**Sistema organizzativo** è un Insieme di risorse.

Sistema informativo → gestisce tutti quei processi che coinvolgono informazioni e risorse

Il sistema informatico è la parte del sistema informativo, tratta informazioni con tecnologie informatiche

Un **Dato** è una *rappresentazione oggettiva,* ciò che è immediatamente presente alla conoscenza; una volta che gli diamo un significato, esso diventa **un'Informazione**.

5 è un dato, se parliamo di temperatura è un’informazione.

Un database è una collezione di dati logicamente correlati da un qualche significato.

**Basi di dati** → dati correlati tra loro

Un **DBMS** è un insieme di programmi che permettono di gestire basi di dati. È un software "general-purpose" che ne facilita la creazione, costruzione e gestione di database.

Caratteristiche di una base di dati:

* Consistenza: i dati devono essere significativi
* Sicurezza: si deve impedire che il BD venga danneggiato
* Integrità: garantire che le operazioni sul DB non provochino inconsistenza dei dati
* Persistenza: i dati non devono andar persi

Un DBMS permette di:

**Definire** → specificando i tipi di dati, la struttura

**Costruire** → memorizzando i dati

**Manipolare** → interrogare il database per trovare dati specifici, aggiornare il database per riflettere cambiamenti nel mondo reale.

**Metadato** dato di più alto livello che fornisce un significato strutturale al nostro database

Tipe Studente=

nome: stringe;

matricola: string;

anno: string;

end;

Multiutente

Un DBMS multiutente deve consentire accesso al database a più utenti contemporaneamente. Utilizza un Software per definire Chi è autorizzato ad accedere o modificare quali dati. Esso deve gestire anche la concorrenza per garantire l'aggiornamento corretto.

Transazione (utente) → esecuzione e memorizzazione di un’attività/funzione

Transazione (sistema) → sequenza indivisibile di operazioni

DBMS vs File Processing

Nel DBMS definiamo una struttura, che può essere utilizzata dai vari programmi, mentre tradizionalmente (FP) i dati vengono conservati in un file. Ogni programma contiene una descrizione della struttura (nei file), col rischio di incoerenza si può scambiare l’anno di iscrizione con l’anno dell’esame.

DBMS: Pro e Contro

PRO → Gestione centralizzata -- Riduzione di ridondanze, errori ed inconsistenze --    Manutenzione semplificata -- Flessibilità

CONTRO → minore efficienza

File Processing: *Pro e Contro*

PRO → Possibilità di scrivere programmi efficienti.

CONTRO → comporta elevato rischio di inconsistenze nei dati e ridondanza dei dati.

***Modello*** fornisce regole di rappresentazione

***Modello***→ Offre un insieme di costrutti per rappresentare la realtà in modo astratto.

*Costruttori* *di tipo* → permettono di definire nuovi tipi, partendo da altri più elementari; ad esempio, il modello relazionale prevede il costruttore **relazione**, che permette di definire insiemi di record omogenei

***Modello logico*** → usato dal DBMS per l’organizzazione dei dati.

***Modelli concettuali*** → permettono di rappresentare i dati in modo indipendente da ogni sistema. Ho un’astrazione superiore, descrivo efficientemente la realtà da strutturare. {il più noto è il modello Entity-Relationship}

Modelli concettuali, perché?

servono per ragionare sulla realtà di interesse e rappresentarla, attraverso efficaci rappresentazioni grafiche. indipendentemente dagli aspetti realizzativi.

 (Utile per avere documentazioni) (uso di editor grafico)

Un modello offre regole di rappresentazione, se tramite quel modello produciamo un progetto espresso secondo le regole di quel modello; abbiamo uno schema fatto secondo quel modello.

Usando le regole di rappresentazione del modello otteniamo uno

*Schema→* descrive la struttura della base di dati. Aspetto intensionale; nel modello relazionale, le intestazioni delle tabelle. (Nome, cognome, voto)

Schema logico: descrizione della base di dati si ha una struttura relazionale. Definendone i campi. Docenza (corso, Nome Docente)

Schema fisico: si occupa della memorizzazione dei dati -- rappresentazione fisica dello schema logico. (una tabella diventa un file)

Schema esterno: livello più vicino all'utente

Indipendenza fisica → Il livello logico è indipendente da quello fisico; una tabella è utilizzata nello stesso modo qualunque sia la sua realizzazione fisica

Vista → piccolo specchietto sulla base di dati

INDIPENDENZA FISICA DEI DATI

Lo schema fisico (interno) può essere cambiato senza dover cambiare lo schema logico (e quindi anche quello esterno). Un cambiamento dello schema fisico può essere per migliorare l’esecuzione di aggiornamento o ritrovamento.

MAPPING

Processo di trasformazione delle richieste e dei risultati.

Permette la comunicazione tra i vari livelli, infatti, da una richiesta dallo schema esterno si arriva allo schema fisico (passando per quello logico), dove risiedono i dati; viene elaborata la richiesta e la inoltra a ritroso.

ARCHITETTURA A 3 Livelli

***VANTAGGI***→ L’architettura a 3 livelli consente facilmente di ottenere una reale indipendenza dei dati Il database risulta più flessibile e scalabile

***SVANTAGGI*** → C'è il problema di dover trasformare le richieste di dati tra i vari livelli. Il Mapping crea overhead causando inefficienze nel DBMS.

Linguaggi di DBMS

Un DBMS deve fornire ad ogni categoria di utenti interfacce e linguaggi adeguati a comunicare gli schemi di database progettati e per manipolare i dati.

***DDL***

Linguaggio di definizione (DDL). Con esso informo il DBMS com’è il database, costruendolo, vuoto, e i vincoli di integrità (restrizioni per definire la validità dei dati)

***DML***

Una volta che il DB viene popolato (riempito) con i dati.  Il DBMS fornisce un linguaggio di manipolazione dati (DML) per consentire agli utenti di manipolare i dati.

Esistono due tipi di DML → Alto / Basso livello

* Alto: Consentono di specificare in maniera concisa operazioni complesse sui database. Sono detti ***dichiarativi***perché l’utente si interessa su quale dato deve essere ritrovato piuttosto che come farlo. Sono detti **set-at-a-time** perché possono specificare e ritrovare molti record in singolo Statement DML (es. l’SQL). Quando è inglobato in un linguaggio general-purpose esso è detto data *sublanguage*; se invece è usato da solo è detto *query-Language;* quest’ultimo è usato soprattutto da utenti meno esperti --Tutto ciò a scapito della flessibilità.
* Basso: devono essere inglobati in un linguaggio di programmazione general-purpose.  Sono detti ***record-at-a-time*** perché tipicamente ritrovano record individuali ;  hanno quindi bisogno di usare costrutti di programmazione  come l’iterazione per trattare il singolo record da un insieme di record.

L’utente pone l’interrogazione allo schema logico, lo schema fisico fornisce la risposta, la quale, passando per lo schema logico, arriva all’utente.

ISTANZA = è lo stato del Database in un preciso istante. Aspetto estensionale

**Modello E-R**

Il modello Entity-Relationship è utile  alla rappresentazione concettuale dei dati ad un alto livello di astrazione. Viene spesso utilizzato nella prima fase della progettazione di una base di dati in cui è necessario tradurre le informazioni risultanti dall'analisi di un determinato dominio in uno schema concettuale. Usando entità, attributi, associazioni.

Insieme dei Costrutti di Rappresentazione che ci mette a disposizione il modello ER. Rappresentiamo le entità ma non le singole istanze(occorrenze), generalizzando.

**Entità** → Classe di oggetti con proprietà comuni (omogenei), ed ha un’esistenza autonoma (Studente). Occorrenza di entità: elemento della classe (Lo studente Rossi). L’entità rappresenta l’aspetto intensionale dato che con essa non andiamo a far riferimento all’istanza, ma in generale.

**Associazioni** → Legami logici tra entità. Una occorrenza di un'associazione binaria è una coppia di occorrenze di entità, una per ciascuna entità coinvolta.

**Attributo →** Proprietà elementare di un’entità o di un'associazione. Associa ad ogni occorrenza di entità o associazione, un valore appartenente al dominio dell’attributo. Cognome è un attributo che associa ad ogni occorrenza di ‘Persona’ un valore preso da un dominio.

Attributo dell’associazione: Caratteristica che lega due entità.

Attributi composti: attributi che raggruppano un insieme di attributi di una entità/associazione (indirizzi)

Cardinalità di associazione →  Coppia di valori associati a ogni entità che partecipa a una relazione. Specificano il numero minimo e massimo di volte che un’occorrenza di un’entità può partecipare all'associazione. Per semplicità usiamo solo tre simboli:  0 e 1 per la cardinalità minima: 0 = partecipazione **opzionale** , 1 partecipazione **obbligatoria**; 1 e N per cardinalità massima.

Cardinalità di attributi →  per indicare : informazioni opzionali o attributi multi-valore.

Identificatore di una entità → a livello E-R

* Abbiamo un identificatore **interno** se gli attributi dell’entità permettono di distinguere univocamente le varie occorrenze dell’entità.
* Se gli attributi non sono sufficienti a formare un identificazione, vi aggiungo attributi di entità esterne che sono però collegate da un’associazione all’entità corrente; abbiamo un identificatore **esterno**.

Una identificazione esterna è possibile solo se l’attività da identificare partecipa alla relazione con una cardinalità (1,1).

Generalizzazione→ Mette insieme più entità (E1,E2..) con una sola(E) che le comprende tutte come casi particolari. E è generalizzatore , E1,E2 sono sottotipi particolari di E. Se E (genitore) è generalizzazione di E1, E2, ..., En (figlie):  ogni proprietà di E è significativa per E1, E2, ..., En ; ogni occorrenza di E1, E2, ..., En è occorrenza anche di E. tutte le proprietà (attributi, relazioni…) dell’entità genitore sono ereditate dalle entità figlie.

Se una generalizzazione ha solo un’entità figlia si parla di **sottoinsieme**

**Tipi di generalizzazioni**

*Totale* se ogni occorrenza dell'entità genitore è occorrenza di almeno una delle entità figlie, altrimenti è parziale

*Esclusiva* se ogni occorrenza dell'entità genitore è occorrenza di al più una delle entità figlie, altrimenti è sovrapposta

Quando ci approcciamo alla Progettazione Logica necessitiamo di: 1) uno schema concettuale (con dizionario dei dati e dei vincoli non esprimibili), 2)aver scelto un modello logico ( gerarchico ,relazionale), 3) informazioni sul carico applicativo ma non dobbiamo scegliere il DBMS specifico.

Carico applicativo: informazioni su come il database verrà usato, quali sono le principali operazioni da effettuare e con quale frequenza; ciò ci serve perché già in questa fase possiamo fare scelte progettuali (in termini di performance) tenendo conto di queste informazioni.

L’obiettivo è quello di tradurre lo schema concettuale in uno schema logico (output), che rappresenta gli stessi dati, ma nel formato del modello logico. Rappresentare gli stessi costrutti in termini di modello relazionale –

Nella fase finale della progettazione logica viene effettuata la **traduzione** dello schema concettuale in termini del modello relazionale. Non è una trascrizione (1°1) poiché il modello concettuale è il più ricco di costrutti, e nel modello logico non potremo trovare sempre una controparte; allora si deve *reinterpretare* nei costrutti del modello logico.

Per la progettazione logica abbiamo due fasi:

1. Si prende lo schema concettuale e lo si ristruttura. Lo si sintetizza per prepararlo alla traduzione verso il modello relazionale, per renderlo più vicino all’implementazione.
2. Ottenuto lo Schema concettuale ristrutturato, lo si può tradurre nel modello logico scelto, applicando una serie di regole di trasformazione. Avremo così lo **schema logico**.

Ristrutturazione schema E-R

Motivazioni: semplificare la traduzione 🡪 poiché lo schema concettuale è troppo fornito di costrutti, che ci fanno capire la semantica dei dati, ma ci allontanano dal modello logico in cui dobbiamo tradurlo. Ottimizzare le prestazioni

Parametri per valutare le prestazioni:

Numero di occorrenze (entità e relazioni) previste 🡪 ho una valutazione in termini di spazio, quanto me ne serve per implementare il Database.

Numero di accessi ad occorrenze di (entità e relazioni) durante un’operazione. Vado a vedere le operazioni più frequenti, e vedo quali entità e associazioni sono maggiormente coinvolte. Stimo il numero di accessi e cerco di migliorare le prestazioni.

*Principio di Pareto ( 80:20 )*

Regola empirica che dice: Ciascun sistema informatico dedica l’80% delle sue risorse elaborative per elaborare il 20% delle operazioni più frequenti.

Mi serve stimare il 20% delle operazioni più frequenti così da avere circa l’80% della potenza del sistema.

