[Appunti Basi di Dati]

Appunti del corso "Basi di Dati" dell'Università degli Studi di Genova

Lorenzo Vaccarecci

UNIVERSITÀ DEGLI STUDI DI GENOVA

Dipartimento di Informatica
Università degli Studi di Genova

2024

Indice

CAPITOLO 1

Modello Relazionale

1.1 Introduzione

Le interrogazioni sulle relazioni possono essere espresse in due formalismi:

- **Algebra relazionale**: le interrogazioni vengono espresse usando operatori specifici alle relazioni.
- **Calcolo relazionale**: le interrogazioni vengono espresse usando formule logiche.

1.2 Relazioni

Un dominio è un insieme (anche infinito) di valori. Indicheremo con $\mathcal{D}$ l'insieme di tutti i domini.

1.2.1 Definizione: Prodotto cartesiano

Siano $D_1, D_2, \ldots, D_k \in \mathcal{D}$ con k domini. Il prodotto cartesiano indicato con $D_1 \times D_2 \times \ldots \times D_k$, è definito come l'insieme $\{(v_1, v_2, \ldots, v_k) \,|v_1 \in D_1, \ldots, v_k \in D_k\}$.
Gli elementi appartenenti al prodotto cartesiano sono detti tuple. Il prodotto cartesiano ha **grado** k.

1.2.2 Definizione: Relazione

Una relazione di k domini è un sottoinsieme finito del prodotto cartesiano, ha **grado** k quindi ogni tupla ha k componenti. La **cardinalità** di una relazione è il numero di tuple appartenenti alla relazione. Una relazione è sempre un insieme finito. Non vi sono tuple duplicate.
La coppia (nome di attributo, dominio) è detta attributo. L'uso di attributi permette di denotare le componenti di ogni tupla per nome piuttosto che per posizione.

1.2.3 Definizione: Schema di relazione

- R un nome di relazione
- $\{A_1, A_2, \ldots, A_n\}$ un insieme di nomi di attributi
- $dom : \{A_1, A_2, \ldots, A_n\} \rightarrow \mathcal{D}$ una funzione totale che associa ad ogni nome di attributo il corrispondente dominio.

La coppia $(R(A_1, A_2, \ldots, A_n), dom)$ è uno schema di relazione. U_r denota l'insieme dei nomi di attributi di R, ovvero $U_r = \{A_1, A_2, \ldots, A_n\}$.

1.2.4 Definizione: Schema di base di dati

Siano $S_1, S_2, \ldots, S_n$ schemi di relazioni distinti, $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ è detto schema di base di dati.

1.2.5 Definizione: Tuple e relazione

Una tupla t definita su una relazione R è un insieme di funzioni totali che associano all'attributo di nome A_i un valore del dominio di tale attributo. Una relazione definita su uno schema di relazione è un insieme finito di tuple definite su tale schema. Tale relazione è anche detta istanza dello schema. $t = [A_1 : v_1, A_2 : v_2, \ldots, A_n : v_n]$ dove $v_i \in dom(A_i)$ con $i = 1, \ldots, n$. Notazione: $t[A_i]$ indica il valore dell'attributo A_i(quindi v_i) nella tupla t.

1.3 Valori nulli

Un aspetto importante nella modellazione dei dati riguarda il fatto che non sempre sono disponibili tutte le informazioni sulle entità del dominio applicativo che vengono rappresentate nella base di dati. L'approccio adottato è quello di introdurre un valore speciale, detto valore nullo, il quale denota la mancanza di un valore.
Nella trattazione assumiamo di denotare il valore nullo con il simbolo '?'. Il valore nullo è un valore accettato in tutti i domini.

1.4 Chiavi

Una chiave di una relazione è un insieme di attributi che distingue fra loro le tuple della relazione. Più formalmente:

1.4.1 Definizione: Chiave e super-chiave

Sia R uno schema di relazione. Un insieme $X \subseteq U_R$ di attributi di R è chiave di R se verifica le seguenti proprietà:

1. Univocità: nella relazione non ci possono essere due tuple distinte che abbiano lo stesso valore per tutti gli attributi della chiave X.

2. Nessun **sottoinsieme proprio** di X verifica la proprietà (1).

Un insieme di attributi che verifica la proprietà (1) ma non la (2) è detto super-chiave di R. Una super-chiave può essere una chiave della relazione.
In una relazione ci possono essere più insiemi di attributi che soddisfano le due proprietà. In tal caso si parla di chiavi candidate. **Una relazione ha sicuramente almeno una chiave (sia primaria che super)**. Nel caso in cui ci sono più chiavi candidate, una di queste viene scelta come chiave primaria su cui il DBMS ottimizza le operazioni.
Un criterio per scegliere la chiave primaria è quello di scegliere la chiave più piccola in termini di numero di attributi o quella più usata nelle interrogazioni. Una chiave non può contenere valori nulli.

1.4.2 Definizione: Chiave esterna

Sia R_1 ed R_2 due relazioni, sia X una chiave per R_1 e Y una chiave per R_2 tale che Y e X contengano lo stesso numero di attributi e di dominio compatibile (*es. interi e reali sono compatibili*). X è una chiave esterna di R_1 su R_2 se per ogni tupla t di R_1 esiste una tupla t' di R_2 tale che $t[X] = t'[Y]$. R_1 viene detta relazione **referente** e R_2 relazione **riferita**.
Vincolo di integrità refernziale: se una tupla t di R_1 fa riferimento ai valori della chiave di una tupla t' di R_2, allora t' deve esistere in R_2. Può essere violata da inserimenti e modifiche nella relazione referente e da cancellazioni e modifiche nella relazione riferita.
Una relazione può contenere più chiavi esterne, eventualmente anche sulla stessa relazione e possono assumere valori nulli.
Notazione:

$$R_1\left(\dots, chiave\_esterna^{R_2}, \dots\right)$$

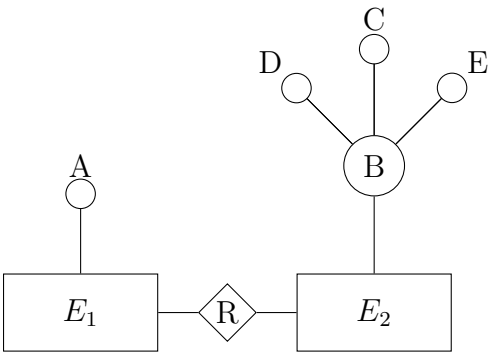
CAPITOLO 2

Modello ER

2.1 Introduzione

Il termine diagramma ER indica la rappresentazione grafica di uno schema concettuale ER. Gli elementi principali di un diagramma ER sono:

- Entità: una collezione di oggetti della realtà che vogliamo modellare che possiedono caratteristiche comuni, graficamente viene rappresentata da un rettangolo
- Relazioni (o associazione): rappresenta un legame logico tra più entità, graficamente viene rappresentata da un rombo
- Attributi: rappresenta una proprietà posseduta da un'entità o da una relazione, è mono-valore se può assumere al più un valore (*es. la matricola di uno studente*), è multi-valore altrimenti (*es. i numeri di telefono di un'azienda*). Può essere a sua volta essere formato da *sotto-attributi* rendendolo composto, un attributo composto può avere domini di diverso tipo (domini composti)



Le istanze di un'associazione sono un sottoinsieme del prodotto cartesiano delle entità coinvolte, non possono esistere duplicati. Le associazioni sono classificate in base al loro grado (il numero di entità in relazione):

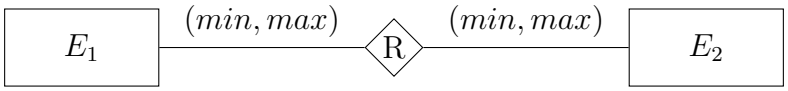
- **Unaria**: un'entità è in relazione con se stessa
- **Binaria**: due entità sono in relazione
- **n-aria**: n entità sono in relazione

Le informazioni sui domini degli attributi non sono direttamente rappresentabili in un diagramma ER, ma possono essere specificate nella documentazione a corredo dei diagrammi ER.

2.2 Vincoli di integrità

2.2.1 Vincoli di cardinalità

I vincoli di cardinalità per associazioni stabiliscono il numero minimo e massimo di istanze di un'associazione a cui le istanze delle entità coinvolte nella associazione possono partecipare. Graficamente, vengono rappresentati come coppia di valori interi (min, max) collocata vicino alla linea che connette l'associazione con ciascuna entità che mette in relazione.



(min, max):

- $min = 0$: l'entità non è obbligata a partecipare all'associazione
- $min = 1$: l'entità è obbligata a partecipare all'associazione
- $max = 1$: l'entità può partecipare al massimo una volta all'associazione
- $max = n$: l'entità può partecipare al massimo n volte all'associazione

Se in un diagramma ER non è specificata la cardinalità di un'associazione, si assume che il vincolo sia $(0, n)$.
Date due entità E_1 e E_2 se:

- $max_{E_1} = max_{E_2} = 1$ allora l'associazione è uno a uno
- $max_{E_1} = 1, max_{E_2} = n$ allora l'associazione è uno a molti
- $max_{E_1} = max_{E_2} = n$ allora l'associazione è molti a molti

Le cardinalità possono essere specificate anche per gli attributi, permettendo di specificare se un attributo è mono o multi-valore, e se è obbligatorio o meno. Se non viene specificata la cardinalità di un attributo, si assume che sia $(1, 1)$.

2.2.2 Vincoli di identificazione

Definire un vincolo d'identificazione per un'entità significa specificare un insieme di attributi e/o entità che posseggono la proprietà di identificare univocamente le istanze dell'entità (identificatori o chaivi).

- Identificatore interno: è costituito da uno o più attributi dell'entità stessa
- Identificatore misto: è costituito da attributi dell'entità e da attributi di altre entità
- Identificatore esterno: è costituito da attributi di altre entità
- Identificatore semplice: è costituito da un solo attributo
- Identificatore composto: è costituito da più attributi

Un'entità le cui istanze vengono identificate mediante l'associazione con altre entità viene chiamata entità debole.

2.3 Gerarchie di generalizzazione

Nel modello ER è possibile organizzare le entità in gerarchia di generalizzazione definendo un insieme di entità dette figlie come specializzazione di un'altra entità detta padre rappresentante le proprietà in comune a tutte le entità figlie.

- Generalizzazione totale: ogni istanza dell'entità padre è anche un'istanza di **almeno** una delle entità figlie
- Generalizzazione parziale: se un'istanza dell'entità padre non è un'istanza di nessuna delle entità figlie
- Generalizzazione esclusiva: se un'istanza dell'entità padre è un'istanza di **al più** una delle entità figlie
- Generalizzazione condivise: se un'istanza dell'entità padre può essere un'istanza di **più** entità figlie

Le generalizzazioni possono essere di quattro tipi diversi: totali esclusive, totali condivise, parziali esclusive e parziali condivise.

CAPITOLO 3

Progettazione Logica

3.1 Introduzione

L'obiettivo principale della progettazione logica è tradurre uno schema ER in uno schema relazionale equivalente. Si articola in due fasi:

- Fase di ristrutturazione dello schema ER: questa fase prevede l'eliminazione dallo schema ER di tutti quei costrutti non direttamente rappresentabili nel modello relazionale, anche questa fase si può suddividere in più sotto-fasi:
  - analisi della ridondanza
  - partizionamento/accorpamento di entità
  - eliminazione degli attributi composti e multi-valore
  - eliminazione delle gerarchie di generalizzazione
- Fase di traduzione dello schema ER: in questa fase, lo schema ER restituito dalla fase di ristrutturazione viene tradotto in un equivalente schema relazionale. La traduzione non è sempre univoca.

