DOMANDE FILTRO

Si illustri il costrutto di IDENTIFICATORE (interno e esterno) nel modello ER

gli identificatori vengono specificati per ciascuna entità e descrivono i concetti che permettono di identificare in maniera univoca le occorrenze delle entità.

- se uno o più attributi dell'entità sono sufficienti a individuare un identificatore, si parla di identificatore interno;
- nei casi in cui l'identificatore di una entità è ottenuto utilizzando altre entità si parla di identificatore esterno.

Si illustri il costrutto di ENTITÀ DEBOLE (o con identificatore esterno) nel modello ER

Una entità E può essere identificata da altre entità solo se tali entità sono coinvolte in una relazione a cui E partecipa con cardinalità (1,1).

Si illustri il costrutto di ATTRIBUTO del modello ER

Descrivono le proprietà elementari di entità e relazioni che sono di interesse ai fini dell'applicazione.

Si illustri il costrutto di ENTITÀ del modello ER

Le entità rappresentanoclassi di oggetti che hanno proprietà comuni ed esistenza "autonoma". Hanno inoltre una identificazione univoca.

Si illustri il costrutto di TUPLA del modello ER

Una tupla su un insieme di attributi X è una funzione *t* che associa a ciascun attributo A appartenente X un valore del dominio dom(A).

Si illustri il costrutto di RELAZIONE del modello ER

Una RELAZIONE rappresenta un legame logico tra due o più Entità

Si illustri il costrutto di RELAZIONE del modello Relazionale

Una relazione su X è un insieme di tuple su X.

Si illustri il costrutto di SUPERCHIAVE del modello Relazionale

Un insieme K di attributi è superchiave di una relazione *r* se non contiene due tuple distinte t_1 e t_2 con $[t_1]=[t_2]$.

Si presenti la definizione di join naturale dell'algebra relazionale

Join naturale (su r1 di schema X1 e r2 di schema X2):

$r1r2 = \{t \mid \exists t1 \in r1: \exists t2 \in r2: t1 = t[X1] \wedge t2 = t[X2] \}$

DEFINIZIONE:

Un **DBMS** è un sistema che gestisce su memoria secondaria collezioni di dati (chiamate "Basi di Dati") grandi, condivise e persistenti, assicurando affidabilità, privatezza e accesso efficiente.

PROGETTAZIONE CONCETTUALE:

Obbiettivo: Rappresentare il contenuto informativo della base di dati in modo formale ma indipendente dall'implementazione (DBMS) a dalle operazioni

PROGETTAZIONE LOGICA:

Obbiettivo: Tradurre lo schema concettuale nello schema logico aderente al modello dei dati del DBMS scelto per l'implementazione. Nella traduzione si tiene conto delle operazioni più frequenti che le applicazioni eseguiranno sulla base di dati.

PROGETTAZIONE FISICA:

Obbiettivo: Completare lo schema logico con i parametri relativi alla memorizzazione fisica dei dati e con gli opportuni metodi d'accesso (INDICI)

DEFINIZIONE:
Una **ENTITA'** rappresenta una classe di oggetti con le seguenti caratteristiche:

- hanno proprietà comuni
- hanno esistenza "Autonoma" rispetto alle altre classi ad oggetti dello schema
- hanno "identificazione univoca"

DEFINIZIONE:
Una **RELAZIONE** rappresenta un legame logico tra due o più Entità

DEFINIZIONE:
Un **ATTRIBUTO** rappresenta proprietà elementari di Entità o Relazioni

DEFINIZIONE:
Data una Relazione R che coinvolge *n* Entità $E_1, \dots, E_n$, i **VINCOLI DI CARDINALITA'** vanno precisati per ogni Entità E_i che partecipa alla relazione R e specificano il numero minimo e il numero massimo di occorrenze di R a cui ogni occorrenza di E_i deve/può partecipare

Valori tipici di **min**:
0: indica che la partecipazione delle occorrenze di E_i alla Relazione R è opzionale
1: indica che la partecipazione delle occorrenze di E_i alla Relazione R è obbligatoria
n >1: indica che ogni occorrenza di E_i deve partecipare ad almeno *n* occorrenze di R

Valori tipici di **max**:
1: indica che al massimo ogni occorrenza di E_i partecipa a una e una sola occorrenza di R
N: indica che ogni occorrenza di E_i può partecipare a più occorrenze di R
n >1: indica che ogni occorrenza di E_i può partecipare al massimo a *n* occorrenze di R

DEFINIZIONE:
L'**IDENTIFICATORE DI ENTITA'** è un insieme di proprietà (attributi o altre Entità legate via Relazione) dell'Entità E che *identificano univocamente* le occorrenze di E

- identificatore INTERNO: uno o più attributi dell'entità sono suff a individuare un identificatore
- identificatore ESTERNO:nei casi in cui l'identificatore di entità è ottenuto utilizzando altre entità

CLASSIFICAZIONE delle Relazioni binarie:

- relazione UNO A UNO: Accade se entrambe le Entità che partecipano alla Relazione presentano cardinalità max pari a 1, indipendentemente dalla cardinalità min
- relazione UNO A MOLTI: Si ha se una delle entità coinvolte nella Relazione presenta cardinalità max pari a 1, mentre l'altra presenta cardinalità max maggiore di 1 (o N)
- relazione MOLTI A MOLTI: Se entrambe le Entità coinvolte nella Relazione presentano Cardinalità max maggiore di 1 (o N)

DEFINIZIONE:
L'**ATTRIBUTO OPZIONALE e/o MULTIVALORE** si ottiene specificando un vincolo di cardinalità sui valori che l'attributo può/deve assumere. Se non si specifica nulla, il vincolo implicito è (1,1)
DEFINIZIONE:
L'**ATTRIBUTO COMPOSTO** premette di raggruppare attributi di Entità (o Relazione) per affinità di significato

DEFINIZIONE:
Una **RELAZIONE TERNARIA** è una Relazione alla quale partecipano 3 Entità, offrono una rappresentazione di serie statiche, o relazioni storicizzate.

DEFINIZIONE:

Una **GENERALIZZAZIONE** rappresenta un legame logico (simile ad una ereditarietà tra classi) tra una Entità Padre E e *n* (*n>0*) Entità Figlie E_1, … , E_n, dove E rappresenta una classe di oggetti più generale rispetto alle classi di oggetti rappresentate da E_1, … , E_n.