*Tavole di carico*

**Tavola Volumi**, contenente una stima delle occorrenze per entità ed associazioni

**Tavola delle operazioni**, elenchiamo il 20% delle operazioni più frequenti, e il tipo: se interattiva o batch

**Tavole accessi**: riportiamo per ognuna delle operazioni più frequenti, quali sono gli accessi sulle occorrenze di entità e associazioni, che bisogna compiere per reperire i dati necessari.

**Contare Occorrenze Associazioni**  
Caratteristica delle associazioni 1-N con partecipazione obbligatoria da parte di entrambi è che  
DIPARTIMENTO(1,1) 🡪 N COMPOSIZIONE 1 🡨SEDE(1,N)

Se dove sta N c’è una partecipazione obbligatoria ( significa che ogni dipartimento è collegato ad una sola occorrenza dell’associazione. Dipartimenti diversi sono collegati eventualmente alla stessa sede, ma sempre attraverso diverse associazioni.) le occorrenze dell’associazione sono uguali al numero delle occorrenze dell’entità che si trovano dal lato N. Vale anche per 1-N, almeno uno deve avere frequenza obbligatoria; conto l’occorrenza dal lato obbligatorio.

L’impiegato genererà tante occorrenze di partecipazione, quanti sono i progetti a cui partecipa l’impiegato.

Ogni progetto genera tante occorrenze di partecipazione, quanti sono gli impiegati che vi lavorano.

Impiegato (0,1) 🡨 partecipazione  🡪 Progetto(1,N)

**Attività della ristrutturazione:**

***Analisi delle ridondanze***: cerchiamo di capire se esse sono o meno utili, si decide se eliminarle o meno in base al numero di accessi che provocano sul 20% delle operazioni più frequenti.

* Vantaggi: semplificazione delle interrogazioni.
* Svantaggi: appesantimento degli aggiornamenti - maggiore occupazione di spazio

Per capire se la ridondanza è utile o no, si analizza al tavola dei volumi, la memoria che occupa questo attributo/associazione. Sviluppiamo la tavola delle operazioni ( quante operazioni, quali operazioni e con quale frequenza).Si vede la tavola degli accessi, dopo aver fatto tutti i calcoli si fanno le stime.

*Forme di ridondanza****:*** Attributi derivabili da altri attributi della stessa entità o associazione da attributi di altre entità o associazioni.

***Eliminazione delle generalizzazioni****:* la generalizzazione ci aiuta a definire la semantica/significato dei dati, però nel modello relazionale, non abbiamo una controparte per rappresentarle, cerchiamo di trasformale usando entità e associazioni; semplificando così lo schema. Abbiamo 3 tecniche:

* Eliminare le entità figlie, accorpando gli attributi e le associazioni delle sotto entità, portandole sull’entità genitore. Gli attributi/associazioni aggiunti hanno valore opzionale. Operazione sempre possibile.

Quando accorpo i figli, non so più una data occorrenza a quale sotto entità faceva parte, aggiungo un nuovo attributo TIPO, per capire la corrispondenza.

* Eliminare il genitore, trasportando tutte le sue proprietà (ID, attributi, partecipazione alle associazioni) nelle entità figli . Possibile solo se la gerarchia è totale. Se ho la gerarchia veicolo e ho solo l’entità automobile e autocarro, se elimino il genitore tutti i motocicli non avranno una rappresentazione, poiché la gerarchia non è totale.

Se il padre era collegato ad una associazione, tutti i figli ora avranno una associazione diversa che si colleghi alla stessa entità a cui era legata il padre.

* Lascio l’entità padre e le figlie, sostituisco la generalizzazione con una serie di associazioni 1-1 per ciascuna entità figlia. Il figlio partecipa all’associazione in modo obbligatorio, mentre il padre opzionale. I figli sono entità deboli.

Pattern di cardinalità, le sotto entità partecipano in modo obbligatorio con massimo1, il padre opzionale max 1.

Posso calcolare 3 volte le tavole degli accessi applicando le 3 tecniche. Vedo il numero di accessi sul 20% di operazioni più frequenti. Vedo quanti valori opzionali(null) si generano.

Posso scegliere anche più tecniche per lo stesso progetto.

***Partizionamento/Accorpamento Di Entità E Associazioni***: può capitare di aver più attributi su una entità e serviva un’entità in più, idem per le associazioni. Si riescono a ridurre gli accessi.

* Partizionamento Verticale Di Entità: Se ci sono operazioni frequenti che accedono solo a dati anagrafici, mentre altre a dati lavorativi, partizioniamo l’entità in 2 entità, l’identificatore finirà in una delle 2. L’associazione sarà 1-1 a partecipazione obbligatoria da parte di entrambe. Si va a partizionare lo schema e non l’istanza, ossia gli attributi(la parte intenzionale dello schema )
* Eliminazione Di Attributi Multi Valore: i valori che possono essere associati all’attributo, possono essere distaccate su un’entità a parte. E tutte le occorrenze saranno associate all’occorrenza dell’entità principale tramite un’associazione. La nuova entità partecipa in modo 1-1 ,dall’altra parte 1-N
* Accorpamento Di Entità/Associazione: Se abbiamo 2 entità diverse collegate da un’associazione, le operazioni più frequenti accedono *sempre* agli attributi di entrambe le entità, costringendoci a navigare nell’associazione, è utile accorparle.

Scompongo quando voglio alleggerire il carico di un’operazione se vi sono attributi che non vengono mai presi in considerazione.

***Scelta degli identificatori primari*** per ogni entità, visto che ci stiamo avviando alla fase verso il modello relazionale. Esso non deve avere valore opzionale, deve essere semplice

## Modelli Logici, Caratteristiche

**Gerarchico e reticolare** : utilizzano riferimenti espliciti (puntatori) fra record

**Relazionale** "è basato su valori" : anche i riferimenti fra dati in relazioni diverse sono rappresentati per mezzo di valori. Quando avremo coincidenza di valori tra diverse tabelle stiamo facendo un collegamento.

Non avremo un puntatore a record per collegare uno studente a uno degli esami, ma avremo l’attributo matricola sia nella struttura studente che in quella esame.

**Relazione matematica**: dati n insiemi, dato il prodotto cartesiano di questi insiemi, una relazione è un sottoinsieme del prodotto cartesiano.

Essendo la **relazione** matematica un insieme, esso gode delle seguenti proprietà: i suoi elementi non sono ordinati, l’insieme non ammette duplicati

Per rendere irrilevante l'ordine delle colonne si usa un etichetta così da identificare il suo ruolo

## Tabelle e Relazioni

Una tabella rappresenta una relazione se :

* + le righe sono diverse fra loro (altrimenti non è un insieme)
  + i valori di ogni colonna sono fra loro omogenei

Una tabella che rispetta queste proprietà, e quindi è una relazione, allora l’ordinamento è irrilevante.

Le colonne della tabella sono identificate da attributi distinti e ad ogni colonna corrisponde un dominio della relazione (l’insieme degli attributi è lo <schema>).

Ogni riga corrisponde a una n-upla e non vi possono essere duplicati.

**Vantaggi** struttura basata su valori

* Indipendenza strutture fisiche🡪maggiore portabilità -- (i puntatori invece cambiano su ogni architettura)
* L’utente finale vede gli stessi dati dei programmatore
* Si rappresenta solo ciò che è rilevante dal punto di vista dell’applicazione (senza overhead)
* I puntatori sono direzionali, i valori no. Un puntatore ha una sorgente e una destinazione, i valori no.

**Schema Di Relazione E Di BD**

Nel modello relazionale i concetti di Schema ed Istanza di Base di Dati si traducono in

* **Schema Di Relazione**: descrive la struttura, col suo nome e i suoi attributiA1,...,An 🡪 R(A1...An)
* **Schema Relazionale** **Di Base Di Dati**: insieme di schemi di relazione. R = {R1(X1), ..., Rk(Xk)}

**Ennuple (Tuple) Di Relazioni**

Una Tupla (su un insieme di attributi X) è una funzione che associa ad un attributo A in X, un valore del suo dominio. t[A] denota il valore della tupla t sull'attributo A.

**Istanze di Relazione e di BD**

* Istanza di relazione su uno schema di relazione R(X): è un insieme r di n-uple definite sugli attributi X
* Istanza di base di dati Relazionale su uno schema R= {R1 (X1 ), ..., Rn (Xn )}: è un insieme di n istanze di relazioni , con ri istanza di relazione definita sullo schema Ri(Xi)). **(Ad ogni schema è associata un’istanza)**

**Valori Nulli**

Informazione incompleta nel modello relazionale 🡪 valore nullo, il quale denota l’assenza di un valore.

t[A], per ogni attributo A, diventa un valore del dominio dom(A) oppure il valore nullo NULL

Esistono 2 Tipi Di Valori Nulli :

* **sconosciuto** se il valore c’è ma non è stato inserito
* **inesistente** se non c’è proprio

I DBMS non distinguono i diversi tipi di valore nullo, un abuso di questi valori può creare inconsistenza

**Vincoli Di Integrità**

Proprietà che deve essere soddisfatta dalle istanze che rappresentano le informazioni. Con essi si ha una descrizione più accurata della realtà, con maggiore qualità.

Vincoli Intrarelazionali: riguardano una sola relazione(tabella)

* **Vincoli Di Valori**: vincoli che riguardano i valori che può assumere un singolo attributo. Es - l’intero voto deve essere compreso tra 18 e 30
* **Vincoli Di Ennupla**: riguardano condizioni sui valori di ciascuna ennupla, la combinazione di valori che un insieme di attributi possono assumere. Indipendente da altre ennuple. Dobbiamo guardare una sola ennupla, per verificare se c’è stata qualche violazione.

Es. prezzo base – Iva – prezzo iva inclusa, questi tre valori non sono indipendenti – voto 27 e c’è la lode, entrambi valori ammissibili ma la combinazione tra loro non è ammissibile.

Vincoli Interrelazionali: coinvolgono due o più relazioni

### Identificazione di ennuple

**Superchiave**: un insieme K di attributi della relazione r forma una Superchiave della relazione r se, non esistono in quell’istanza di relazione 2 ennuple che hanno la stessa combinazione di valori su quell’insieme di attributi K.

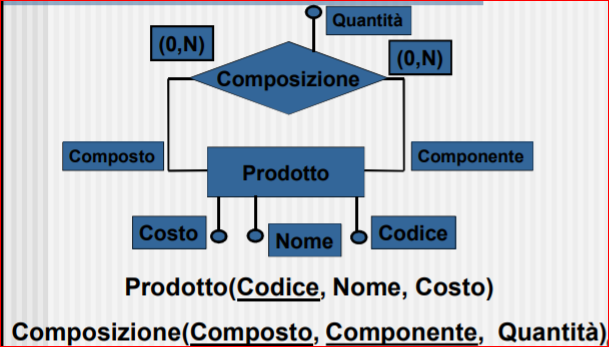
L’Insieme di attributi k è chiave Candidata per l’istanza di relazione r, se oltre ad essere Superchiave per r è anche Superchiave minimale per r. **Superchiave minimale**: non contiene un sottoinsieme dei suoi attributi che forma una Superchiave. Matricola-Nome, contiene all’interno un'altra Superchiave! Se la rimozione di uno dei suoi attributi , fa sì che i rimanenti non abbiamo la proprietà di super key.

Una Superchiave è chiave se nessuno dei suoi attributi può essere eliminato, poiché si perde la proprietà di chiave.

Attributo **primo** se questo appartiene ad almeno una chiave candidata, non primo altrimenti

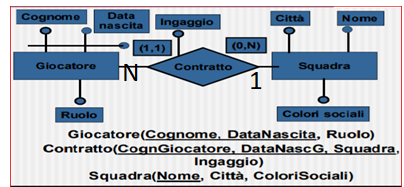
Associazioni molti a molti

Nell’associazione molti a molti introduciamo una relazione, una tabella , non solo per le entità ma anche per l’associazione (joint table).

Associazioni ricorsive

Nella joint table usiamo i nomi dei Ruoli, come chiave esterna.

Ogni Tupla punta a 2 prodotti. Se 1 prodotto è composto da 5 altri prodotti, avremo 5 ennuple che puntano allo stesso composto con lo stesso codice prodotto, però ognuna di essa punta ad un diverso prodotto componente.

Associazioni 1 a molti

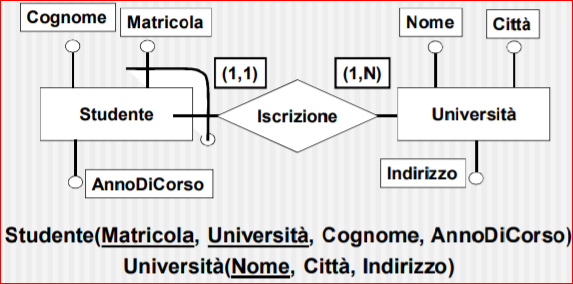
Non è necessario ricorrere alla joint table. Posso far a meno della tabella contratto. Posso mettere la chiave di giocatore in Squadra, ma avrò un attributo multi-valore. In giocatore metto una chiave esterna, la chiave di squadra, poiché ogni giocatore gioca in una squadra.