3.2 Fase di ristrutturazione

3.2.1 Analisi della ridondanza

Uno schema ER presenta delle ridondanze quando un'informazione viene rappresentata sia esplicitamente nello schema sia può essere derivata da altre informazioni presenti nello schema. Nei diagrammi ER la presenza di ridondanza dovrebbe essere limitata solo a quei casi in cui sia possibile ottenere un significativo beneficio in termini di tempo di esecuzione delle interrogazioni.

3.2.2 Partizionamento/accorpamento di entità

Lo schema ER può essere ulteriormente ristrutturato partizionando od accorpando entità ed associazioni sulla base dell'analisi del carico di lavoro. Un'entità E può essere partizionata in due entità E_1, E_2 una delle quali identificata esternamente dall'altra, collegate mediante un'associazione uno a uno. Questa operazione può essere conveniente quando alcune operazioni frequenti coinvolgono solo un sottoinsieme degli attributi di E. Viceversa, due entità E_1, E_2 collegate da un'associazione uno a uno, possono essere accorpate in un'unica entità contenente gli attributi di E_1 e E_2 nel caso in cui operazioni frequenti abbiano la necessità di accedere ad entrambi gli insiemi di attributi.

3.2.3 Eliminazione degli attributi composti e multi-valore

Il modello relazionale consente la specifica solo di attributi semplici e mono-valore. E' quindi necessario ristrutturare lo schema ER generato dalla fase di progettazione concettuale per eliminare eventuali attributi composti e multi-valore in esso presenti. L'eliminazione di un attributo composto A da un'entità E può avvenite in due modi:

- Eliminando i sotto-attributi di A e considerando l'attributo composto come un attributo semplice
- Considerando tutti i sotto-attributi di A come attributi di E

Quest'ultima soluzione richiede ovviamente una ridefinizione del dominio dell'attributo. La modellazione di attributi multi-valore mediante attributi a valore semplice richiede la definizione di una nuova entità, collegata all'entità di partenza tramite un'opportuna associazione, in cui l'attributo multi-valore è rappresentato mediante un attributo mono-valore che identifica l'entità. Il vincolo di cardinalità dell'associazione è quello dell'attributo multi-valore oggetto di ristrutturazione.

3.2.4 Eliminazione delle gerarchie di generalizzazione

Consideriamo un'entità E generalizzazione di un insieme di entità $E_1, \ldots, E_n$

- Eliminazione delle entità figlie: Le entità figlie vengono eliminate ed i loro attributi vengono inseriti nell'entità padre come attributi opzionali.
  - Generalizzazione totale: l'attributo portato al padre non può essere nullo
  - Generalizzazione parziale: un valore nullo indica un'istanza dell'entità padre che non era istanza di alcuna delle entità figlie
  - Generalizzazione condivise: l'attributo sarà multi-valore in quanto un'istanza dell'entità padre potrebbe essere istanza di più entità figlie
- Eliminazione dell'entità padre: consiste nell'eliminare l'entità padre E e nell'inserire i suoi attributi in ciascuna delle entità figlie. Tale procedimento è applicabile solo nel caso in cui la generalizzazione sia **totale**.
  - Generalizzazione esclusiva: è necessario aggiungere un vincolo per indicare che non possono esistere istanze di due entità figlie distinte aventi lo stesso valore per gli identificatori.

  Ogni associazione a cui partecipava l'entità padre l'entità padre viene inoltre sostituita con n nuove associazioni, una per ogni entità figlia.
- Sostituzione della generalizzazione con associazioni: le entità coinvolte nella generalizzazione non vengono modificate, mentre la gerarchia di generalizzazione viene sostituita da n associazioni uno a uno, ognuna delle quali lega l'entità padre con una diversa entità figlia. Le entità figlie sono identificate esternamente dall'entità padre e partecipano obbligatoriamente alle associazioni create mentre la partecipazione dell'entità padre è opzionale.
  - Generalizzazione esclusiva: un'istanza del padre non può partecipare contemporaneamente a due o più associazioni

– Generalizzazione totale: ogni istanza dell'entità padre deve partecipare obbligatoriamente ad almeno un'associazione

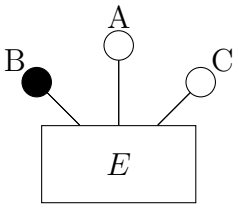
In generale, la scelta di accorpare le entità figlie nell'entità padre comporta uno spreco di memoria per la presenza dei valori nulli. La soluzione di eliminare l'entità padre consente un risparmio di memoria rispetto alla soluzione di eliminare le entità figlie in quanto evita il problema dei valori nulli.

3.3 Fase di traduzione

La fase di traduzione può essere suddivisa nelle seguenti sotto-fasi:

- Traduzione delle entità
- Traduzione delle associazioni
- Traduzione dei vincoli di integrità
- Ottimizzazioni finali

3.3.1 Traduzione delle entità

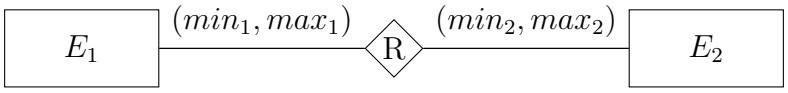


→ E(B,A,C);

Nel caso di chiavi esterne si scrivono in fondo (di solito) e in corsivo indicando anche la tabella di provenienza per una maggiore comprensione della traduzione: E(K,...,FK^{E_i}).
I criteri per la scelta della chiave primaria si possono sintetizzare come segue:

- Non devono contenere valori nulli
- Devono essere composti da pochi attributi
- Sono gli attributi più utilizzati nelle interrogazioni per accedere alle entità

3.3.2 Traduzione delle associazioni



Consideriamo che E_1 ha gli attributi A,B e E_2 ha gli attributi C,D. La chiave di E_1 è K e la chiave di E_2 è L.
Casi:

- Associazione uno a uno: si "elimina" una delle due entità e si accorpa con l'altra: $E_1(\underline{K, L}, A, B, C, D)$
- Associazione uno a molti: l'entità "da sola" riceve la chiave dell'altra entità come chiave esterna: $E_1(\underline{K}, A, B, L), E_2(\underline{L}, C, D, K^{E_1})$
- Associazione molti a molti: si crea una nuova relazione con le chiavi delle entità coinvole e gli eventuali attributi dell'associazione: $R(\underline{K, L})$

CAPITOLO 4

Algebra Relazionale

4.1 Introduzione

L'algebra è composta da cinque operazioni di base:

- Proiezione
- Selezione
- Prodotto cartesiano
- Unione
- Differenza

Ogni operazione ha come argomento una o due relazioni e restituisce come risultato una relazione; è pertanto possibile applicate un'operazione al risultato di un'altra operazione (proprietà di chiusura). In riferimento alla notazione per nome, viene introdotta un'ulteriore operazione, di **ridenominazione**, che permette di modificare i nomi degli attributi.

4.2 Ridenominazione

La ridenominazione di una relazione R indicata con

$$\rho_{A_1,A_2,\ldots,A_n \leftarrow B_1,B_2,\ldots,B_n}(R)$$

ridenomina l'attributo di nome A_i con il nome $B_i, i = 1, \ldots, n$. La ridenominazione è corretta se il nuovo schema di relazione per R ha attributi con nomi tutti distinti.

4.3 Proiezione

La proiezione di una relazione R su un insieme $A = A_1, A_2, \ldots, A_n \subseteq U_R$ di nomi di attributi di R, indicata con:

$$\Pi_{A_1,A_2,\ldots,A_n}(R)$$

è una relazione di grado n le cui tuple hanno come attributi solo gli attributi specificati in A. Il risultato della proiezione è un insieme di tuple senza duplicati.

4.4 Selezione

La selezione su una relazione R, dato un predicato F su R, indicata con:

$$\sigma_F(R)$$

genera una relazione che contiene tutte le tuple di R che verificano F. F può essere una combinazione degli operatori booleani $\wedge, \vee, \neg$ e operatori relazionali di confronto $=, \neq, <, >, \leq, \geq$.
Lo schema (ed il grado) della relazione risultato sono uguali a quelli rella relazione operando.

4.5 Prodotto cartesiano

Il prodotto cartesiano di due relazioni R e S, indicato con:

$$R \times S$$

sono tutte le possibili combinazione delle tuple di R con le tuple di S. La relazione risultato ha grado pari alla somma dei gradi delle relazioni operandi e la cardinalità è il prodotto delle cardinalità degli argomenti.
Il prodotto cartesiano può essere applicato solo se le due relazioni R e S hanno schemi disgiunti.

4.6 Unione

l'unione delle relazioni R ed S, indicata con:

$$R \cup S$$

è l'insieme delle tuple che sono in R od in S. **Le due relazioni devono avere lo stesso schema**.
Le tuple duplicate vengono eliminate dal risultato.

4.7 Differenza

La differenza delle relazioni R ed S, indicata con:

$$R - S$$

è l'insieme delle tuple che sono in R ma non in S. **Le due relazioni devono avere lo stesso schema e stesso grado**.

4.8 Intersezione

L'intersezione delle relazioni R ed S, indicata con:

$$R \cap S = R - (R - S)$$

restituisce le tuple che sono sia in R che in S. **Le due relazioni devono avere lo stesso schema**.

4.9 Join

Il join (detto anche theta-join) di due relazioni R ed S sugli attributi A di R e B di S, indicato con:

$$R \bowtie_{A\theta B} S$$

dove θ è un operatore relazionale di confronto, è definito dall'espressione algebrica:

$$\sigma_{A\theta B}(R \times S)$$

il join pertanto è un prodotto cartesiano seguito da una selezione.
Il join può essere applicato solo se le due relazioni R ed S hanno schemi disgiunti. *Il join si chiama equi-join se l'operatore θ è l'uguaglianza.*

4.9.1 Join naturale

$$R \bowtie S$$

è un join in cui si considerano solo gli attributi comuni tra R ed S e poi elimina i duplicati dalla relazione risultato. Nel caso particolare $U_R = U_S$, il join naturale degenera nell'intersezione, mentre, nel caso $U_R \cap U_S = \emptyset$, degenera nel prodotto cartesiano.