→ Proprietà delle occorrenze delle Entità rappresentate da una generalizzaizione:

- proprietà INTENSIONALE: Tutte le proprietà (attributi, Relazioni, identificatori) specificate sull'Entità Padre, sono implicitamente anche proprietà delle occorrenze delle Entità Figlie
- proprietà ESTENSIONALE: Ogni occorrenza delle Entità Figlie è anche occorrenza dell'Entità Padre

→ Proprietà delle GENERALIZZAZIONI:

- Una generalizzazione si dice TOTALE (t) se ogni occorrenza del Padre è anche occorrenza dei almeno un'Entità Figlia
- Una generalizzazione si dice ESCLUSIVA (e) se ogni occorrenza del Padre è al più anche occorrenza di una sola Entità Figlia
- Una generalizzazione NON TOTALE si dice PARZIALE (p)
- Una generalizzazione NON ESCLUSIVA si dice SOVRAPPOSTA (s)

→ Uso delle generalizzazioni sugli schemi:

- Per fattorizzare proprietà comuni ad un'Entità dello schema
- Per specializzare Entità dello schema

STRATEGIE POSSIBILI DI PROGETTO:

- ***TOP-DOWN***:
  - Considerare i requisiti globalmente e produrre uno schema COMPLETO ma contenente pochi concetti astratti
  - Raffinare i concetti per inglobare nello schema tutti i dettagli descritti nei requisiti
- ***BOTTOM-UP***:
  - Scomporre i requisiti in parti elementari (frasi che descrivono un concetto)
  - Produrre degli schemi per le parti elementari
  - Fondere gli schemi prodotti al passo precedenti per ottenere lo schema finale
- ***INSIDE-OUT***:
  - Indovinare i requisiti dei concetti guida
  - produrre gli schemi per i concetti guida
  - integrare gli schemi prodotti per i concetti guida generando lo schema concettuale finale

ANALISI DI QUALITA' DELLO SCHEMA CONCETTUALE:

- **Verifica di Correttezza**: Uno schema è CORRETTO se usa propriamente i costrutti del modello dei dati adottato
  - Errori SINTATTICI: si verificano quando i costrutti sono usati senza rispettare le regole sintattiche del modello
  - Errori SEMANTICI: si verificano quando i costrutti sono usati in modo non aderente ai requisiti
- **Verifica di Completezza**: Uno schema è COMPLETO se rappresenta tutti i dettagli descrittti nei requisiti, e se tutte le eventuali operazioni descritte nelle specifiche possono essere eseguite a partire dai dati dello schema
- **Verifica di Minimalità**: Uno schema è MINIMALE se ogni concetto descritto nei requisiti è rappresentato nello schema una solo volta
- **Verifica di Leggibilità**: Uno schema è LEGGIBILE se rappresenta i requisiti in modo naturale al valore inserito

1. Anomalie di AGGIORNAMENTO: si verifica quando per modificare il valore di un attrubuto si è obbligati a modificare tale valore su più tuple della BDD
2. Anomalie di INSERIMENTO: si verifica quando per inserire una nuova tupla è necessario inserire valori al momento NON conosciuti (sostituibili da valori nulli) per gli attributi NON disponibili
3. Anomalie di CANCELLAZIONE: si verificano quando per cancellare una tupla è necessario cancellare valori ancora validi per un sottoinsieme di attributi

Cosa rappresenta il **VALORE NULLO**?

I casi possibili sono

- Il valore dell'attributo è sconosciuto, quindi il valore per l'attributo esiste, ma non è al momento noto alla BDD
- Il valore dell'attributo è inesistente, quindi il valore dell'attributo per quella tupla non esiste
- il valore dell'attributo è sconosciuto o inesistente

CLASSIFICAZIONE dei vincoli:

- **VINCOLI DI DOMINIO**: impongono una restrizione ai valori che un attributo può assumere
- **VINCOLI DI TUPLA**: impongono una restrizione alla combinazione di valori che una tupla può assumere
- **VINCOLI INTRA-RELAZIONALI**: impongono una restrizione all'insieme di tuple che rappresentano istanze della Relazione (di questa categoria fanno parte i *Vincoli di Chiave*)
- **VINCOLI INTER-RELAZIONALI**: impongono restrizioni agli insiemi di tuple che rappresentano istanze di due o più Relazioni
- **VINCOLI DI CHIAVE**:
  - *SUPERCHIAVE*:
    Data una Relazione di schema $R(x)$, un insieme di attributi k, con $k \subseteq x$ è SUPERCHIAVE per $R(x)$ se per ogni istanza r di $R(x)$ vale la seguente condizione:
    $$\forall\ t,t' \in r \quad t \neq t' \Rightarrow t[k] \neq t'[k]$$
  - *CHIAVE oppure CHIAVE CANDIDATA*:
    Data una Relazione di schema $R(x)$,un insieme di attributi k, con $k \subseteq x$ è CHIAVE CANDIDATA se è una SUPERCHIAVE per $R(x)$ e vale la seguente condizione:
    $$\nexists\ k' \subseteq k : k' \text{ è SUPERCHIAVE per } R(x)$$
  - *CHIAVE PRIMARIA*:
    Data una Relazione di schema $R(x)$,un insieme di attributi k, con $k \subseteq x$ è CHIAVE PRIMARIA se è la CHIAVE CANDIDATA scelta per identificare univocamente le tuple delle istanze di $R(x)$

- **VINCOLI DI INTEGRITA' REFERENZIALE**: Un VIR tra un insieme di Attributi $Y=\{A_1, \ldots, A_p\}$ di $R_1(x_1)$ $(Y \subseteq x)$ e un insieme di Attributi $K=\{B_1, \ldots, B_p\}$ Chiave Primaria di $R_2(x_2)$, è soddisfatto se per ogni istanza r_1 di $R_1(x_1)$ e r_2 di $R_2(x_2)$ *vale:*
  $$\forall\ t \in r_1 : \exists\ s \in r_2 : \ \forall\ i \in \{1, \ldots, p\} : t[A_i] = s[B_i]$$

TRANSAZIONI (1,2,3)

Principale caratteristica di un DBMS:
Un DBMS è un sistema TRANSAZIONALE, che definisce un meccanismo per la definizione ed esecuzione di TRANSAZIONI.

DEFINIZIONE:
Una **TRANSAZIONE** è un'unità di lavoro svolto da un programma applicativo per la quale si vogliono garantire alcune proprietà.
La principale caratteristica è che essa o va a buon fine causando effetti sulla Base di Dati oppure abortisce e non ha nessun effetto sulla BdD.

