina 1, aggiornamento p' e p

Gestione dei trabocchi: Se una tupla va inserita in una pagina non satura, l'operazione si conclude senza problemi. Quando si deve inserire una tupla in una pagina i-esima satura a cui è associato un certo valore p', si procede come segue:

1. Se $p^{'} = p$ l'operazione interessa la pagina dei dati e il vettore B.
   a. Aggiorno p di 1 e si raddoppia B e gli elementi di B raddoppiato contengono lo stesso riferimento alla pagina dati. Ora $p^{'} < p$ e si procede come nel prossimo caso per trattare la pagina.
2. Se $p^{'} < p$ l'operazione interessa solo la pagina dati.
   a. La pagina interessata viene sdoppiata; Alla vecchia e nuova pagina viene associato il valore $p^{'} = p^{'} + 1$

b. Ridistribuisco le tuple, convertendo in bit la loro chiave k basandomi rispetto all'indice vettore.

Se a seguito di un operazione di cancellazione, il contenuto di due pagine 'vicine' può essere memorizzato ina sola di esse, le due pagine si fonodono; alla nuova pagina viene associato un valore di $p^{'} = p^{'} - 1$ e si aggiorna l'indice B. Due pagine sono dette **vicine** se sono identificate dallo stesso valore $p^{'}$ e le tuple in esse contenuti differiscono solo per il $p^{'}$-esimo bit. Se si fondono le uniche due pagine 'vicine' con $p^{'} = p$ si dimezza B.
Osservazioni: Anche se come nell'hashing virtuale si utilizza il raddoppio, quest'ultimo viene effettuato non sulla size delle pagine totali, ma sul vettore B che è possibile mantenere in memoria centrale.
Inoltre potremo utilizzare un organizzazione ad albero (invece di un vettore di indirizzi) per realizzare la struttura secondaria (Vettore B) evitando così di "rappresentare tutte le cellette".

IL PROFESSORE HA DETTO DI VEDERE TALE CASISTICA DA SOLI SUL LIBRO MA SUL LIBRO NON C'è SCRITTO UN CAZZO.

19.2 Svantaggi Hashing

L'Hashing è una tecnica d'indicizzazione procedurale, ovvero è sufficiente memorizzare una procedura per creare gli indirizzi e non gli indirizzi stessi; per tale motivo viene detta tecnica d'indicizzazione di tipo **Primario** ovvero che la stessa funzione hash a determinare ove memorizzare la tupla.
Questo determina però **l'impossibilità** di preservare relazioni di ordine, di effettuare eventuali ricerche basate su range o ricerche rispetto a un insieme differente di chiavi.

19.3 Strutture ad Albero per dati permanenti

Per i motivi descritti in precedenza, si è scelta una via alternativa a quella procedurale, ovvero l'utilizzo di strutture ad albero generalizzate per l'utilizzo in memoria secondaria.(**Tecnica d'Indicizzazione Tabellare**
Perché ho specificato "per l'utilizzo in memoria secondaria"? Perché gli alberi AVL utilizzati in memoria temporanea non sono adatti per due motivi.

- La ricerca di un elemento dell'insieme può comportare un elevato numero di accessi alla memoria permanente.
- Gli algoritmi di bilanciamento potrebbero risultare molto onerosi per alberi memorizzati in memoria permanente.

Una soluzione al primo problema può venire memorizzando opportunamente più nodi dell'albero binario in unica pagina della memoria permanente. Rimane il problema però di come far evol-

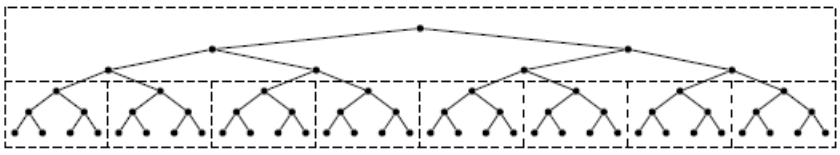


Figure 25: Albero ove ogni pagina contiene 7 nodi

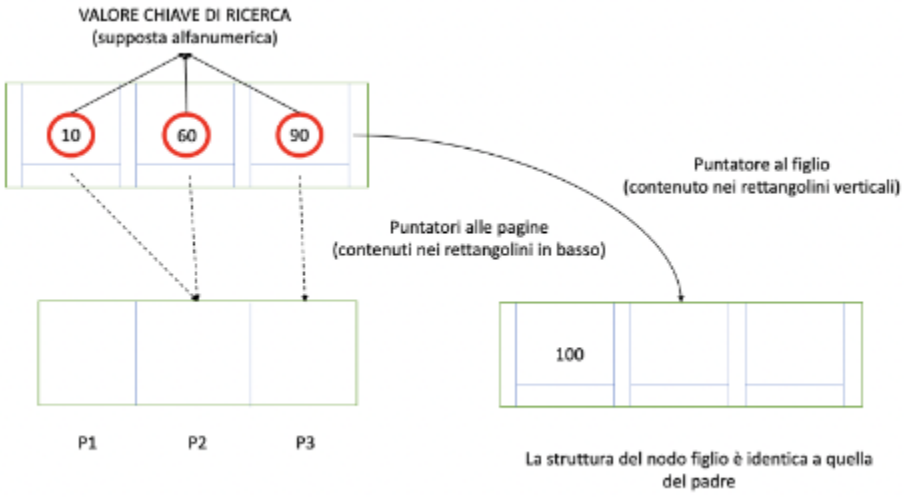
vere la struttura in caso d'inserimenti e cancellazioni in modo da evitare sbilanciamenti che ne pregiudicano le prestazioni.

Nella prossime pagine capiremo come raggruppare i nodi nelle pagine e contemporaneamente mantenere l'albero bilanciato grazie a una struttura detta **B-tree**.

20 Lezione N20 06/12/2021

20.1 B-tree

La struttura B-tree è un albero di ricerca che permette di compiere operazioni d'inserimento, cancellazione e ricerca in tempi armotizzati logaritmicamente, garantendo il bilanciamento ovvero che per ogni nodo, le altezze dei sottoalberi destro e sinistro differiscono al più di una unità.
In particolare si ha che ogni nodo è rappresentato da una pagina memorizzata in memoria secondaria, contenente riferimenti, scritti in maniera ordinata, a più pagine in cui sono memorizzate le tuple.



Ogni nodo all'interno di tale struttura può avere più figli, difatti per ogni chiave ho un sotto-albero destro e sinistro ed è bene osservare che per x chiavi, all'interno di uno stesso nodo, si hanno al più $x + 1$ figli. Un nodo è detto foglia se non ha puntatori a destra e sinistra per ogni entry.
In un B-tree i nodi sono omogenei, ovvero hanno tutti la medesima struttura, e non necessariamente deve essere occupata ogni sua posizione. In fase iniziale va scelto il grado del B-tree, ovvero il numero massimo di figli che può avere un nodo.
Tale parametro è importantissimo e dipende dal numero massimo di chiavi che è possibile inserire in un solo nodo.
Presa in considerazione la dimensione di una pagina (4kb) e dei puntatori (32/64 bit), due parametri che dipendono strettamente dal sistema in un uso, si riesce a ottenere il numero di chiavi che possono essere memorizzate all'interno di una stessa pagina, al variare del numero di bit necessari a rappresentare il valore della chiave stessa così da ottenere il grado del B-tree (Tale procedura è svolta in automatico dal sistema)

20.1.1 Ricerca

Il B-tree è un albero di ricerca, quindi vale il seguente criterio di ordinamento:

Presa una qualunque chiave il sotto-albero destro associato contiene valori di

chiave più grandi del valore corrente e il sotto-albero sinistro associato contiene valori di chiave più piccoli del valore corrente.