4.10 Divisione

L'operazione di divisione, date due relazioni R ed S, con insiemi di attributi $U_R \subset U_S$ indicata con:

$$R \div S$$

L'idea intuitiva è che l'operazione di divisione è utile per determinare le tuple per cui in una relazione c'è una corrispondenza con tutte le tuple di un'altra relazione. Il grado della relazione risultato è $h - k$ dove h è il grado di R e k è il grado di S.
La divisione è **commutaiva** e **associativa**. La divisione è un'operazione derivata dall'algebra relazionale. La divisione di due relazioni R ed S è definita come:

$$R \div S = \Pi_D(R) - \Pi_D((\Pi_D(R) \times S) - R)$$

CAPITOLO 5

Linguaggio SQL

5.1 Tipi

- `INTEGER`: interi a 32 bit
- `SMALLINT`: interi a 16 bit
- `BIGINT`: interi a 64 bit
- `NUMERIC`: numeri decimali, usa due parametri: la precisione (il numero totale di cifre) e la scala (il numero di cifre dopo la virgola) → `NUMERIC(precisione,scala)` i valori di default sono 1 (precisione) e 0 (scala)
- `DECIMAL`: come `NUMERIC` ma le cifre possono essere $\geq$ p
- `REAL`: numeri in virgola mobile a precisione singola
- `DOUBLE PRECISION`: numeri in virgola mobile a precisione doppia
- `FLOAT`: numeri in virgola mobile a precisione personalizzata → `FLOAT(p)` con $1 \leq p \leq n$ dove n dipende dall'implementazione
- `CHARACTER`: stringhe di lunghezza fissa → `CHARACTER(n)`, di default è 1
- `CHARACTER VARYING`: stringhe di lunghezza variabile → `CHARACTER VARYING(n)` dove n è la lunghezza massima
- `DATE`: data in formato `YYYY-MM-DD`
- `TIME`: tempo in formato `HH:MM:SS`, può essere specificata la precisione dei secondi → `TIME(p)` con $0 \leq p \leq 6$
- `TIMESTAMP`: concatenazione di `DATE` e `TIME`
- `INTERVAL`: durata temporale in riferimento ad uno o più dei qualificatori tra `YEAR`, `MONTH`, `DAY`, `HOUR`, `MINUTE`, `SECOND` → `INTERVAL 'valore' qualificatore`
- `BOOLEAN`: valore booleano `TRUE`,`FALSE` o `UNKNOWN`
- `BLOB`: dati binari di lunghezza variabile
- `CLOB`: dati di testo di lunghezza variabile

5.2 Creazione di tabelle

```
1        CREATE TABLE nome_tabella(
2            nome_colonna tipo_colonna,
3            ...
4        );
5
```

5.2.1 Obbligatorietà di colonne

Per la specifica dell'obbligatorietà di una colonna si utilizza la clausola `NOT NULL`:

```
CREATE TABLE nome_tabella(
    nome_colonna tipo_colonna NOT NULL,
    ...
);

```

5.2.2 Chiavi

Per la specifica di una chiave primaria si utilizza la clausola `PRIMARY KEY`:

```
CREATE TABLE nome_tabella(
    nome_colonna tipo_colonna PRIMARY KEY,
    ...
);

```

Per le chiavi alternative si usa `UNIQUE`. Se si hanno chiavi composte da più attributi:

```
CREATE TABLE nome_tabella(
    nome_colonna1 tipo_colonna,
    nome_colonna2 tipo_colonna,
    ...
    PRIMARY KEY(nome_colonna1,nome_colonna2)
);

```

(stessa cosa per `UNIQUE`).
Per la specifica di una chiave esterna si utilizza la clausola `FOREIGN KEY`:

```
CREATE TABLE nome_tabella(
    nome_colonna tipo_colonna,
    ...
    FOREIGN KEY(nome_colonna)
        REFERENCES nome_tabella_esterna(nome_colonna_esterna)
);

```

La chiave esterna può avere delle "azioni" da eseguire in caso di cancellazione (`ON DELETE`):

- `NO ACTION`: la cancellazione di una tupla non è permessa se esistono riferimenti ad essa
- `CASCADE`: la cancellazione di una tupla comporta la cancellazione delle tuple che fanno riferimento ad essa
- `SET NULL`: la cancellazione di una tupla comporta l'impostazione a `NULL` del valore della chiave esterna delle tuple che fanno riferimento ad essa
- `SET DEFAULT`: la cancellazione di una tupla comporta l'impostazione al valore di default del valore della chiave esterna delle tuple che fanno riferimento ad essa

In caso di modifica (`ON UPDATE`) le "azioni" funzionano in modo simile.

5.3 Cancellazione di tabelle

```
DROP TABLE nome_tabella [CASCADE | RESTRICT];
```

L'opzione `CASCADE` permette di eliminare anche le tabelle che fanno riferimento alla tabella da eliminare, mentre `RESTRICT` impedisce la cancellazione se esistono tabelle che fanno riferimento alla tabella da eliminare.

5.4 Modifica di tabelle

5.4.1 Aggiunta di colonne

```
ALTER TABLE nome_tabella
    ADD nome_colonna tipo_colonna;
```

5.4.2 Modifica di colonne

```
ALTER TABLE nome_tabella
    ALTER COLUMN nome_colonna SET DATA TYPE tipo_colonna;
```

5.4.3 Cancellazione di colonne

```
ALTER TABLE nome_tabella
    DROP COLUMN nome_colonna [CASCADE | RESTRICT];
```

5.5 Interrogazioni

5.5.1 Selezione

```
SELECT [DISTINCT] lista_attributi
FROM nome_tabella
WHERE condizione;
```

La clausola `DISTINCT` permette di eliminare i duplicati.
Nella selezione ci si può riferire alle colonne mettendo prima il nome della tabella a cui appartengono seguito da un punto:

```
SELECT nome_tabella.nome_colonna
FROM nome_tabella;
```

Questo può aiutare in caso di colonne con lo stesso nome in tabelle diverse o semplicemente per una maggiore chiarezza.
Se ci si vuole riferire a tutte le colonne di una (o più) tabelle si può usare il simbolo `*`:

```
SELECT *
FROM nome_tabella;
```

Nel `WHERE` si possono usare i connettivi logici `AND`, `OR` e `NOT`.

5.5.2 Operatori di confronto

Condizioni su intervalli di valori

L'operatore `BETWEEN` permette di ritrovare le tuple che contengono valori di una colonna in un intervallo specificato:

```
        SELECT *
        FROM nome_tabella
        WHERE nome_colonna BETWEEN valore1 AND valore2;

```

`BETWEEN` è l'abbreviazione di:

```
        SELECT *
        FROM nome_tabella
        WHERE nome_colonna >= valore1 AND nome_colonna <= valore2;

```

Può essere usato con il `NOT`.

Ricerca di valori in un insieme

L'operatore `IN` permette di determinare le tuple che contengono uno tra i valori di un insieme spiecificato:

```
        SELECT *
        FROM nome_tabella
        WHERE nome_colonna IN (valore1,valore2,...);

```

E' l'abbreviazione di:

```
        SELECT *
        FROM nome_tabella
        WHERE nome_colonna = valore1 OR nome_colonna = valore2 OR ...;

```

Può essere usato con il `NOT`.

Condizioni di confronto per stringhe di caratteri

L'operatore `LIKE` permette di eseguire alcune sepmlici operazioni di *pattern matching* su colonne di tipo stringa:

- `%`: rappresenta una sequenza di zero o più caratteri
- `_`: rappresenta un singolo carattere

```
        SELECT *
        FROM nome_tabella
        WHERE nome_colonna LIKE '__d%';  -- trova la tupla che inizia
    con due caratteri qualsiasi, seguita da 'd' e poi da zero o piu'
    caratteri

```

5.5.3 Espressioni e funzioni aritmetiche

- `+`: somma
- `-`: sottrazione
- `*`: moltiplicazione
- `/`: divisione
- `ABS(`n`)`: valore assoluto di n
- `MOD(`n`,`m`)`: resto della divisione di n per m

5.5.4 Espressioni e funzioni per stringhe

- `||`: concatenazione
- `LENGTH(`s`)`: lunghezza della stringa s
- `LOWER(`s`)`: trasforma la stringa s in minuscolo
- `UPPER(`s`)`: trasforma la stringa s in maiuscolo
- `SUBSTR(`s`,`m`,`n`)`: sottostringa di s a partire dalla posizione m di lunghezza n, se n è omesso la sottostringa è fino alla fine di s
- `TRIM` $s1$ `FROM` $s2$: rimuove gli spazi bianchi da $s2$, se $s1$ è specificato rimuove i caratteri di $s1$ da $s2$

5.5.5 Espressioni e funzioni per date e tempi

- `CURRENT_DATE`: data corrente
- `CURRENT_TIME`: tempo corrente
- `CURRENT_TIMESTAMP`: data e tempo correnti
- `EXTRACT(`*campo* `FROM` *data*`)`: estrae il campo specificato da una data o un tempo (*es.* `EXTRACT(YEAR FROM CURRENT_DATE)` restituisce l'anno corrente)

5.5.6 Ordinamento dei risultati di una query

```
SELECT *
FROM nome_tabella
ORDER BY nome_colonna [ASC | DESC];

```

Ordina i risultati in base ai valori della colonna specificata in ordine crescente (`ASC`) o decrescente (`DESC`).

5.5.7 Operazione di join

- `CROSS JOIN`: prodotto cartesiano tra due tabelle

```
SELECT *
FROM tabella1
CROSS JOIN tabella2;

```

- `JOIN ON`: prodotto cartesiano tra due tabelle con una condizione

```
SELECT *
FROM tabella1
JOIN tabella2
ON tabella1.colonna = tabella2.colonna;
```

- `JOIN USING`: prodotto cartesiano tra due tabelle con una condizione su una o piu' colonne comuni

```
SELECT *
FROM tabella1
JOIN tabella2
USING (colonna_comune1, ...);
```

- `NATURAL JOIN`: prodotto cartesiano tra due tabelle con una condizione su tutte le colonne con lo stesso nome

```
SELECT *
FROM tabella1
NATURAL JOIN tabella2;
```

Outer join

Con i `JOIN` non si ha traccia delle tuple di una tabella che corrispondono a nessuna tupla dell'altra tabella. Per ovviare a questo problema si usano le `OUTER JOIN` che aggiunge al risultato anche le tuple che non hanno corrispondenza completandole con valori nulli. Il `JOIN` originario è detto `INNER JOIN`.

- `FULL`: tutte le tuple di entrambe le tabelle che non hanno corrispondenza vengono completate ed inserite nel risultato
- `LEFT`: solo le tuple della tabella a sinistra che non hanno corrispondenza vengono completate ed inserite nel risultato
- `RIGHT`: solo le tuple della tabella a destra che non hanno corrispondenza vengono completate ed inserite nel risultato

5.5.8 Funzioni di gruppo

- `MAX`: massimo valore di un insieme di valori
- `MIN`: minimo valore di un insieme di valori
- `SUM`: somma di un insieme di valori
- `AVG`: media di un insieme di valori
- `COUNT`: cardinalità di un insieme

Esistono anche le funzioni `STDEV` e `VAR` per calcolare la deviazione standard e la varianza. La funzione `COUNT` può avere tre tipi di argomenti:

- un nome di una colonna: conta il numero di valori non nulli nella colonna specificata
- un nome di colonna preceduto dal qualificatore `DISTINCT`: conta il numero di valori distinti non nulli nella colonna specificata
- `*`: conta il numero di tuple nel risultato

5.5.9 Raggruppamento

```
SELECT colonna1
FROM tabella
GROUP BY colonna1;
```

Nel `GROUP BY` si possono usare solo le colonne che compaiono nella clausola `SELECT`. Raggruppa le tuple in base ai valori della colonna specificata.
E' possibile specificare condizioni di ricerca su gruppi di tuple utilizzando la clausola `HAVING`:

```
SELECT colonna1
FROM tabella
GROUP BY colonna1
HAVING condizione;
```

5.5.10 Sotto-interrogazioni

```
SELECT *
FROM tabella
WHERE colonna IN (SELECT colonna FROM tabella2);
```

Per le sotto-interrogazioni si possono usare i quantificatori `ALL` e `ANY`:

- `ANY`: restituisce vero quando la valutazione dell'operatore di confronto è vera per almeno una delle tuple restituite dalla sotto-interrogazione, restituisce falso altrimenti o se la sotto-interrogazione non restituisce tuple
- `ALL`: restituisce vero quando la valutazione dell'operatore di confronto è vera per tutte le tuple restituite dalla sotto-interrogazione o se la sotto-interrogazione non restituisce tuple, restituisce falso altrimenti

`IN` è equivalente a `= ANY`.
`NOT IN` è equivalente a `<> ALL`.

