

Gestione dei Big Data

Matteo Aprile

Professore: Marco Zappatore, Antonella Longo

INDICE

I	Libri di testo consigliati	1	VIII Introduction to Transaction Processing Concepts and Theory	9
II	Databases	2	VIII-A Introduzione alle transazioni	9
II-A	Definizioni di base	2	VIII-B Problemi durante le transazioni	10
II-B	Tipologie di DB	2	VIII-C Le operazioni di una transazione	10
II-C	Ciclo di vita del DB	2	VIII-D Proprietà delle transazioni	10
II-D	Livelli di un DB	2	VIII-E Scheduling basato sulla recoverability	11
II-E	Data Base Managment System	2	VIII-F La serializzazione	11
II-F	Mini-world	2	IX NOSQL (Not Only SQL)	11
III	Database System Concepts and Architecture	3	IX-A Introduzione	11
III-A	Definizioni sui modelli	3	IX-B CAP Theorem	11
III-B	Definizioni fondamentali	3		
III-C	Schema	3		
III-D	Tipologie di DBMS	3		
IV	Data Modeling Using the Entity-Relationship (ER) Model	4		
IV-A	Entity-Relationship (ER)	4		
V	The Enhanced Entity-Relationship (EER) Model	5		
V-A	Superclassi e sottoclassi	5		
V-B	Graficazione superclassi e sottoclassi	5		
V-C	Notazioni	5		
V-D	Terminologia	6		
VI	Basic SQL	6		
VI-A	Statement - SELECT	6		
VI-B	Statement - WHERE	6		
VI-C	Statement - ORDER BY	6		
VI-D	Statement - INSERT	6		
VI-E	Statement - GROUP BY	7		
VI-F	Statement - HAVING	7		
VI-G	Statement - UNION	7		
VI-H	Statement - JOIN	7		
VI-I	Values - NULL	7		
VI-J	Function - Aggregate	7		
VI-K	Function - Window	7		
VI-L	Function - String	7		
VII	Distributed Database Concepts	8		
VII-A	Distributed Databases	8		
VII-B	Forme di trasparenza	8		
VII-C	Affidabilità e disponibilità	8		
VII-D	Autonomia	8		
VII-E	Frammenti	8		
VII-F	DDBMS	8		

I. LIBRI DI TESTO CONSIGLIATI

- Fundamental of Database Systems, 7th ed, Elmasri, Navathe
- Data Warehouse Design, Rizzi, Golfarelli big dataL concepts 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 **sarà 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 certa velocità
- **DataBase**: raccolta di dati altamente organizzati, intercorrelati e strutturati. È una struttura con dei collegamenti strutturati tra i dati
- **DBMS Data Base Management 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 relazioni
- **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 dati
- 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 Management 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 entità**, cioè i **concetti di base** che rappresentano una parte delle cose che inseriremo nel DB relazionale. Poi andremo a **connettere tra loro le entità**, 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

STUDENT

Name	Student_number	Class	Major
------	----------------	-------	-------

COURSE

Course_name	Course_number	Credit_hours	Department
-------------	---------------	--------------	------------

PREREQUISITE

Course_number	Prerequisite_number
---------------	---------------------

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
--------------------	---------------	----------	------	------------

GRADE_REPORT

Student_number	Section_identifier	Grade
----------------	--------------------	-------

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 dictionary**: 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
- **clients**: per far accedere gli utenti
- **DBMS server**: per eseguire 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:

- **entità' (sostantivi):** oggetti o cose specifiche presenti nel mini-world che bisogna rappresentare
- **relazioni (verbi):** collegano le entità'. 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 (proprietà):** **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
- **entità' 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 attributo, 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 **entità' al singolare**
- **verbi alla terza persona** e attivi o passi per capire da che parte si deve leggere la relazione

- per la **cardinalità** mi chiedo per un solo elemento quante entità potrà 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 incomberci in **relazioni di livello più alto** nel caso in cui ci trovassimo a descrivere relazioni con complessità alto. In generale **si cerca di evitare e di farlo con relazioni binarie** per evitare complicazioni nell'implementazione.

