# Gestione dei Big Data

Matteo Aprile
Professore: Marco Zappatore, Antonella Longo

## INDICE



## I. LIBRI DI TESTO CONSIGLIATI

- Fundamental of Database Systems, 7th ed, Elmasri, Navathe
- Data Warehaouse Design, Rizzi, Golfarelli big dataL consepts technology and architecture 1st ed balusamy abirami gadomi

## II. Databases

### A. Definizioni di base

Le definizioni di base da sapere sono:

- dato: **insieme di fatti conosciuti, registrati e con un significato**. È detto dato grezzo visto che si suppone che andrò ad elaborarlo, questo dato **sara' poi archiviato**, sarà un **fatto conosciuto** cioè avremo:
  - **eventi con un significato** per un dato tipologia di utenti
  - sorgente che **produce i dati** con una cerca velocità
- DataBase: **raccolta di dati altamente organizzati, intercorrelati e strutturati**. È una struttura con dei collegamenti strutturati tra i dati
- DBMS Data Base Managment System: insieme di **programmi per accedere ai dati e farci delle operazioni** di 4 tipi: creazione, recupero, aggiornamento e cancellazione, ciclo **CRUD**. Ne favorisce anche il mantenimento.
- mini-world: **parte del mondo reale alla quale si riferiscono i dati presi** andando a **limitare la modellazione** in un numero n di concetti
- DataBase System: insieme di DBMS con i dati
- astrazione: **separare i dati dai collegamenti tra le entità per disporle in un modello** senza che esso si occupi di come salvare i dati
- modello concettuale: formato da entità e relaizoni
- modello fisico: definizione dei tipi dato e dove sono **conservati**
- controllo della concorrenza: garantire che tutte le transazioni sono **correttamente eseguite**
- recovery: se la transazione è stata eseguita è stata conservata nel database

### B. Tipologie di DB

Esistono molti tipi di DB:

- numerici o testuali
- multimediali
- Geographic Information Systems (GIS)
- Data Warehouses

### C. Ciclo di vita del DB

È opportuno vedere un concetto di base dei dati, cioè il loro ciclo di vita. Il più semplice è:

1) **acquisizione** (scattered data)
2) **aggregazione** (integrated data)
3) **analisi** (knowledge)
4) finisce in un **applicazione** che genera dei "log data" che saranno poi acquisiti come scattered data

Da un punto di vista computazionale queste fasi si devono prendere in un altro modo:

1) storage dei data
2) formattazione e pulizia
3) capire cosa dicono i dati
?) se non mi bastano i dati che ho posso integrare dei dati

### D. Livelli di un DB

Quando si ha un DB abbiamo 3 livelli da considerare

1) fisico: dove sono **salvati i dati**
2) logico: indica come i dati sono **collegati tra loro**
3) view: **rappresentazione** che sarà diversa per ogni tipo di utente

### E. Data Base Managment System

Un DBMS offre l'opportunità di:

- **salvataggio** dei dati
- **definizione** modelli dati
- **manipolazione** dei dati
- **processare** e condividere i dati

Per quanto riguarda l'interazione con i DB avremo 2 strumenti:

- query: accede a parti differenti di dati e formula una richiesta
- transazioni: legge dei dati ed aggiorna alcuni valori, salvandoli nel DB

### F. Mini-world

Avremo bisogno di identificare delle entita', cioè i **concetti di base** che rappresentano una parte delle cose che inseriremo nel DB relazionale. Poi andremo a connettere tra loro le entita', dette relazioni (relationships) (ER), ne derivano delle tabelle dette relation.

Il tutto da derivare dai requisiti e non dall'esperienza personale.

Le tabelle create dalle entità conterranno i dati che ho a disposizione. Saranno divisi in:

- righe (record)
- colonne (attributi)
- celle (dati grezzi)

Si verrà quindi a creare un catalogo con vincoli, tipo di dati e la relazione di appartenenza degli attributi.