La ricerca avviene come in un qualsiasi albero di ricerca:

1. Accedo alla radice
2. Scandisco i valori delle chiavi:
   a. Il valore è contenuto nella pagina acceduta (nodo), ho finito
   b. Il valore non è contenuto nella pagina acceduta (nodo), non ho finito.
3. Verifico se esiste un possibile puntatore al nodo successivo tale per cui si ha:

   $$K_{sn} < K_{Cercata} < K_{dx}$$

   Ovvero il sotto-albero conterrà valori più grandi del valore di chiave a sinistra (se esiste) e più piccoli del valore di chiave a destra (Se esiste).
   Se non vi sono figli concludo che la chiave cercata non è memorizzata all'interno della struttura.
4. Accedo al nodo figlio, e GOTO 2.

Il costo della ricerca è pari all'altezza dell'albero. Però se l'albero non è bilanciato la profondità potrebbe essere lineare al numero dei nodi, si tratterà quindi di un albero degenere con costi esorbitanti. In realtà tale casistica, non avviene mai, vista l'efficienza in termini di bilanciamento del B-tree.
Generalmente un albero si dice bilanciato quando:

Per ogni nodo, il sotto-albero destro e sotto-albero sinistro differiscono al più di un elemento.

In un B-tree la proprietà di bilanciamento è ancora più forte:

Tutte le foglio sono posizionate alla medesima altezza.

Da ciò ne consegue che per ciascun nodo, il sotto-albero destro e sinistro hanno la stessa profondità e quindi il costo della ricerca diventa logaritmico::

$$O(log_x(n)) \quad \text{x indica il grado dei nodi}$$

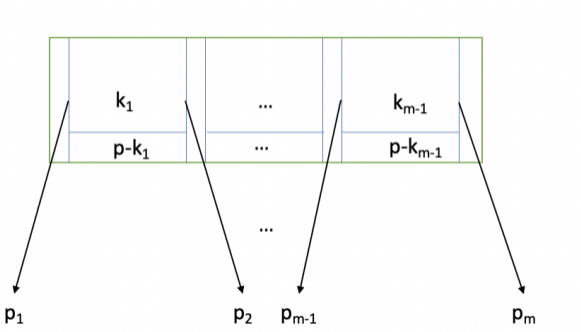
I costi logaritmici, pur analizzando una struttura in memoria secondaria, sono indice di efficienza data la base molto elevata del logaritmo.
Inoltre è importante sottolineare che la maggior parte dei nodi sono posizionati alla base dell'albero, quindi volendo i nodi dal penultimo al l'ultimo livello in su possono essere memorizzati in memoria centrale (occupano relativamente poco spazio) e trasportati in memoria secondaria e sovrascritti in memoria secondaria in determinati intervalli di tempo (Per evitare di perdere informazioni in caso di guasti).
Nel concreto i costi del B-tree non sono logaritmici e anche qui bisogna cercare di addensare le tuple, che si vogliono indicizzare, tutte in pochi nodi per evitare che la struttura sia sparsa.

Nota bene in un DMBS il B-tree è utilizzato per ottimizzare la ricerca e tramite linguaggio SQL è posibile creare una struttura con grado compatibile alla chiave scelta nel seguente modo:

```
1 CREATE INDEX
2 ON nome_tabella(Chiave)
```

20.1.2 Proprietà B-tree

– Ogni nodo è caratterizzato dalla seguente struttura:



– I puntatori ai figli del nodo sono al più m (Grado del B-tree)

– I valori delle chiavi di ricerca e i puntatori alle pagine (ove sono memorizzate le tuple associate) contenuti nel nodo sono al più $m - 1$

– Le chiavi sono ordinate (InsertionSort)

– Il sotto albero puntato da p_i contiene chiavi maggiori di k_{i-1} se $i > 1$ e minori di k_i se $i < m$

– Se un nodo ha x chiavi ha al più $x + 1$ figli

– Se un nodo ha x figli allora ha $x - 1$ chiavi

– Tutte le foglie sono alla stessa altezza (Forte proprietà di bilanciamento)

$$\forall \quad nodo \quad h_{sn} - h_{dx} = 0$$

– Ogni nodo non radice possiede almeno $\frac{m}{2} - 1$ chiavi (Sulla radice non può essere messo un vincolo sul numero di chiavi contenute).

20.1.3 Inserimenti B-tree

Presa un tupla e la radice di un B-tree effettuo i seguenti passi:

1. se la radice è foglia (non ha figli) inserisco in maniera ordinata:
   a. Se c'è posto allora ho finito
   b. Se non c'è spazio, allora prelevo tutte le chiavi in maniera ordinata e il mediano lo promuovo di un livello ponendo i restanti valori come nodi figli; in particolare i valori a sinistra vengono lasciati nel nodo esploso precedentemente, i valori a destra vengono inseriti in un nuovo nodo fratello (di quello esploso).
2. Se la radice non è foglia (presenta figli) allora scorro l'albero secondo il criterio di ordinamento fino a trovare un nodo foglia (le foglie stanno alla medesima altezza quindi il costo è pari all'altezza del B-tree).
   a. Se ci sono posti non occupati inserisco il valore in maniera ordinata.
   b. Se non ci sono posti (Nodo esploso) allora effettuo la promozione del mediano (quindi passa al nodo superiore) e i valori a destra li inserisco in un nodo fratello.

Se il nodo in cui deve essere promosso l'elemento mediano trabocca, si effettua la medesima procedura in maniera iterativa fin quando non si sono gestiti tutti i trabocchi.

Si osservi che l'ordine ha un costo trascurabile poiché per leggere una pagina, quest'ultima viene portata in memoria centrale, successivamente all'inserimento viene ordinata tramite InsertionSort e trascritta in memoria secondaria (Poiché la pagina è stata modificata) .

Procedendo in tal maniera l'albero cresce dal basso. Per quanto riguarda il costo dell'inserimento:

- Inserire un elemento nel nodo foglia corrispondente (senza trabocchi) ha costo pari ad h ove h è l'altezza dell'albero.
- Se la foglia corrispondente è piena si ha la necessità di creare nuove pagine (Split della foglia), quindi si effettuano 2 scritture. Supponendo che se una pagina è stata letta, non dobbiamo rileggere allora il costo sarà pari ad h accessi + 2 scritture.
  Nel caso in cui promuovendo l'elemento mediano si abbiano altri casi di overflow, queste 2 scritture vanno fatte per ogni livello dell'albero. Caso peggiore tutti i nodi ai livelli superiori vanno in overflow e devo quindi creare una nuova radice $2h + 1$ accessi alle pagine dell'albero.

Analizzando i costi precedentemente ottenuti è facile osservare che il numero di accessi è lineare all'altezza del B-tree, e questo ne consegue che è logaritmico rispetto al numero di elementi presenti (Come già spiegato precedentemente h è un valore piccolissimo vista il valore elevato della base del logaritmo).
Inoltre i nodi che sono presenti all'ultimo livello sono al più pari:

$$m^{h-1}$$

Invece i nodi presenti dal penultimo livello in su (per progressione geometrica):

$$\frac{m^{h-1} - 1}{m - 1}$$

In particolare stiamo osservando che il numero di nodi all'ultimo livello è al più m volte più grande del numero di nodi dal penultimo livello in su. Quindi la porzione di albero dal penultimo livello in su è molto più piccola tanto da stare in memoria centrale diminuendo i costi, e portata in memoria secondaria (in caso di guasti e quindi evitando perdite d'informazioni) a intervalli regolari.