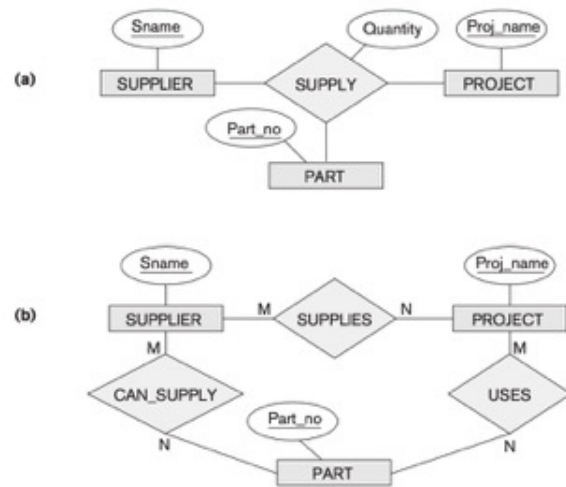


Figura 2. Relazione di livello alto

V. THE ENHANCED ENTITY-RELATIONSHIP (EER) MODEL

A. Superclassi e sottoclassi

L'idea è di andare a creare una **gerarchia**:

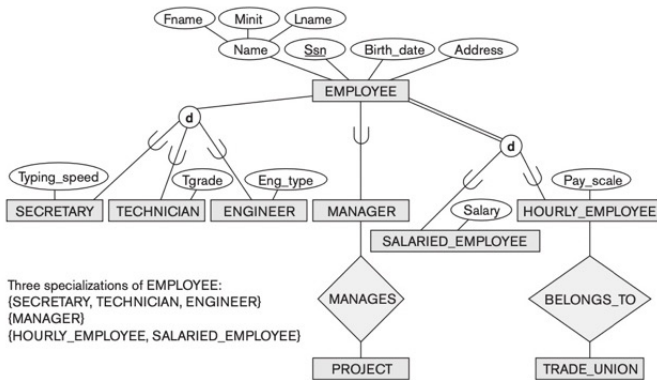


Figura 3. Gerarchia con disjoint

Per modellarlo mi chiedo quali siano le **caratteristiche che hanno in comune alcune entità**, allora tutti gli **attributi in comune vanno nella superclasse**. **Ogni entità 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.

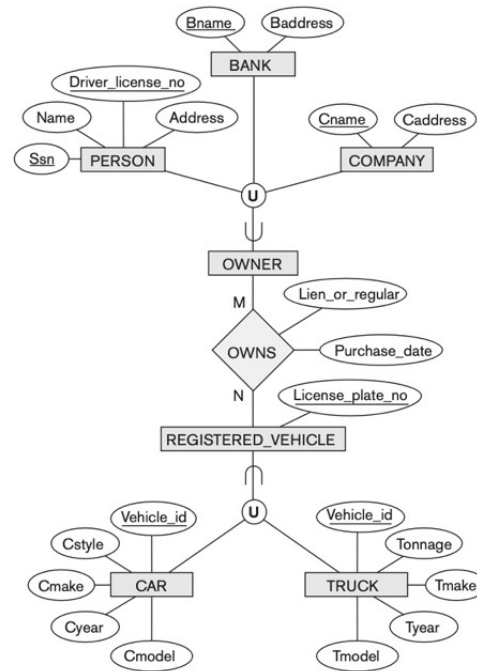


Figura 4. Gerarchia con disjoint

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:

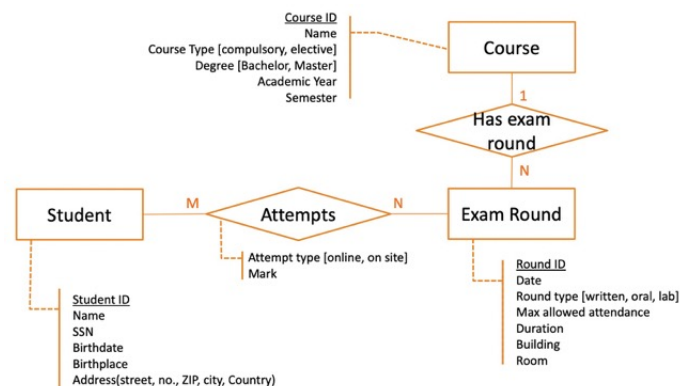


Figura 5. Notazione dei diagrammi ER

se un **attributo può avere solo un numero finito di valori** si usa:

...{..., ...}

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

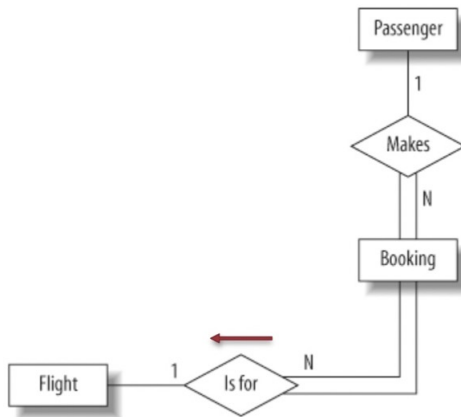


Figura 6. Reificazione ed orientamento della relationship

conviene usare delle relation con un nome univoco.