## III. Database System Concepts and Architecture

### A. Definizioni sui modelli

Le definizioni di base da sapere sono:

- Data Model: insieme di **concetti che descrivono struttura, operazioni e vincoli** applicati al DB
- Data Model Structure and Constraints: abbiamo dei **costrutti che definiscono come collegare gli elementi** definiti da: entità, record e tabella
- Data Model Operation: di base (**CRUD**) o definite dall'utente
- modello dal concettuale: di **alto livello** e semantico
- modello fisico: di basso livello, definisce **come i dati sono salvati**
- modello implementativo: usati nel DBMS
- modello autodescrivente: basati su XML

### B. Definizioni fondamentali

- DataBase schema: descrizione del database in termini di struttura, tipo dati e vincoli
- schema diagram: **visione rappresentativa** del DB schema

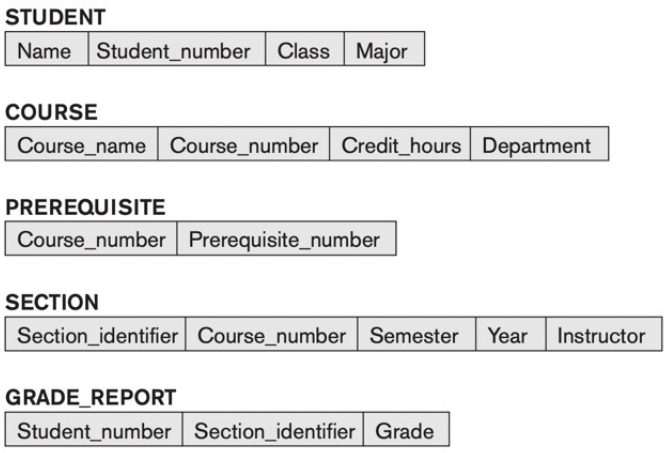



Figura 1. Schema diagram

- schema construct: insieme tra **schema e dati** dei DB
- database state: **snapshot** in istante t del DB, si definisce quindi ai suoi contenuti
- valid state: si definisce funzionante se il suo contenuto **soddisfa i vincoli** per quello schema
- data dictioraty: insieme per salvare schema e altre info

### C. Schema

Possiamo avere 3 livelli di schema:

1) **interno (fisico)**: come i dati devono essere salvati e come posso accederci
2) **concettuale**
3) **esterno**: per descrivere le view dell'utente

Per passare da uno schema ad un altro ho bisogno di un mapping per capire a cosa corrisponde un elemento. Avremo:

- logic data independence: se voglio **cambiare lo schema concettuale** senza cambiare quello fisico
- physical: devo **cambiare lo schema fisico** senza cambiare quello concettuale

### D. Tipologie di DBMS

Possiamo avere più tipologie di DBMS:

- centralized: dove abbiamo tutta l'**elaborazione su un unico nodo**
- 2-tier: si specializza in termini di server per ogni blocco di funzionalità che devo offrire
- cliets: per far accedere gli utenti
- DBMS server: per eseguiire query e transazioni tramite API

# IV. Data Modeling Using the Entity–Relationship (ER) Model

## A. Entity-Relationship (ER)

Partendo dal mini-world serve capire i requisiti utili. Bisognerà far gestire, all'applicazione, alcuni dati per poi visualizzarli (requisiti relazionali).

La procedura sarà:

1) acquisizione dei data requirements
2) conversione in un modello concettuale
3) applicazione dell'algoritmo di mapping
4) DBMS si occupa di physics design ed internal schema

in parallelo avremo la gestione delle transazioni del mini-world estraendo i functional requirements per effettuare una functional analysis che genera delle transazioni ad alto livello.