20.1.4 Cancellazione nel B-tree

Presa il valore di chiave di una tupla per effettuare una cancellazione della stessa all'interno del B-tree si eseguono i seguenti passi:

1. Eseguo una ricerca
   a. Se non c'è ho finito
   b. Se c'è continuo con il passo 2.
2. Rimuovo direttamente poiché non cambia l'ordinamento effettuato precedentemente.
3. Verifico che il riempimento minimo sia garantito, ovvero che ogni nodo non radice abbiamo almeno $\frac{m}{2} - 1$ chiavi.

a. Se è garantito ho finito.
b. Non è garantita la proprietà, prelevo i valori del nodo corrente, il valore del padre e i valori del fratello adiacente; se ci sono valori sufficienti allora effettuo il passo 4 altrimenti passo 5.

4. Ho abbastanza valori quindi effettuo una rotazione, il valore del padre viene declassato nel nodo che non garantisce la proprietà; il valore a estrema sinistra del fratello viene promosso al nodo padre.(Non vi è propagazione ma interagiamo solo con i 3 nodi).

   È come se prendessi il mediano dei valori considerati e lo promuovessi a nuovo padre.

5. Non ho abbastanza valori quindi effettuo un accorpamento dei nodi adiacenti e quindi l'eliminazione si sposta a livello superiore.
   Prendo gli elementi interessati (valore del nodo corrente, il valore del padre e il valore del fratello adiacente) e li accorpo in unico nodo, eliminando il valore del padre dal nodo a livello superiore.
   Questa volta vi è propagazione ai livelli superiori e quindi si ha la necessità se la proprietà e valida per il nodo padre (Se è radice no) e quindi devo applicare ricorsivamente il passo 5. (Mano a mano che andiamo avanti l'albero scende in altezza)

   L'accorpamento funziona e non vi è overflow poiché il nodo corrente se siamo a tale passo non rispetta la proprietà di riempimento minimo, il nodo adiacente rispetta la proprietà ma non vi sono sufficienti valori quindi ha esattamente $\frac{m}{2} - 1$ valori e quindi accorpandoli insieme al padre creano un nuovo nodo pieno.

20.1.5 Verifica Altezza Albero

Ipotizzando un B-tree con le seguenti caratteristiche:

- **Grado** m
- **Altezza** h
- **Riempimento massimo** quindi ogni nodo ha esattamente m figli

Indicando con p_{max} il numero di pagine con riempimento massimo si avrà:

$$p_{max} = 1 + m + m^2 + ... + m^{h-1} = \frac{m^h - 1}{m - 1}$$

Il numero di tuple che possono essere indicizzate equivale a:

$$N_{max} = (m - 1) * p_{max} = m^{h-1}$$

Il numero di tuple dipende esponenzialmente, secondo h, dal grado del B-tree.
Supponiamo ora l'albero sia di riempimento minimo. Al primo livello abbiamo una pagina, la radice, che non ha vincoli di riempimento e avrà quindi 2 figli. Dal secondo livello in poi, avendo riempimento minimo, si avranno $\frac{m}{2}$ figli da ciò ne consegue un numero minimo di pagine pari a:

$$p_{min} = 1 + 2 + 2\frac{m}{2} + 2(\frac{m}{2})^2 + ... + 2(\frac{m}{2})^{h-2}$$

$$p_{min} = 1 + 2(\frac{m}{2})^{h-1} - 2 = 2(\frac{m}{2})^{h-1} - 1$$

Il numero di tuple invece che memorizzo, trascurando la radice, sarà di $\frac{m}{2} - 1$ per ogni nodo:

$$N_{min} = 1 + (p_{min})(\frac{m}{2} - 1) = 2(\frac{m}{2})^{h-1} - 1$$

Analizziamo ora le relazioni ricavate:

– **Lower Bound**:
  Il numero di tuple sarà pari a:

$$N \leq m^h - 1$$

  Quindi l'altezza del B-tree sarà al minimo:

$$h \geq log_m(N + 1)$$

– **Upper Bound**:
  Il numero di tuple sarà pari a:

$$N \geq 2(\frac{m}{2})^{h-1} - 1$$

  Quindi l'altezza sarà al massimo:

$$h \leq 1 + log_{m/2}(\frac{N+1}{2})$$

21 Lezione n21 10 12 2021

21.1 B^+-tree

In alcuni database commerciali si utilizza una variante del B-tree, ovvero B^+-tree. Si raffigura la struttura di un nodo per carpirne le principali differenze: La figura mostrata rappresenta

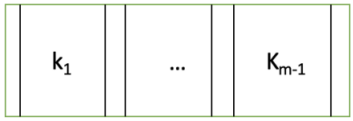


Figure 26: Nodo non foglia

la struttura di un **nodo non foglia** il quale si differenzia da un nodo non foglia del B-tree per la mancanza, per ogni chiave, il puntatore alle pagine ove trovare la tupla corrispondente (Il criterio di ordinamento è il medesimo). Questo permette di utilizzare lo spazio in eccesso (precedentemente assegnato ai puntatori alle pagine) per indicizzare ulteriori chiavi (A parità di contesto il grado della nuova struttura è più alto)

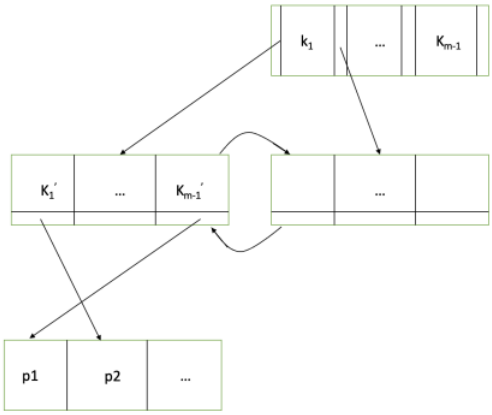


Figure 27: Struttura B^+-tree

Il puntatore ai figli è sostituito con un puntatore alle pagine per tutti i **nodi al livello foglia**; Inoltre a tale livello tutti i nodi sono caratterizzati da ulteriori puntatori (in avanti e all'indietro) verso i nodi fratelli, e dato che tutte le chiavi presenti ai livelli intermedi sono riportate anche all'ultimo livello, si elabora una lista concatenata contenente tutte le chiavi in ordine lessicografico (Permettendo così anche una ricerca per range).

In precedenza abbiamo sottolineato che il grado di tale struttura è più alto rispetto al B-tree, e perché tale caratteristica è vantaggiosa? Perché L'albero è più basso e quindi ne consegue un maggiore scarto tra al'ultimo livello e la restante parte dell'albero e contrasta inoltre la necessità di duplicare le chiavi ai livelli intermedi in un nodo foglia. Difatti se l'albero è basso e la maggiorparte delle chiavi si trova all'ultimo livello, allora le chiavi duplicate sono molto piccole (Si osservi che dal penultimo livello al primo, l'albero viene memorizzato in memoria centrale)

In generale l'indice e le pagine ove sono memorizzate le tuple sono separati e questo permette di associare più indici al medesimo gruppo di pagine. Inoltre si ha la possibilità di ordinare le

tuple in ordine lessicografico rispetto a una determinata chiave candidata e ciò risulterebbe vantaggioso nel caso in cui tramite ricerca su range si volesse accedere a più tuple contemporaneamente, ma ciò è possibile solo se il criterio di ordinamento delle tuple non differisce da quello utilizzato per l'indice (Medesima chiave di ricerca); In tale situazione L'indice è detto **clustered**.