D. Terminologia

termine informale	termine formale
table	relation
column header	attribute
all possible column values	domain
row	tuple "i...i"
table definition	schema of a relation
populated table	state of the relation

Tabella I
TABELLA DELLE ORE DI LAVORO

Relation Name	Attributes
STUDENT	Name, Ssn, Home_phone, Address, Office_phone, Age, Gpa
Tuples	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 (clausole):

```
1 SELECT <attribute list>
2 FROM <table list>
3 [ WHERE <condition> ]
4 [ 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**:

```
1 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:

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

Si possono usare delle condizioni di tipo:

- numerico:

```
1 WHERE Dno = 5
```

- pattern matching tra stringhe:

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

se non è un'occorrenza esatta usiamo:

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

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

vado a selezionare da Statement gli attributi column

```
1 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.

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

F. Statement - HAVING

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

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

G. Statement - UNION

Usato per combinare il risultato di 2 o più statement SELECT

```
1 SELECT City FROM Customers
2 UNION
3 SELECT City FROM Suppliers
4 ORDER BY City;
```

H. Statement - JOIN

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

```
1 SELECT Orders.OrderID, Customers.CustomerName,
   Orders.OrderDate
2 FROM Orders
3 JOIN Customers
4 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:

```
1 SELECT column_names
2 FROM table_name
3 WHERE column_name IS NULL;
```

J. Function - Aggregate

Eseguono un'operazione su n valori dati e possono essere:

- COUNT
- AVG
- SUM
- MAX
- MIN

```
1 SELECT COUNT(column_name)
2 FROM table_name
3 WHERE condition;
```

L. Function - String

- CONCAT
- LEN
- UPPER
- LOWER

K. Function - Window

- LAG
- LEAD

VII. DISTRIBUTED DATABASE CONCEPTS

A. Distributed Databases

I dati utilizzati nelle infrastrutture dei big data **devono essere ACID**. Questi sistemi distribuiti 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 bisognerà 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. Sarà completa se siamo in grado attraverso l'operazione di union di riavere la relazione iniziale completa

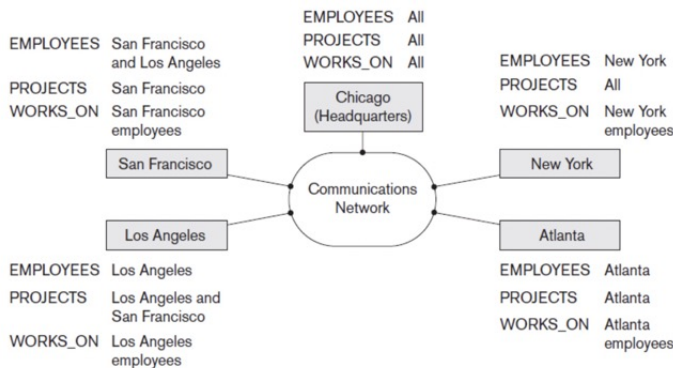


Figura 8. Partizione orizzontale di un db

- **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 possibilità di scalare i dati in modo orizzontale o verticale** a seconda dell'uso che dobbiamo farne. Parliamo di:

- **horizontal 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 **unità 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. DDBMS

Potrebbero esserci dei problemi:

- 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 caso bisognerà ricominciare 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**.

Quando noi **eseguiamo una query in ambiente distribuito** possiamo usare il:

query mapping: per capire **quali dati vengono utilizzati** dalla query e **dove si trovano** sui vari frammenti eseguendo quindi una **mappatura**. Successivamente **esegue la query in modo decentralizzato**

Da notare che il DDBMS permette l'**ottimizzazione delle query**, indicano i valori più richiesti su un database. Per esempio l'utilizzo di indici sulle chiavi primarie ed ereditate per velocizzare il processo della query.