Per la scelta degli elementi avremo:

- entita' (sostantivi): **oggetti o cose specifiche** presenti nel mini-world che bisogna rappresentare
- relazioni (verbi): **collegano le entita'**. Il **grado di tipo** della relazione è il **numero di partecipanti a quella relazione**, identificando quante volte la relazione viene percorsa.
  Può:
  - essere **ricorsiva** se si riferisce ad una stessa entità
  - avere un suo attributo definito dall'azione che sta compiendo
- attributi (proprieta'): **descrittori** per ogni entità
- record: **insieme degli attributi** che si danno ad un entità
- dato singolo: ha un unico valore
- dato composto: dati da un **insieme di più descrittori**, notazione: ...(... , ...)
- dato multivalore: attributi che hanno **n-uple di valori**, notazione: ...
- attributo chiave: **identificare univocamente tutti i record**. Si può usare anche un'unione tra attributo chiave e un altro attributo
- entita' debole: entità che **da sola non può esistere**, quindi dipende da un entità più forte. Le sue relationship saranno deboli anche esse. Questa entità **non ha un attributo chiave** ma ha almeno un **attributo in comune con l'entità forte**.
- vincoli: ci sono dei concetti che fungano da vincoli
  - **impliciti**: come è definito il modello dati (es: non posso avere una lista come valore di un attiributo, allora userò n colonne per quanti sono i possibili numeri di telefono)
  - **espliciti**: aggiunti dal modellista (es: cardinalità min max)
  - **semantici**: vincoli aggiunti dal programmatore che farà l'applicativo sul quale si base il nostro db (es: la psw deve avere un tot di caratteri e non altri)

Piccoli accorgimenti da avere:

- scritto da **sx a dx** e dall'altro verso il basso
- nomi delle **entita' al singolare**
- **verbi alla terza persona** e attivi o passi per capire da che parte si deve leggere la relazione
- per la **carcinalita'** mi chiedo per un solo elemento quante entità puotrà avere dell'altro a cui è relazionato. Può essere rappresentata tramite:
  - **vincoli di dipendenza esistenziale**: 1:1, 1:N, M:N dove bisogna mettere la cardinalità nel lato opposto
  - **min max**: dico che posso avere da un min a un max di record che percorrono la relazione dando in vincolo di intervallo (sarà di aiuto a chi farà il database quando dovrà gestire un warning)

I database NoSQL saranno esenti da una modellazione così pesante.

Potremmo incombere in relazioni di livello piu' alto nel caso in cui ci trovassimo a descrivere relazioni con complessità alto. In gnerale si cerca di evitare e di farlo con relazioni binarie per evitare complicazioni nell'implementazione.

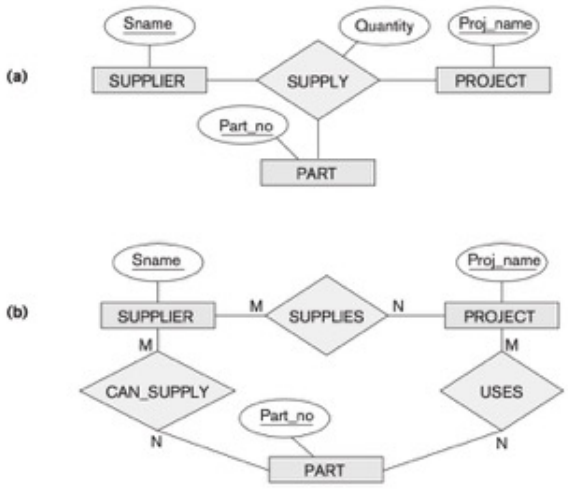



Figura 2. Relazione di livello alto

# V. The Enhanced Entity–Relationship (EER) Model

## A. Superclassi e sottoclassi

L'idea è di andare a creare una gerarchia:

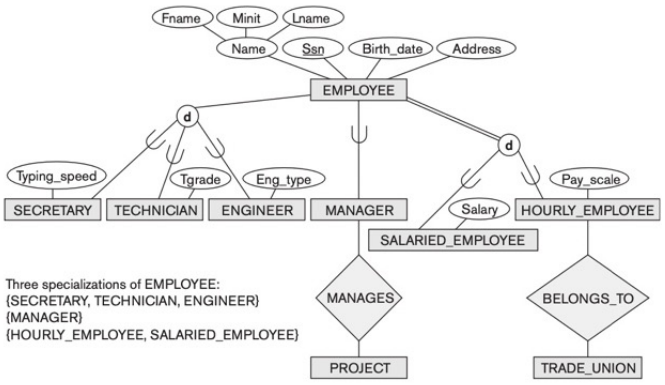



Figura 3. Gerarchia con disjoint

Per modellarlo mi chiedo quali siano le caratteristiche che hanno in comune alcune entita', allora tutti gli attributi in comune vanno nella superclasse. Ogni entita' DEVE avere i suoi attributi specifici ma non ho un attributo chiave dato che viene preso dalla superclasse.