**Soluzione più compatta**

Giocatore(Cognome, DataNasc, Ruolo, Squadra, Ingaggio) metto anche attributo dell’associazione

Squadra(Nome, Città, ColoriSociali)

REGOLA: prendo l’entità al lato N e ci metto la chiave esterna, poiché questa entità punta ad un solo valore che si trova all’altro lato; non avrò chiave esterna multi-valore.

In questo caso essendo 1 a N, metto in studente la chiave di università. Visto che studente è entità debole, il nome dell’università sarà chiave primaria per studente.

N

1

Relazioni 1 a 1

È preferibile mettere la chiave esterna dove c’è la partecipazione obbligatoria.

Se partecipano entrambi obbligatorio o opzionale si vede la chiave più semplice da trasportare

**Algebra relazionale** contiene Insieme di operatori che operano su relazioni e ne producono altre

notiamo che le relazioni sono insiemi, e quindi ha senso definire su di esse gli operatori insiemistici tradizionali di unione, differenza e intersezione. Essi sono applicabili solo a relazioni definite sugli stessi attributi.

***Ridenominazione*** : operatore monadico. Produce un risultato che modifica lo schema lasciando inalterata l'istanza dell'operando. REN NewNames ← OldNames (Operando)

cambia il nome degli attributi specificati in OldNames con quelli specificati in NewNames

***Selezione*** : operatore monadico che produce un risultato con lo stesso schema dell’operando. Si ha un sottoinsieme delle ennuple dell'operando che soddisfano una condizione

SEL Condizione (Operando) il risultato contiene le ennuple dell'operando che soddisfano la condizione

***Proiezione:*** monadico. produce un risultato che contiene parte degli attributi dell'operando, un sottoinsieme dello schema.

PROJ ListaAttributi (Operando)

**Join naturale** correla dati in relazioni diverse, sulla base di valori uguali in attributi con lo stesso nome.

Produce una relazione definita sull’unione degli attributi degli operandi e le sue tuple sono ottenute combinando le tuple degli operandi con valori uguali.

In generale, il join naturale r1(X1) e r2(X2) è una relazione definita su X1 unito X2 come segue:

* r1 join r2 = { t su X1X2 | esistono t1 ∈ r1 e t2 ∈ r2 con t[X1] = t1 e t[X2] = t2 }

il risultato è dato da un insieme di ennuple t tale che esistono una ennupla t1 in r1 ed una ennupla t2 in r2 tale che se prendiamo la porzione di t relativa agli attributi di X1, questa coincide con t1; mentre la porzione di t relativa al sottoinsieme di attributi X2 coincide con la ennupla proveniente da r2 che si chiama t2.

* r1 join r2 = { t su X1X2 | t[X1] ∈ r1 e t[X2] ∈ r2 }

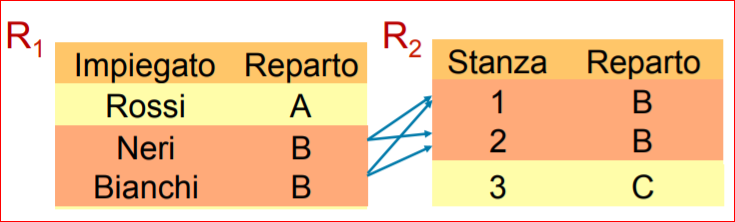
t[X1]: valore che la tupla t ha per gli attributi X1

Prendiamo tutte le possibili combinazioni delle ennuple di r1 e r2, e le uniamo solo quando il valore dell’attributo comune di r1 e r2 coincide.

**Join completi e incompleti**

Abbiamo un join completo se ogni ennupla contribuisce al risultato

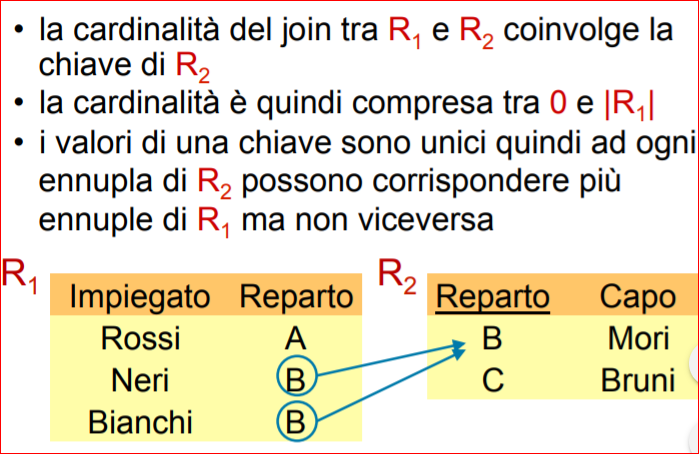
**Cardinalità join** ( Numero di ennuple che possiamo trovare )

Se congiungiamo due relazioni

1. **0 ≤ |R1 JOIN R2| ≤ |R1| × |R2|** 
   * + 0 se join vuoto
     + r1xr2 join completo

|R1 JOIN R2| = 4

1. se il join coinvolge una chiave di R2, allora il numero di ennuple è compreso fra zero e |R1| **0 ≤ |R1 JOIN R2| ≤ |R1|**

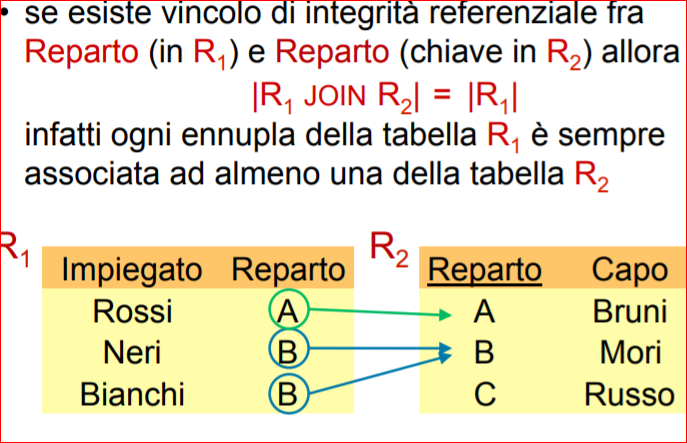


Se effettuiamo il join su un attributo comune di r1 e r2 ed esso è chiave di r2. Provando tutte le combinazioni possibili. Presa una ennupla di r1 sull’attributo comune può matchare al massimo con una ennupla di r2 ( può essere anche 0 ). Siccome l’attributo comune su r2 è chiave, non ci possono essere più match, non ci possono essere due ennuple con la stessa chiave

1. se il join coinvolge una chiave di R2 e un vincolo di integrità referenziale, allora il numero di ennuple è pari a |R1|

**|R1 JOIN R2| = |R1|**

* + Se l’attributo comune oltre ad essere chiave di R2 è anche chiave esterna in R1. L’attributo comune punta alla chiave candidata di R2. Avendo un vincolo di integrità referenziale su R2 abbiamo che ogni ennupla di R1 avrà un valore dell’attributo comune ( chiave esterna ) che troviamo sicuramente in ennupla di R2; Quindi non abbiamo riferimenti inesistenti. Ogni ennupla di R1 avrà un solo attributo di valore comune ad una ennupla di R2.



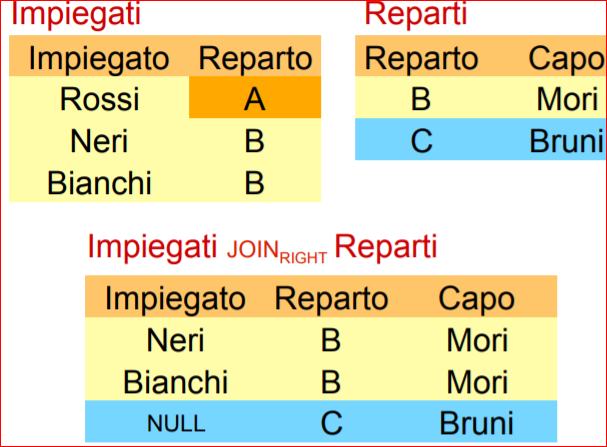
**Join esterno**

Il **join esterno** prendono tutte le ennuple che fanno match, quelle che non fanno match le prende le estende, con valori nulli, le ennuple che verrebbero tagliate fuori da un join (interno)

Esistono tre versioni: sinistro, destro, completo

**Join sinistro**: prende le ennuple che fanno match in più riporta tutte quelle dell’operando sinistro che non hanno fatto match e mette a null gli attributi che provengono dall’operando destro

**Join destro**: riporta le ennuple che fanno match, riporta poi tutte quelle dell’operando di destra che non hanno fatto match, imposta a null gli attributi che provengono dall’operando sinistro

**Join completo**: riporta le ennuple che fanno match, quelli di sinistra e a destra che non fanno match li imposta a null

**Prodotto cartesiano**

* un join naturale su relazioni senza attributi in comune.
* Contiene tutte le possibili combinazioni di ennuple su R1 e R2. Le ennuple sono pari al prodotto delle cardinalità di R1 e R2 (degli operandi)(le ennuple sono tutte combinabili).

Lo possiamo vedere come un join naturale che non ha nessun attributo in comune.

Esso ha senso quasi solamente in seguito ad una selezione: SELCondizione (R1 JOIN R2)

L'operazione viene chiamata **theta-join** e indicata con  R1 JOINCondizione R2

La condizione C è spesso una congiunzione  (AND) di atomi di confronto A1ϑ A2 dove ϑ è uno  degli operatori di confronto. A1 e A2 sono attribuiti di R1 ed R2.

Così posso unire le ennuple del primo operando con quelle del secondo sulla base della condizione booleana

Se l'operatore di confronto nel theta-join è sempre l'uguaglianza (=) allora si parla di **equi-join.**

Due **espressioni** sono **equivalenti** se producono lo stesso risultato qualunque sia l'istanza attuale della base di dati

Push selections (se A è attributo di R2 ) sempre meglio fare una selezione e poi la join.

SEL A=10 (R1 JOIN R2) = R1 JOIN SEL A=10 ( R2)

Abbiamo 2 interrogazioni equivalenti però quella con join è ottima, poiché effettua meno operazioni, poiché prima riduciamo le tuple e poi facciamo join su un insieme ridotto di tuple.

Viste

Rappresentazione di una o più tabelle dello schema, a cui è associata un query.

È una rappresentazione derivata, da relazioni presenti nel nostro database, la vista è definita per mezzo di interrogazioni. Si usano per definire gli schemi esterni.

**Viste** **materializzate**: vista che una volta eseguita viene trasformata in una vera tabella e quindi memorizzata, è permanente il suo contenuto.

* vantaggi: dati immediatamente disponibili per le interrogazioni
* svantaggi: ridondanza (duplico dati), possibilità di inconsistenza dati; se in seguito ad un aggiornamento, aggiorno una sola copia, l’altra sarà inconsistente. Ho anche spreco di spazio

Un interrogazione su una vista viene eseguita prima "ricalcolando" la vista

Viste (virtuali), motivazioni

Schema esterno: ogni utente vede solo i dati di suo interesse o i dati che è autorizzato a vedere

Strumento di programmazione:

* si può semplificare la scrittura di interrogazioni evitando espressioni complesse o sotto espressioni ripetute
* L'utilizzo di viste non influisce sull'efficienza delle interrogazioni

**5 Operatori Minimali Dell’algebra Relazionale**

Con questi 5 operatori si possono emulare tutti gli altri operatori. Minimali perché da soli possono simularli, ma un loro sottoinsieme non permetterà di emulare qualche operazione.

Selezione, Proiezione, Prodotto Cartesiano, Unione, Sottrazione.

A ⋂ B = A u B – ( A - B ) – ( B – A ) = A – ( A – B ) = B – ( B – A )

In informatica, SQL (Structured Query Language) è un linguaggio per database basati sul modello relazionale, progettato per le seguenti operazioni:

creare e modificare schemi di database (DDL = Data Definition Language);

inserire, modificare e gestire dati memorizzati (DML = Data Manipulation Language);

creare e gestire strumenti di controllo e accesso ai dati (DCL = Data Control Language).

SQL è un linguaggio per interrogare e gestire basi di dati mediante l'utilizzo di costrutti di programmazione denominati query.

La clausola **select** specifica gli elementi dello schema della tabella risultato, che rappresenta la selezione degli attributi delle tabelle elencate nella clausola from.

Clausola **from** Quando si desidera formulare un'interrogazione che coinvolge righe appartenenti a più di una tabella, si pone come argomento della clausola fro m l'insieme di tabelle alle quali si vuole accedere.

Clausola **where** La clausola where ammette come argomento una espressione booleana costruita combinando predicati con gli operatori and , or e not .