5.5.11 Sotto-interrogazioni correlate

Una sotto-interrogazione è detta correlata se il risultato della sotto-interrogazione dipende dal valore di una colonna della query esterna. Per fare riferimento alle colonne delle tuple della query esterna si usano degli alias:

```
SELECT *
FROM tabella T
WHERE colonna IN (SELECT colonna
                  FROM tabella2
                  WHERE tabella2.colonna = T.colonna);
```

oppure

```
SELECT *
FROM tabella AS T
WHERE colonna IN (SELECT colonna
```

```
                        FROM tabella2
                        WHERE tabella2.colonna = T.colonna);

```

Operatori

Le sotto-interrogazioni correlate sono spesso usate in combinazione con gli operatori `EXISTS` e `NOT EXISTS`:

- `EXISTS`: restituisce vero se la sotto-interrogazione restituisce almeno una tupla, falso altrimenti

```
            SELECT *
            FROM tabella
            WHERE EXISTS (SELECT *
                          FROM tabella2
                          WHERE tabella2.colonna = tabella.colonna)
;

```

- `NOT EXISTS`: restituisce vero se la sotto-interrogazione non restituisce alcuna tupla, falso altrimenti

```
            SELECT *
            FROM tabella
            WHERE NOT EXISTS (SELECT *
                              FROM tabella2
                              WHERE tabella2.colonna = tabella.
colonna);

```

5.5.12 Operatori insiemistici

Un'interrogazione (o sotto-interrogazione) può essere costituita da una o più sotto interrogazioni connesse dall'operatore `UNION`. Tale operatore restituisce tutte le tuple distinte restituite da almeno una delle sotto-interrogazioni a cui è applicata.

- `UNION`: restituisce l'unionde di due insiemi di tuple, elimina i duplicati
- `INTERSECT`: restituisce l'intersezione di due insiemi di tuple, elimina i duplicati
- `EXCEPT (o MINUS)`: restituisce la differenza tra due insiemi di tuple, elimina i duplicati

5.6 Operazioni di aggiornamento

5.6.1 Inserimento

```
        INSERT INTO nome_tabella
        VALUES (valore1,valore2,...);

```

E' possibile anche inserire una nuova tupla con i valori di alcune colonne specificate:

```
        INSERT INTO nome_tabella (colonna1,colonna2,...)
        VALUES (valore1,valore2,...);

```

I valori delle colonne possono essere generati anche con una sotto-interrogazione:

```
INSERT INTO nome_tabella (colonna1,colonna2,...)
sotto-interrogazione;
```

La clausola di proiezione della sotto-interrogazione deve includere colonne di tipo compatibile con le colonne della tupla o delle tuple da inserire. Quando usiamo una sotto-interrogazioine all'interno di un comando `INSERT` viene inserito un numero di tuple uguale alla cardinalità del risultato della sotto-interrogazione.

5.6.2 Cancellazione

Il comando `DELETE` permette la cancellazione di tuple da una data relazione. Le tuple da cancellare sono specificate tramite una condizione di ricerca; se non è specificata alcuna condizione di ricerca, vengono cancellate **tutte** le tuple presenti nella relazione oggetto del comando.

```
DELETE FROM nome_tabella
[WHERE condizione];
```

5.6.3 Modifica

Il comando `UPDATE` permette l'esecuzione di modifiche ad una o più colonne delle tuple di una relazione. Le sotto-interrogazioni possono essere usate non solo per determinare le tuple da modificare ma anche per determinare i valori da assegnare alle colonne da modificare.

```
UPDATE nome_tabella
SET colonna1 = valore1, colonna2 = valore2, ...
[WHERE condizione];
```

5.7 Vincoli di integrità

SQL permette la specifica di alcuni vincoli di integrità semantica, dove con vincolo intendiamo una proprietà che un insieme di dati deve verificare. E' possibile innanzitutto disitnguere tra vincoli statici e vincoli di transizione. Un vincolo statico riguarda uno stato della base di dati (*es. "la valutazione relativa ad un film deve essere compresa tra 0 e 5"*) mentre un vincolo di transizione mette in relazione stati diversi della base di dati (*es. "non è possibile modificare la data di restituzione di un video assegnandogli una data precedente a quella memorizzata"*).

- Vincoli su singola relazione: Sono i vincoli che riguardano una singola relazione
  - Vincoli su singola tupla: vincoli di questo tipo coinvolgono gli attributi di una singola tupla
    - Vincoli su singolo attributo (*es.* `NOT NULL`)
    - Vincoli su attributi multipli
  - Vincoli su tuple multiple di una stessa relazione

    - Dipendenze funzionali: permettono di modellare correlazioni tra i valori di determinati attributi in tuple diverse e di esprimere che un attributo od un insieme di attributi siano chiave di una relazione
    - Vincoli di aggregazione: impongono che una determinata funzione aggregata calcolata su un insieme di tuple verifichi una data relazione di confronto rispetto ad un valore dato

- Vincoli su relazioni multiple: sono vincoli che coinvolgono tuple di relazioni diverse. Un importante tipo di vincolo in questa classe è l'integrità referenziale

Quando si crea una tabella è possibile usare il comando `CHECK` per definire vincoli arbitrari su colonna o su relazione, che corrispondono a vincoli su singola tupla della classificazione precedente. Per definire un vincolo `CHECK` su una colonna si usa la seguente sintassi:

```
CREATE TABLE nome_tabella(
    nome_colonna tipo_colonna CHECK (condizione),
    ...
);

```

Mentre per definire un vincolo `CHECK` su una relazione si usa la seguente sintassi:

```
CREATE TABLE nome_tabella(
    ...
    CHECK (condizione),
    ...
);

```

La relazione vuota soddisfa sempre tutti i vincoli `CHECK`.
Una possibilità prevista da SQL per tutti i vincoli associati alle definizioni di relazioni è quella di assegnare un nome al vincolo premettendo alla specifica del vincolo la clausola `CONSTRAINT nome_vincolo`. Il nome del vincolo deve essere unico rispetto ad altri nomi di vincoli definiti per la stessa relazione:

```
CREATE TABLE nome_tabella(
    ...
    CONSTRAINT nome_vincolo CHECK (condizione),
    ...
);

```

5.8 Asserzioni

SQL prevede il meccanismo delle asserzioni per specificare vincoli su più tuple o relazioni. Le condizioni che possono essere espresse solo tramite asserzioni sono quelle relative al numero minimo di tuple (che soddisfano una certa condizione) contenute in una tabella.

```
CREATE ASSERTION nome_asserzione
CHECK (condizione);

```

Definire tali vincoli di integrità come asserzioni è concettualmente più corretto, ed, in generale, porta ad una verifica più efficiente.
Un'asserzione può essere rimossa tramite il comando `DROP ASSERTION nome_asserzione`.

5.9 Controllo di vincoli di integrità

SQL consente di specificare sia vincoli d'integrità con valutazione immediata, cioè valutati dopo ogni comando di manipolazione dei dati, sia vincoli la cui valutazione è differita al termine dell'esecuzione di una sequenza di operazioni di manipolazione dei dati, che constituiscono una transazione. Quando i vincoli sono immediatamente verificati, una violazione del vincolo causa la non esecuzione del comando che ne ha causato la violazione (eventuali modifiche parziali vengono annullate). Nel caso in cui i vincoli siano valutati alla fine della transazione, invece, la violazione del vincolo comporta l'abort della transazione. Tutte le operazioni nella transazione vengono cioè annullate, poichè non vi è modo di stabilire quale operazione ha causato la violazione del vincolo.
Nella definizione di un vincolo è possibile specificare se dovrà essere sempre valutato dopo ogni singola operazione SQL (`NOT DEFERRABLE`, di default è questa), viceversa la valutazione di un vincolo può essere differita alla fine della transazione (`DEFERRABLE`). Un vincolo `DEFERRABLE` può essere specificato con le opzioni

- `INITIALLY IMMEDIATE` (default): la verifica avviene dopo ogni istruzione SQL
- `INITIALLY DEFERRED`: la verifica avviene alla fine della transazione

La modalità di controllo di un vincolo può essere cambiata dinamicamente:

```
1         SET CONSTRAINTS {lista vincoli | ALL} {DEFERRED | IMMEDIATE};
2
```

Un vincolo di integrità è violato se la valutazione della condizione di controllo restituisce `FALSE`, viceversa un vincolo di integrità non è violato se la valutazione della condizione restituisce `TRUE` o `UNKNOWN`.

5.10 Dati derivati e Viste

5.10.1 Colonne derivate

I valori per la colonna derivata sono calcolati ed assegnati automaticamente ogni volta che una nuova tupla è inserita nella relazione e mantenuti aggiornati in seguito ad aggiornamenti a tuple della relazione. Nel comando `CREATE TABLE` o nella clausola `ADD COLUMN` del comando `ALTER TABLE` si può specificare una colonna derivata con la seguente sintassi:

```
1         colonna_derivata [tipo_colonna] GENERATED ALWAYS AS (
  espressione)
2
```

5.10.2 Derivazione di relazioni

SQL prevede la possibilità di definire una nuova relazione basandosi su relazioni già definite. A livello di schema è possibile, mediante la clausola opzionale `LIKE` del comando `CREATE TABLE`, copiare la struttura completa di una o più relazioni esistenti nella definizione di una nuova relazione.
SQL permette inoltre, mediante la clausola opzionale `AS` del comando `CREATE TABLE`, di creare una relazione con la stessa struttura di un'interrogazione. Se viene specificata la clausola `WITH DATA`, la relazione così creata è popolata con le tuple risultato della valutazione dell'interrogazione, altrimenti viene creata una relazione vuota.

```
CREATE TABLE nome_tabella
[WITH DATA] AS (SELECT ...)
```

5.11 Viste

In SQL è possibile definire viste alternative degli stessi dati. Una vista è una relazione virtuale attrvaerso cui è possibile "vedere" i dati memorizzati nelle relazioni "reali" (dette di base). Il comando di definizione di una vista ha la seguente sintatti:

```
CREATE VIEW nome_vista [(lista colonne)]
AS interrogazione
[WITH [{LOCAL | CASCADED}] CHECK OPTION];
```

Una vista può essere cancellata tramite il comando:

```
DROP VIEW nome_vista {CASCADE | RESTRICT};
```

5.11.1 Interrogazioni su viste

Una volta definita, una vista è parte dello schema e può essere utilizzata nelle interrogazioni come una relazione di base.

5.11.2 Aggiornamenti su viste

Un ulteriore aspetto da tenere in considerazione riguarda le operazioni di aggiornamento eseguite attrvaerso le viste. L'esecuzione di un'operazione di aggiornamento su una vista viene propagata alla relazione su cui la vista è definita.

1. E' possibile eseguire il comando `DELETE` sulla vista V se l'interrogazione Q:
   - E' un'interrogazione su una singola relazione R
   - Non contiene la clausola `GROUP BY`, l'opzione `DISTINCT`, operatori insiemistici, né alcuna funzione di gruppo
   - Eventuali sotto-interrogazioni presenti nella clausola di selezione non fanno riferimento, né mediante correlazione né elencandola esplicitamente nella clausola `FROM`, alla relazione R
2. E' possibile eseguire il comando `UPDATE` su una colonna C di V se Q:
   - Soddisfa tutte le condizioni per il comando `DELETE`
   - C non è definita tramite un'espressione o funzione
3. E' possibile eseguire il comando `INSERT` su V se Q:
   - Soddisfa tutte le condizioni per il comando `UPDATE`
   - Qualsiasi colonna per cui valga il vincolo `NOT NULL` è inclusa nella vista

Per assicurare quindi che le tuple inserite tramite una vista siano accettate solo se verificano la condizione dell'interrogazione di definizione della vista, si deve specificare la clausola `CHECK OPTION` nel comando di definizione della vista. L'opzione `CHECK OPTION` può essere specificata con le opzioni `LOCAL` o `CASCADED`

- `LOCAL CHECK OPTION`: solo la clausola `WHERE` della vista in cui compare l'opzione viene verificata
- `CASCADED CHECK OPTION` (default): le clausola `WHERE` della vista in cui compare l'opzione e di tutte le viste su cui si basa eventualmente la sua definizione vengono verificate

CAPITOLO 6

Normalizzazione

La teoria della normalizzazione genera un nuovo schema equivalente al precedente ma normalizzato (senza ridondanze). Questa teoria si basa sull'analisi di determinati vincoli di integrità noti come dipendenze funzionali.