## B. Graficazione superclassi e sottoclassi

La graficazione avrà per:

- specializzazione diretta: si ha un segmento
- gerarchia (IS-A): si ha un segmento con un nodo con:
  - **d -> disjoint**: NON POSSO avere un'entità che è contemporaneamente due o più sottoentità (**solo una**)
  - **o -> overlap**: posso avere un'entità che è contemporaneamente due o più sottoentità (**almeno una**)
  - **U - > union**: **raggruppa** entità di tipo diverso

le quali potranno avere partecipazioni totali o parziali che indicano se la superclasse deve o meno scegliere tra le sottoclassi.

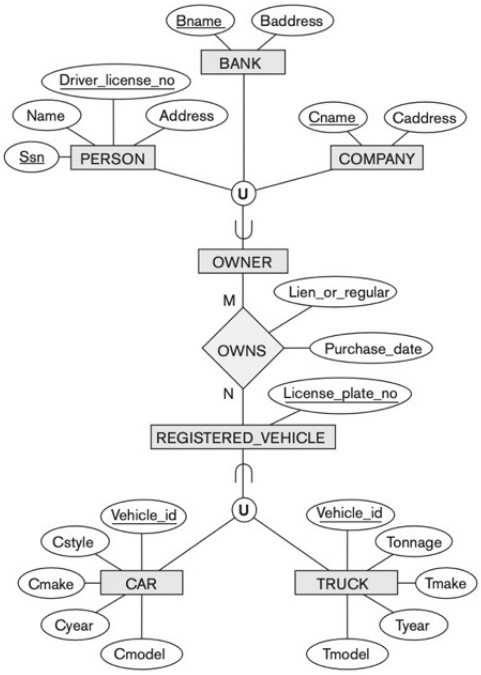



Figura 4. Gerarchia con disjoint

Il motivo della modellazione è la presenza di alcune fasi per i sistemi di gestione delle informazioni:

- studio di fattibilità
- analisi dei requisiti
- modellazione e design
- prototipo (ciclico)
- implementazione

Per i relazionali le fasi sono:

- application requirements
- modello concettuale
- modello logico
- modello fisico

## C. Notazioni

Nella creazione del modello usiamo la notazione:

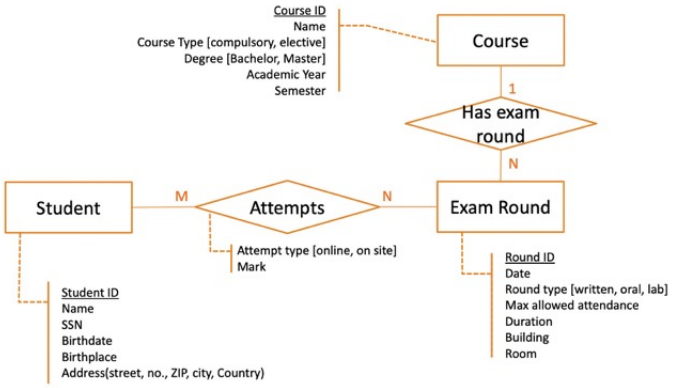



Figura 5. Notazione dei diagrammi ER

se un attributo puo' avere solo un numero finito di valori si usa:

...[..., ...]

Se ho bisogno di sostituire una connessione logica con un entita' la chiamo: reificazione. La si usa se si ha la necessita' di creare un entita' sulla quale si baseranno altre relationship. Se sbaglio il verso delle relationship metto una freccia.