Per riuscire a **ridurre la quantità di dati che devo trasmettere**, usiamo un costrutto particolare: **semi-join** che **riduce il numero di tuple che trasmetto mandando soltanto la colonna di join** e prendendo unicamente i valori che mi vengono dalla join della tabella di partenza.

Nei DDBMS relazionali sono usati i **database federati** dove 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. Abbiamo **3 dimensioni per il database distribuito**:

- autonomia
- distribuzione
- eterogeneità

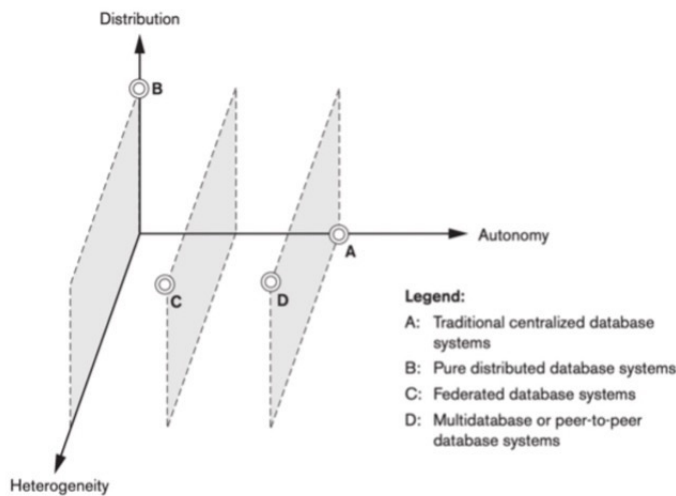


Figura 9. Dimensioni del DDBMS

VIII. INTRODUCTION TO TRANSACTION PROCESSING CONCEPTS AND THEORY

A. Introduzione alle transazioni

Il concetto di transazione è in **contrapposizione con il concetto di query** dato che:

- **query**: esecuzione di iscrizioni per ottenere delle informazioni
- **transazione**: **descrive l'unità di elaborazione del DB** cioè un insieme di query con 2 stadi fissi (inizio e fine)

Ci troveremo in un **sistema transazionale** se ho 2 **utenti concorrenti** (multiuser DBMS) che vogliono accedere allo stesso DB. Quindi se avrò dei DB molto ampi, con centinaia di possibili utenti che vogliono accedervi contemporaneamente dovrò avere 2 **requisiti** prestazionali:

- alta **disponibilità**
- alta **velocità di risposta**

oppure posso usare il **multiprogramming** consentendo al OS di mandare in esecuzione istruzioni di più processi in parallelo o in interleaved (interlacciato).

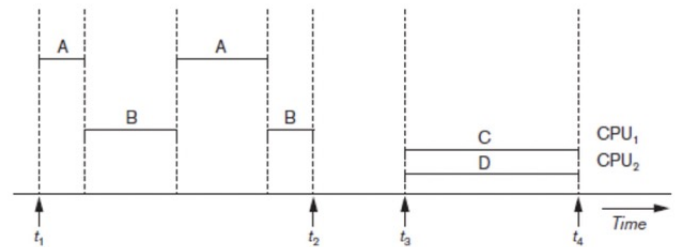


Figura 10. Processo paralleo e interleaved

I **tipi di transazione** che possiamo avere sono in:

- sola lettura
- lettura e scrittura

e possono avvenire su item, record, attributi di un database, o anche su un blocco del disco. Quando si eseguono queste operazioni, i **buffer vengono salvati nel DBMS** che una volta pieni vengono sostituiti tramite il **last recently used**. Tutto ciò **indipendentemente dalla granularità dell'item**, con notazione:

```
operazione_item(x)
```

e chiameremo:

- **read set**: tutto quello letto in una transazione
- **write set**: tutto quello scritto in una transazione