6.1 Ridondanze ed anomalie

Le anomalie che vogliamo eliminare tramite il processo di normalizzazione dipendono dalla presenza di dati ridondanti in una base di dati relazionale.

- Anomalie di modifica: si verificano quando si deve modificare lo stesso dato in più tuple
- Anomalie di cancellazione: si verificano quando si cancella una tupla che contiene informazioni che non si vogliono perdere
- Anomalie di inserimento: si verificano quando si vuole inserire una nuova tupla ma non si hanno tutti i dati necessari

L'obiettivo della teoria della normalizzazione è, quindi, quello di:

- Identificare situazioni che possono generare anomalie, partendo dall'analisi di alcuni tipi di vincoli di integrità validi per lo schema di partenza. Tali vincoli sono noti come dipendenze funzionali
- Identificare quali proprietà, rispetto alle dipendenze funzionali, uno schema deve soddisfare affincnhé non sia soggetto ad anomalie. Tali proprietà portano alla definizione di forme normali per gli schemi relazionali
- Fornire strumenti per trasformare lo schema relazionale di partenza in uno schema equivalente al precedente che soddisfi una forma normale e quindi non sia soggetto a determinati tipi di anomalie. Questo processo è noto come scomposizione e prevde la sostituzione di una relazione che presenta anomalie con un insieme di relazioni, che non presentano tale problema

La normalizzazione può portare ad inefficienze nelle prestazioni, nel caso in cui le interrogazioni richiedano frequentemente di combinare i dati suddivisi in relazioni distinte.

6.2 Dipendenze funzionali

Una dipendenza funzionale descrive un legame di tipo funzionale esistente tra gli attributi di una singola relazione, più formalmente:

Sia $R(A_1, \ldots, A_n)$ uno shcme di relazione. Siano X ed Y sottoinsiemi di U_R. Sia r un'istanta di $R(A_1, \ldots, A_n)$. r soddisfa $X \to Y$ se, per ogni coppia di tuple t_1 e t_2 in r, vale la seguente condizione: se $t_1[X] = t_2[X]$, allora $t_1[Y] = t_2[Y]$.
X determina funzionalmente Y in $R(A_1, \ldots, A_n)$, indicato con $X \to_{R(A_1,\ldots,A_n)} Y$ (o con $X \to Y$ in assenza di ambiguità) se qualunque istanza r di $R(A_1, \ldots, A_n)$ soddisfa $X \to Y$. $X \to R(A_1, \ldots, A_n)Y$ è chiamata dipendenza funzionale su U_R.
In modo compatto:

$$\forall r \in R \text{ se } t_1[X] = t_2[X] \text{ allora } t_1[Y] = t_2[Y]$$

Esempio:
`titolo` $\to$ `regista`, si legge come "uno stesso titolo non può avere registi diversi".

- Dipendenze funzionali banali: sempre valide
- Dipendenze funzionali derivate: sono implicate da altre dipendenze funzionali note

Esempio:

1. `titolo regista` $\to$ `genere`
2. `colloc` $\to$ `titolo regista`
3. `colloc` $\to$ `genere`

La terza è una dipendenza funzionale derivate dalle prime due.

6.2.1 Chiusura di un insieme di dipendenze funzionali

Indicheremo con F^+ l'insieme delle dipendenze funzionali logicamente implicate da F pertanto $F^+ = \{X \to Y | F \models X \to Y\}$.

Regole di Armstrong

Nel seguito D è un insieme di attributi:

- Riflessività: sia $Y \subseteq X \subseteq D$ allora $X \to Y$
- Additività: sia $Z \subseteq D$. Se $X \to Y$ allora $XZ \to YZ$
- Transitività: se $X \to Y$ e $Y \to Z$ allora $X \to Z$
- Unione: se $X \to Y$ e $X \to Z$ allora $X \to YZ$
- Pseudotransitività: se $X \to Y$ e $WY \to Z$ allora $WX \to Z$
- Scomposizione: sia $Z \subseteq Y$. Se $X \to Y$ allora $X \to Z$

Diciamo che $X \to Y$ è derivabile da F, indicato con $F \vdash X \to Y \iff F \models X \to Y$.

6.2.2 Chiusura di un insieme di attributi

Sia F un insieme di dipendenze funzionali su un insieme di attributi D. Sia $X \subseteq D$. La chiusura di X rispetto ad F, denotata con X^+, è l'insieme $\{A \in D | F \vdash X \to A\}$.
Esempio: $D = \{colloc, titolo, regista, anno, genere, valutaz, tipo\}$

1. `titolo regista` $\to$ `anno`
2. `titolo regista` $\to$ `genere`
3. `titolo regista` $\to$ `valutaz`
4. `colloc` $\to$ `titolo regista`
5. `colloc` $\to$ `tipo`

Prendiamo $X =$ colloc, $X^+ =$ {titolo,regista,tipo,anno,genere,valutaz,colloc}

6.2.3 Chiave

Sia $R(A_1, \ldots, A_n)$ uno schema di relazione, K un sottoinsieme di $A_1, \ldots, A_n$ e F un insieme di dipendenze funzionali su $A_1, \ldots, A_n$. K è una chiave per R se:

1. $K \to A_1, \ldots, A_n \in F^+$
2. Non esiste $Y \subset K$ tale che $Y \to A_1, \ldots, A_n \in F^+$

In poche parole:

$$K^+ = \{A_1, \ldots, A_n\}$$

Esempio:
Considerando l'esempio di prima, $X =$ colloc è chiave perchè $R \subseteq X^+$
Se un attributo non compare mai a destra di una dipendenza funzionale in F, questo attributo farà <u>certamente</u> parte della chiave.

6.3 Forme normali

6.3.1 Forma normale di Boyce-Codd (BCNF)

Sia $R(A_1, \ldots, A_n)$ uno schema di relazione. Sia F un insieme di dipendenze funzionali su U_R. $R(A_1, \ldots, A_n)$ è in forma normale di Boyce-Codd rispetto ad F se per ogni dipendenza funzionale $X \to Y \in F$, con $Y \not\subseteq X$, allora X è una chiave o super-chiave di R.

6.3.2 Terza forma normale (3NF)

Attributo primo

Un attributo A_i è un attributo primo di R rispetto a F se A_i è contenuto in una chiave di R.

Terza forma normale

Sia $R(A_1, \ldots, A_n)$ uno schema di relazione. Sia F un insieme di dipendenze funzionali su U_R. $R(A_1, \ldots, A_n)$ è in terza forma normale rispetto ad F se per ogni dipendenza funzionale $X \to Y \in F$, con $Y \not\subseteq X$, allora X è una chiave o super-chiave di R oppure $\forall A \in Y$, A è un attributo primo per $R(A_1, \ldots, A_n)$.

6.4 Scomposizione di schemi relazionali

La scomposizione di uno schema di relazione $R(A_1, \ldots, A_n)$ è la sua sostituzione con un insime di schemi relazionali:

$$\Sigma = \{R_1, \ldots, R_k\}$$

tali che:

$$U_R = U_{R_1} \cup \ldots \cup U_{R_k}$$

Gli schemi di attributi dei vari schemi R non devono essere necessariamente disgiunti, pertanto uno stesso attributo di R può comparire in più schemi R_i.

6.4.1 Proprietà

Lossless Join

Se una relazione viene scomposta, è importante che sia possibile riottenere esattamente la stessa relazione eseguendo il join naturale delle relazioni in cui è stata scomposta (lossless join):

$$r = \pi_{R_1}(r) \bowtie \ldots \bowtie \pi_{R_k}(r)$$

cioè r è il join naturale delle sue proiezioni sugli schemi $R_1, \ldots, R_k$. *Con r istanza di R.* Una scomposizione per uno schema di relazione ha la proprietà di lossless join se soddisfa una delle seguenti implicazioni:

- $(U_{R_1} \cap U_{R_2}) \to (U_{R_1} - U_{R_2})$
- $(U_{R_1} \cap U_{R_2}) \to (U_{R_2} - U_{R_1})$

Se gli schemi di una scomposizione sono disgiunti, la scomposizione non è lossless join.

Proiezione di insiemi di dipendenze funzionali

Sia D un insieme di attributi. Sia F un insieme di dipendenze funzionali su D. Sia $Z \subseteq D$. La proiezione di F su Z, denotata con $\Pi_Z(F)$, è l'insieme $\{X \to Y | X \to Y \in F^+ | XY \subseteq Z\}$. Una scomposizione preserva un insieme di dipendenze funzionali F se l'unione di tutti gli insiemi di dipendenze $\Pi_{R_i}(F), i = 1, \ldots, k$, implica logicamente tutte le dipendenze in F.

Scomposizione che preserva le dipendenze

Sia $R(A_1, \ldots . A_n)$ uno schema di relazione. Sia $\Sigma = \{R_1, \ldots, R_k\}$ una scomposizione per $R(A_1, \ldots . A_n)$. Sia F un insieme di dipendenze funzionali su U_R. Σ preserva le dipendenze in F se:

$$\bigcup_{i=1,\ldots,k} \Pi_{R_i}(F) \models F$$

CAPITOLO 7

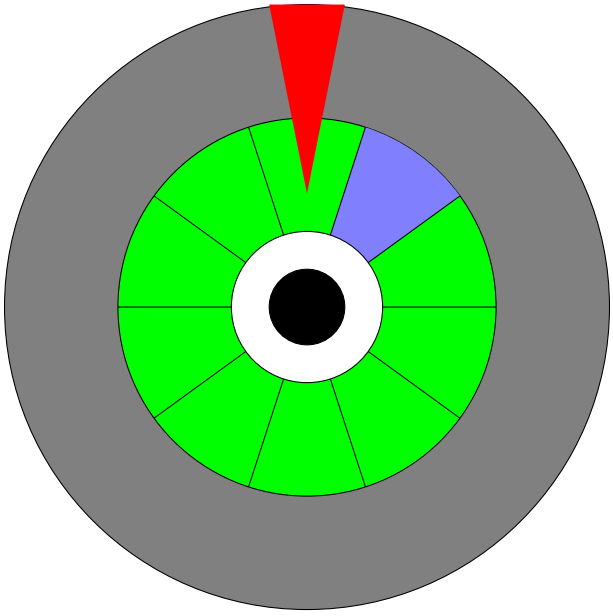
Gestore delle strutture di memorizzazione

7.1 Gerarchia delle memorie

La gerarchia può essere vista come una piramide, i dischi magnetici saranno alla base (lento, economico e capiente) mentre la memoria principale sarà in alto (veloce, costosa e di piccola capacità). Per calcolare il tempo di accesso a una memoria si può usare la formula:

$$T_{accesso} = Latenza + \frac{\text{Dimensione dati da trasferire}}{\text{Velocità di Trasferimento}}$$

7.2 Disco magnetico



- Tracce: superficie in cui l'informazione è memorizzata (colore verde, cerchio intermedio)
- Settore: porzione di traccia (colore blu)
- Cilindro: insieme di tracce allineate verticalmente
- Piatto: colore grigio, cerchio esterno
- Asse di rotazione: asse attorno al quale ruota il piatto in senso orario (colore nero, cerchio interno)

Un disco è composto da uno o più piatti, per leggere e scrivere utilizza delle testine (una per ogni piatto, nel disegno di colore rosso). Le testine si muovono allo stesso tempo e non sono indipendenti l'una dall'altra.
Se i dati sono memorizzati su uno stesso cilindro (anche di tracce diverse) possono essere recuperati molto più velocemente che non dati distribuiti su diversi cilindri (il movimento della testina è molto lento).