PROPRIETA' ACIDE (Proprietà delle transazioni):

1. **ATOMICITA'**: una transazione è una unità di esecuzione INDIVISIBILE. O viene eseguita completamente oppure non viene eseguita affatto.
   Se una transazione viene interrotta prima del commit, il lavoro fin li eseguito deve essere disfatto, ripristinando la situazione in cui si trovava la BdD prima dell'inizio della transazione.
   Se invece viene interrotta all'esecuzione del commit, significa che il commit viene eseguito con successo, e il sistema deve assicurare che la transazione abbia effetto sulla BdD.
2. **CONSISTENZA**: l'esecuzione di una transazione non deve violare i vincoli di integrità.
   Se si verifica una violazione di un vincolo, il sistema può:
   - abortire l'ultima operazione e restituire all'applicazione una segnalazione d'errore, in questo modo si ha una reazione dell'applicazione alla violazione.
   - al commit se un vincolo di integrità viene violato, la transazione viene abortita senza possibilità da parte dell'applicazione di reagire.
3. **ISOLAMENTO**: l'esecuzione di una transazione deve essere INDIPENDENTE dalla contemporanea esecuzione di altre transazioni.
   Il rollback di una transazione non deve creare rollback a catena di altre transazioni che si trovano in esecuzione contemporaneamente.
   Il sistema dovrà quindi regolare l'esecuzione concorrente.
4. **PERSISTENZA**: l'effetto di una transazione che ha eseguito il commit non deve andare perso.
   Il sistema deve essere in grado, in caso di guasto, di garantire gli effetti delle transazioni che al momento del guasto avevano già eseguito un commit.

Esecuzione concorrente di transazioni
L'esecuzione concorrente di transazioni senza controllo può generare anomali, è quindi necessario introdurre dei meccanismi di controllo per evitarle.

Anomalie tipiche:
- lettura sporca
- lettura inconsistente
- perdita di update
- aggiornamento fantasma

Notazione:
- indichiamo con t_i una transazione
- $r_i(x)$ è un operazione di lettura eseguita dalla transazione t_i sulla risorsa x
- $w_i(x)$ è un operazione di scrittura eseguita dalla transazione t_i sulla risorsa x

DEFINIZIONE
un **BLOCCO CRITICO** è una situazione di blocco nell'esecuzione di transazioni dovute alla gestione dei lock sulle risorse.

DEFINIZIONE

Uno **SCHEDULE** rappresenta una sequenza di operazioni di lettura e scrittura eseguite su risorse della BdD da diverse transazioni concorrenti.
Esso rappresenta una possibile esecuzione concorrente di diverse transazioni.

- **schedule SERIALE**: è uno schedule dove le operazioni di ogni transazione compaiono in sequenza senza essere inframmezzate da operazioni di altre transazioni.
  (es. r1(x)w1(x)r2(x)w2(x) è uno schedule SERIALE)
- **schedule SERIALIZZABILE**: è uno schedule "equivalente" ad uno schedule seriale.

DEFINIZIONE

Dato uno schedule S si dice che un'operazione di lettura $r_i(x)$, che compare in S, **LEGGE_DA** un operazione di scrittura $w_j(x)$, che compare in S, se $w_j(x)$ precede $r_i(x)$ in S e non vi è alcuna operazione $w_k(x)$ tra le due.
(es. S1: r1(x)w1(x)r2(x)w2(x) → LEGGE_DA(S1) = { (r2(x), w1(x)) })

DEFINIZIONE

Dato uno schedule S si dice che un'operazione di scrittura $w_i(x)$ che sta in S, è una **SCRITTURA FINALE** se è l'ultima operazione di scrittura della risorsa x in S.
(es. S1: r1(x)r2(x)w1(x)w2(x) → SCRITTURE_FINALI(S1) = {w2(x)})

DEFINIZIONE

Due schedule S1 e S2 sono **VIEW-EQUIVALENTI** $S1 \approx_V S2$ se possiedono le stesse relazioni LEGGE_DA e le stesse SCRITTURE_FINALI.
L'algoritmo di test di view-equivalenza tra due schedule è di complessità *lineare.*

DEFINIZIONE

Uno schedule S è **VIEW-SERIALIZZABILE** (VSR) se esiste uno schedule seriale S' tale che $S' \approx_V S$
L'algoritmo di test di view-serializzabilità tra due schedule è di complessità *esponenziale.*
es. schedule VSR:

S1: r1(x)r2(x)w1(x)w2(z)r1(z)w2(y)w1(y) → LEGGE_DA(S1) = {(r1(z),w2(z)) }
(S1 è schedule VSR) → SCRITTURE_FINALI(S1) = {w1(y), w2(z), w1(x) }

S2: r2(x)w2(z)w2(y)r1(x)w1(x)r1(z)w1(y) → LEGGE_DA(S1) = {(r1(z),w2(z)) }
(S2 è schedule seriale) → SCRITTURE_FINALI(S1) = {w1(y), w2(z), w1(x) }

es. schedule NON VSR (anomalia: perdita di update):

S : r1(x)r2(x)w2(x)w1(x) → LEGGE_DA(S) = Ø
→ SCRITTURE_FINALI(S) = {w1(x) }

S1: r1(x)w1(x)r2(x)w2(x) → LEGGE_DA(S1) = {(r2(x),w1(x)) }
→ SCRITTURE_FINALI(S1) = {w2(x) }

S2: r2(x)w2(x)r1(x)w1(x) → LEGGE_DA(S2) = {(r1(x),w2(x)) }
→ SCRITTURE_FINALI(S1) = {w1(x) } → **S non è VSR**

DEFINIZIONE

Conflitto: dato uno schedule S, una coppia di operazioni (a_i, a_j) che compaiono in S rappresenta un conflitto se valgono le seguenti condizioni:

1) $i \neq j$ ***2)*** *almeno una è un'operazione di scrittura* ***3)*** agiscono sulla stessa risorsa ***4)*** a_i precede a_j *in S*