Se un indice è clustered non si è costretti a rappresentare tutte le chiavi ma è sufficente memorizzarne solo alcune:
Preso un valore, vado in avanti rispetto al valore più piccolo e all'indietro rispetto al valore

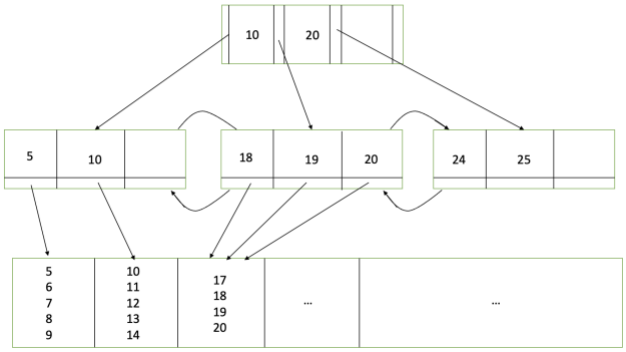
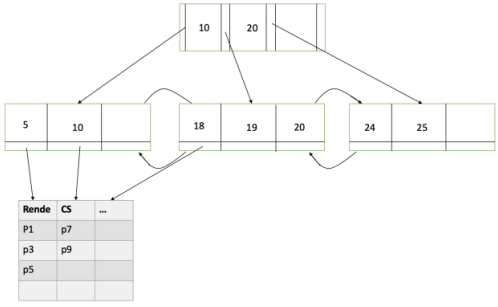


Figure 28: Indice Sparso

più grande, trovo la pagina associata e scandisco i valori fino a quando non trovo un valore più grande di quello cercato.
Un indice di tal tipo è detto **Sparso** e si utilizza per ridurre le dimensioni dell'indice (rispetto a uno denso) ed è il lato positivo.
Il lato negativo è che in fase di ricerca, per accorgersi che un record con una determinata chiave non è presente è necessario accedere alle pagine (Se l'indice è ben strutturato abbiamo bisogno di un solo accesso).

Un indice clustered può essere sparso o denso, il secondo indicizza tutti i record presenti nelle pagine.

Si noti che se l'attributo scelto, sulla quale basare l'indice, può essere qualunque ma nel caso non sia una chiave candidata allora per rendere compatibile la scelta effettuata con la struttura utilizzata si ha la necessità d'introdurre una struttura intermedia, una tabella:



La tabella è memorizzata nelle pagine e contiene ogni valore di chiave a cui viene associata la pagina ove trovarla. Le foglie del B^+-tree puntano alle pagine ove è memorizzata la tabella.

21.2 Modello Relazionale e Modello Semi-Strutturato

- **Modello Relazionale**:
  Il modello relazionale è un modello fortemente strutturato. Ciò vuol dire che qualunque cosa noi vogliamo rappresentare dobbiamo codificarla in una struttura rigida, ben precisa. Tutte le tuple sono uniformate e hanno la stessa struttura. Inoltre la struttura dei dati tipicamente non varia. Un'altra caratteristica è che lo schema e i dati sono ben distinti. Lo schema è più piccolo dei dati.
- **Modello Semi-Strutturato**:
  La differenza rispetto a quello strutturato è che la struttura è lasca. Ciò vuol dire che non tutti i dati che rappresentano informazioni riguardanti lo stesso concetto hanno la stessa forma. La struttura di ciascun dato può variare nel tempo. Lo schema può essere codificato nei dati. Lo schema può avere dimensioni paragonabili ai dati. Si chiama semi-strutturata perché la struttura non è totalmente assente, è solo meno rigida.
  Un modello de-struttrato potrebbe essere il testo libero.

21.3 XML

È il linguaggio standard per rappresentare i modelli semi/strutturati. XML è l'acronimo di Xtend-able-Markup Language. Questo linguaggio utilizza i delimitatori (anche detti Tag) per definire il tipo del dato che si sta codificando.

```
<biblioteca>
    <libro>
        <titolo> La Divina Commedia </titolo>
        <autore> Dante Alighieri </autore>
    </libro>
    <libro>
        <ISBM> 394-2445-2432 </ISBM>
        <autori>
            <autore> Pippo </autore>
            <autore> Topolino </autore>
        </autori>
        <titolo> Basi di Dati </titolo>
        <casaED> McGrawHILL </casaED>
    <libro>
<biblioteca>
```

Si osservi che tra due Tag è contenuto un elemento, e il nome del Tag può essere scelto autonomamente. Tutto ciò che viene scritto è codificato in UNICODE o ASCII. Questo modello è pensato per favorire la condivisione. Inoltre le informazioni possono essere rappresentate anche sotto forma di attributi.
Un **attributo** è un informazione che descrive l'elemento che sto rappresentando.

```
<biblioteca>
    ...
    <libro anno="1980" prezzo "29 euro">
        <titolo> La Divina Commedia </titolo>
        <autore> Dante Alighieri </autore>
    </libro>
    ...
<biblioteca>
```

È possibile associare una struttura comune ai vari elementi anche se stiamo parlando di un Modello Semi-strutturato. Difatti per guidare le applicazioni che devono compilare il

documento in XML, si correda una **guida di dati** sfruttando il linguaggio **Data Type Definition** DTD.
Più propriamente DTD è un oggetto associato ai documenti XML che indica quanto e come è definita la struttura dei suoi elementi.

```
<!ELEMENT biblioteca (libro*, curatore*)>
<!ELEMENT libro(ISBM?, titolo, autori | autore, casaED?)>
<!ELEMENT autori(autore+)>
<!ELEMENT ISBM #PCDATA>
<!ELEMENT autore #PCDATA>
<!ELEMENT casaED #PCDATA>
<!ELEMENT curatore #PCDATA>
<!ATTLIST libro identificatore ID>
<!ATTLIST libro prezzo CDATA>
<!ATTLIST libro curatore IDREF>
<!ATTLIST curatore identificativo ID>
```

- **?**: elementi e attributi che possono essere o meno presenti.
- *****: attributi che possono essere presenti in un numero indefinito
- — : OR (Sbarretta verticale ma che viene codificata da Latex come sbarretta orizzontale)
- +: Indica che è presente almeno un attributo
- **PCDATA**: Parsed Character Data, indica che è presente del testo e non dei veri e propri attributi.
- **ATTLIST** indica gli attributi che possono essere presenti in un elemento.
  - **ID**: identificatore di oggetto (univoco in locale, ovvero sul file ove ci si trova)
  - **CDATA**: caratteri
  - **IDREF**: riferimento ad un altro elemento.

Si ha la possibilità anche di inserire altro oltre al testo all'interno di uno stesso elemento:

```
    <!ELEMENT a ((b|c|d|#PCDATA)*)>
```

Tale elemento è detto **Mixed Element** e la sua definizione deve necessariamente essere il più essenziale possibile. Avremo in XML:

```
    <a>
        String1
        <b>...</b>
        String2
        <c>...</c>
    <a>
```

Esistono altri modi per definire "La guida ai dati" di un documento XML detto **XML schema** linguaggio che, essendo più espressivo e completo, permette di essere più preciso. Inoltre possiamo trovare una guida completa di XML sul sito www.w3.org/standards/xml/core.

22 Lezione n22 13 12 2021

22.1 Rappresentazione di un documento XML

I documenti XML hanno una struttura gerarchica e possono essere interpretati concettualmente come una struttura ad albero.
È facile osservare che le caratteristiche principali di un documento XML sono:

- I tag d'inizio e fine, che delimitano gli elementi sono nidificati correttamente.
- Un singolo elemento "radice" contiene tutti gli altri.