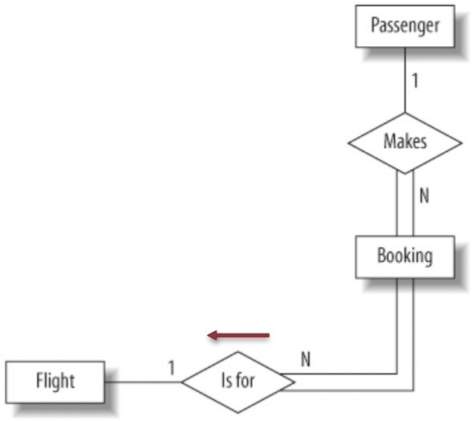


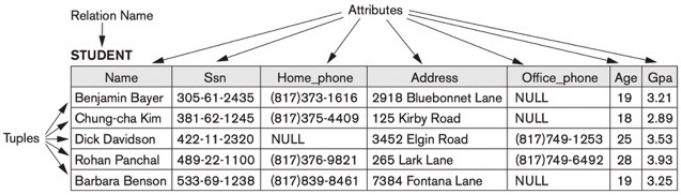

Figura 6. Reificazione ed orientamento della relationship

conviene usare delle relation con un nome univoco.

### D. Terminologia

| termine informale | termine formale |
|---|---|
| table<br>column header<br>all possible column values<br>row | relation<br>attribute<br>domain<br>tuple "¡...¿" |
| table definition<br>populated table | schema of a relation<br>state of the relation |

Tabella I
TABELLA DELLE ORE DI LAVORO




| Name | Ssn | Home_phone | Address | Office_phone | Age | Gpa |
|---|---|---|---|---|---|---|
| Benjamin Bayer | 305-61-2435 | (817)373-1616 | 2918 Bluebonnet Lane | NULL | 19 | 3.21 |
| Chung-cha Kim | 381-62-1245 | (817)375-4409 | 125 Kirby Road | NULL | 18 | 2.89 |
| Dick Davidson | 422-11-2320 | NULL | 3452 Elgin Road | (817)749-1253 | 25 | 3.53 |
| Rohan Panchal | 489-22-1100 | (817)376-9821 | 265 Lark Lane | (817)749-6492 | 28 | 3.93 |
| Barbara Benson | 533-69-1238 | (817)839-8461 | 7384 Fontana Lane | NULL | 19 | 3.25 |

Figura 7. termini formali per le tabelle

## VI. BASIC SQL

SQL è un linguaggio che consente di accedere al db in varie modalità ed ha la funzione di creare e gestire i db.

### A. Statement - SELECT

Usato per recuperare informazioni dal db. la sua struttura ha 3 clausole (claude):

```
SELECT <attribute list>
FROM <table list>
[ WHERE <condition> ]
[ ORDER BY <attribute list> ];
```

Molto utile usare gli alias (AS) per:

- andare a **definire i campi che ci serviranno in modo da dividere gli attributi di una tabella con quelli di un altra**
- **accedere ad una stessa tabella ma con 2 alias diversi** perchè per esempio uno rappresenta l'impiegato e l'altro il supervisore
- **rinominare gli attributi**:

```
EMPLOYEE AS E(Fn, Mi, ...)
```

Keyword da poter usare:

- **DISTINCT**: restituisce solo valori distinti (diversi) nel set di risultati

### B. Statement - WHERE

Esprime una condizione, se manca è possibile fare il prodotto cartesiano se si usa:

```
SELECT Ssn, Dname
FROM EMPLOYEE, DEPARTMENT
```

Si possono usare delle condizioni di tipo:

- numerico:

```
WHERE Dno = 5
```

- pattern matching tra stringhe:

```
WHERE Ssn LIKE "yes"
```

se non è un occorrenza esatta usiamo:

  - %: indica una qualsiasi sottostringa
  - _: indica un solo carattere in una specifica posizione

```
WHERE column IN (SELECT Statement)
```

vado a selezionare da Statement gli attributi column

```
WHERE column BETWEEN value1 AND value2
```

vado a selezionare i valori tra value1 e value2 gli attributi column

### C. Statement - ORDER BY

Per ordinare i risultati con DESC o ASC.