Il comando SELECT la clausola DISTICT serve ad estrarre una sola volta ogni diversa occorrenza di un valore all'interno di un dato campo.

Lo standard SQL prevede cinque operatori **aggregati**: count , sum, max, min e avg .

Gli operatori si applicano sulle righe che soddisfano la condizione presente nella clausola where e hanno il seguente significato:

• sum: restituisce la somma dei valori posseduti dall'espressione;

• max e min : restituiscono rispettivamente il valore massimo e minimo;

• avg : restituisce la media dei valori (vale a dire, il risultato della divisione di sum per count)

La *Select* deve essere omogenea perché usando Operatori aggregati e semplici avrò che l’operatore aggregato mi tira fuori un solo valore, quello semplice prende i valori di tutte le tuple

Il linguaggio non offre un meccanismo per gestire questa eterogeneità e perciò la sintassi SQL non ammette che nella stessa clausola select compaiano funzioni aggregate ed espressioni al livello di riga, a meno che non si faccia uso della clausola group by

**Interrogazioni con raggruppamento**

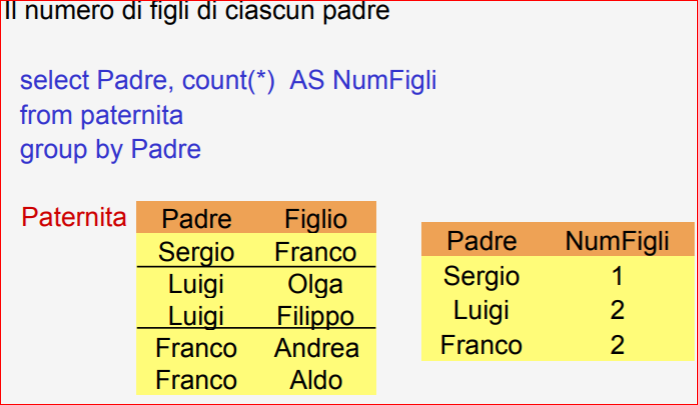
Molto spesso sorge l'esigenza di applicare l'operatore aggregato separatamente a sottoinsiemi di righe. Per poter utilizzare in questo modo l'operatore aggregato, SQL mette a disposizione la clausola **group by,** che permette di specificare come dividere le tabelle in sottoinsiemi. La clausola ammette come argomento un insieme di attributi e l'interrogazione raggrupperà le righe che possiedono gli stessi valori per questo insieme di attributi.

La tabella ottenuta viene poi analizzata, dividendo le righe in insiemi caratterizzati dallo stesso valore degli attributi che compaiono come argomento della clausola group by .

La sintassi SQL impone che, in un'interrogazione che fa uso della clausola group by, possa comparire come argomento della select solamente un sottoinsieme degli attributi usati nella clausola group by

Usare attributi semplici e aggregati (Clausola Mista) è possibile con la GROUP BY. Si può invocare solo l’attributo semplice che è stato usato per il raggruppamento, o un sottoinsieme se nel GROUP BY abbiamo più attributi.

Solo padre si può mettere nella select come attributo semplice



La capacità dei DBMS di “ottimizzare” le interrogazioni rende (di solito) non necessario preoccuparsi dell'efficienza quando si specifica un'interrogazione ,è perciò più importante preoccuparsi della chiarezza.

Scorretta select padre, avg(f.reddito), p.reddito

from persone f join paternita on figlio = f.nome join

persone p on padre =p.nome

group by padre

corretta select padre, avg(f.reddito), p.reddito

from persone f join paternita on figlio = f.nome join

persone p on padre =p.nome

group by padre, p.reddito

io vorrei nome padre , reddito medio figli , reddito padre. Raggruppiamo per padre, quindi avremo gruppo di padre con stesso reddito(padre) è scorretta perché il DBMS non sa che il reddito è uguale, non abbiamo raggruppato per quel attributo!

**Condizioni sui gruppi**

Una applicazione può aver bisogno di considerare solo i sottoinsiemi che soddisfano certe condizioni. Se le condizioni che i sottoinsiemi devono soddisfare sono verificabili al livello delle singole righe, allora basta porre gli opportuni predicati come argomento della clausola where. Se invece le condizioni sono di tipo aggregato, sarà necessario utilizzare un nuovo costrutto, la clausola **having** .

La clausola having descrive le condizioni che si devono applicare al termine dell'esecuzione di un'interrogazione che fa uso della clausola group by. Ogni sottoinsieme di righe costruito dalla group by fa parte del risultato dell'interrogazione solo se il predicato argomento della having risulta soddisfatto.

**HAVING O WHERE**? 🡪 Per sapere quali predicati di un'interrogazione che fa uso del raggruppamento vanno dati come argomento della clausola where e quali come argomento della clausola having , basta rispettare il seguente criterio:

solo i predicati in cui compaiono operatori aggregati devono essere argomento della clausola having. Where lo uso quando posso prima scremare le tuple

estrarre i dipartimenti per cui la media degli stipendi degli impiegati che lavorano nell'ufficio 20 è superiore a 25 mila euro.

select Dipart

from Impiegato

where Ufficio = 20

group by

Dipart having avg(Stipendio ) >25

**Sintassi** SELECT

SELECT ListaAttributiOEspressioni

FROM ListaTabelle

[ WHERE CondizioneSemplice]

[ GROUP BY ListaAttributiDiOrdinamento

[ HAVING CondizioniAggregate ]]

[ ORDER BY ListaAttributiDiOrdinamento]

**Interrogazioni di tipo insiemistico** {(union | intersect | except ) [ all ]

Gli operatori insiemistici, al contrario del resto del linguaggio, assumono come default di eseguire una eliminazione dei duplicati poiché rispetta molto meglio il tipico significato di questi operatori. Qualora nell'interrogazione si voglia preservare i duplicati, sarà sufficiente utilizzare l'operatore con la parola chiave **all.** Un'altra osservazione è che SQL richiede che gli attributi siano in pari numero e che abbiano domini compatibili.

La corrispondenza tra gli attributi non si basa sul nome ma sulla *posizione* degli attributi. Se gli attributi hanno nome diverso, il risultato normalmente usa i nomi del primo operando.

Except prende il risultato della prima e da questi ci sottrai il risultato della seconda query

Intersect

**Interrogazioni nidificate**

Nella clausola where si possono avere predicati semplici (Finiti) o espressioni complesse che comprende anche il risultato di una sotto interrogazione.

La parola chiave **any** specifica che la riga soddisfa la condizione se risulta vero il confronto (con l'operatore specificato) tra il valore dell'attributo per la riga e almeno uno degli elementi restituiti dall'interrogazione.

Una riga soddisfa la condizione se risulta vero il confronto fra il valore dell’attributo per la riga e almeno uno degli elementi restituiti dalla sotto espressione. Vedo se il valore è uguale ad uno dei valori tuple della query

La parola chiave **all** invece specifica che la riga soddisfa la condizione solo se tutti gli elementi restituiti dall'interrogazione nidificata rendono vero il confronto.

Una riga soddisfa la condizione se risulta vero il confronto fra il valore dell’attributo per la riga e tutti gli elementi restituiti dalla sotto espressione

**IN**( Sotto espressione )

Una riga soddisfa la condizione se il valore dell’attributo per la riga è contenuto negli elementi restituiti dall’interrogazione. L’elemento è contenuto nell’insieme di valori della sotto query

La sintassi richiede la compatibilità di dominio tra l'attributo restituito dalla interrogazione nidificata e l'attributo con cui avviene il confronto.

La forma nidificata è “meno dichiarativa” ma talvolta più leggibile (no alias)

Le sotto interrogazioni non possono contenere operatori insiemistici (“l’unione si fa solo al livello esterno”)

**Regole di divisibilità**

Non è possibile fare riferimenti a variabili definite in blocchi più interni

Se non uso alias e ci sono due volte la stessa tabella, se faccio riferimento a un suo attributo, si assume riferimento alla variabile più “vicina”

In un blocco si può fare riferimento a variabili definite in blocchi più esterni, nei blocchi più interni si può far riferimento a variabili definite in blocchi esterni. Se stanno allo stesso livello, non si possono “Vedere”

**Operatore Logico Exists**

Questo operatore ammette come parametro un'interrogazione nidificata e restituisce il valore vero solo se la l’interrogazione restituisce almeno una tupla. **L’interrogazione interna viene eseguita una volta per ciascuna ennupla dell’interrogazione esterna**

**Modifica dei dati in SQL**

La parte di Data Manipulation Language comprende i comandi che permettono di modificare la base di dati sono insert , delete e update .

**Inserimento**

insert into NomeTabella [ Lista Attributi ]

{ values (listaValori) | SelectSQL}

Con la lista attributi definisco gli attributi che inserisco, così se il numero di valori è inferiore rispetto alla #totale di attributi, esso avrà valore nullo. se l'inserimento viola un vincolo di not null definito sull'attributo, l'inserimento viene rifiutato.

Possiamo aggiungere degli insiemi di righe, estratti dal contenuto della base di dati.

**Cancellazione**

Il comando delete elimina righe dalle tabelle della base di dati

delete from NomeTabella [where Condizione]

Quando la condizione argomento della clausola where non viene specificata, il comando cancella tutte le righe dalla tabella, altrimenti vengono rimosse solo le righe che soddisfano la condizione. Qualora esista un vincolo di integrità referenziale con politica di cascade in cui la tabella viene referenziata, allora la cancellazione di righe dalla tabella può comportare la cancellazione di righe appartenenti ad altre tabelle

drop table NomeTabella [comando]

elimina tutte le righe dalla tabella NomeTabella lo schema della base di dati viene modificato, eliminando dallo schema la tabella DIPARTIMENTO

**Modifica 🡪 Update**

update NomeTabella

set Attributo={ Espressione | SelectSQL | null | default )

{Attributo=( Espressione \ SelectSQL | null | default ) }

[ where Condizione ]

Il comando di update permette di aggiornare uno o più attributi delle righe di NomeTabella che soddisfano l'eventuale Condizione. Se il comando non presenta la clausola where , come al solito si suppone che la condizione sia soddisfatta e si esegue la modifica su tutte le righe. Il nuovo valore cui viene posto l'attributo può essere:

* il risultato della valutazione di un'espressione sugli attributi della tabella, che può anche far riferimento al valore corrente dell'attributo che verrà modificato dal comando;
* il risultato di una generica interrogazione SQL;
* il valore nullo;
* il valore di default per il dominio.

Il vincolo **CHECK** assicura che tutti i valori presenti in una colonna soddisfino determinate condizioni. Una volta definito, se il nuovo valore soddisfa il vincolo **CHECK**, il database inserirà solo una nuova riga o ne aggiornerà una esistente. Il vincolo **CHECK** viene utilizzato per assicurare la qualità dei dati. Specifica di vincoli di ennupla. Ad ogni vincolo di check si può associare un nome (attraverso la clausola constraint).

**Asserzioni**

Grazie alla clausola check è possibile definire le asserzioni, ossia vincoli a livello di schema. Le asserzioni possiedono un *nome* , tramite il qual e possono essere eliminate esplicitamente dallo schema con l'istruzione drop

create assertion Nome Asserzione check (Condizione )

**Viste**

create view NomeVista [ ( ListaAttributi ) ] as SelectSQL

[ with [ local | cascaded ] check option ]

Le viste vengono definite in SQL associando un nome e una lista di attributi al risultato dell'esecuzione di una interrogazione. Nell'interrogazione che definisce la vista possono comparire anche altre viste.

**Modificare i contenuti della vista**

Su certe viste è permesso effettuare operazioni di modifica, che verranno tradotte

negli opportuni comandi di modifica al livello delle tabelle di base da cui la vista

dipende. Si può effettuare aggiornamenti su viste che sono definite su singole tabelle.

La **check option** specifica che possono essere ammessi aggiornamenti a condizione che le tuple dopo l’aggiornamento non devono violare le condizioni di appartenenza a quella vista; e vi sia in considerazione una sola tabella, senza join.

se si assegna a un attributo della vista un valore che rende falso uno dei predicati di selezione la modifica sarà vietata.

**Controllo dell'accesso**

In SQL è possibile specificare chi (utente) e come (lettura, scrittura, …) può utilizzare la base di dati (o parte di essa).

Oggetto dei *privilegi* (diritti di accesso) sono di solito le tabelle, ma anche altri tipi di risorse, quali singoli attributi, viste.

Un amministratore/creatore ha tutti i privilegi.