- Command Overhead: tempo impiegato a impartire comandi al drive
- Seek Time: tempo impiegato dal braccio[1] a posizionarsi sulla traccia desiderata
- Settle Time: tempo impiegato per la stabilizzazione del braccio
- Rotational Latency: tempo di attesa dal primo settore da leggere

7.2.1 Tempo di trasferimento

Il tempo di trasferimento è il tempo impiegato per trasferire un certo numero di byte e si riferisce alla velocità con cui si trasferiscono byte dai (sui) piatti sulla (dalla) cache del controller. Dipende dalla velocità di trasferimento (o Transfer Rate Tr).
Un blocco (o pagina) è una sequenza contigua di settori su una traccia, costituisce l'unità di I/O per il trasferimento dei dati tra il disco e la memoria principale.
Il compito del gestore delle strutture di memorizzazione è quello di ridurre i tempi di latenza.

7.3 Il Database Fisico

A livello fisico di un DB consiste in un insieme di file, ognuno dei quali viene visto come una collezione di pagine di dimensione fissa.

LIVELLO LOGICO	LIVELLO FISICO
Relazione (tabella)	File
Tupla	Record

7.3.1 File

Un file è una sequenza di record, è detto file con record a lunghezza fissa se tutti i record hanno la stessa dimensione in byte, altrimenti parliamo di file con record a lunghezza variabile.

7.3.2 Record

Un record è costituito da un insieme di valori collegati e in generale ogni record include un header che può contenere:

- Identificatore della relazione cui il record appartiene
- Identificatore univoco RID del record nel DB
- Timestamp che indica quando il record è stato inserito o modificato per l'ultima volta
- Lunghezza record

[1] Il braccio è quello che tiene tutte le testine allineate

7.4 Allocazione dei file

7.4.1 Allocazione contigua

- I blocchi dei file sono allocati in blocchi di disco contigui quindi non si deve muovere troppo la testina
- Rende molto efficiente le letture dell'intero file
- Gli aggiornamenti sono costosi (se si ha un file pieno e si vuole aggiungere un record, devo spostare tutti i blocchi in un'altra posizione di lunghezza +1)

7.4.2 Allocazione concatenata

- Ogni blocco di un file contiene un puntatore al successivo blocco del file
- Gli aggiornamenti sono molto efficienti
- La lettura dell'intero file è molto lenta
- Utilizzo di bucket non necessariamente contigui ma vicini (possibilmente nello stesso cilindro), per gruppi di record tra loro collegati

7.5 Organizzazione dei record nei file

- File heap: i record vengono memorizzati in ordine di inserimento
- File ordinato su X: i record vengono memorizzati in ordine rispetto ad un campo X
- File hash su X: i record vengono memorizzati in ordine parziale rispetto ad un campo X (vengono raggruppati in base al valore di X)

File heap

colloc	dataNol	codCli	dataRest
1111	01-Mar-2006	6635	02-Mar-2006
1115	01-Mar-2006	6635	02-Mar-2006
1118	10-Mar-2006	6642	11-Mar-2006
1117	02-Mar-2006	6635	06-Mar-2006
1118	02-Mar-2006	6635	06-Mar-2006
1111	04-Mar-2006	6642	05-Mar-2006
1119	08-Mar-2006	6635	10-Mar-2006
1120	08-Mar-2006	6635	10-Mar-2006
1116	08-Mar-2006	6642	09-Mar-2006
1121	15-Mar-2006	6635	18-Mar-2006
1122	15-Mar-2006	6635	18-Mar-2006
1113	15-Mar-2006	6635	18-Mar-2006
1129	15-Mar-2006	6635	20-Mar-2006
1119	15-Mar-2006	6642	16-Mar-2006
1126	15-Mar-2006	6610	16-Mar-2006
1112	16-Mar-2006	6610	18-Mar-2006
1114	16-Mar-2006	6610	17-Mar-2006
1128	18-Mar-2006	6642	20-Mar-2006
1124	20-Mar-2006	6610	21-Mar-2006
1115	20-Mar-2006	6610	21-Mar-2006
1124	21-Mar-2006	6642	22-Mar-2006
1117	21-Mar-2006	6610	?
1127	22-Mar-2006	6635	?
1125	22-Mar-2006	6635	?
1122	22-Mar-2006	6642	?
1113	22-Mar-2006	6642	?
1116	21-Mar-2006	6610	?

File ordinato su dataNol

colloc	dataNol	codCli	dataRest
1111	01-Mar-2006	6635	02-Mar-2006
1115	01-Mar-2006	6635	02-Mar-2006
1117	02-Mar-2006	6635	06-Mar-2006
1118	02-Mar-2006	6635	06-Mar-2006
1111	04-Mar-2006	6642	05-Mar-2006
1119	08-Mar-2006	6635	10-Mar-2006
1120	08-Mar-2006	6635	10-Mar-2006
1116	08-Mar-2006	6642	09-Mar-2006
1118	10-Mar-2006	6642	11-Mar-2006
1121	15-Mar-2006	6635	18-Mar-2006
1122	15-Mar-2006	6635	18-Mar-2006
1113	15-Mar-2006	6635	18-Mar-2006
1129	15-Mar-2006	6635	20-Mar-2006
1119	15-Mar-2006	6642	16-Mar-2006
1126	15-Mar-2006	6610	16-Mar-2006
1112	16-Mar-2006	6610	18-Mar-2006
1114	16-Mar-2006	6610	17-Mar-2006
1128	18-Mar-2006	6642	20-Mar-2006
1124	20-Mar-2006	6610	21-Mar-2006
1115	20-Mar-2006	6610	21-Mar-2006
1124	21-Mar-2006	6642	22-Mar-2006
1116	21-Mar-2006	6610	?
1117	21-Mar-2006	6610	?
1127	22-Mar-2006	6635	?
1125	22-Mar-2006	6635	?
1122	22-Mar-2006	6642	?
1113	22-Mar-2006	6642	?

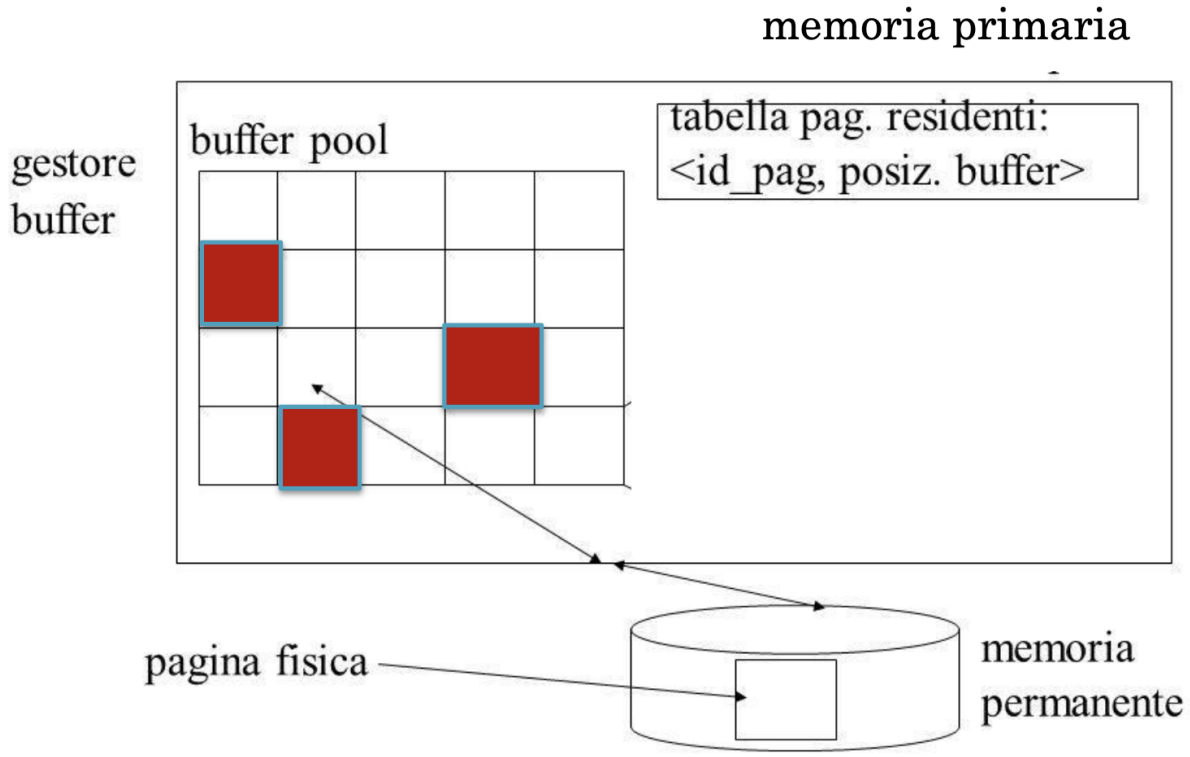
File hash su dataNol

colloc	dataNol	codCli	dataRest
1117	02-Mar-2006	6635	06-Mar-2006
1118	02-Mar-2006	6635	06-Mar-2006
1111	04-Mar-2006	6642	05-Mar-2006
1119	08-Mar-2006	6635	10-Mar-2006
1120	08-Mar-2006	6635	10-Mar-2006
1116	08-Mar-2006	6642	09-Mar-2006
1111	01-Mar-2006	6635	02-Mar-2006
1115	01-Mar-2006	6635	02-Mar-2006
1118	10-Mar-2006	6642	11-Mar-2006
1121	15-Mar-2006	6635	18-Mar-2006
1122	15-Mar-2006	6635	18-Mar-2006
1113	15-Mar-2006	6635	18-Mar-2006
1129	15-Mar-2006	6635	20-Mar-2006
1119	15-Mar-2006	6642	16-Mar-2006
1126	15-Mar-2006	6610	16-Mar-2006
1112	16-Mar-2006	6610	18-Mar-2006
1114	16-Mar-2006	6610	17-Mar-2006
1124	20-Mar-2006	6610	21-Mar-2006
1115	20-Mar-2006	6610	21-Mar-2006
1128	18-Mar-2006	6642	20-Mar-2006
1124	21-Mar-2006	6642	22-Mar-2006
1116	21-Mar-2006	6610	?
1117	21-Mar-2006	6610	?
1127	22-Mar-2006	6635	?
1125	22-Mar-2006	6635	?
1122	22-Mar-2006	6642	?
1113	22-Mar-2006	6642	?

7.6 Gestione del buffer

L'obiettivo è minimizzare il numero di accessi alla memoria non volatile, che è molto lenta. Il buffer viene gestito dal DBMS. Mantenere pi'u blocchi possibili in memoria principale in modo da evitare riletture da memoria non volatile.

Il buffer è organizzato in pagine, che hanno la stessa dimensione delle pagine/blocchi su disco.