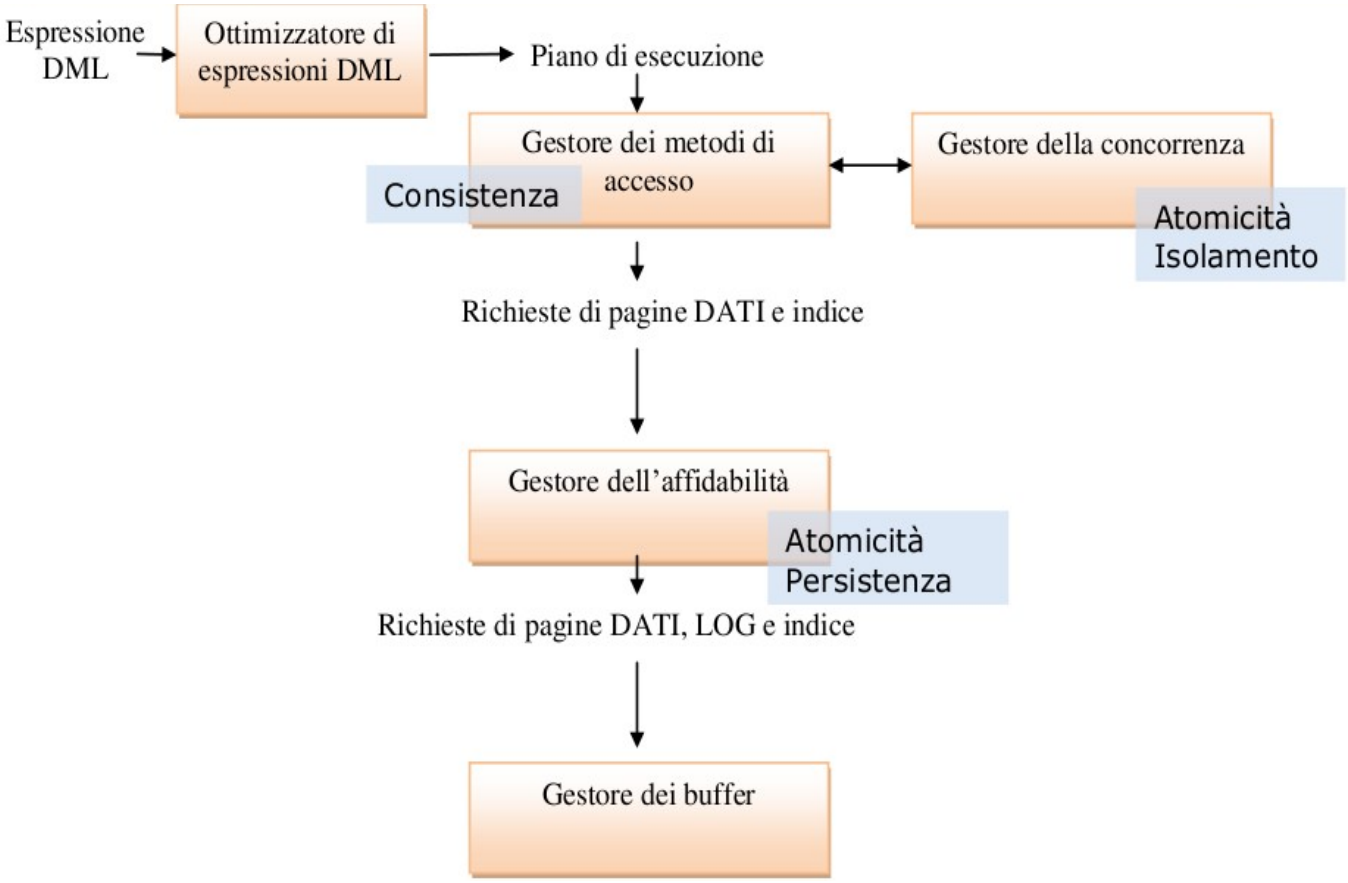
DEFINIZIONE

Due schedule sono **CONFLICT-EQUIVALENTI** $S1 \approx_C S2$ se hanno le stesse operazioni e gli stessi conflitti

DEFINIZIONE

Uno schedule S è **CONFLICT-SERIALIZZABILE** (CSR) se esiste uno schedule seriale S' tale che $S' \approx_C S$
se l'ordine degli elementi di un conflitto è invertito, i due ordini non sono equivalenti.

Architettura di un DBMS



Struttura del Dizionario di Pagina

- Tuple di lunghezza fissa: il dizionario non è necessario, si deve solo memorizzare la dimensione delle tuple e l'offset del punto iniziale.
- Tuple di lunghezza variabile: il dizionario memorizza l'offset di ogni tupla presente nella pagina e di ogni attributo di tutte le tuple.

La lunghezza massima di una tupla = dimensione massima dell'area disponibile su una pagine, altrimenti va gestito il caso di tuple memorizzate su più pagine.

PAGINA DATI

Inserimento di una tupla:

- se esiste spazio contiguo sufficiente: inserimento semplice
- se non esiste spazio contiguo ma esiste spazio sufficiente: riorganizzare lo spazio ed eseguire un inserimento semplice
- se non esiste spazio sufficiente: operazione rifiutata

La cancellazione è sempre possibile, anche senza riorganizzare

FILE SEQUENZIALE

Inserimento di una tupla:

- individuare la pagina P che contiene la tupla che precede, nell'ordine della chiave, la tupla da inserire
- inserire la tupla nuova in P; se l'operazione non va a buon fine, aggiungere una nuova pagina alla struttura: la pagina contiene una nuova tupla, altrimenti si prosegue
- aggiustare la catena dei puntatori

Cancellazione di una tupla:

- individuare la pagina P che contiene la tupla da cancellare
- cancellare la tupla da P

DEFINIZIONE

Gli **INDICI** sono strutture ausiliare, utilizzate per aumentare le prestazioni degli accessi alle tuple memorizzate nelle strutture fisiche. Tali strutture velocizzano l'accesso casuale via chiave di ricerca. La chiave di ricerca è un insieme di attributi utilizzati dall'indice nella ricerca.

Indici su file sequenziali

- **INDICE PRIMARIO:** in questo caso la chiave di ordinamento del file sequenziale coincide con la chiave di ricerca dell'indice.
- **INDICE SECONDARIO:** in questo caso invece la chiave di ordinamento e la chiave di ricerca sono diverse.

INDICE PRIMARIO

Usa una chiave di ricerca che coincide con la chiave di ordinamento del file sequenziale.

Ogni record dell'indice primario contiene una coppia $<v_i, p_i>$:

- $\mathbf{v_i}$ è il valore della chiave di ricerca;
- $\mathbf{p_i}$ è il puntatore al primo record nel file sequenziale con chiave $\mathbf{v_i}$.

1) INDICE DENSO: per ogni occorrenza della chiave nel file, esiste un corrispondente record nell'indice
2) INDICE SPARSO: solo per alcune occorrenze delle chiavi presenti nel file esiste un corrispondente record nell'indice, tipicamente una pagina.

INDICE SECONDARIO

Usa una chiave di ricerca che NON coincide con la chiave di ordinamento del file sequenziale.