Un privilegio è caratterizzato da:

* la risorsa cui si riferisce
* l'utente che concede il privilegio
* l'utente che riceve il privilegio
* l'azione che viene permessa
* la trasmissibilità del privilegio (se l’utente può trasmettere a sua volta ad altri)

**Tipi di privilegi offerti da SQL**

* **insert**: permette di inserire nuove ennuple
* **update**: permette di modificare il contenuto
* **delete**: permette di eliminare oggetti
* **select**: permette di leggere la risorsa
* **references**: permette la definizione di vincoli di integrità referenziale verso la risorsa (può limitare la possibilità di modificare la risorsa) con l’utilizzo di no action (in un'altra tabella quindi, posso vietare la cancellazione della tupla se c’è un'altra tupla che fa riferimento ad essa)
* **usage**: permette l'utilizzo di un oggetto in una definizione (per esempio, di un dominio posso consentire di usare o meno il dominio vuoto)

**Comandi per concedere e revocare privilegi**

**Concessione di privilegi:**

grant <Privileges |all privileges> on Risorsa to Utenti[with grant option]

elenco di privilegi da concedere o concederli tutti

Risorsa tabella, dominio

to a quale utente

*with grant option* specifica se il privilegio può essere trasmesso ad altri utenti

**Revoca di privilegi**

revoke Privilegi on Risorsa from Utenti [ restrict | cascade ]

**restrict** (di default) specifica che il comando non deve essere eseguito qualora la revoca dei privilegi all’utente comporti qualche altra revoca (dovuta ad un precedente grant option)

**cascade** invece forza l’esecuzione del comando

**Transazioni**

Insieme di operazioni da considerare indivisibile ("atomico"). Un sistema che mette a disposizione un meccanismo per la definizione e l'esecuzione di transazioni viene detto sistema transazionale.

Tutto il codice che viene eseguito all'interno di una transazione gode di proprietà particolari, le cosiddette proprietà acide delle transazioni: atomicità, consistenza, isolamento e persistenza ACID.

**Atomicità:** L’Atomicità rappresenta il fatto che una transazione è un'unità indivisibile di esecuzione; o vengono resi visibili tutti gli effetti di una transazione, oppure la transazione non deve avere alcun effetto sulla base di dati. In pratica, non è possibile lasciare la base di dati in uno stato intermedio.

Se durante l'esecuzione delle operazioni si verifica un errore e una delle operazioni non può essere portata a compimento, allora il sistema deve essere in grado di ripristinare lo stato della base di dati, disfacendo il lavoro svolto dalle istruzioni eseguite fino a quel momento.

**Consistenza:** La consistenza richiede che l'esecuzione della transazione non violi i vincoli di integrità definiti sulla base di dati. Quando il sistema rileva che una transazione sta violando uno dei vincoli, per esempio che si sta inserendo una tupla con un campo chiave avente un valore già presente nella tabella, il sistema interviene per annullare la transazione

**Isolamento:** L'isolamento richiede che l'esecuzione di una transazione sia indipendente dalla contemporanea esecuzione di altre transazioni. In particolare, si richiede che il risultato dell'esecuzione concorrente di un insieme di transazioni sia analogo al risultato che le stesse transazioni otterrebbero qualora ciascuna di esse fosse eseguita da sola.

**Persistenza:** La persistenza invece richiede che l'effetto di una transazione non venga perso per nessun motivo

L'inizio di una transazione è rappresentato dal comando **star transaction** . le operazioni che seguono non vengono subito scritte sulla base di dati, ma dopo essersi accertati che le operazioni siano andate a buon fine, si rendono definitive

Il termine della transazione è rappresentato da due istruzioni particolari

**commit** [work]: le operazioni specificate a partire dall'inizio della transazione vengono eseguite e rese definitive sulla base di dati

**rollback** [work]: si può annullare gli effetti del lavoro svolto dalla transazione, indipendentemente dalla sua complessità e riporta allo stato prima della transazione

**SQL immerso**

le istruzioni SQL sono “immerse” nel programma sorgente scritto nel linguaggio “ospite” le istruzioni SQL.

Un precompilatore viene usato per analizzare il programma e tradurlo in un programma nel linguaggio ospite (sostituendo le istruzioni SQL con chiamate alle funzioni di una API del DBMS, leggibili dal linguaggio ospite).

Le istruzioni vengono distinte da quelle normali tramite un opportuno separatore, la stringa EXEC SQL, sono le porzioni di interesse del precompilatore. Preleva le istruzioni in cui compare exec sql le analizza e le trasforma i blocchi di codice SQL in blocchi di codice nel linguaggio ospite( c ).

Le variabili del programma possono essere usate come “parametri” nelle istruzioni SQL precedute da “ : ”.

Devo verificare che la connessione sia andata a buon fine, utilizzo una struttura che è definita nella libreria, essa è SQLCA e permette di gestire la comunicazione tra il programma e il DBMS. Ha un campo SQLCODE che contiene un intero, rappresenta l’esito dell’ultima operazione:

zero 🡪 successo, altro 🡪 no

Una coppia di comandi BEGIN DECLARE SECTION ed END DECLARE SECTION Sono necessari se si desidera poi utilizzare le variabili del programma come parametri per i comandi SQL.

FASI

Scriviamo il programma più blocchi sql, che viene sottoposto al Precompilatore il quale analizza solo i blocchi SQL e li trasforma in blocchi in c.

Così Abbiamo codice sorgente precompilato che aderisce alla sintassi del linguaggio ospite, il quale viene compilato

Viene prodotto il codice oggetto e nella fase di collegamento, le librerie e API usate vengono collegate; viene generato infine il codice eseguibile.

I linguaggi di programmazione accedono agli elementi di una tabella scandendone le righe una a una, utilizzando quello che viene detto un approccio tuple-oriented. Al contrario, SQL è un linguaggio di tipo set-oriented, che opera su intere tabelle, non su singole righe, e che restituisce come risultato di una interrogazione un'inter a tabella.

Un importante problema che caratterizza l'integrazione tra SQL e i normali linguaggi di programmazione è il cosidetto problema del conflitto d'impedenza. Sappiamo che Il risultato di una select è costituito da zero o piú ennuple: zero o una: il risultato puó essere gestito in un record

Più ennuple? l’insieme (in effetti, la lista) non è gestibile facilmente in molti linguaggi.

Quindi si ricorre alla tecnica del **Cursore**: tecnica per trasmettere al programma una ennupla alla volta.

Si crea una struttura che si frappone tra programma e DBMS, una buffer che contiene il risultato della query(insieme di tuple) con un puntatore ad una specifica ennupla, e da programma si possono gestire le ennuple spostandosi col puntatore. Trasmette le ennuple al programma una alla volta

Operazioni sui cursori

Definizione del cursore

declare NomeCursore [ scroll ] cursor for Select

//cursore per la qury select…

Esecuzione dell'interrogazione

open NomeCursore

Utilizzo dei risultati (una ennupla alla volta)

fetch NomeCursore into ListaVariabili

Disabilitazione del cursore

close cursor NomeCursore

Accesso alla ennupla corrente (di un cursore su singola relazione a fini di aggiornamento) nella clausola where

current of NomeCursore

Le interrogazioni per cui è garantito che venga restituita una sola riga (per esempio perché nella condizione compare un predicato di uguaglianza con una chiave) vengono dette query scalari; per queste interrogazioni è possibile utilizzare una semplice interfaccia tra SQL e il linguaggio di programmazione , che non richiede di definire un cursore. Si può infatti in questo caso fare uso della clausola into , mediante la quale si stabilisce in modo diretto a quali variabili del programma debba essere assegnato il risultato della interrogazione.

**SQL dinamico**

In diverse situazioni sorge la necessità di permettere all'applicazione di definire al momento dell'esecuzione le interrogazioni da effettuare sulla base di dati. L’ SQL Embedded, permette l'uso di SQL dinamico, con questi comandi è possibile costruire un programma che esegue delle istruzioni SQL costruite al momento dell'esecuzione del programma.

Nel caso di SQL statico, i comandi SQL vengono gestiti da un preprocessore che analizza la struttura del comando e ne può costruire una traduzione nel linguaggio interno del sistema. In questo modo il comando non deve essere analizzato e ottimizzato ogni volta che viene richiesta la sua esecuzione. Ciò porta dei considerevoli vantaggi in termini di prestazioni.

**Esecuzione immediata**

Mediante il comando di **execute** immediate si richiede l'esecuzione di una istruzione SQL, specificata direttamente. Può essere utilizzato solo per comandi che non richiedono parametri né in ingresso né in uscita, come per esempio alcuni comandi di inserimento e cancellazione.

execute immediate SQLStatement

**Fase di preparazione**

II comando **prepare** analizza un'istruzione SQ L e la traduce. Esso associa alla traduzione dell'istruzione un nome , che può essere poi usato dagli altri comandi:

prepare NomeComando from IstruzioneSQL

Per eseguire un comando che è stato pre-elaborato da una prepare si usa il comando di execute, con la seguente sintassi:

execute NomeComando [ into ListaTarget ]

[ using ListaParametri ]

*La lista dei target* contiene l'elenco dei parametri in cui deve essere scritto il risultato dell'esecuzione del comando è opzionale qualora il comando SQL non restituisca dei valori.

*La lista dei parametri* specifica invece quali sono i valori che devono essere assunti dai parametri variabili della lista (anche questa parte può non essere presente qualora il comando SQ L sia privo di parametri).

**Call Level Interface (CLI)**

È uno standard. Il programmatore dispone direttamente di una libreria di funzioni per realizzare il dialogo con la base di dati. Rispetto alla soluzione SQL Embedded , con la CLI si dispone normalmente di uno strumento più flessibile, esso è indipendente dal DBMS. (JDBC)

Il modo general e d'uso di questi strumenti è il seguente

1. Si utilizza un servizio della CLI pe r creare un a connessione con il DBMS .

2. Si invia sulla connessione un comando SQL che rappresenta la richiesta.

3. Si riceve come risposta una struttura relazionale in un formato ; la CLI permette di analizzare e descrivere la struttura del risultato.

4. Si chiude la connessione e si rilasciano le strutture dati utilizzate

**Stored procedure**: è un programma che viene memorizzato all'interno della base di dati. Essa ha un nome e dei parametri ed è utilizzabile come se fosse un comando SQL.

* La Stored procedure evita al client di riscrivere query complesse offrendo la possibilità di richiamare una procedura archiviata all'interno del database.
* La compilazione di una Stored procedure avviene una volta sola, al suo inserimento. Ogni volta che la procedura viene richiamata viene semplicemente eseguita e questo aumenta significativamente le prestazioni.
* Le Stored procedure aumentano il carico di lavoro per il server

**Trigger**: detto anche regola attiva. Ogni trigger si attiva quando occorre uno specifico evento all'interno della base di dati; se è soddisfatta una data condizione , allora il trigger esegue un'azione stabilita.

Un trigger è composto da :

un *evento* insert delete su tabelle, e update

*condizione* è rappresentata da un generico predicato

*l'azione* del trigger è rappresentata da un singolo comando

Una delle applicazioni più classiche dei trigger è rappresentata dalla gestione di vincoli di integrità generici. L'approccio si basa sulla scrittura di trigger che:

(1) sono sensibili a tutti gli eventi che possono introdurre violazioni nei vincoli,

(2) verificano nella condizione se le violazioni sono effettivamente presenti e in tal caso

(3) eseguono le azioni che hanno la responsabilità di intervenire sulla base di dati per eliminar e le violazioni.

* Lo scopo è normalmente quello di specificare le reazioni che il sistema deve mettere in atto in presenza di situazioni prestabilite.

**JDBC**

È una API (Application Programming Interface) di Java (intuitivamente: una libreria) per l'accesso a basi di dati, in modo indipendente dalla specifica tecnologia.

JDBC è una interfaccia, realizzata da classi chiamate driver: l'interfaccia è standard, mentre i driver contengono le specificità dei singoli DBMS

Funzionamento :

• Caricamento del driver

• Apertura della connessione alla base di dati

• Richiesta di esecuzione di istruzioni SQL

• Elaborazione dei risultati delle istruzioni SQL

La **normalizzazione** è usata per valutare la qualità degli schemi di basi di dati, privo di difetti.

Quando una relazione non è normalizzata: presenta ridondanze, problemi in aggiornamenti/cancella

Esistono delle misure informali di qualità:

1. Semantica degli attributi di una relazione: Tutti gli attributi hanno associato un significato chiaro.

Non combinare attributi da entità e relazioni differenti in una singola tabella!

1. Riduzione dei valori ridondanti nelle tuple 🡪 poiché si può avere sia un’occupazione di spazio maggiore, sia anomalie di: Cancellazione – Inserimento – Modifica.

Anomalie di Inserimento : non si inserire una tupla in una tabella se non si hanno a disposizione altri dati di altre tabelle.

Anomalie di Cancellazione: se in una tabella mettiamo dati diversi , che andavano su più tabelle se cancelliamo una tupla di questa tabella possiamo perdere altri dati, che , si potevano gestire su più tabelle.

Anomalie di Modifica: se si modifica un attributo, si dovranno modificare tutti gli altri, Altrimenti il database diventa inconsistente.