Dopo l'esecuzione di una interrogazione, il **Buffer Manager** (BM) controlla prima nel buffer e se la pagina non è presente:

- Cerca una pagina nel buffer libera
- Se non è presente, cerca una pagina da sostituire
- Se la pagina da sostituire è stata modificata, la scrive su disco
- A questo punto la pagina è libera e può essere sovrascritta

Quando una pagina è presente nel buffer, le operazioni di lettura e scrittura possono essere effettuate su di essa.
Accedere alle pagine nel buffer invece che alle corrispondenti pagine su disco influenza notevolmente le prestazioni.

7.6.1 Politiche di sostituzione

Il BM sceglie quale politica usare in base alle informazioni che ha.

LRU (Least Recently Used) : La pagina meno recentemente usata viene sostituita.

MRU (Most Recently Used) : La pagina più recentemente usata viene sostituita. Spesso viene usata questa politica perchè il sistema sa che dovrà rileggere questo blocco ma non nell'immediato futuro.

7.7 Indici

7.7.1 Indici ad albero (o ordinati)

Terminologia

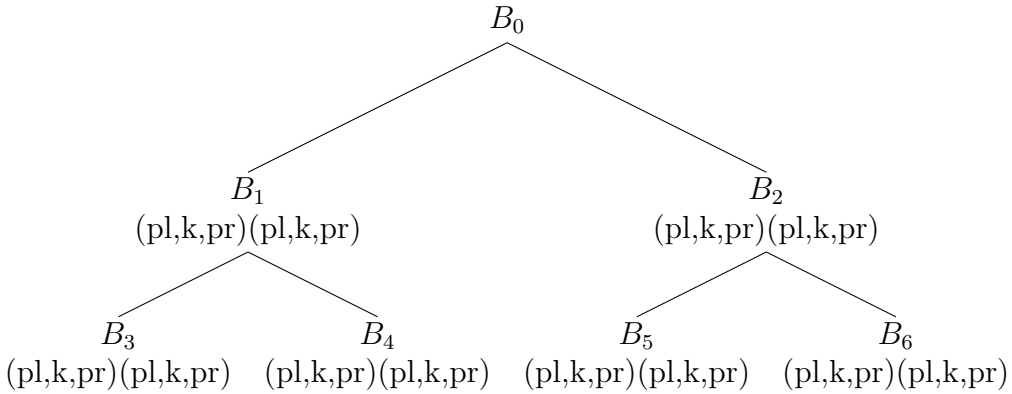
- **Organizzazione primaria**: l'insieme dei file dei dati
- **Organizzazione secondaria**: l'insieme dei file per indici ordinati

Le coppie (k_i, r_i) vengono memorizzate in un file su disco (**organizzazione secondaria**), ordinate rispetto ai valori della chiave k_i. Se l'indice è molto piccolo, può essere tenuto in memoria principale. Altrimenti è necessario tenerlo su disco, quindi per rendere più efficiente l'accesso alle coppie dell'indice si usa una struttura multilivello.
L'indice assume una **struttura ad albero** in cui ogni nodo dell'albero corrisponde a un blocco dati.

Requisiti

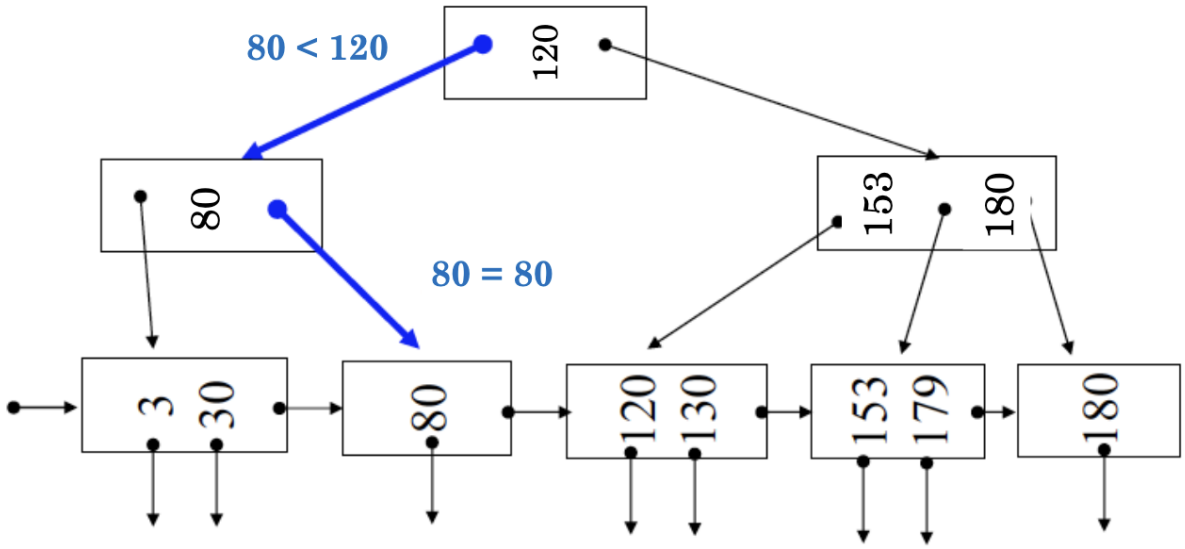
- Bilanciato: l'indice è bilanciato rispetto ai blocchi e non ai singoli nodi (il cammino dalla radice a ogni blocco deve essere distante uguale)
- Occupazione minima: per evitare un sotto-utilizzo della memoria viene stabilito un limite inferiore all'utilizzazione dei blocchi
- Efficienza di aggiornamento: il costo delle operazioni di aggiornamento è comunque limitato

In un indice ad albero ogni nodo corrisponde a un blocco e il costo delle operazioni (ricerca, inserimento e cancellazione) in tali strutture è lineare nell'altezza dell'albero e logaritmico nel numero di elementi memorizzati nell'indice. Il numero massimo di elementi (coppie) memorizzabili in un nodo è $m-1$ dove m dice quanti puntatori al massimo sono contenuti in un nodo (m può essere maggiore di 2 quindi l'albero non è solo binario). Il numero minimo di elementi è $\lceil \frac{m}{2} \rceil - 1$ ovvero l'albero è "pieno" almeno al 50%.



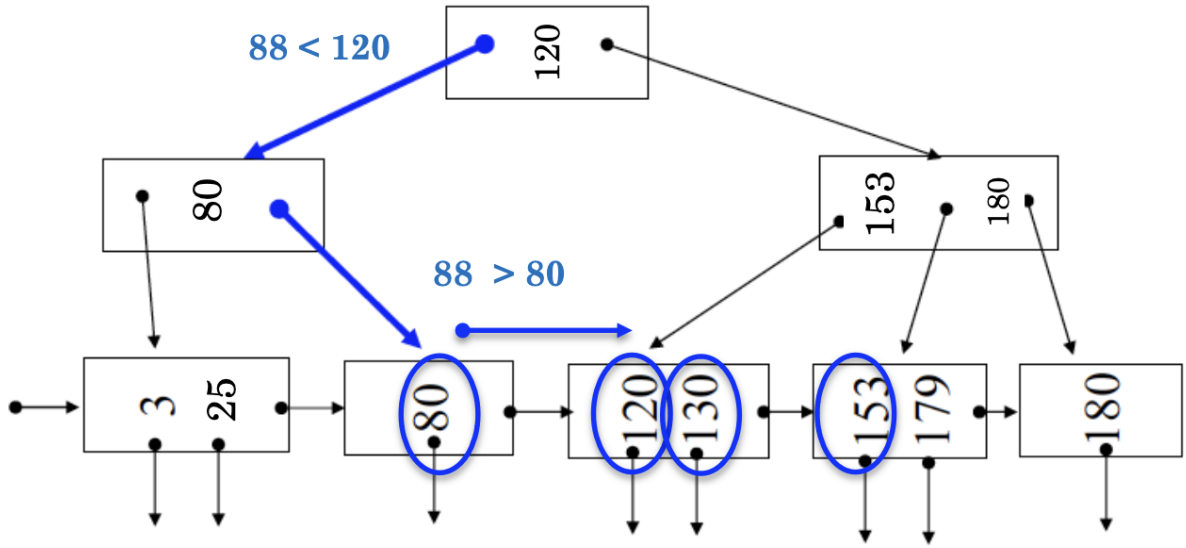
Nelle parentesi ci sono i puntatori ai figli. Nel caso delle foglie i puntatori punteranno all'organizzazione primaria. I nodi foglia hanno anche i puntatori ai nodi foglia successivi come una lista. I nodi puntano ai figli in base al valore della chiave (*pl* è il puntatore sinistro, *pr* è il puntatore destro).

Ricerca per uguaglianza



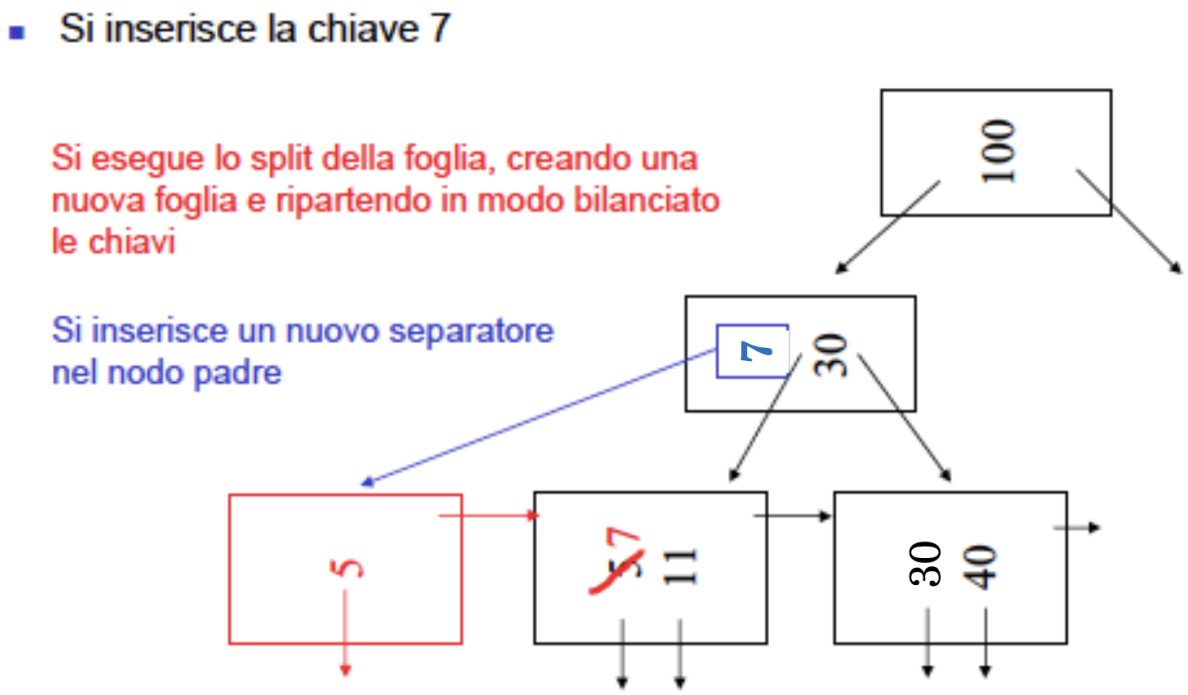
1. La radice viene caricata nel buffer e il valore viene confrontato con il valore di ricerca (K) se è minore si scende a sinistra, altrimenti centrale o destra
2. Si scende nel sottoalbero e viene controllato che il valore sia quello ricercato, se lo è scendo nel sottoalbero puntato dal numero ricercato.
3. Ritorno al passo 1 fino a quando il nodo non punta all'organizzazione primaria. Se il valore non esiste non si accede ai file dati.