Ogni record dell'indice secondario contiene una coppia $<v_i, p_i>$:

- $\mathbf{v_i}$ è il valore della chiave di ricerca;
- $\mathbf{p_i}$ è il puntatore al bucket di puntatori che individuano nel file sequenziale tutte le tuple con valore di chiave $\mathbf{v_i}$.

Gli indici secondari sono sempre DENSI.

B+-TREE e HASHING

È una struttura ad albero, dove ogni **nodo** corrisponde ad una **pagina della memoria secondaria**; i legami tra i nodi diventano puntatori a pagina, ed ogni nodo ha un numero elevato di figli, così che tipicamente ci sono pochi livelli e molti nodi foglia.
L'albero è **bilanciato**, cioè la lunghezza dei percorsi che collegano la radice ai nodi foglia è costante.
Inserimenti e cancellazioni non alterano le prestazioni dell'accesso ai dati, infatti l'albero si mantiene bilanciato.

Struttura di un B+-tree (fan-out = n) :

NODO FOGLIA (vincolo di riempimento)
Ogni nodo foglia contiene un numero di valori chiave ***#chiavi*** vincolato come segue:

$$\lceil \frac{(n-1)}{2} \rceil \leqslant chiavi \leqslant (n-1)$$

NODO INTERMEDIO (vincolo di riempimento)
Ogni nodo intermedio contiene un numero di puntatori **#puntatori** vincolato come segue

$$\lceil \frac{n}{2} \rceil \leqslant puntatori \leqslant n$$

Il costo di una ricerca nell'indice, in termini di numeri di accessi alla memoria secondaria, risulta pari al numero di nodi acceduti nella ricerca.
Tale numero in una struttura ad albero è pari alla profondità dell'albero, che nel B^+-tree è indipendente dal percorso ed è funzione del fan-out e del numero di valori chiave presenti nell'albero ***#valoriChiave***

STRUTTURE AD ACCESSO CALCOLATO (Hashing)

Si basano su una funzione di hash che mappa le chiavi sugli indirizzi di memorizzazione delle tuple nelle pagine dati della memoria secondaria;

$h: K \rightarrow B$, dove K è il dominio delle chiavi, e B è il dominio degli indirizzi

Una caratteristica di una buona funzione di Hashing è di distribuire in modo UNIFORME e CASUALE i valori chiave nei bucket.

Operiazione di RICERCA(dato un valore di chiave K trovare la corrispondente tupla):

- calcolare $b = h(f(K))$ {costo: zero}
- Accedere al bucket b {costo: 1 accesso a pagina}
- Accedere alle tuple attraverso i puntatori del bucket {costo: m accessi a pagina con $m<n$}

Operazione di INSERIMENTO e CANCELLAZIONE: di complessità simile alla Ricerca

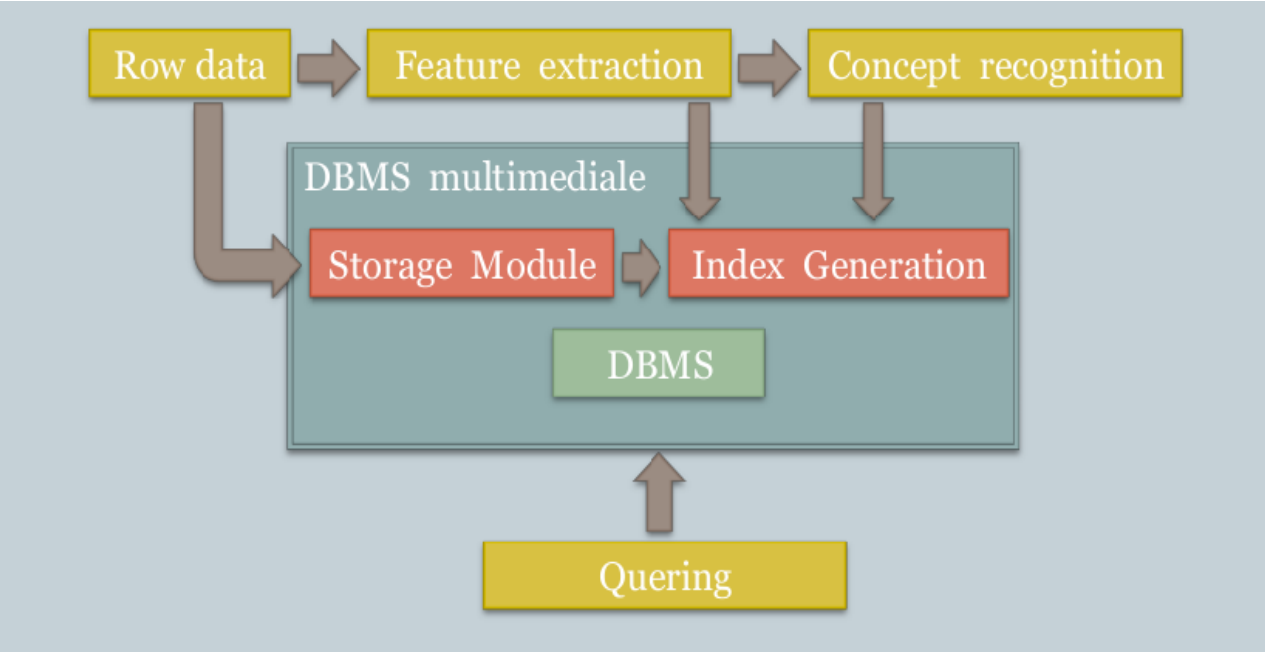
La struttura ad accesso calcolato funziona se i bucket conservano un basso coefficiente di riempimento. Infatti il problema delle strutture ad accesso calcolato è la gestione delle COLLISIONI.

Una **COLLISIONE** si verifica quando, dati due valori di chiave K_1 e K_2 con $K_1 \neq K_2$, risulta : $h(f(K_1))=h(f(K_2))$
Un numero eccessivo di collisioni porta alla saturazione del bucket corrispondente.

DBMS MULTIMEDIALI

I **DBMS multimediali** sono DBMS che consentono di memorizzare e recuperare dati di natura mutlimediale: testo, immagini, suoni, animazioni, voce, video, ecc.
Hanno problematiche differenti rispetto ai DBMS tradizionali:

- dimensione dati molto maggiore
- gestione dei media continui
- complessità della modellazione
- interrogazioni basate su predicati non esatti, approssimate



CODIFICA DI HUFFMAN e LEMPEL-ZIV-WELCH (LZW)

Tipi di compressione:

- **Senza perdita (lossless)**: permettono di ricostruire perfettamente la rappresentazione del dato originale, conservando tutta l'informazione;
- **Con perdita (lossy)**: permettono di ricostruire solo in parte la rappresentazione del dato originale, quindi parte dell'informazione va persa.