1. Riduzione dei valori NULL nelle tuple: Se nel disegno di uno schema di database raggruppiamo molti attributi in una relazione, può capitare che alcuni degli attributi non riguardano tutte le tuple. Pertanto, possiamo avere molti valori NULL. Esso può avere diverse interpretazioni:
   * + - 1. L'attributo non si applica alla tupla
         2. Il valore dell'attributo per la tupla non è noto
         3. Il valore dell’attributo non è stato ancora registrato.
2. Tuple spurie 🡪 se volessimo ottenere delle informazioni con un join si ottengono anche tuple

spurie, rappresentanti informazioni errate. Solitamente poiché l’attributo che collega le due tabelle su cui cerchiamo di fare join non è né chiave primaria, né chiave esterna in nessuna delle due relazioni.

Linea Guida: Progettare gli schemi di relazione in modo da poter effettuare JOIN con condizioni di uguaglianza su attributi che sono o chiave primaria o chiave esterna, in modo da non generare tuple spurie.

Sommario: Abbiamo mostrato che una cattiva progettazione dello schema di un database può portare ad una serie di problemi:

* + Anomalie che implicano un maggiore sforzo nell’inserimento e nella modifica di una tabella, e che possono portare a perdite accidentali di dati durante la cancellazione.
  + Spreco di spazio a causa dei valori null.
  + Generazione di tuple spurie o non valide durante operazioni di join.

**Dipendenze Funzionali**

Vincolo che si definisce in fase di progettazione e descrive legami di tipo funzionale tra gli attributi di una relazione. Deve valere su tutte le possibili istanze dello schema (proprietà intenzionale)

Una dipendenza funzionale, denotata da X 🡪 Y, tra due insiemi di attributi X e Y specifica un vincolo sulle possibili tuple che possono formare una istanza.

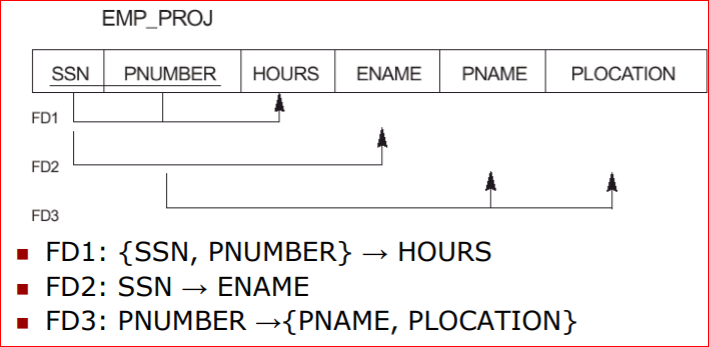
Il vincolo stabilisce che se X 🡪 Y, allora V t1 , t2 in r tali che t1X = t2X, deve valere t1Y = t2Y .

Se X = Data … t1X: prendere il valore di Data della tupla t1

Data X 🡪 Età Y ogni volta che la data di nascita è uguale anche l’età è uguale!

Prese due tuple, se hanno valore uguale su X, devono avere valore uguale su Y.

t1X = t2X è la nostra ipotesi, e deve valere nella tesi che t1Y = t2Y.



1) Ogni volta che ssn e pnumber sono uguali, allora il numero di ore è uguale.

2) Ogni volta che ssn è uguale anche il nome del dipendente è lo stesso

3) Stesso numero di progetto implica che nome di progetto e luogo progetto è lo stesso

**Regole di inferenza(Deduzione)**

Date le n dipendenze funzionali che si hanno (F: l’insieme delle dipendenze funzionali) , esse non sono tutte quelle possibili, ce ne possono essere altre.

Così partendo da quelle definite, si va a determinare la chiusura dell’insieme (F) , è un meccanismo automatico per ricavare nuove dipendenze funzionali a partire da quelle note. Quindi nuove dipendenze funzionali possono essere dedotte a partire da altre

F╞ X🡪Y non era presente in F ma è stata inferita/dedotta a partire da quelle definite in F

X🡪Y è nuova ed è inferita da F

{X,Y} 🡪 Z è abbreviato in XY🡪Z

**Regole di inferenza fondamentali**

1. Regola riflessiva se X è un soprainsieme di Y, si ha che X🡪Y

X=Nome, Cognome Y=Nome

Possiamo dire che Nome, Cognome 🡪 Nome

Dette anche dipendenze funzionali banali



1. Regola incrementale

se X🡪Y se aggiungiamo gli stessi attributi ambi membri, varrà lo stesso la relazione.



1. Regola transitiva

Date due DF valide, per la transitività è valida anche X🡪Z

CF 🡪 Data ascia e Data Nascita 🡪 età **allora** CF 🡪 età



1. Regola di Decomposizione

Si può decomporre il lato dx della dipendenza

CF 🡪 Nome, Cognome si può dire che CF 🡪 Nome



1. Regola di Unione

Visto che a sx si ha lo stesso insieme di attributi, si possono unire i lati dx

CF 🡪 Nome e CF 🡪 Cognome allora CF 🡪 Nome, Cognome



1. Regola Pseudo transitiva

Mi devo portare indietro W per il ragionamento della transitività

CF🡪 data nascita , data nascita indirizzo 🡪 citta CF indirizzo 🡪citta

**Regole di inferenza di Armstrong**

È stato provato che le regole da 1 a 3 sono regole di inferenza corrette e complete

Corretta: applicandole a DF valide avrò sempre DF valide

Completa: la chiusura di F (l’insieme di tutte le possibili dipendenze inferibili da F) può essere determinata usando solo IR1 - IR3 (regole di inferenza di Armstrong)

Tipicamente prima si specificano le dipendenze funzionali F a partire dalla semantica degli attributi e poi usa le regole di inferenza (di Armstrong) per inferire dipendenze funzionali aggiuntive.

**Normalizzazione dei dati**

La normalizzazione dei dati può essere vista come un processo che consente di decomporre schemi di relazione non ottimali in schemi più piccoli, tali da garantire la mancanza di Anomalie.

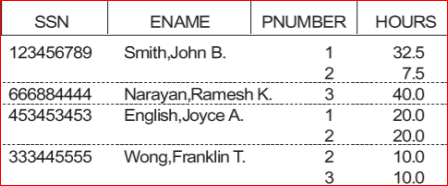
Le forme normali forniscono un metodo di analisi dello schema, basandosi sulle chiavi e sulle DF

Esistono varie forme normali le più importanti sono la 1,2,3; abbiamo poi la forma normale di BoyceCodd (BCNF), definita sulla base di 3NF.

*Prima forma normale* (**1NF**) : il suo obbiettivo è di non consentire attributi multi-valore, composti e loro combinazioni. L’attributo non può essere ulteriormente scomponibile.

Nominativo, non va bene, poiché può essere scomposto in (nome, cognome)

Indirizzo neanche poiché può essere ( indirizzo , CAP, citta, nazione)

Per SSN e ENAME abbiamo più istanze, non va bene poiché i valori delle singole istanze devono essere atomici. Ogni attributo deve avere un unico valore.

Per poter garantire la **1NF** 🡪 mettiamo tutti gli attributi atomici con la chiave assegnata.

Generiamo un'altra relazione in cui mettiamo la chiave dello schema generale, gli attributi che non avevano valore atomico e aggiungendo un'altra chiave riusciamo a rendere atomici questi valori. Funziona poiché la chiave ora sarà SSN,PNUMBER.

*Seconda Forma Normale* (**2NF**) È basata sul concetto di dipendenza funzionale piena - parziale.

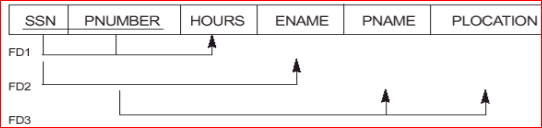
Definizione:

* Una dipendenza funzionale X → Y è **piena** se la rimozione di qualche attributo A da X implica che la dipendenza non vale più.
* Una dipendenza funzionale X → Y è **parziale** se qualche attributo A di X può essere rimosso e la dipendenza vale ancora.

ES: CF,NOME 🡪 COGNOME …. è parziale, se tolgo nome, vale ancora.

**Seconda Forma Normale** : Uno schema di relazione R è in 2NF se ogni attributo non primo A in R ha una dipendenza funzionale piena dalla chiave primaria di R.

*Non esiste una DF che dipendente parzialmente dalla chiave primaria dello schema di relazione*



DF1: hours (attributo non primo) dipende da SSN e PNUMBER(chiavi).

DF 2:dipendono parzialmente rispetto alla chiave primaria.

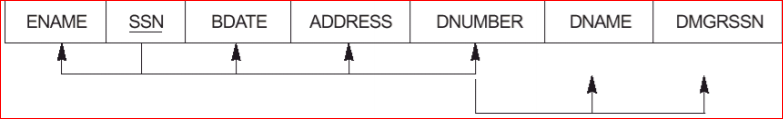
Non è in 2NF! Causa … DF2, DF3.

* Lo schema se non è in 2NF, può essere normalizzato in un certo numero di tabelle in 2NF, così che gli attributi non primi che dipendono parzialmente dalla chiave si trovino su un’altra relazione.

Terza Forma Normale (**3NF**) : È basata sul concetto di *dipendenza transitiva*

Una dipendenza funzionale X → Y in uno schema R è una dipendenza **transitiva** se esiste un insieme di attributi Z che non è sottoinsieme di alcuna chiave di R e valgono X → Z e Z→ Y.

Dato uno schema e definita la chiave. Per garantire la 3NF, vogliamo che non esistano dipendenze transitive rispetto alla chiave primaria

Non è in 3NF poiché è transitiva rispetto a SSN, poiché

SSN → DNUMBER e DNUMBER → DMGRSSN

DNUMBER non è sottoinsieme di alcuna chiave della relazione dell’esempio

**Terza Forma Normale** : uno schema di relazione R è in 3NF se è in 2NF e nessun attributo non primo di R è transitivamente dipendente dalla chiave primaria.

Come si rende in 3NF? Faccio un altro schema con DNUMBER,DNAME,DMGSSN.

**Definizione generale di 2NF e 3NF**

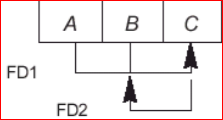
Uno schema R è in **2NF generale** se e solo se ogni attributo non primo A in R non è parzialmente dipendente da nessuna chiave di R, né primaria né candidata.

Uno schema di relazione R è in **3NF generale** se, ogni volta che vale in R una DF X → A

Si deve avere che X è una Superchiave di R oppure A è un attributo primo

**Forma Normale di BoyceCodd (BCNF)**

È derivata dalle 3NF generale

Uno schema R è in BCNF se per ogni DF X🡪A allora X è una Superchiave di R.

Domanda… è più restrittiva o meno?

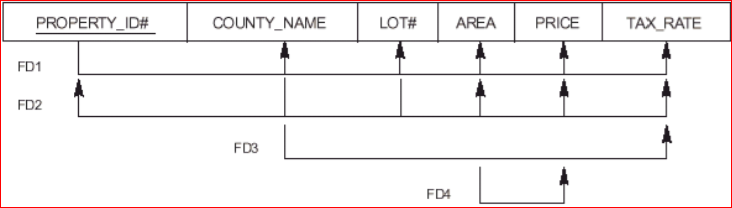
{A,B} chiave.

FD1{A,B} 🡪 C

FD2 C🡪B

È in 3NF poiché C dipende da una Superchiave (FD1), per la FD2, B è un attributo primo!

Non è in BCNF poiché non vale che X è Superchiave, dato che C non è Superchiave nella FD2



Chiavi candidate: PROPERTY\_ID# e {COUNTY\_NAME, LOT#}

Chiave primaria: PROPERTY\_ID#

Abbiamo : FD1,FD2, FD3: COUNTY\_NAME → TAX-RATE

FD4: AREA → PRICE

È in 2NF generale? No, poiché la FD3 ha un attributo non primo, TAX-RATE che dipende parzialmente dalla chiave candidata.

Per renderlo 2NF generale, si crea una nuova relazione con COUNTY\_NAME, TAX-RATE.

È in 3NF generale? FD1,FD2 valgono poiché la X (PROPERTY\_ID# ) è Superchiave, Key candidat.

FD4 vale? No, perché area non è Superchiave e price non è attributo primo.

Quindi creiamo una nuova relazione AREA,PRICE.

**Progettazione fisica**

**Strutture**

**SEQUENZIALI**: Esiste un ordinamento fra le ennuple

* *seriale*: ordinamento fisico ma non logico (sono ordinate sul disco, ma non sono ordinate in modo logico, così da leggerle sequenzialmente)
* *ordinata*: ogni tupla ha una posizione sul disco in base al valore di un campo detto chiave (ordinamento fisico con quello di un campo)

**SERIALE:** Gli inserimenti vengono effettuati: in coda o al posto di record cancellati.

La gestione è molto semplice, ma spesso inefficiente.