Queste caratteristiche assomigliano a quelle degli alberi. Però per definizione un albero è un grafo non orientato connesso e senza cicli, ma più propriamente la struttura che andremo a ottenere è un **grafo diretto aciclico** (Stiamo rilassando la definizione di albero per semplicità).
In particolare rappresentando elementi, sotto elementi, testo e attributi dall'elemento radice e sino al livello più basso otteniamo:

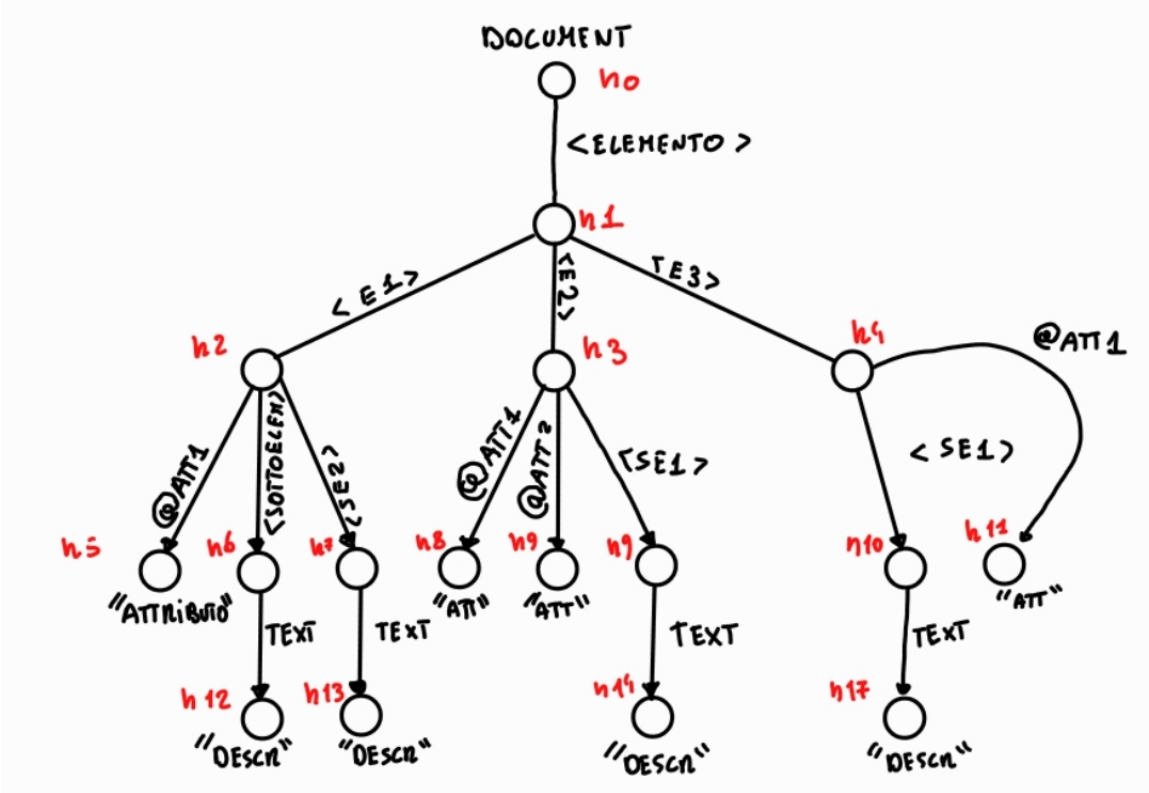


Figure 29: Esempio

Si osserva che gli attributi sono rappresentati come nodi, ma distinti tramite "@" per indicare che sono figli che non hanno successori.

22.2 XPath

XPath definisce una sintassi che identifica una o più componenti interni (Elementi, attributi, text ecc.) di un documento XML e viene ampiamente ed esclusivamente utilizzato per accedere

ai dati codificati in XML (Estrapolare Dati non Riorganizzarli) ottimale per effettuare le Query. In XPath abbiamo diverse relazioni tra i nodi:

- **Parent**:
  Permette di salire di un livello (accedere al genitore di un nodo) nell'albero dal nodo corrente.
- **Child o /** Consente di scendere di un livello (accedere al figlio di un nodo) nell'albero dal nodo corrente.
- **Siblings** Permette di accedere ai fratelli del nodo corrente.
- **Ancestor** Permette di salire di un qualunque numero di livelli dal nodo corrente (Antenati).
- **Descendant o //** Permette di scendere di un qualunque numero di livelli dal nodo corrente (Discendenti).
- **Ancestor-or-self** Variante di Ancestor, include anche il nodo corrente.
- **Descendant-or-self** Variante di Descendant, include anche il nodo corrente.

XPath consente di accedere a un informazione di nostro interesse, esprimendo (grazie all'utilizzo anche di relazioni) i cammini che devono essere percorsi a partire dalla radice dell'albero per raggiungere il nodo che contiene l'informazione.

$$Document("MioXML")/Child :: Elemento//Child :: E1/Child :: SE1$$

$$Document("MioXML")/Elemento/E1/SE1$$

$$Document("MioXML")/Elemento/E1/SE1/Text()$$

Si osserva che le prime due sono espressioni sono equivalenti (la seconda è scritta in forma ridotta) e nella terza per ottenere il testo associato a un nodo viene utilizzata la funzione **Text()** (Nelle prime viene restituito <SE1> Descr </SE1>).

22.2.1 Filtro di Selezione

Per esprimere una condizione si utilizza il **Filtro Di Selezione** che consente di esprimere una condizione che deve essere verificata per restituire il nodo.

$$Document(...)//E2[./SE1/Text() = x]$$

Per ciascun nodo E2, vado avanti e cerco il figlio SE1, se esiste verifico il contenuto e se la comparazione restituisce true allora restituisco E2.

$$Document(...)//E2[./SE1]/@Att1$$

$$Document(...)//E2[SE1]/@Att1$$

La comparazione qui indica che per tutti i nodi E2 viene verificato se esiste almeno un nodo SE1 e se esite, allora dal nodo E2 viene estrapolato Att1 (Le due espressioni sono equivalenti, quando il primo nodo è un nodo figlio non è necessario esprimere ./).

$$Document(...)//E2[./SE1 == Descr]$$

Qui viene indicato che per ogni nodo E2 per la quale viene verificata la comparazione, allora viene restituito. La query funziona poiché prima che venga effettuata la comparazione, il

membro destro e il membro sinistro vengono convertiti nei loro valori; in particolare il nodo SE1 viene convertito nel suo valore testuale che altro non è la concatenazione delle stringhe presenti al suo interno (Per nodi che hanno solo Text coincide con il valore restituito dalla funzione Text()).
Nel caso in cui esistessero più cammini E2/SE1 non vi è problema, perché ambo i membri vengono convertiti prima ancora della comparazione nella sequenza di valori, e la comparazione viene soddisfatta se esiste una coppia di un elemento preso dalla sequenza a sinistra e di un elemento preso dalla sequenza a destra che soddisfa la proprietà.