CODIFICA DI HUFFMAN

CODIFICA:

Si applica il seguente algoritmo:

1. si ordinano i simboli in ordine decrescente di probabilità;
2. si unificano i due simboli in fondo alla lista, si sommano le probabilità e si reintroduce tale probabilità nella lista, mantenendo l'ordine;
3. si ripete il passo precedente fino a quando la lista sarà formata da un solo elemento;
4. si etichetta ogni biforcazione con uno 0 e un 1;
5. la codifica di ogni simbolo dell'alfabeto è data dal percorso dalla radice dell'albero fino alla foglia contenente il simbolo;
6. per le proprietà degli alberi, ogni foglia ha un unico percorso che non sarà mai il prefisso di un altro percorso

DECODIFICA:

La decodifica è univoca e mai ambigua.
Il primo bit di ogni simbolo coincide con la radice dell'albero;
Ogni bit successivo guida il decodificatore fino ad una foglia.

- si codificano sequenze di simboli di lunghezza variabile con codici di lunghezza fissa;
- non è necessario trasmettere la tabella di codifica, in quanto viene ricostruita durante la decodifica;

Algoritmo di codifica:

- si analizza il dizionario dei simboli con tutti i simboli dell'alfabeto sorgente con la corrispondente codifica;
- si inizializza a "" la variabile PREFISSO
- si legge il primo carattere dello STREAM di INPUT nella variabile C.

1. se la sequenza PREFISSO+C è presente nel dizionario
   - allora PREFISSO = PREFISSO + C;
   - altrimenti:
     - output = output + code (Prefisso)
     - inserire la sequenza PREFISSO+C nel dizionario e codificarla
     - PREFISSO = C
2. Se esistono ancora caratteri in INPUT leggere un carattere e tornare al punto precedente altrimenti
   - output = output + code (Prefisso)

Algoritmo di decodifica:

- si inizializza il dizionario dei simboli con tutti i simboli dell'alfabeto sorgente ordinati, e si codificano in stringhe di bit
- si inizializza a "" la variabile PREFISSO
- **si legge la prima posizione della codifica dello STREAM di INPUT nella variabile CW**
- si supponga che esista una funzione DECODE(c) che dato un codice *c* restituisce la corrispondente stringa di simboli del dizionario.

1. Si produce in output la stringa di simboli DECODE(CW)
2. PW = CW; sposto CW sulla posizione successiva
3. se CW è presente nel dizionario allora:
   - si produce io output la stringa di simboli DECODE(CW)
   - Prefisso = DECODE(PW)
   - Char = DECODE(CW)[1]
   - Si aggiunge la stringa Prefisso+Char nel dizionario
4. Altrimenti
   - Prefisso = DECODE(PW)
   - Char = DECODE(PW)[1]
   - Si produce in output Prefisso+Char e si aggiunge al dizionario
5. Se l'input non è terminato tornare al punto 2.

JPEG (**J**oint **P**hotographic **E**xperts **G**roup)

È uno standard creato nel 1990 con l'intenzione di migliorare e sostituire i formati di immagine già esistenti.
La compressione JPEG è di tipo "ibrido", combina infatti DCT e quantizzazione.
Supporta colori a 8 e a 24 bit: un'immagine JPEG può avere da 256 a oltre 16,7 milioni di colori.
Un file JPEG occupa meno spazio di un file GIF della stessa immagine, infatti adotta un rapporto di compressione più elevato. L'algoritmo di compressione JPEG è più complesso rispetto a GIF, e richiede un tempo di decompressione dell'immagine maggiore.

JPEG standard fornice 4 modi di operare:

1. codifica **sequenziale** basata su DCT: ogni componenete dell'immagine è codificata in un singolo scan (left-to-roght, top-to-bottom).
2. codifica **progressiva** basata su DCT: l'immagine è codificata in scan successivi al fine di produrre una decodifica della stessa più veloce in caso di tempi di trasmissione elevati.
3. codifica **lossless**: l'immagine è codificata in modo da garantirne un'esatta riproduzione.
4. codifica **gerarchica**: l'immagine è codificata in multipla risoluzione.

JPEG - ENCODER

Fase 1 – Partizionamento dell'immagine:

- i pixel dell'immagine originale sono raggruppati in blocchi 8x8 elementi
- gli elementi di ciascun blocco shiftati da interi senza segno, nel range $[0, 2^p -1]$, a interi con segno, nell'intervallo $[-2^{p-1}, 2^{p-1} - 1]$ (con *p*, precisione dell'immagine)

Fase 2 – Trasformazione FDCT:

- ogni blocco 8x8 può essere visto come un segnale discreto a 64 punti. Tale segnale è dato in input alla trasformazione FDCT, la cui definizione matematica è la seguente:

$$F(u,v)=\frac{1}{4}\cdot C(u)C(v)\sum_{x=0}^{7}\sum_{y=0}^{7}f(x,y)\cdot\cos\frac{(2x+1)\cdot u\cdot\pi}{16}\cdot\cos\frac{(2y+1)\cdot v\cdot\pi}{16}$$

dove *C(u)* e *C(v)* sono così definiti:

$$C(u),C(v)=\begin{cases}\frac{1}{2} & se\ \ u,v=0\\ 1 & altrimenti\end{cases}$$

L'output della FDCT è un insieme di 64 coefficienti che rappresentano le ampiezze dei segnali base in cui il segnale originale è stato scomposto.
FDCT comprime l'immagine concentrando la maggior parte dell'energia contenuta nel segnale nelle componenti di bassa frequenza (si tratta di una compressione *lossless*).

Fase 3 – Quantizzazione:

- I 64 coefficienti DCT sono quantizzati uniformemente mediante una tabella di quantizzazione a 64 elementi rappresentati da interi nell'intervallo [1-255]
- la quantizzazione riduce l'ampiezza dei coefficienti DCT il cui contributo è nullo o comunque basso per la qualità dell'immagine. La quantizzazione scarta dunque informazioni che non sono rilevanti al fine della visualizzazione. Si tratta di compressione *lossy.*
- La quantizzazione è un **mapping molti-a-uno** e si calcola mediante l'equazione:

$$F_q(u,v)=Round\left(\frac{F(u,v)}{Q(u,v)}\right)$$

dove Q(u,v) rappresenta il coefficiente di quantizzazione associato al coefficiente DCT.