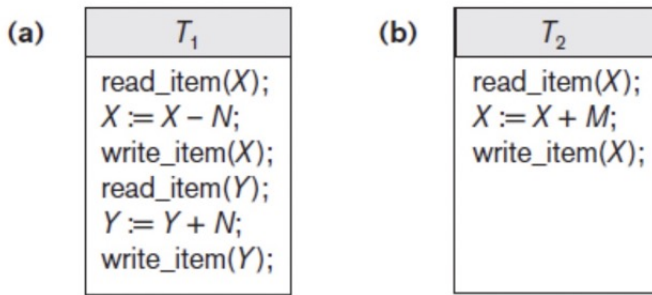


Figura 11. Due transazioni indipendenti

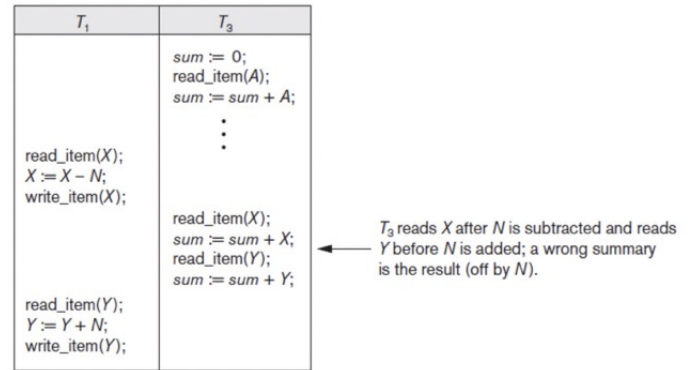


Figura 14. Incorrect Summary Problem

B. Problemi durante le transazioni

Possono essere:

- **lost update**: quando le esecuzioni della transazione i sono interleaved con la $i + 1$

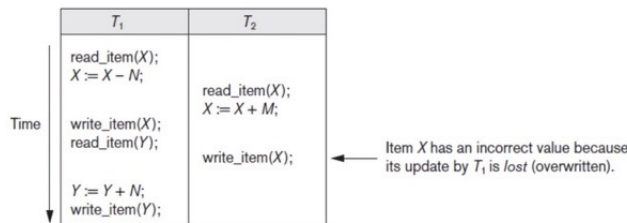


Figura 12. Lost Update Problem

con T_2 che legge il valore originale e l'aggiornamento di T_1 viene sovrascritto a quello di T_2

- **temporary update (dirty read)**: quando una transazione accede ad un dato e poi fallisce e nello stesso tempo un'altra accede allo stesso dato, questa continua a leggere il valore della transazione fallita (dirty data)

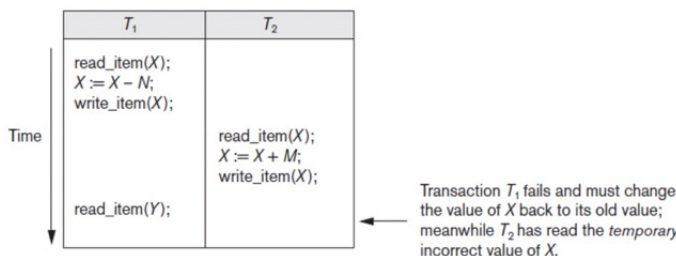


Figura 13. Dirty Read Problem

- **incorrect summary**: quando cerco di accedere a dei dati che stanno essendo modificati leggendo alcuni dati già modificati e altro no

- **unrepeatable read**: quando si effettua una lettura doppia del dato ma il valore della funzione ripetuta ad intervalli di tempo differenti e' diversa

C. Le operazioni di una transazione

Possono essere:

- **BEGIN_TRANSACTION**: fa iniziare la transazione
- **READ or WRITE**: operazioni di lettura o scrittura sul DB
- **END_TRANSACTION**: segna la fine della transazione e delle operazioni di lettura/scrittura. fa anche un **check sui cambiamenti effettuati** come può essere l'interleaved
- **COMMIT_TRANSACTION**: segnala una **fine con successo** e l'invio del commit delle transazioni, ottenendo un **commit point** che affiancato all'Id della transazione mi permette di capire quali di queste sono fallite
- **ROLLBACK**: segnala un **fail della transazione** e quindi la successiva **cancellazione dei cambiamenti**

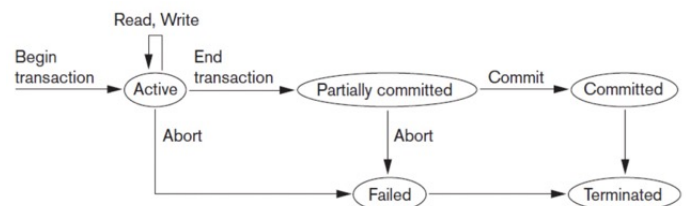


Figura 15. Operazioni delle transazioni

D. Proprietà delle transazioni

Le proprietà fondamentali per ogni transazione sono dette **ACID**:

- **Atomicity**: dove o esegui la transazione intera o non la esegui
- **Consistency**: quando la transazione termina non lascia inconsistenza
- **Isolation**: dove la transazione non interferisce con altre transazioni
- **Durability/permanency**: dalla transazione devo avere delle modifiche permanenti nel DB

E. Scheduling basato sulla recoverability

Lo scheduling **contiente**:

- **ordine di esecuzione delle operazioni** da tutte le transazioni
- operazioni di **transazioni diverse**

Potremo avere due **operazioni in conflitto** se:

- appartengono a **transazioni diverse**
- accedono allo **stesso elemento X**
- ha un'operazione di **write**

F. La serializzazione

È importante perché **non permette l'interleaved** tra le transazioni, eliminando i problemi sopra citati. Se presa **come unica risorsa non e' realizzabile**, allora ci si chiede quali siano le schedulazioni che si posso serializzare (solo n transazioni).

IX. NOSQL (NOT ONLY SQL)

A. Introduzione

Sono dei **database distribuiti** con:

- storage semistrutturato
- alta replicazione dei dati
- alte prestazioni

Non sarà richiesto uno schema ER, e ha come **tipi**:

- documentali
- chiave valore
- colonnari
- a grafo
- ibridi
- a oggetti
- xml

B. CAP Theorem

Se ho modelli diversi di DB vorrei poter avere **diversi livelli di consistenza dello stato del DB** e voler forzare la transazione. Allora dovremo avere:

- **consistenza**: in tutti i DB distribuiti **ciscuna istanza deve fornire gli stessi dati**
- **disponibilita'**: **un fail del nodo master non deve intaccare gli altri**

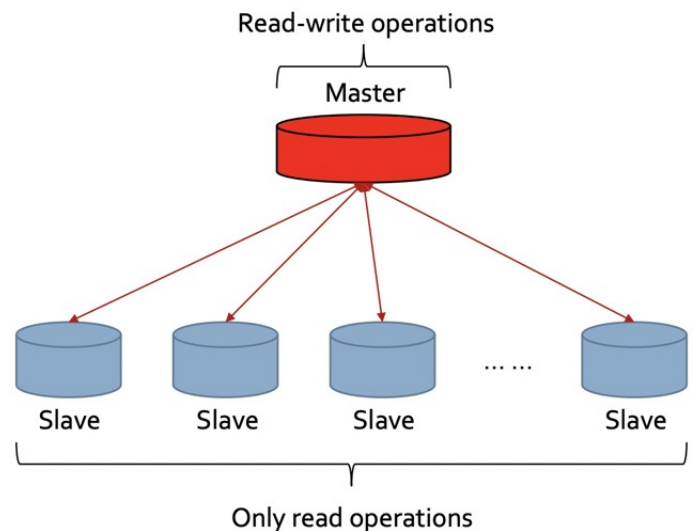


Figura 16. Master and slave

- **partizione**: il sistema continua a **funzionare nonostante la perdita di messaggi, o problemi di connessione**

Se ho un sistema distribuito **non possono garantire tutte e 3 le cose**, allora possono essere garantite **solo 2 alla volta**:

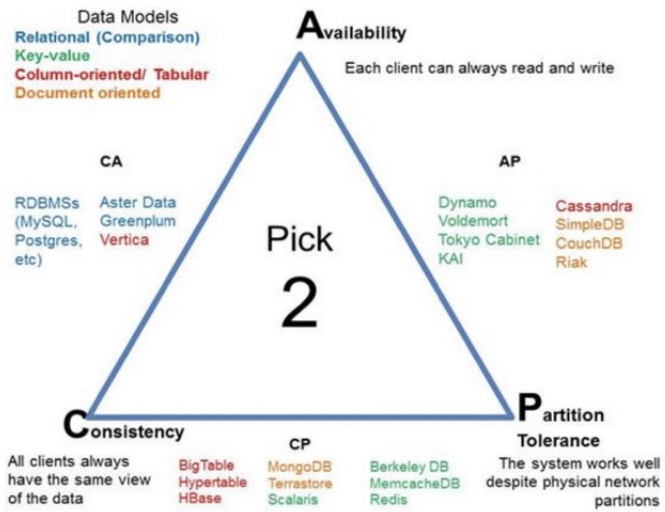


Figura 17. CAP theorem