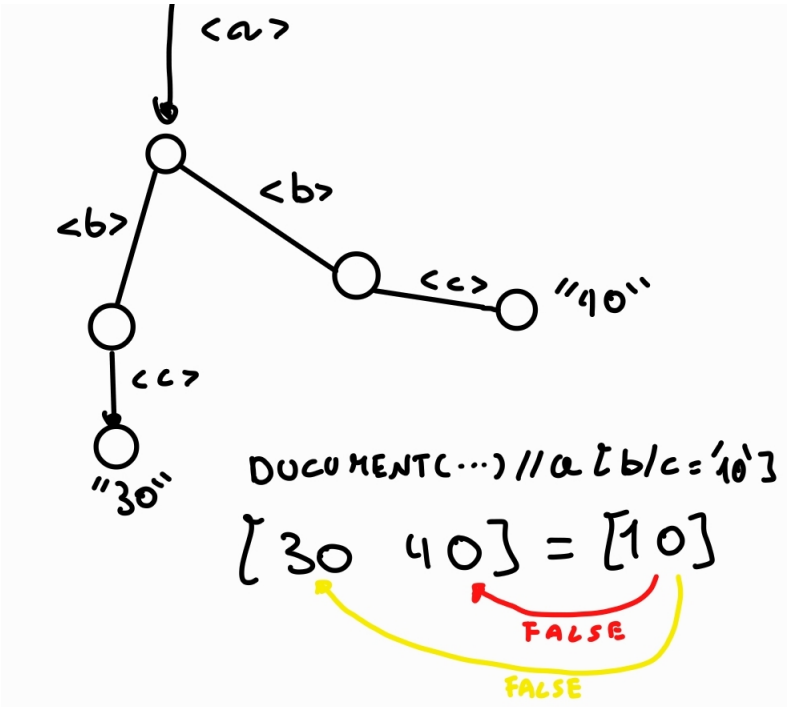


Figure 30: Esempio

22.2.2 WildCard

Si ha la possibilità di utilizzare la wildcard che permette di semplificare ulteriormente l'espressione da scrivere. In Informatica sappiamo che la wildcard "*" è utilizzata per interpretare una sequenza di caratteri alfanumerici. Qui viene indicata per interpretare più nodi.

$$Document(...)//*[./SE1]/@Att1$$

23 Lezione n23 16 12 2021

/Ricorda che successivamente a un'espressione XPath, il documento ottenuto è un documento XML solo se ben formato/

23.1 Linguaggio XQuery

XQuery è un linguaggio d'interrogazione per dati espressi in XML, e può essere dunque utilizzato per accedere a documenti semi-strutturati. XQuery è un linguaggio case sensitive, vi è una sintassi ma l'ordine delle clausole non è fisso come in MySQL. **Esempio:**

$$return < result > \{EspressioneXpath\} < /result >$$

Restituisce un documento XML ben formato contente gli elementi definiti all'interno dell'espressione.

Si noti che sono presenti le seguenti clausole:

- **Return:** costruisce il risultato dell'espressione XQuery.
- **For:** Permette di associare una variabile ad espressione e quindi di scannerizzare documenti XML.
- **Where:** filtra l'elenco di associazioni in base ad una condizione
- **Let:** costruisce un alias dell'intero risultato di un'espressione. (Crea un elenco con tutte le associazioni possibili). (Non sono sicuro su questo)

Vari esempi su documento word

23.2 Transazioni

Un sistema basato su DBMS rispetto a un sistema di gestione di dati è caratterizzato dalla gestione delle Transazioni.
Una **Transazione** è un programma in esecuzione che estrapola dati da una base di dati ed eventualmente modifica lo stato di quest'ultima.
Si supponga di avere una base di dati con la seguente configurazione:

Fattura(<u>numero,anno</u>,cliente,data)
Cliente(<u>CF</u>,nome,..)
Composizione(<u>nFattura,aFattura,prodotto</u>,qta,prezzoU)
Prodotto(<u>codice</u>,nome,...)

Per inserire una fattura:

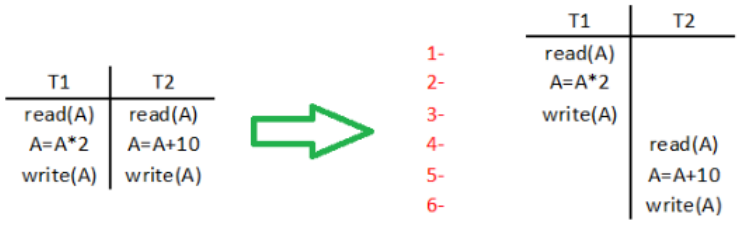
- Inserire 1 tupla in fattura
- Forse fare l'inserimento in cliente
- Forse fare n inserimenti in prodotti
- Fare n inserimenti in Composizione

L'inserimento della tupla in Fattura è una Transazione, concettualmente può essere vista come un'unica operazione che o è svolta interamente o non deve essere seguita, composta da più statement di basso livello.
Per delimitare le transazioni viene utilizzato **COMMIT** che rende definitivo tutto ciò che è stato scritto precedentemente. Se la transazione per qualsiasi motivo non viene completata con successo è necessario ripristinare la base di dati a uno stato consistente precedente (**ROLL BACK**).
Le proprietà garantite da un motore transazionale sono 4 e vengono sintetizzate tramite la sigla ACID:

- **Atomicità**: ha succesos se tutti gli statement della transazione hanno successo.
- **Consistenza**: una transazione agiste sullo stato consistente di una base di dati e lascia la base di dati in uno stato consistente.
- **Isolamento**: più transazioni che concorrentemente lavorano sulla base di dati sono isolate tra loro, viene quindi gestita la concorrenza delle modifiche.
- **Durabilità**: dopo il commit, se questa ha successo, lo stato del DBMS è definitivo e dura nel tempo.

23.2.1 Scheduling Seriale e Serializzabile

Lo **Schedule** è l'assegnazione di un numero ordinale a ogni operazione di un set d'istruzioni.



Dato un insieme di transazioni, dare un ordine di esecuzione sequenziale è detto **Scheduling Seriale**; si osservi che l'esecuzione di una transazione non può essere interrotta, ciò permette di garantire la massima consistenza (no race condition) ma con efficienza pessima.
Uno schedule S è **Serializzabile** se esiste S' sulle stesse transazioni di S tale che:

$$S \equiv S'$$

Ossia $\forall$ istanza del database possibile si ha:

$$S(D) = S'(D)$$

T1	T2
read(A)	read(A)
A=A+10	tmp=0,2*A
write(A)	A=A+tmp
read(B)	write(A)
B=B-10	read(B)
write(B)	B=B-tmp
	write B

	T1	T2
1-	read(A)	
2-	A=A+10	
3-	write(A)	
4-		read(A)
5-		tmp=0,2*A
6-		A=A+tmp
7-		write(A)
8-	read(B)	
9-	B=B-10	
10-	write(B)	
11-		read(B)
12-		B=B-tmp
13-		write B

I due possibili schedule seriali, per le due transazioni T_1 e T_2 sono:

$$T_1, T_2 \qquad T_2, T_1$$

Si osservi che i due schedule seriali non sono equivalenti perché non portano al medesimo risultato. Lo schedule a destra non è seriale in quanto le operazioni di T1 sono bloccate dall'esecuzione di T2 e viceversa, ma è serializzabile in quanto presenta il medesimo risultato dello schedule seriale T_1, T_2.

Uno schedule serializzabile è ragionevole se produce il medesimo risultato di uno seriale.