Ricerca per intervallo



Molto simile alla ricerca per uguaglianza solo che se non c'è controllo qual è la prima foglia a soddisfare la condizione (ad esempio nell'immagine il numero 88 è maggiore del numero più "vicino" quindi so che tutti i nodi a destra di quel nodo potrebbero soddisfare la condizione).

Inserimento



Lo split avviene nel primo nodo pieno che si incontra. Per nodo pieno si intende che non c'è spazio per inserire un nuovo separatore.

7.7.2 Indici Clusterizzati e non Clusterizzati

Un indice ad albero è clusterizzato se il file dei dati è ordinato rispetto alla chiave di ricerca, in caso contrario è non clusterizzato. Le operazioni di inserimento, cancellazione e aggiornamento nel file ordinato sono facilitate dalla presenza dell'indice. Al più un indice può essere clusterizzato per tabella. Se l'indice è su un campo chiave, che sia clusterizzato o meno, punteranno a un solo blocco. Se l'indice è su un campo non chiave, l'indice punterà a blocchi diversi.

Indici su più attributi

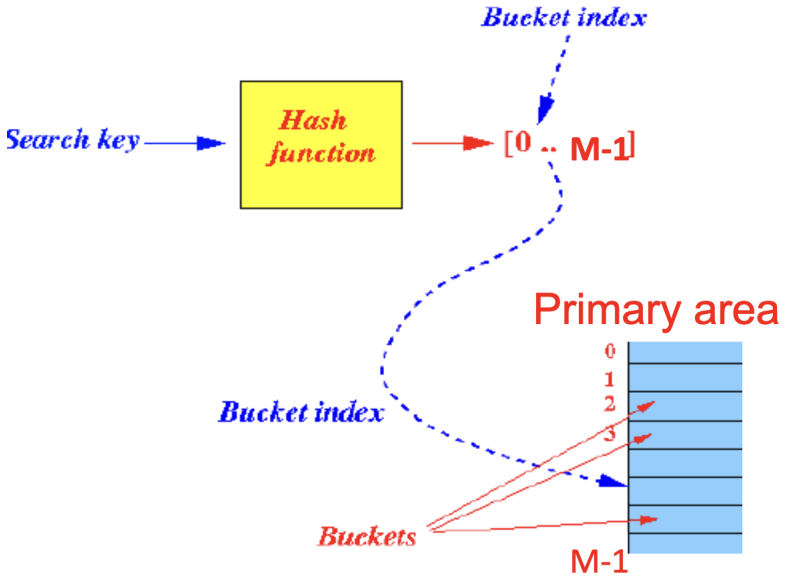
Indice la cui chiave di ricerca è costituita da più attributi, altrimenti si chiama indice su singolo attributo.

7.7.3 Indici Hash

Una funzione hash ottimale deve distribuire uniformemente le chiavi nello spazio negli indirizzi e distribuire casualmente le chiavi. Una funzione è detta perfessa se non vengono prodotti trabocchi. Può essere sempre definita disponendo di un'area primaria con capacità complessiva pari al numeri dei record da memorizzare. Le funzioni hash operano su insiemi di chiavi intere (se le chiavi sono alfanumeriche si può associare un id prima di applicare la trasformazione).
La funzione hash è quindi definita come:

$$h(k) = k \mod M$$

Per avere una buona distribuzione delle chiavi è opportuno che M sia un numero primo oppure ≤ 20.
L'uso di indici ad albero ha lo svantaggio di richiedere la scansione di una struttura dati, memorizzata su disco, per localizzare i dati. Questo perché le associazioni (k_i, r_i) vengono mantenute in forma esplicita, come record in un file Gli indici hash al contrario mantengono le associazioni (k_i, r_i) in modo implicito, tramite l'uso di una funzione hash, definita sul dominio della chiave di ricerca.



A ogni valore della funzione hash corrisponde un indirizzo in area primaria.
Esempio:
$I_A(R)$ $H(D_A \to \{0, \ldots, 2\})$, la funzione H restituisce il numero del bucket.

- $H(1) = 1$
- $H(2) = 2$
- $H(3) = 0$
- $H(5) = 2$
- $H(10) = 1$

Bucket index:

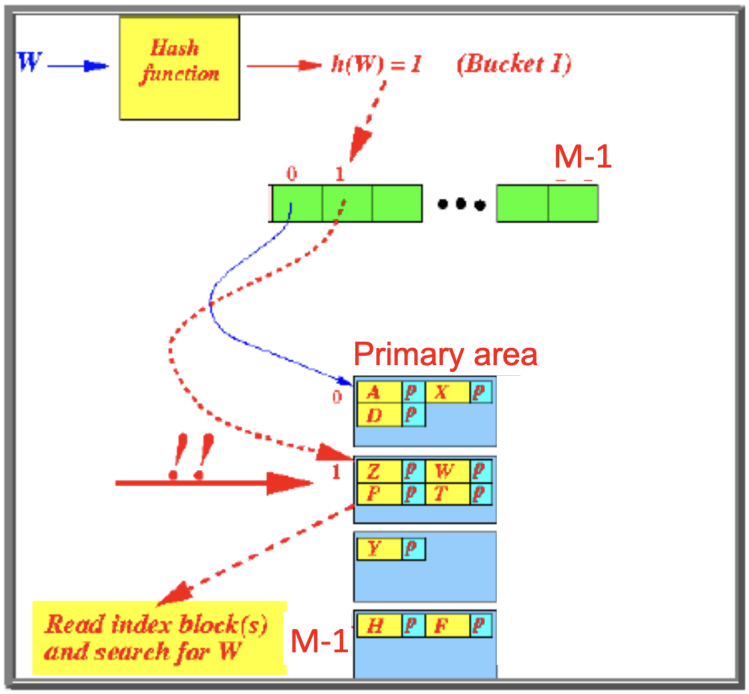
0	1	2

A	B
1	a
2	a
3	b
5	c
10	d
3	f

$\Rightarrow$

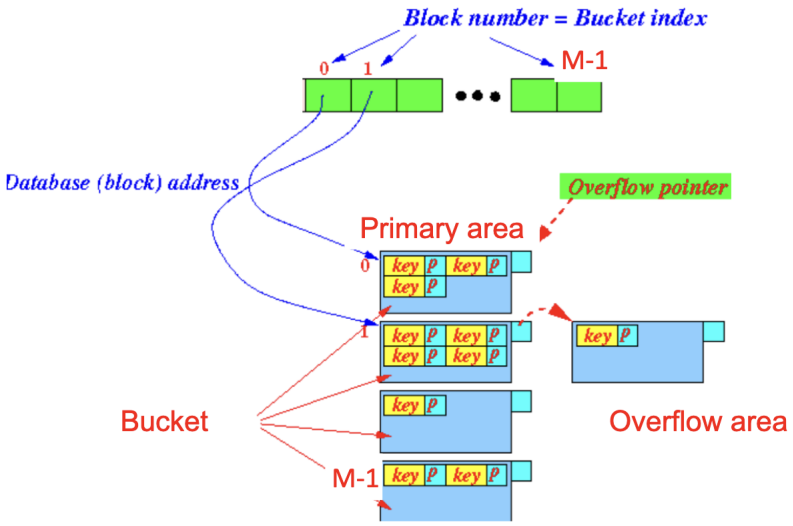
Bucket 0	
3	b
3	f
Bucket 1	
1	a
10	d
Bucket 2	
2	a
5	c

Ricerca per uguaglianza



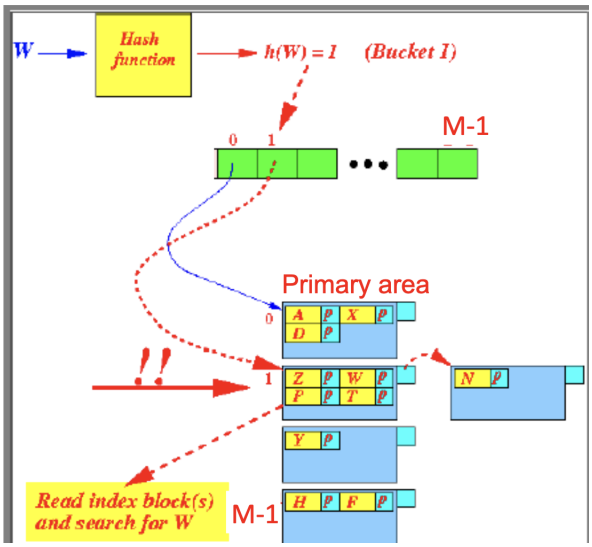
Non supporta la ricerca per intervallo in quanto dovrei conoscere tutti i valori dell'intervallo in quanto la funzione hash non mantiene l'ordine.

Inserimento e Trabocchi



Se il bucket è pieno, si alloca un nuovo blocco (*trabocco*) dove c'è spazio chiamato area di overflow e se anche l'area di overflow è piena se ne crea un'altra e così via.

Ricerca

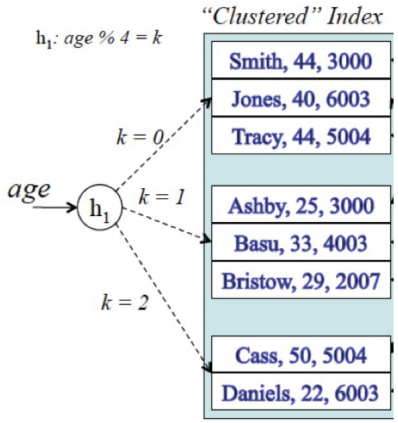


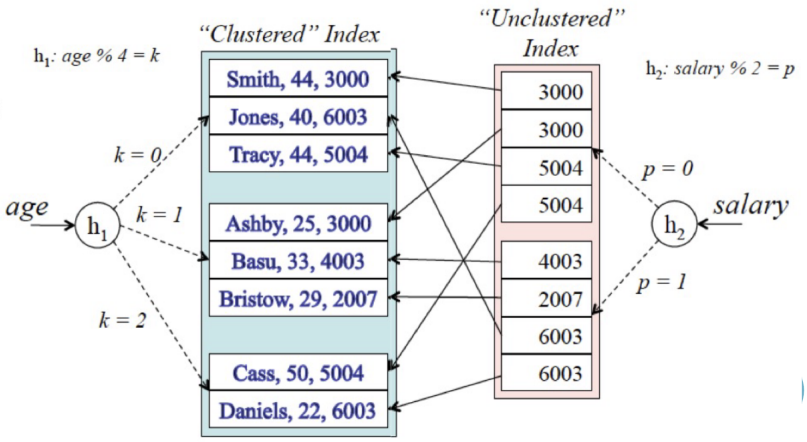
In generale i costi di una qualsiasi operazione usando un indice hash sono:

- In assenza di overflow, il costo di accesso a indice è costante
- In presenza di overflow, il costo non è facilmente determinabile
  - Quanti blocchi di overflow per il blocco acceduto?
  - Dove sono memorizzati?

Indici Hash clusterizzati e non clusterizzati

E' clusterizzato se i record con chiavi simili sono memorizzati nello stesso bucket. In caso contrario è non clusterizzato.
Finora abbiamo considerato solo gli indici clusterizzati.
Un file dei dati di tipo hash è sempre associato a un indice hash clusterizzato.
In presenza di un indice hash **clusterizzato** l'organizzazione primaria corrisponde ai record memorizzati nell'area primaria + i record memorizzati negli overflow.
Esempio:





Gli indici non clusterizzati sono indici multilivello.
Per selezioni di uguaglianza sono preferibili gli indici hash, per le selezioni di range sono preferibili gli indici ad albero.