**SEQUENZIALE ORDINATA:** Ogni tupla ha una posizione su disco basata sul valore di un campo “Chiave”.

I dati venivano riposti in un *file principale*, le modifiche venivano raccolte in *file differenziali*. Tali file venivano periodicamente fusi

*Principali problemi*: inserimenti o modifiche aumentano lo spazio fisico utilizzato – richiedono riorganizzazioni periodiche. Effettuando queste operazioni, la dimensione del file differenziale accresce e aumenta lo spazio occupato.

**STRUTTURE ORDINATE :** Permettono ricerche binarie, ma solo fino ad un certo punto (come troviamo la "metà" del file su disco?). quindi si usano con gli indici.

**FILE HASH:** Permettono un accesso diretto molto efficiente

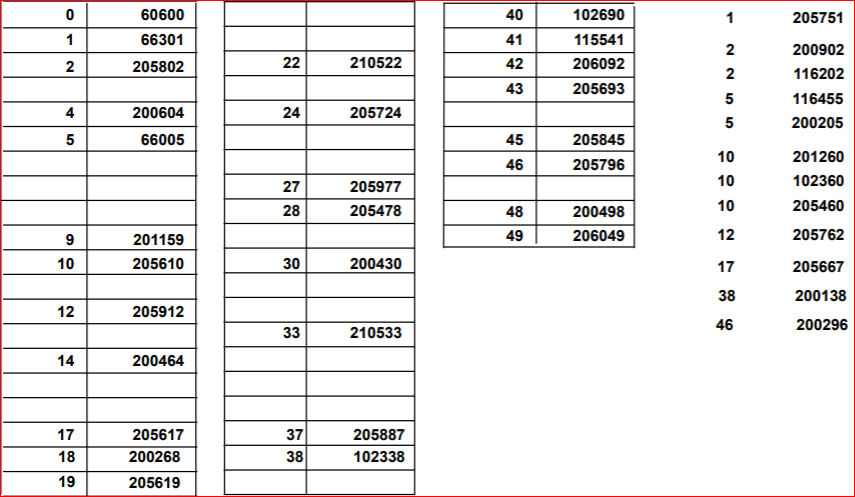
*Tavola Hash*: Obiettivo: accesso diretto ad un insieme di record sulla base del valore di un campo (supponendolo chiave).

Se i possibili valori della chiave sono in numero simile al numero di record allora usiamo un array.

Se i possibili valori della chiave sono molti di più di quelli effettivamente utilizzati e volendo continuare ad usare qualcosa di simile ad un array, ma senza sprecare spazio, possiamo pensare di trasformare i valori della chiave in possibili indici di un array:

* **Funzione Hash:** 
  + associa ad ogni valore della chiave un "indirizzo". Poiché il numero di possibili chiavi è maggiore del numero di possibili indirizzi, esiste la possibilità di collisioni (chiavi diverse che corrispondono allo stesso indirizzo).

Buone funzioni hash distribuiscono in modo casuale e uniforme, riducendo le probabilità di collisione (che si riduce aumentando lo spazio ridondante)



ESEMPIO

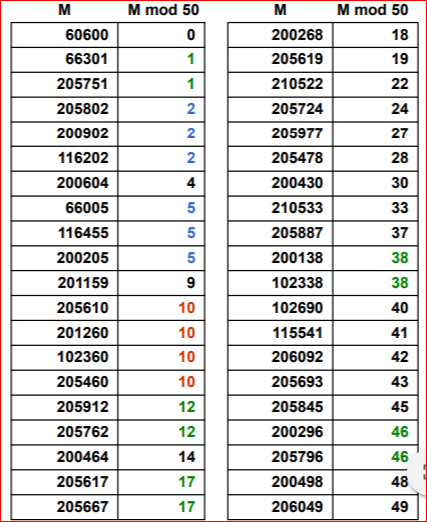
40 record

Tavola hash con 50 posizioni:

* 1 collisione a 4
* 2 collisioni a 3
* 5 collisioni a 2
* 20 record senza collisione

40 = 4+3\*2+5\*2+20

numero medio di accessi:

(28x1+8x2+3x3+1x4)/40 = 1,425

* Varie tecniche:

– posizioni successive disponibili

– tabella di overflow (gestita in forma collegata)

– utilizzo di più funzioni hash

Nota:

* le collisioni ci sono (quasi) sempre

al crescere della taglia delle collisioni decresce la probabilità che vi siano

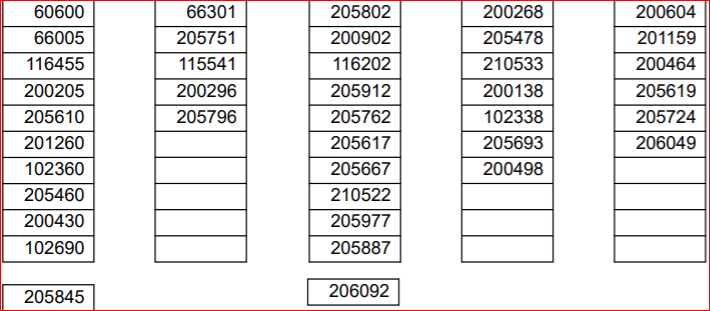
Abbiamo 12 record di overflow: nel caso 1 collisione a 4 il 1° ha trovato posto, gli altri 3 no.

2 collisioni a 3 : i primi 2 trovano posto, ne restano 4 ecc.

**File Hash:** si basa sull'organizzazione in blocchi dei dischi usando l’idea a della tavola hash

Ogni blocco contiene più record. Fattore di blocco: quanti record può contenere un blocco.

Possiamo usare “mod 5” 0 1 2 3 4

L’hashing lo faremo sui blocchi, utilizzeremo 5 indirizzi di blocco; e all’interno del blocco effettuiamo una ricerca sequenziale per trovare il blocco libero.

*ESEMPIO* 🡪 ogni blocco può contenere fino a 10 record, un record può andare in overflow, se quel blocco è già pieno, e si allocano al di fuori della tavola hash.

In media devo fa 5 accessi per quelli che trovano posto. Due soli record in overflow

Per quelli che non trovano posto, faccio 10 accessi, poiché li esamino tutti, ne faccio un altro in quello di overflow.

numero medio di accessi: (42/40) = 1,05

in 5 casi 1 accesso

5 volte 2 accessi

..

6 accessi li faccio 4 volte

7 accessi 3 volte

8/9/10/11 accessi 2 volte

**Collisioni, stima**

– Numero di record esistenti: T

– Numero di blocchi: B

– Fattore di blocco: F

– Coefficiente di riempimento: T/(F x B)

Al crescere di F, diminuisce la lunghezza media delle catene di overflow.

Se aumenta il coefficiente di riempimento abbiamo poco spazio ridondante a disposizione, aumenta la lunghezza media delle catene di overflow.

**File hash, osservazioni**Le collisioni (overflow) sono di solito gestite con blocchi a puntatori, è più efficiente per l'accesso diretto basato su valori della chiave con condizioni di uguaglianza.

È efficiente solo con file la cui dimensione non varia molto nel tempo, ma non quando lo spazio ridondante è poco e si esaurisce nel tempo.

**Indici di file**

Struttura per l'accesso ai record di un file sulla base dei valori di un campo detto chiave ma non è necessariamente identificante.

Un indice è un altro file, con record a due campi: 1)chiave, 2)puntatore ai record o ai blocchi del file principale, che sto indicizzando. Il file indice è ordinato secondo i valori della chiave

**Tipi di indice**

* *Indice primario*: è costruito su un campo rispetto al quale i record del file principale sono ordinati. Si può anche evitare di effettuare tutti i riferimenti, potevamo anche memorizzare un indirizzo per ogni blocco, così da avere un indice sparso (più facile da manutenere)
* *Indice secondario*: costruito su un campo con ordinamento diverso da quello del file principale; posso avere tanti indici secondari quanti sono gli attributi.
* *Indice denso*: il file indice contiene tutti i valori del campo chiave del file principale, quindi per indici su campi *identificanti*, un riferimento per ciascun record del file principale. Per *campi pseudo* chiave invece avremo per ogni valore mettiamo un record, ma ad ogni record dell’indice saranno associato più record del file principale, tutti quelli che condividono lo stesso valore del campo.
* *Indice sparso:* il file indice contiene solo alcuni valori della chiave anche per indici su campi identificanti, quindi si ha un numero di riferimenti inferiore rispetto ai record del file.

• Un indice primario di solito è sparso, invece se denso possiamo eseguire operazioni sugli indirizzi, senza accedere ai record

• Un indice secondario deve essere denso

• Si possono usare, come detto, puntatori ai blocchi oppure puntatori ai record

– I puntatori ai blocchi sono più compatti

– I puntatori ai record permettono di semplificare alcune operazioni (effettuate solo sull'indice, senza accedere al file se non quando indispensabile)

**Dimensioni dell'indice**

• L numero di record da memorizzare , del file principale

• B dimensione dei blocchi

• R lunghezza dei record (fissa)

• K lunghezza del campo pseudo chiave

• P lunghezza degli indirizzi (ai blocchi)

N. di blocchi per il file principale (circa): NF = L / (B/R)

B/R = num di record entrano in un blocco

N. di blocchi per un indice denso: ND = L / (B/(K+P))

La dimensione di un record del file indice è K+P Fattore di blocco = B/(K+P)

N. di blocchi per un indice sparso: NS = NF/ (B/(K+P))

Memorizziamo un record per ogni blocco. NF = Numero blocchi per memorizzare L record del file.

**ESEMPIO**

• L numero di record nel file 1.000.000

• B dimensione dei blocchi 4KB

• R lunghezza dei record (fissa per semplicità) 100B

• K lunghezza del campo chiave 4B • P lunghezza degli indirizzi (ai blocchi) 4B

NF = L / (B/R) = ~ 1.000.000/(4.000/100) = 25.000

ND = L / (B/(K+P)) = ~ 1.000.000/(4.000/8) = 2.000

NS = NF/ (B/(K+P)) = ~ 25.000/(4.000/8) = 50

**Caratteristiche degli indici**Accesso diretto (sulla chiave) efficiente , forniscono un ordinamento logico sui record del file

Modifiche della chiave, inserimenti, eliminazioni inefficienti, oltre all’overhead di spazio.

**Indici su campi non chiave**

Ci sono (in generale) più record per un valore della (pseudo)chiave.

Indice *primario sparso* con puntatori a blocchi solo a blocchi con valori “nuovi”

Indice *primario denso*:

* una coppia con valore della chiave e riferimento per ogni record (valori della chiave ripetuti)
* valore della chiave seguito dalla lista di riferimenti ai record con quel valore

Indice *secondario denso:* indice sul campo in cui il file principale non è ordinato

* una coppia con valore della chiave e riferimento per ogni record (valori della chiave ripetuti)
* per ogni valore della chiave l’indice contiene un record con riferimento al blocco di una struttura intermedia che contiene riferimenti ai record

**Indici multilivello**Gli indici sono file essi stessi e quindi ha senso costruire indici sugli indici, L’indice è ordinato e quindi l’indice sull’indice è primario. Il tutto a più livelli, fino ad avere un livello con un solo blocco. Usando indici sparsi, così da diminuire la grandezza delle tabelle.

I livelli sono di solito abbastanza pochi, perché l'indice è ordinato, i successivi saranno sparsi e i record dell'indice sono piccoli

Gli indici sono poco flessibili in presenza di dinamicità(cancellazioni/inserimenti)

**Albero binario di ricerca :** Per ogni nodo il sottoalbero sinistro contiene solo etichette minori di quella del nodo e il sottoalbero destro etichette maggiori.

**Albero di ricerca di ordine P**

Ogni nodo ha (fino a) P figli (puntatori) e (fino a) P-1 etichette (chiavi), in ordine crescente da sx/dx

In genere un nodo corrisponde di solito ad un blocco e quindi ogni nodo intermedio ha un numero di figli pari al fattore di blocco dell’indice (fan-out)

**B-TREE**

Albero di ricerca in cui ogni nodo corrisponde ad un blocco viene mantenuto perfettamente bilanciato (tutte le foglie sono allo stesso livello), come?

Cercando di riempirlo fino al 70% ed effettuando riorganizzazioni.

**Split e merge ( Inserimento ed eliminazione di tuple )**

Inserimenti ed eliminazioni sono precedute da una ricerca fino ad una foglia.

* inserimenti, se c'è posto nella foglia, ok, altrimenti il nodo va suddiviso, con necessità di un puntatore in più per il nodo genitore; se non c'è posto, si sale ancora, eventualmente fino alla radice. Il riempimento rimane sempre superiore al 50%
* le eliminazioni possono portare a riduzioni di nodi, se si va in under flow(50%) col nodo foglia, si unisce il nodo foglia con un altro vicino. Ciò comporta la cancellazione della coppia chiave-puntatore dal padre; ciò può propagarsi di conseguenza fino alla radice.