### D. Statement - INSERT

Usata per inserire dei nuovi dati nei DB.

### *E. Statement - GROUP BY*

Usato per raggruppare uno o più attributi in base ad un certo valore.

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

### *F. Statement - HAVING*

Usata come un WHERE, quindi filtro, ma usando delle funzioni aggregate

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

### *G. Statement - UNION*

Usato per combinare il risultato di 2 o più statement SELECT

```
SELECT City FROM Customers
UNION
SELECT City FROM Suppliers
ORDER BY City;
```

### *H. Statement - JOIN*

Usata per unire più risultati di operazioni disseminate tra più tabelle.

```
SELECT Orders.OrderID, Customers.CustomerName,
    Orders.OrderDate
FROM Orders
JOIN Customers
ON Orders.CustomerID=Customers.CustomerID;
```

### *I. Values - NULL*

Rappresenta un valore nullo che può dare problemi nel caso di machine learning o raccoglimento di dati.

Usandolo come statement:

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

### *J. Function - Aggregate*

Eseguono un operazioe su n valori dati e possono essere:

- COUNT
- AVG
- SUM
- MAX
- MIN

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

### *K. Function - Window*

- LAG
- LEAD

### *L. Function - String*

- CONCAT
- LEN
- UPPER
- LOWER

## VII. Distributed Database Concepts

### A. Distributed Databases

I dati utilizzati nelle infrastrutture dei big data devono essere ACID. Queste infrastrutture sono composte da nodi che collaborano per compiere un task. In queste infrastrutture andremo a distribuire le risorse sui nodi che cooperano in modo da avere ridondanza di dati.

Esiste una relazione logica tra questi database connessi, ma non tutti i nodi devono essere omogenei quindi possiamo al concetto di DISTRIBUTED DBMS che deve gestire l'avere modelli dati connessi.

Per quanto riguarda le query bisognera' riorganizzarle per poter gestire i nodi distribuiti.

### B. Forme di trasparenza

Abbiamo varie forme di trasparenza rispetto all'utente:

- organizzazione dei dati:
  - local transparency
  - naming transparency
- ridondanza: usata per ridurre la mancanza del servizio
- frammentazione dei dati:
  - **partizione orizzontale**: abbiamo una partizione delle tuple. La struttura dati è la stessa ma cambiano i dati salvati

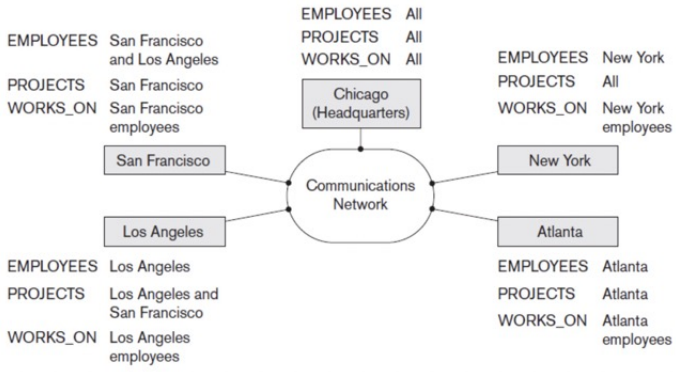



Figura 8. Partizione orizzontale di un db

  - **partizione verticale**: frammentazione del modello dati attraverso la partizione degli attributi che possono essere dislocati su più nodi

### C. Affidabilità e disponibilità

Per poter rispettare affidabilità e disponibilità usiamo la ridondanza.