- Tutti i coefficienti quantizzati sono ordinati in una sequenza a "zig-zag", in modo tale da facilitare la successiva fase di *Entropy Encoding* ponendo i coefficienti di bassa frequenza prima di quelli ad alta.

Fase 4 – Entropy Encoder:

- L'*Entropy Encoder* aggiunge ulteriore compressione ai coefficienti Dct quantizzati, codificandoli in modo più compatto.
- Il processo si abbassa sulla codifica di Huffman:
  - la sequenza "zig-zag" di coefficienti DCT quantizzati è convertita in una seq. intermedia di simboli
  - la seq. Intermedia è codificata in una sequenza binaria mediante l'uso di tabelle (di Huffman)

Il **JPEG Decoder** esegue i processi inversi rispetto quelli applicati nel JPEG Encoder, invertendone l'odine:

- *Entropy Decoder*: prende la sequenza bianria e la trasforma, prima in sequenza intermedia di simboli, poi nella sequenza a "zig-zag" rappresentante i 64 coefficienti DCT quantizzati.
- *Dequantizer*: prende i 64 coefficienti quantizzati e li trasforma in 64 coefficienti DCT semplici, seguendo l'espressione:

$$F_q^{INV}(u,v) = F_q(u,v) \cdot Q(u,v)$$

- *Inverse DCT (IDCT)*: prende in input i 64 coefficienti DCT e ricostruisce in output il segnale discreto di 64 punti rappresentante un blocco 8x8 dell'immagine originale.

$$f(x,y) = \frac{1}{4} \cdot \sum_{u=0}^{7} \sum_{v=0}^{7} C(u)C(v) \cdot F(u,v) \cdot \cos\frac{(2x+1) \cdot u \cdot \pi}{16} \cdot \cos\frac{(2y+1) \cdot v \cdot \pi}{16}$$

$$C(u), C(v) = \begin{cases} \frac{1}{2} & se\ u,v = 0 \\ 1 & altrimenti \end{cases}$$

K-d TREE

Il **K-d Tree** è un albero binario di ricerca multi-dimensionale in cui ogni nodo indicizza un punto in *k* dimensioni.

DEFINIZIONE

Sia T la radice dell'albero ed N un generico nodo, il **LIVELLO di N** è una funzione definita ricorsivamente:

- level(N) = 0 se N è la radice T dell'albero
- level(N) = level(P_N)+1 se N non è la radice e P_N è il padre di N

Caratteristiche strutturali

- questi alberi generalmente NON sono bilanciati, e nel caso peggiore, possono degenerare in una lista;
- per evitare questo problema, i punti dovrebbero essere inseriti casualmente in modo che il valore atteso della profondità dell'albero sia logaritmico;
- oppure si possono applicare le tecniche di ribilanciamento per mantenere il tempo di ricerca logaritmico.

OPERAZIONI SU K-d Tree

- **Point Query** ---> ricerca di un punto P = (x_P, y_P)
  il tempo di una **PQ** è logaritmico nel numero totale di nodi, nel caso di un albero bilanciato, tuttavia, come già accennato, potrebbe anche diventare lineare nel caso in cui K-d Tree sia degenerato in una lista
- **Range Query** ---> ricerca di tutti i punti P_i che appartengono ad una regione **R** passata in ingresso, che ssumiamo essere rettangolare. Supponiamo inoltre che sia nota la regione di spazio associata ad un nodo
- **Inserimento** ---> l'inserimento del nodo relativo ad un punto P consiste in:
  La creazione di un K-d Tree consiste nella creazione del nodo radice con relativo aggiornamento dei campi statici e puntatori a NULL, seguita dall'inserimento dei nodi successivi. Con una procedura ottimizzata si costruisce un K-d Tree in $O(n \log n)$, dove n è il numero di punti.
- **Cancellazione** ---> cancellazione del nodo relativo ad un punto P
  - 1^sol: cancellare il nodo e reinserire tutti quelli dei suoi sottoalberi (troppo onerosa).
  - 2^sol: il nodo da rimuovere viene segnato come "eliminato" in modo che non sia preso in considerazione nelle successive ricerche, e quando l'alberò dovrà essere ricostruito, tutti i nodi marcati saranno rimossi.
  - 3^sol: cercare un nodo candidato *R* nel sottoalbero destro (o sinistro) di N, rimpiazzare i campi statici di N con quelli di R, e infine cancellare R.

XML

Linguaggio di marcatura proposto dal W3C, definisce una sintassi generica per contrassegnare i dati di un documento elettronico con marcatori (tag) semplici e leggibili.

Dato che XML è più restrittivo di HTML, per quanto riguarda il posizionamento dei tag e il modo in cui vengono scritti, ogni documenti XML deve essere **ben formato**, vale a dire:

- deve avere una sola radice
- l'annidamento dei tag deve essere corretto
- tutti i tag aperti devono essere chiusi
- i valori degli attributi devono essere specificati tra virgolette

SINTASSI

Ogni elemento è caratterizzato da:

- una marca/tag iniziale (<*nome_elemento*>)
- una marca/tag finale (</*nome_elemento*>)

Un contenuto può essere un valore atomico o un valore strutturato attraverso altri elementi XML
XML è sensibile alla differenza tra maiuscole e minuscole

Non ci sono regole prefissate su quando usare attributi e quando usare elementi per rappresentare informazioni in XML.
Tuttavia se XML viene usato come sintassi per la specifica di informazione **semi-strutturata** è evidente che gli elementi forniscono una rappresentazione più accurata del dato semi-strutturato.

es. Contenuto con attributi

```
<person sex="female">
        <firstname>Anna</firstname>
        <lastname>Smith</lastname>
</person>
```

es. Contenuto con elementi

```
<person>
        <sex>female</sex>
        <firstname>Anna</firstname>
        <lastname>Smith</lastname>
</person>
```

SVANTAGGI dell'uso di attributi:

- non possono contenere più valori (gli elementi si)
- non possono contenere informazione ultriormente strutturate ad albero (gli elementi si)
- non possono essere facilmente estese (gli elementi si)

Gli attributi sono difficili da leggere e gestire; meglio quindi utilizzare gli elementi per le informazioni, e gli attributi per la meta-informazione *(identificatori).*

VALIDAZIONE dei documenti XML

Per ogni insieme di documenti XML che debba rappresentare una certa categoria di informazione è possibile definire un documento XML che ne descrive la sintassi specifica: vale a dire, quali sono i tag ammessi, qual'è la struttura dei tag e dei loro attributi, ecc...

Ogni documento XML è **valido** se è ben formato e rispetta la sintassi specificata nel suo file XSD.
(la SINTASSI di un documento XML (file .XML) si descrive attraverso uno SCHEMA di documento XMLSchema (file .XSD))

Caratteristiche della classe **PreparedStatement** della libreria JDBC:
usando la classe PreparedStatement è possibile ottimizzare l'esecuzione di una interrogazione che deve essere fatta più volte. Tale classe consente di inserire parametri nell'interrogazione (attraverso il simbolo "?") e di valorizzarli, usando specifici metodi (*setInt(pos,valoreInt) , setString(pos,valoreString) , …*) prima dell'effettiva esecuzione dell'interrogazione stessa. In questo modo si lascia ai metodi della classe PreparedStatement il compito di convertire i valori dai tipi JAVA ai tipi SQL.

CGI (Common Gateway Interface) è la prima tecnica applicabile per realizzare pagine web dinamiche.

LIMITI DELL'APPROCCIO CGI
• Richiede la creazione di un nuovo processo per ogni richiesta: tale creazione richiede tempo non trascurabile e necessita inoltre di uno specifico spazio di indirizzamento in memoria virtuale.
• Il processo CGI attivato richiede necessariamente una nuova connessione al DBMS e la sua corrispondente chiusura a fine programma.

Tecniche alternative a CGI :
la tecnologia Servlet di Sun (ora Oracle), Java Server Pages (jsp), Hypertext Preprocessor (php).

SERVLET (Sun): si basa sul linguaggio JAVA.
Vantaggi rispetto a CGI:
- Ogni richiesta genera un thread e non un processo.
- È possibile mantenere connessioni aperte con il DBMS tra una richiesta e l'altra.
- Portabilità JAVA.

HTML (HYPERTEXT MARKUP LANGUAGE) è il linguaggio usato per specificare IPERTESTI (documento con struttura non sequenziale, costituito da varie parti fra loro collegate, tali legami sono detti LINK e consentono di navigare nell'ipertesto).
Ogni pagina web diventa un file HTML; l'HTML consente di specificare:
- La formattazione del testo per la presentazione
- La struttura della pagina (sezioni, tabelle, liste, ecc.)
- I legami con altre pagine (LINK) o con sottoparti della pagina
- Il contenuto informativo della pagina

MVC e Servlet-centric

In tale approccio la progettazione di una applicazione web viene divisa in tre livelli:
- "Presentation layer" (VIEW): specifica la modalità e la forma di presentazione dei dati all'utente.
- "Application Logic layer" (MODEL): specifica le procedure di estrazione da DB ed elaborazione dei dati
- "Control layer" (CONTROLLER): specifica il flusso dell'applicazione e controlla l'interazione tra gli altri livelli.

Adottando l'approccio MVC e la tecnologia Servlet-JSP, un'applicazione web può essere realizzata secondo l'approccio visto a lezione ovvero quello servlet-centric.
L'approccio servlet-centric prevede di utilizzare le pagine JSP solo per la presentazione e delegare il controllo ad una o ad un gruppo di servlet.
Le servlet quindi:
- gestiscono le richieste (vengono cioè invocate tramite URL)
- elaborano i dati necessari a soddisfare le richieste (interagendo con la classe DBMS.java e utilizzando i Java Data Bean come componenti per rappresentare le informazioni di interesse)
- trasferiscono il controllo alla JSP designata a presentare i risultati.

Il primo passo verso l'architettura MVC-2 è quello di separare l'interazione con il DBMS dall'elaborazione e presentazione dei dati.

Una pagina JSP può essere vista come uno "schema di pagina Web" dove:
le parti statiche sono scritte in HTML e le parti dinamiche sono generate attraverso porzioni di codice Java.
Il "linguaggio" JSP fornisce 4 gruppi principali di
marcatori speciali:
- Direttive (directives)
- Scripting:
    Dichiarazione (declarations)
    Espressione (expressions)
    Scriptlet
- Azioni (Actions)
- Commenti

Le pagine JSP vengono "gestite" da un componente operante sul web server chiamato JSP container.
Questo componente traduce le JSP in servlet Java, che poi esegue.

La metodologia della progettazione è strutturata in 4 fasi:
1. Organizzazione in pagine web del contenuto informativo
2. Definizione della navigazione tra le pagine web (specifica dei LINK)
3. Corrispondenza tra il contenuto delle pagine e la basi di dati (interrogazioni SQL)
4. Presentazione (aspetto grafico delle pagine: fogli di stile CSS)

1. La struttura delle pagine web e il loro contenuto informativo possono essere specificati definendo SCHEMI di PAGINA.
Linguaggio per la specifica di schemi di pagina:
page schema <nomeSchema> [unique](
    <nomeAttributo>: <TIPO>;)
<TIPO> :== {string, integer, date, time, real, <LISTA>}
<LISTA> :== list_of (<nomeAttr>: <TIPO>;)

3. La terza prevede la specifica della corrispondenza tra la base di dati e il contenuto informativo degli schemi di pagina. Nella sintassi proposta tale corrispondenza si precisa per ogni schema di pagina <nomeSchema> come segue:
DB to page schema <nomeSchema>
parameter(<nomepar>, ...,<nomepar>)
(<nomeAttr>, ...,<nomeAttr>: <QUERY_SQL>;)
dove i parametri dovranno comparire nelle interrogazioni SQL con la sintassi ?nomepar? e gli attributi <nomeAttr> devono essere tutti e soli gli attributi dello schema di pagina <nomeSchema>.

Java Bean

E' un componente nella tecnologia Java. Con il termine componente si indicano essenzialmente quelle classi che possono essere utilizzate in modo standard in più applicazioni. Lo scopo dei componenti è dare la possibilità di riutilizzare in modo sistematico gli oggetti in contesti diversi, aumentando la produttività. La definizione di Java Bean è volutamente generica. Un Java Bean è semplicemente una classe Java che risponde a due requisiti:
- ha un costruttore con zero argomenti;
- soddisfa una serie di direttive relative ai suoi metodi e alle sue variabili.

Parliamo di Java Data Beans quando i Java Beans vengono usati per la gestione del risultato di interrogazioni su una base di dati.
I Java Data Bean sono quindi dei componenti fondamentali nella strutturazione di una applicazione web centrata sui dati secondo l'architettura MVC-2. I Java Data Bean, insieme ad una o più classi Java, permettono di separare la logica dell'applicazione dalla parte di controllo e di presentazione delle informazioni.