**Esempio 1**

Calcolo dell’ordine p di un B-Tree su disco:

– Campo chiave V =9 byte

– Dimensione blocchi su disco B = 512 byte

– Puntatore a record di Pr = 7byte

– Puntatore a blocco di P = 6byte

Ogni nodo del B-Tree, contenuto in un blocco del disco, può avere al più p puntatori ad albero, p-1 puntatori a record e p-1 valori del campo chiave di ricerca.

(p\*P)+((p-1)\*(Pr + V)) <= B

p\*P per memorizzare i puntatori a blocco prendo Pr perché è più grande (caso pessimo)

(p\*6)+((p-1)\*(7+9)) ≤ 512 🡪 (22 \* p) ≤ 528 🡪 p ≤ 24

Quindi avrò che p è al massimo 24, non posso fare un albero di grado superiore a 24. Scelgo 23.

**Esempio 2**Calcolo del numero di blocchi in un B-Tree. Supponiamo che il campo di ricerca su cui il file principale non sia ordinato, ma che sia chiave. Supponiamo che ogni nodo sia pieno per il 69%. (massimo a 70% si arriva).

Ogni nodo avrà in media p \*0.69 = 23 \* 0.69 = circa 16 puntatori a sottoalbero (fan out medio, numero di record/puntatori che può avere un nodo) e quindi 15 valori del campo chiave di ricerca.

–Radice: 1 nodo 15 entry = (valori del campo chiave) 16 puntatori a sottoalbero

– livello 1: 16 nodi 240(16\*15)entry 256 (16\*16) puntatori

– livello 2: 256 nodi 3840(256\*15) entry 4096 puntatori

– livello 3: 4096 nodi 61440 (4096 \*15) entry ogni nodo ha 16 puntatori

Per ogni livello:

– # entrate livello i esimo = [# puntatori livello (i-1)-esimo]\*15

**B tree e B+ tree**

• B+ tree:

le chiavi compaiono tutte nelle foglie .I puntatori a record del file principale sono contenuti nei nodi foglia. (e quindi quelle nei nodi intermedi sono comunque ripetute nelle foglie).

Le foglie sono collegate in una lista. Ottimi per le ricerche su intervalli. Usati nei DBMS

*B+ tree primario* 🡪le ennuple possono essere contenute nelle foglie

*B+ tree secondario* 🡪 le foglie contengono puntatori alle ennuple del file principale

• B tree: Le chiavi che compaiono nei nodi intermedi non sono ripetute nelle foglie

Anche i nodi intermedi contengono ennuple

**Esempio 2**

*Calcolo dell’ordine p di un B+ -Tree su disco:*

– Campo di ricerca di V =9 byte

– Dimensione blocchi su disco B=512 byte

– Puntatore a record di Pr = 7 byte

– Puntatore a blocco di P = 6 byte

Un nodo interno di un B+ Tree, contenuto in un singolo blocco, può avere fino a p puntatori ad albero e p-1 valori del campo di ricerca; il puntatore a record lo troverò nella foglia, con la chiave.

p = Fan out dell’albero.

(p\*P)+((p-1)\*V) ≤ B 🡪 (p\*6)+((p-1)\*9) ≤ 512 🡪 (15 \* p) ≤ 521

Il massimo intero che soddisfa Ia disuguaglianza è p=34, che è maggiore di 23 (ordine del B-Tree precedente). Quindi ne risulta un fan-out maggiore e più entrate in ogni nodo interno.

Ordine dei nodi foglia. L’ordine Pleaf dei nodi foglia è:

(Pleaf \* (Pr + V)) + P (Pnext) ≤ B 🡪 (Pleaf \* (7+9)) + 6 ≤ 512 🡪 (Pleaf \* 16) ≤ 506 🡪 Pleaf = 31

Quindi ogni nodo foglia può tenere al massimo 31 combinazioni dei valori della chiave e puntatore a record. Abbiamo un B+ Tree secondario, dove memorizzo i puntatori a record e non i record

Pnext : puntatore al blocco che contiene la foglia successiva.

*Calcolo del numero di blocchi e di livelli in un B+ Tree*

Supponiamo che ogni nodo dell’albero sia pieno per il 69%.

* Ogni nodo interno avrà in media p \* 0.69 = 34 \* 0.69 = circa 23 puntatori (fan-out medio) e quindi 22 valori del campo di ricerca.
* Ogni nodo foglia avrà in media Pleaf\*0.69 = 31\*0.69 = circa 21 puntatori a record dati.

Partendo dalla radice vediamo quanti valori e puntatori esistono in media a ogni livello:

Radice : 1 nodo 22 entry 23 puntatori

Livello 1: 23 nodi 506 (22\*23) entry 529 (23\*23) puntatori

Livello 2: 529 nodi 11638 (22\*529) entry 12167 puntatori

Livello foglie: 12167 nodi 255507(21\*12167) puntatori a record.

**Inserimento in un B+ Tree**

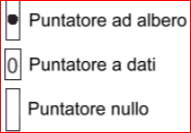
Inizialmente Ia radice è l’unico nodo dell’albero: essendo quindi un nodo foglia conterrà anche i puntatori ai dati.

Se si cerca di inserire un entry in un nodo foglia pieno, esso va in overflow e deve essere scisso:

Le prime \*j= [(pleaf + 1)/2]\* entry sono mantenute nel nodo, mentre le rimanenti sono spostate in un nuovo nodo foglia.

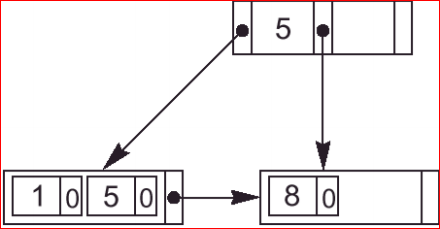
– Il j-esimo valore di ricerca (quello mediano) è replicato nel nodo padre, Nel padre viene creato un puntatore al nuovo nodo.

Potrà capitare che il padre vada in overflow, e si scinde il nodo e si procede come prima.

Esempio di inserimento Vogliamo inserire dei record in un B+ -Tree di ordine p = 3 e Pleaf =2,

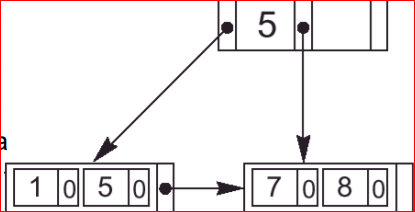
nella sequenza 8, 5, 1, 7, 3, 12.

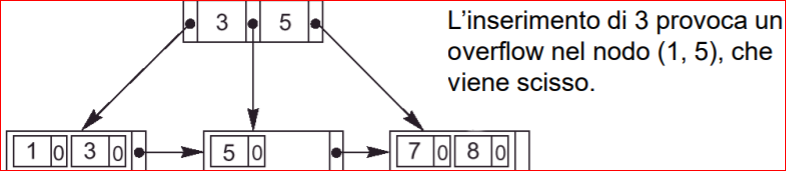


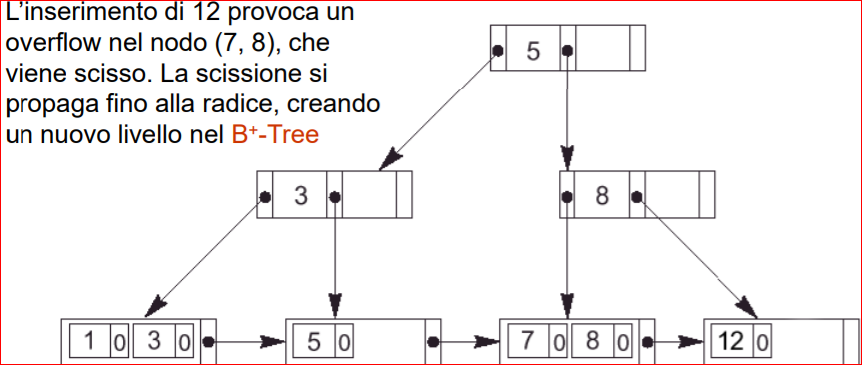
L’inserimento dei valori 8 e 5 non provoca overflow: sono entrambi inseriti nella radice

L’inserimento di 1 provoca overflow poiché p=3 poiché , in ogni nodo ci possono essere 2 valori.

Il nodo è scisso ed il valore mediano è ripetuto in un nuovo nodo radice. Data la regola \*

L’inserimento di 7 non provoca overflow. Si noti che tutti i valori sono a livello foglia, perché i data pointer sono tutti a quel livello. Alcuni sono replicati nei nodi interni per guidare la ricerca





Creo un altro nodo col 12, provo ad inserire l’8 nel padre, ma ho overflow e creo un nodo intermedio, (2+1)/2,

**Cancellazione in un B+ -Tree**

Un entry è cancellata sempre a livello foglie. Se essa ricorre anche in un nodo interno, allora è rimossa anche da quello e al suo posto si inserisce il valore immediatamente alla sinistra del valore rimosso nel nodo foglia. La cancellazione di valori può causare l’under flow di un nodo.

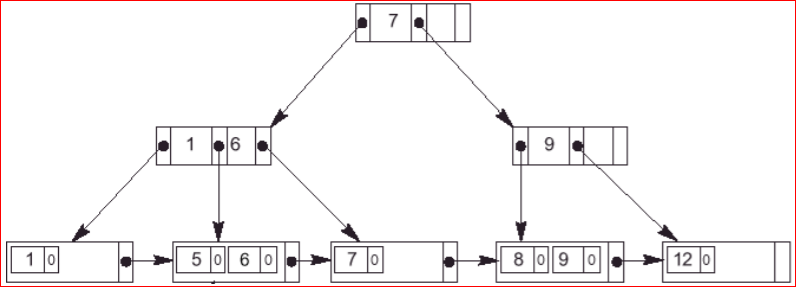
• Approccio comune:

– Si tenta di ridistribuire le entrate con il fratello sinistro;

– Se non è possibile, si tenta di ridistribuire con il fratello destro

– Altrimenti si fondono i tre nodi in due nodi foglia. L'under flow in questo caso si può trasmettere ai nodi interni, poiché sono necessari un puntatore ad albero ed un valore di ricerca in meno. Se la propagazione arriva alla radice, si riduce il numero di livelli dell’albero

Dato il seguente B+ -Tree, vogliamo cancellare i record 5, 12 e 9.

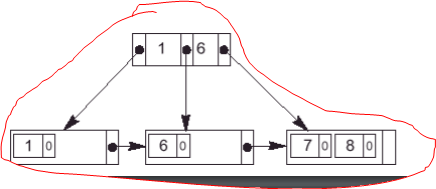


La cancellazione di 5 non pone alcun problema

La cancellazione di 12 viene risolta con una ridistribuzione col nodo sx

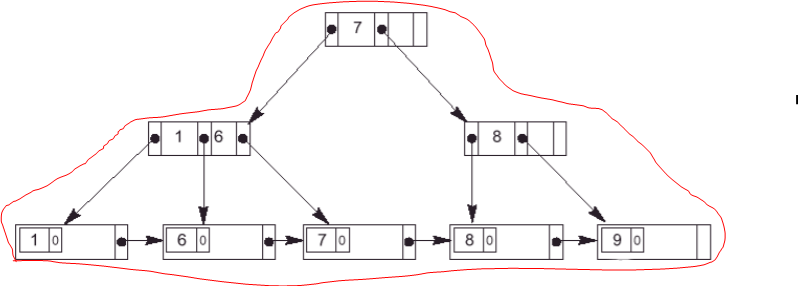
Ho under flow, vado nel nodo di sinistra prendo il 8 e lo metto nel nodo dove ho avuto under flow.

Cambia anche il valore del padre

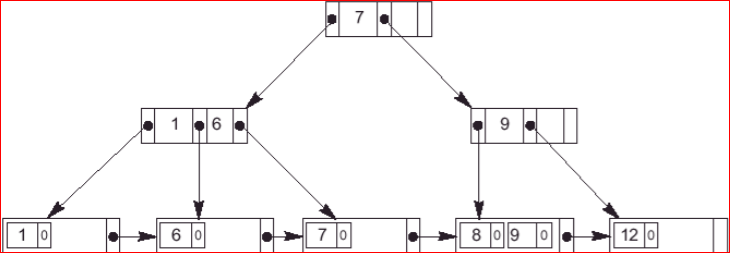


La cancellazione di 9 causa un under flow. Il merge con il fratello sinistro causa un altro under flow ,fondo 7 e 8, che porta ad una riduzione del numero di livelli, dato che cancello anche il padre

la



Progettazione fisica è la fase finale del processo di progettazione di basi di dati;

* in input c’è lo schema logico e informazioni sul carico applicativo
* in output restituisce schema fisico, costituito dalle definizione delle relazioni con le relative strutture fisiche ( indici, alberi bilanciati )