Eseguiamo una distribuzione dei dati per avere la possibilita' di scalare i dati in modo orizzontale o verticale a seconda dell'uso che dobbiamo farne. Parliamo di:

- horizzontal scalability: poter **espandere il numero di nodi** del sistema
- vertical scalability: poter espandere la **capacità di un singolo nodo**
- partition tolerance: il sistema è **capace di operare anche se partizionato**

### D. Autonomia

La caratteristica principale della distribuzione è l'autonomia di ogni nodo rispetto agli altri, quindi lo deve essere anche il loro contenuto, tenendo conto di:

- design autonomy: rappresenta indipendenza dei modelli dei dati e la **gestione delle transazioni tra i nodi**
- communication autonomy: dà la possibilità di condividere le informazioni
- execution autonomy: mi assicura che posso **eseguire una query**

Il vantaggio delle architetture distribuite è una migliore facilità di sviluppo, disponibilità e performance.

### E. Frammenti

I frammenti sono le unita' logiche dei database e possono essere:

- frammentazione orizzontale: divide le relazioni in orizzontale (**tuple**)
- frammentazione verticale: divide le relazioni in verticale (**colonne**)
- frammentazione orizzontale completa: se tramite **union** possiamo ricostruire la nostra lezione completamente
- frammentazione verticale completa: usiamo la **join**

Spesso nella ricostruzione di un db possiamo trovare sia frammentazioni verticali che orizzontali, parliamo allora di frammentazione dello schema.

Un DDBMS è in grado di gestire una frammentazione usando un ALLOCATION SCHEMA che specifica come i frammenti del db sono distribuiti su quali nodi.

Quando abbiamo dei dati possiamo avere la necessità di ridondanza. Questo non è il metodo migliore per chi si occupa di grandi moli di dati, infatti bisognerà fare delle repliche a caldo che ogni tempo t su un altro sito. In questi casi ci viene in soccorso il database journal che tiene traccia dei log/modifiche fatte al db, andando a prevenire eventuali errori dato che si potrà accedere ad una versione immediatamente precedente dei dati.

Potremo avere delle repliche totali o parziali, replicando struttura e dati a secondo dello schema di replica presente del DDBMS.

### F. Problemi per DDBMS

Potrebbero essere:

- eccessive copie dei dati
- two phase commit: dove un nodo quando ne aggiorna un altro ha la possibilità che fallisca e gli sia mandato un ACK -1, in questo casi bisognerà rincominciare tutto dall'inizio

Nella gestione delle ridondanza si utilizza un sito primario che può essere sostituito, nel caso di malfunzionamenti, da uno secondario, andando a gestire le tabelle in modo parallelo. Per capire chi sarà il nuovo responsabile del locking si usa un sistema a voto.

Un sistema distribuito 'e un sistema che ha risorse di elaborazione che possono essere dislocati su nodi che potrebbero

anche essere geograficamente distanti. La relazione che esiste tra questi diversi nodi ‘e di tipo logico, tutti questi sono omoge- nei. Il principio principale ‘e che l'utente finale, che pu‘o essere anche un programmatore, non deve accorgersi che il database ‘e di tipo distribuito. La trasparenza si puo‘ applicare sia sotto il punto di vista della locazione fisica che anche dal punto di vista del naming delle tabelle, ovvero non c'‘e bisogno che io abbia chiamato le tabelle nello stesso modo. Altre propriet‘a importanti sono la Replication Transparency che permette di replicare i dati nel caso di disservizio, l'utente finale ovvia- mente non si rendera‘ conto di questo processo. Anche la Fragmentation Transparency ‘e un metodo di frammentazio- ne che puo‘ essere orizzontale se abbiamo su due database diversi la stessa struttura dati ma differenti righe e verticale se disponiamo di due strutture dati differenti ma in relazione logica tra loro (come se stessimo tagliando verticalmente la tabella).