24 Lezione n24 17 12 2021

24.1 Conflict Serializable e View Serialazable

Dato S, l'insieme di tutti gli schedule serializzabili, il database solitamente genera uno schedule appartenente alle seguneti classi:

- **Conflict-Serializable**
- **View-Serializable**

24.1.1 Conflict Serializable

Proposizione 1:
Uno schedule S è **Conflict Serializable** se è **Conflict Equivalent** ad uno schedule seriale S':

$$S \equiv_c S'$$

Proposizione 2:
Uno schedule S è **Conflict Equivalent** a uno schedule S' se esiste una sequenza di scambi di operazioni non in conflitto tramite la quale si ottiene S' da S.
Proposizione 3:
Due operazioni sono in conflitto quando avvengono sulla medesima risorsa e almeno una è una write. Dato uno schedule è possibile dimostrare la sua equivalenza a uno seriale,

tramite scambi di operazioni non in conflitto, se ha successo possiamo affermare che esso sia Conflict Serializable (CS). Ma esiste un'ulteriore strategia, ovvero tramite l'utilizzo del **Grafo delle precedenze**. I nodi rappresentano le transazioni e ogni arco dalla transazione X alla transazione Y, indica che X che Y hanno una coppia di operazioni in conflitto tale che la prima si trova in X e la seconda in Y. Disegnato il grafo con i relativi archi possono verificarsi due situazioni possibili:

- Se il grafo ottenuto è ciclico, lo schedule analizzato **non è CS** e contiene quindi vincoli non ammissibili
- Se il grafo ottenuto è aciclico (Condizione sufficiente e necessaria), contiene vincoli ammissibili e da quest'ultimi è possibile ottenere uno schedule seriale. Quindi lo schedule **è CS**.

L'aciclicità può essere verificata tramite ordinamento topologico.

Si consideri il seguente schedule, indicando solo operazioni di read e write: non è CS in quanto

l'operazione "1" non può essere scambiata con la precedente o successiva. Inoltre il valore finale di A è gestito da T3, che non fa alcuna read ed è detta quindi **Blind Write**.

24.1.2 View Serializable

Proposizione 1:
Uno schedule S è **View Serializable** se è **View Equivalent** ad uno schedule $S^{'}$

$$S \equiv_v S^{'}$$

Proposizione 2:
Due schedule S e $S^{'}$ sono **View Equivalent** se:

1. $\forall$ risorsa x, le transazioni che leggono il valore iniziale di x in S leggono tale valore anche in S' (e viceversa).
2. $\forall$ risorsa x, se T_i legge il valore di x scritto da T_j in S, allora T_i legge il valore di x scritto da T_j in $S^{'}$
3. $\forall$ risorsa x, la transazione (se esiste) che scrive il valore di x in S' scriveva il valore finale di x in $S^{'}$ e viceversa.

24.2 Problema di Recuperabilità

Il problema della **Recuperability** si occupa di garantire sempre il ripristino di UNO stato consistente, in caso di errore (Lo stato consistente può essere qualunque).
Tale problema è strettamente legato alle **Commit**.
Nello specifico, data una coppia di transazioni, tale che una legge un valore scritta dall'altra, affinché ci sia recuperabilità, la prima transazione a fare commit deve essere quella che scrive.
Ma ciò non è sufficiente, poiché è importante non trascurare l'efficienza. Infatti supponendo che nel momento in cui venga effettuata la commit di una transazione, si crei un errore, diventa necessario fare Roll Back; per cui devono essere annullate tutte le operazioni effettuate dalla transazione corrente e tutte quelle effettuate da transazioni che dipendono dalla corrente. Pensando di avere n transazioni, vengono effettuati RollBack a cascata.
Nelle basi di dati si cerca di ottenere schedule privi di effetti a cascata, e ciò è possibile se utilizzata una condizione sufficiente:

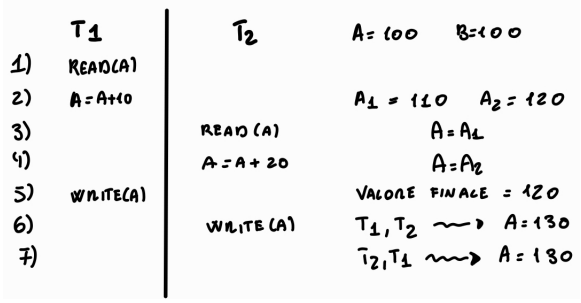
$\forall$ coppia di transazione, tale che la seconda legge un valore scritto dalla prima, la prima deve effettuare commit prima della lettura stessa.

Si garantisce così che la seconda transazione legga una risorsa già consolidata.

25 Lezione n25 20 12 2021

25.1 Lucchetti

Supponiamo di avere due transazioni T_1, T_2 e uno schedule così composto:

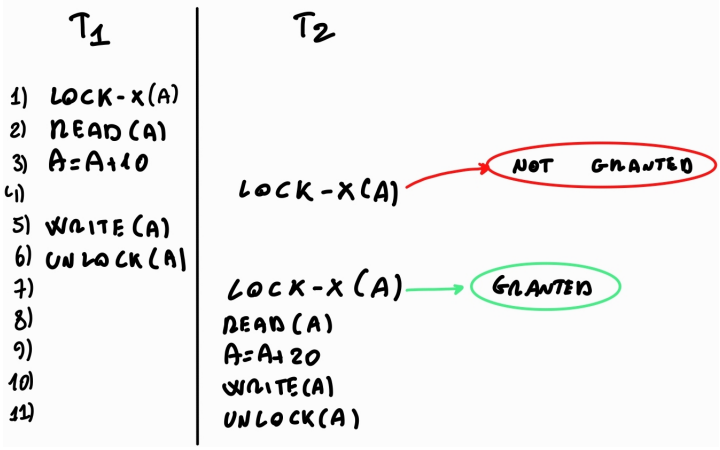


Tale schedule non è serializzabile, difatti i valori finali delle risorse A e B non corrispondono a possibili schedule seriabili, e inoltre vi sono istruzioni in conflitto tale per cui non è possibile ottenere uno schedule seriale tramite la loro permutazione.
Si introducono così dei lucchetti idonei alla lettura e alla scrittura

- **Lock-x**: Anche detto Lock esclusivo (exclusive) utilizzato per le operazioni di scrittura.
- **Lock-s**: Anche detto Lock condiviso (shared) utilizzato per le operazioni di lettura e permette difatti a più "entità" di accedervi contemporaneamente. (Più chiavi esistenti per aprire lo stesso lucchetto).

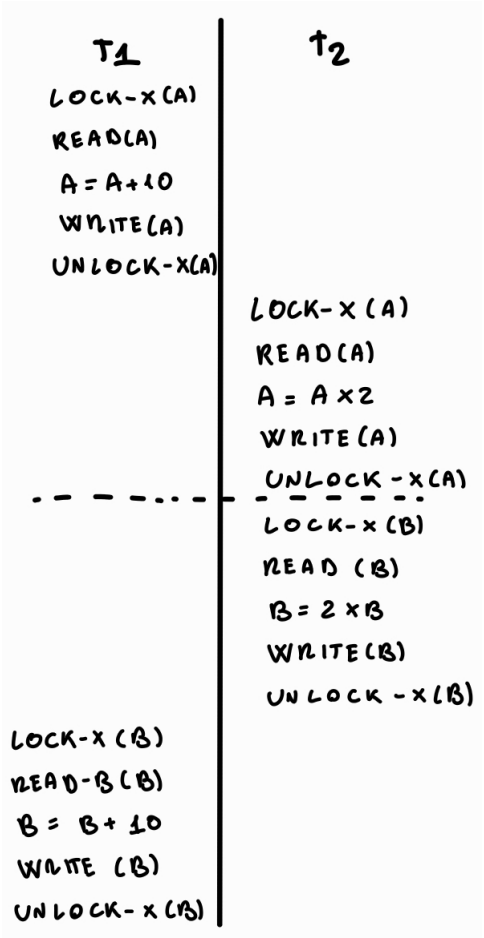
Utilizzando i lucchetti introdotti vi sono due possibili situazioni:

1. *Lock-s* attivo tutte le richieste di *Lock-x* in attesa mentre quelle di *Lock-s* vengono accettate (Garanted).
2. *Lock-x* attivo tutte le richieste di *Lock-s* e *Lock-x* vengono messe in attesa.



Inserendo tali strumenti nello schedule precedentemente visionato ed effettuando le commit dopo le unlock dei lucchetti e non prima, si osserva come sono i lock stessi a comunicare allo schedule chi e quando può accedere alla risorsa. Apparentemente si è raggiunti l'isolamento cercato, ma in realtà no.
Poiché anche utilizzando tutto in maniera corretta e assicurando così la consistenza, non è

detto che lo schedule ottenuto sia serializzabile (isolamento non garantito). Si raffigura un esempio:



Una prima soluzione potrebbe essere dichiarare tutte le risorse utilizzate da parte di ogni transazione, ma tutto ciò risulterebbe inefficiente oltre a causare altri problemi. Difatti introduciamo un protocollo che permette di assicurare l'isolamento oltre che alla consistenza.

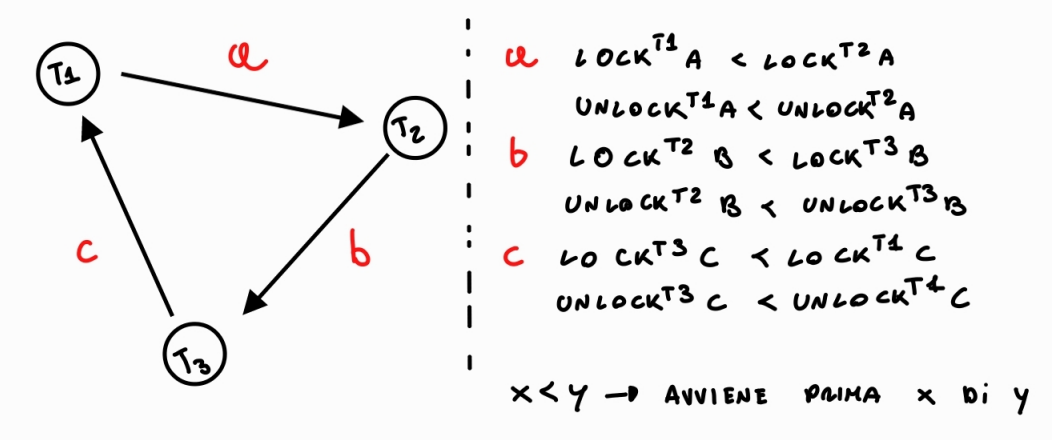
25.1.1 2PhaseLocking

2PL è un protocollo di richiesta a due fasi:

1. Fase **Growing**: Durante tale fase il numero di lucchetti posseduti da ogni transazione, può solo aumentare.
2. Fase **Shrinking**: Durante tale fase il numero di lucchetti posseduti, da ogni transazione, può solo diminuire.

Le due regole di fase, sopra riportate, possono essere riassunte come: non acquisire mai un lock dopo che è stata già effettuata la prima unlock. La proprietà di serializzabilità è garantita per uno schedule che rispetta tale regola.

Dimostrazione:
Dati T_1, T_2, T_3 (transazioni) che aderiscono al protocollo di 2PL e sia S uno schedule generato seconda la semantica corretta dei lucchetti, supponiamo per assurdo che S non sia *Conflict-Serializable* e da ciò ne consegue che il *Grafo delle dipendenze* è ciclico (Contiene almeno un ciclo).



Mettendo in cascata le relazioni ottenute dagli archi che caratterizzano il grafo sopra riportato di si ottiene:

$$unlock^{T1}(A) < Lock^{T1}(B)$$

Ovvero che T1 effettua l'unlock sulla risorsa A prima di una Lock sulla risorsa B, ma ciò è un assurdo poiché smentisce l'ipotesi di partenza, e quindi il grafo è aciclico e lo schedule S è *Conflict-Serializable*.

Domanda di esame, richiesta la medesima dimostrazione ma per "n" Transazioni caso generale.

25.1.2 Strict2PL e StrongStrict2PL

Two Phase Locking non garantisce Recuperabilità poiché non chiarisce nulla riguardo la posizione delle commit. Ecco perché è stata introdotta una variante, o meglio, il protocollo **Strict Two Phase Locking**, all'interno del quale viene introdotto il seguente vincolo:

Le unlock dei lucchetti esclusivi devono essere effettuate dal comando commit.

E ciò permette di garantire recuperabilità perché questo significa che le Lock esclusive vengono effettuate SOLO su risorse consistenti e definitive.

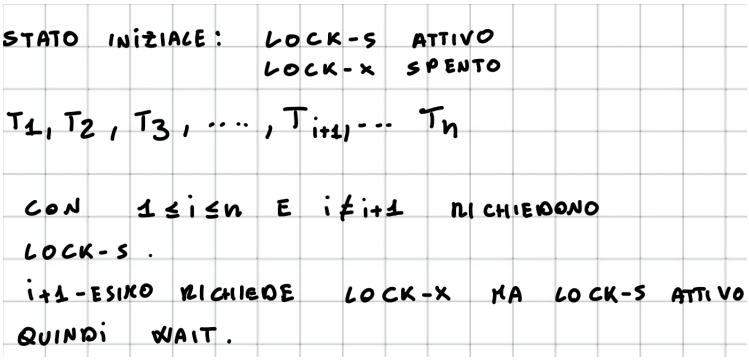
Esiste anche un ulteriore variante, **Strong Strict Two Phase Locking** che sostituisce il vincolo precedentemente riportato con il seguente:

Le unlock di tutti i lucchetti (Anche gli shared) devono essere effettuate dal comando commit.

Dal punto di vista dell'efficienza vi è un netto peggioramento, però si ha un miglioramento per quanto riguarda la complessità algoritmica che sta dietro al DMBS per la gestione dei lock.

25.2 Starvation e DeadLock

Si ipotizzi una situazione di tal tipo:



Si osserva quindi che T_{i+1} aspetterà in maniera indefinita (situazione di starvation) di accedere in maniera esclusiva alla risorsa 'A'. Tale problematica può essere risolta tramite semplici politiche di Aging, ma è bene sapere che possono occorrere durante l'utilizzo di una base di dati.

Un altro evento possibile è il **Dead Lock**, stato che occorre quando ogni transazione di un gruppo attende che un' altra transazione, inclusa se stessa, esegua il rilascio di un Lock-x. Talle problematica può essere risolta identificando le transazioni in attesa da molto tempo, una volta trovate si impone la failure e successivamente rieseguite. Il sintomo utilizzato è l'attesa elevata, perché l'identificazione del ciclo sarebbe troppo oneroso.

Si mette in evidenza che uno stato di stallo (DeadLock) può verificarsi su una risorsa se e solo se si verificano 4 condizioni contemporaneamente:

1. Mutua Esclusione
2. No prelazione
3. Attesa Circolare
4. Sospensione e attesa

25.3 Garanzia Serializabilità

Nelle Basi di dati può essere selezionato il livello d'isolamento desiderato; in particolare si ha:

1. Serializable
2. Repeatable read
3. Read Committed
4. Read Uncomitted

Al variare del livello scelto vi è una variazione dei controlli totali effettuati dal DBMS, in particolare minimi (5) e massimi (1) da cui ne consegue un Efficienza/Velocità massima (5) e minima (1).
ESAME: Per l'orale bisogna sapere, in cosa consiste ogni livello, le principali differenze e le possibili inconsistenze di dati che possono verificarsi.