img

Questo esempio rappresenta una frammentazione orizzontale (sharding),parliamo di frammentazione orizzontale completa se siamo in grado attraverso l'operazione di union di riavere la relazione iniziale completa (in generale per lavorare sulle colonne utilizziamo l'operazione di join, sulle righe la union). I motivi per cui si vuole distribuire un database sono differenti, dobbiamo essere in grado di fornire avaiability e Raliability ovvero delle caratteristiche relative alla disponibilit‘a del dato ed alla robustezza della struttura oltre che per questioni di scalabilit‘a. Si parla di Partition Tolerance quando il sistema ‘e in grado di operare nonostante sia partizionato. Questo ‘e derivato direttamente dal cut teorem, questo afferma che un sistema partizionato puo‘ funzionare come se non lo fosse ma non pu‘o garantire consistenza e la disponibilit‘a insieme, a differenza dei sistemi non partizionati che utilizzano un sistema ACID: Atomicit‘a , consistenza, integrit‘a, durabilit‘a dei dati, propriet‘a disponibili facilmente in un database relazionale non distribuito. Altra caratteristica importante in un sistema frammentata ‘e quella che assicura che un nodo pu‘o decidere autonomamente se condividere in suo dato con altri nodi. In un sistema frammentato ogni tabella dovra‘ necessariamente avere la sua chiave primaria e queste chiavi primarie saranno sfruttate come parametro per effettuare una join. La frammentazione viene gestita attraverso quello che viene chiamato allocation schema e ti fornisce informazioni riguardanti la posizione dei frammenti sui vari nodi. Un altro fattore importante ‘e la replicabilita‘ del sistema, quindi un sistema frammentato deve essere facilmente ricostruibile. Altro meccanismo importante ‘e quello del two phase commit, supponendo di avere un database main e uno doppione e supponiamo di voler eseguire una transazione, in questo caso dovr‘o ricevere non uno ma due messaggi di modifica effettuata con successo, se qualcosa dovesse andare storto necessito di effettuare un rollback su entrambe i database e quindi tornare alle condizioni precedenti. Generalmente esiste un sistema main a cui ci si rivolge e se questo va down allora l'infrastruttura sara‘ responsabile di reindirizzare al server che permettere l'accesso ai dati che abbiamo richiesto al main, quindi questo server dovr‘a avere una replica di quei dati richiesti in precedenza. Esistono differenti metodi di scelta di questi server se un server main va offline. Adesso poniamoci la domanda: quali sono i passi che vengono realizzati quando noi eseguiamo una query in ambiente distribuito ? Analizziamo la prima tecnica importante che ‘e il query mapping, questo ‘e il termine che si usa per rappresentare la metodologia che permette di capire quali dati vengono utilizzati dalla query e dove questi dati si trovano sui vari frammenti. Una volta effettuata la mappatura il dbms effettuer‘a la query in modo decentralizzato, quindi questa verr‘a spezzata, svolta e riorganizzata. Ovviamente il ddbms supporta l'ottimizzazione delle query. Nell'ottimizzazione sono importanti gli indici, questi sono degli strumenti messi a disposizione del dbms che permettono l'ottimizzazione delle query, indicano i valori piu‘ richiesti su un database. E‘ norma apportare degli indici sulle chiavi primarie ed ereditate. La procedura di inserimento e utilizzo degli indici pu‘o velocizzare il processo svolto da una query anche di 100 o 1000 volte. L'obiettivo dell'ottimizzazione sulle query ‘e quello di ridurre la quantita‘ di dati che io devo trasmettere al fine di ottenere il risultato. Utilizziamo un costrutto particolare che ‘e la semi-join, questa riduce il numero di tuple che io mi trasmetto mandando soltanto la colonna di join e prendendo unicamente i valori che mi vengono dalla join della tabella di partenza. L'omogeneita‘ dei data influenza questi parametri di ottimizzazione. Altro concetto molto utilizzato in ambito dei ddbms relazionali ‘e quello del database federato. Questo database federato funziona in questo modo: ho un modello dati condiviso da tutti ma i singoli nodi dispongono di modelli dati indipendenti, quindi ogni nodo mantiene uno schema generale che permette di comunicare con gli altri, ma a livello individuale il nodo pu‘o presentare comportamenti diversi rispetto ai suoi simili. Si pu‘o classificare il database distribuito su 3 differenti dimensioni:

- Autonomia
- Distribuzione
- Eterogeneità

La rappresentazione dei dati ‘e delineata dal data-model e dal data-set. Nella figura seguente ‘e possibile visualizzare le differenti dimensioni: