

Data Science e Tecnologie per le Basi di Dati

“What’s not okay is when you fail and you stay down.
Whoever stays down is a loser.
Winners will fail and get up.
You always get up, that is a winner. That is a winner.”

Davide Antonino Giorgio – s262709

Breve richiamo sulle basi dati di tipo OLAP e OLTP: Gli strumenti **OLAP** si differenziano dagli **OLTP** per il fatto che i primi hanno come obiettivo la performance nella ricerca e il raggiungimento di interrogazioni quanto più articolate sia possibile; i secondi, invece, mirano ad una garanzia di integrità e sicurezza delle transazioni.

1. I Dati

Oggi produciamo un grandissimo quantitativo di dati. Molti dati sono prodotti dagli utenti, altri da sensori, web server, file di log, internet of things.

La particolarità dei dati è descritta dalle 5 V:

1. **Variety**: I dati sono vari e in formati differenti (testo, audio, foto, video etc..)
2. **Velocity**: I dati hanno un veloce rate di generazione e spesso devono essere processati altrettanto velocemente
3. **Veracity**: I dati devono essere accurati, consistenti e rilevanti
4. **Value**: I dati hanno un enorme valore e sono raccolti da tutte le aziende
5. **Volume**: La quantità di dati cresce in modo esponenziale nel tempo

2. Data Science Process

Il processo sequenziale per arrivare allo studio dei dati è descritto dall’immagine sottostante



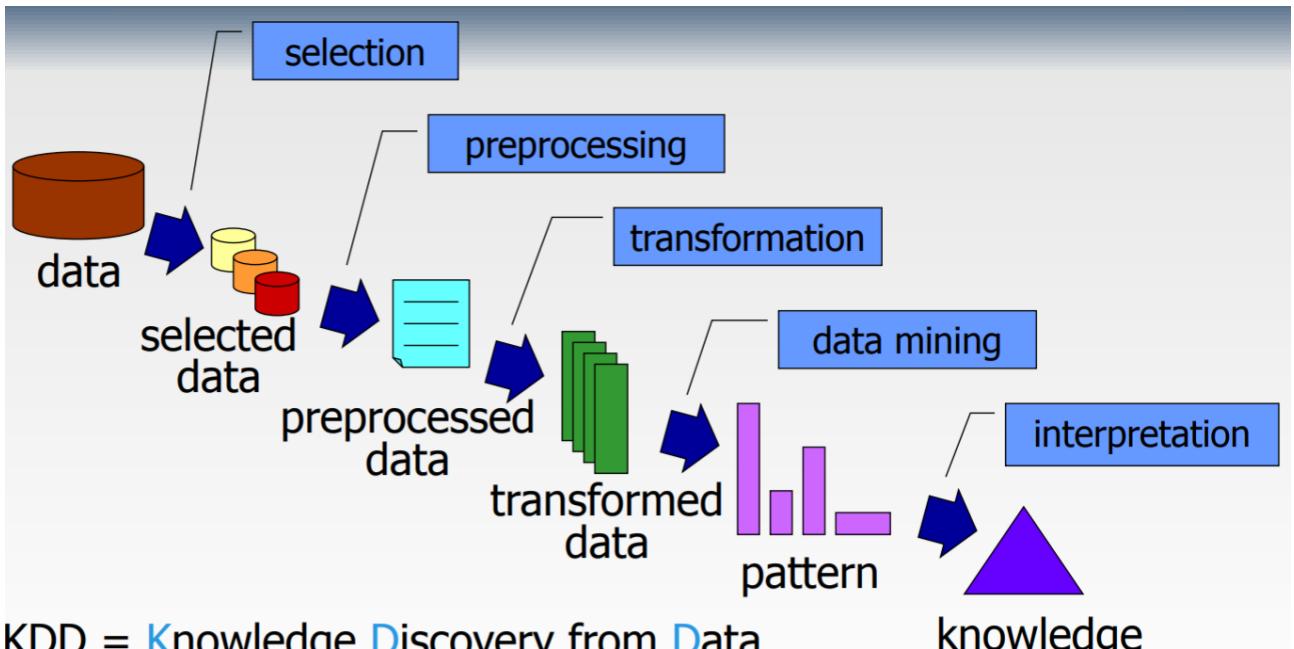
Ogni singolo passo è descritto da precise informazioni:

- **Generation**: Questo passo è legato alla generazione dei dati che può avvenire in maniera attiva (generati da utenti), passiva (transazioni bancarie o registri commerciali) o automatica (sensori e IoT)
- **Acquisition**: In questa fase i dati vengono raccolti, trasmetti e pre-processati eliminando così la parte non utile dei dati
- **Storage**: In questa fase i dati vengono immagazzinati nelle infrastrutture (HDD, SSD) e organizzati nei sistemi di Data Management o Programming models.
- **Analysis**: È la parte che si divide in Objectives (analisi descrittiva) e in Methods (associazioni, classificazioni, clustering)

Di rilevante importanza è il [Data Mining](#) che è l'insieme delle tecniche e metodologie che hanno per oggetto l'estrazione di informazioni utili da grandi quantità di dati in modo automatico tramite l'uso di modelli astratti denominati *pattern*. Ad esempio, lo studio un [clickstream](#) raccolto durante la visita di un sito web di e-commerce. Tali dati sono utilizzati per la profilazione degli utenti da parte delle aziende al fine di poter offrire dei servizi aggiuntivi mirati al cliente in modo da poter incrementare le vendite.

2.1 KDD – Knowledge Discovery in Database process

Il “[Knowledge Discovery in Database process](#)” (KDD) descrive il processo per l'estrazione delle informazioni utili dai dati.



KDD = [Knowledge Discovery from Data](#)

Inizialmente bisognerà selezionare di dati utili alla nostra ricerca scartando i dati di “rumore”, successivamente bisognerà prepararli in dati processati tramite una fase di preprocessing. Solo adesso i dati potranno essere trasformati per poi essere analizzati. L'analisi dei dati avviene sempre insieme agli esperti del settore, ad esempio un ingegnere informatico in Ducati lavorerà affiancato ad altri ingegneri meccanici che potranno assicurare la corretta integrità dei dati raccolti. Inoltre, i dati reali sono sempre “sporchi”, ad esempio possono contenere errori di misura che dovranno essere eliminati o considerati nell'estrazione finale.

3. Tecniche di Data Mining

Per l'analisi automatica dei dati si usano alcune tecniche di Data Mining. Qui sono presentate le più usate ma ne esistono svariate.

3.1 Regole di Association

Si cerca spesso di cercare una correlazione fra le informazioni ricorrenti, ad esempio lo studio del [Market basket](#) per la disposizione dei prodotti negli scaffali dei supermercati. Tale studio è una base dati transazionale. La casualità della comparsa ricorrente di più elementi insieme si studia per generare la “Raccomandation”, sfruttata per consigliare alcuni prodotti sugli scaffali dei supermercati. Ciò avviene anche nei siti di e-commerce come Amazon.

3.2 Regole di Classification

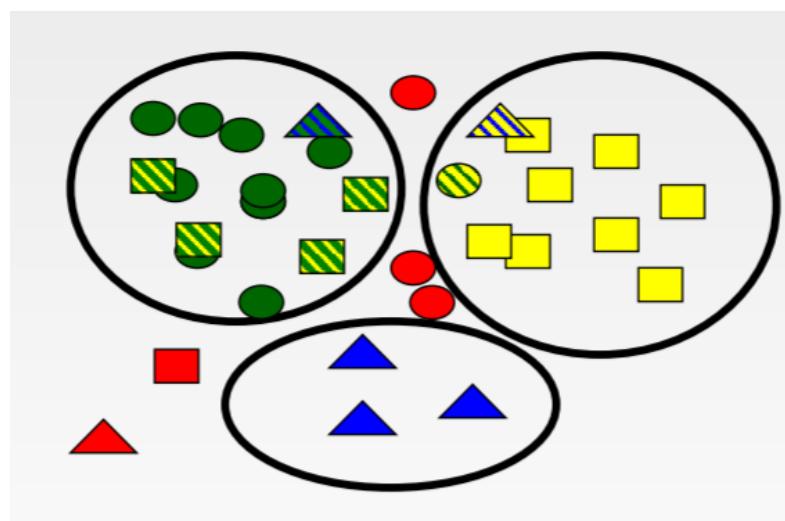
È un modo di predire e classificare un tipo d'informazione, ad esempio le e-mail di spam. Tale operazione è effettuata tramite l'assegnazione di un'etichetta di classe identificata da vecchi dati (dati di training) già classificati in precedenza.

Un algoritmo di base dunque su un modello già esistente per comprendere e classificare i nuovi dati in ingresso. Tale modello può essere costruito con tecniche comprensibili o no. Le tecniche non comprensibili sono denominate [Black Box](#).

Tali algoritmi sono utilizzati per svariati utilizzi come le reti neurali e il riconoscimento facciale.

3.3 [Clustering](#)

È un modo per separare gli oggetti fra loro in base al fatto se sono simili o no. Ciò avviene in base allo studio di alcune caratteristiche simili che contraddistinguono certi oggetti. Ad esempio, tali algoritmi sono utilizzati nella galleria di “Google foto” per selezionare immagini simili in base ad alcune ricerche (ad esempio cercando la parola “mare”)



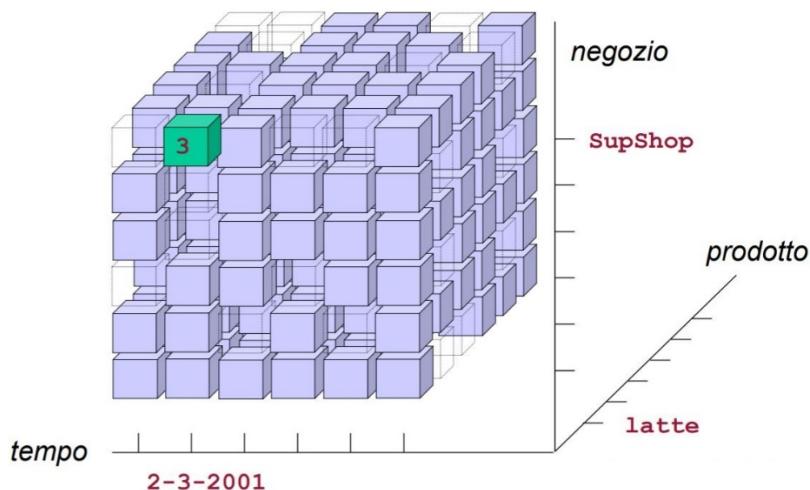
4. Introduzione ai Data Warehouse

Nelle aziende nasce il bisogno di studiare i dati raccolti in modo da supportare le decisioni aziendali. Per far fronte a tali richieste nasce la Business intelligence.

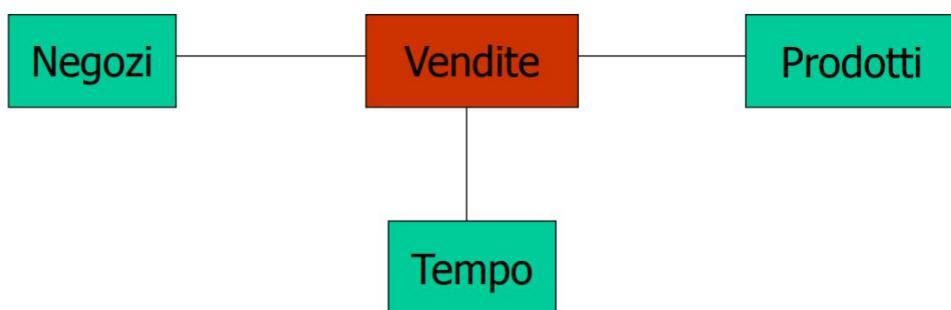
La maggior parte delle aziende possiede già grandi quantità di dati che possono permettere la definizione di strategie vincenti oppure studiare l'evoluzione della domanda di un dato prodotto.

Il Data warehouse è appunto una base dati che nasce per il supporto alle decisioni aziendali. Tale base dati è separata dalla base dati operativa dell'azienda in modo da permettere ricerche complesse che potrebbero ridurre le prestazioni generali del database. Questa base dati è sviluppata per servire uno specifico servizio d'interesse, integrando dati di più basi dati per osservare meglio il problema monitorato.

Un passo importante è la consistenza dei dati iniziali quando devono essere inseriti nel data warehouse. Inoltre, i dati contenuti nel data warehouse sono storici quindi dipendenti dal tempo. L'aggiornamento di un dato non sovrascrive il vecchio ma la base dati mantiene anche le informazioni precedenti. Un modo per rappresentare una base dati potrebbe essere la rappresentazione multidimensionale.



In realtà vi sono numerosi modi per rappresentare una base dati, uno di questi è il modello a stella.



Tipicamente i data warehouse sono spesso rappresentati con un modello a stella come quello sovrastante.

5. Architetture per i data warehouse

La costruzione di un data warehouse può essere divisa in più livelli. Tipicamente non si sviluppano mai in un solo livello, si parla dunque di architetture a due o più livelli poiché queste sono maggiormente scalabili e separano in modo diverso i dati in ingresso dai dati oggetto d'analisi.

Le sorgenti esterne vengono lavorate tramite degli algoritmi definiti [ETL](#) che permettono la l'estrazione, la trasformazione e il caricamento dei dati.

I data warehouse solitamente non sono mai “globali” a livello aziendale ma spesso si preferisce sviluppare più data warehouse per dividere la mole di lavoro che comporta lo sviluppo di un progetto di tali dimensioni.

I dati specifici di un certo compartimento di dati (ad esempio le vendite oppure tutti gli impiegati) sono contenuti in dei database specifici chiamati [Data Mart](#) che contengono tutte le informazioni di un settore prefissato. Nello sviluppo di una base dati bisogna considerare anche lo spazio per i metadati e dei dati per la gestione del data warehouse.

I server per la costruzione dei data warehouse possono essere di tipo:

- ROLAP -> Base dati in relazione estesa
- MOLAP -> Dati in forma matriciale
- HOLAP -> Ibridi

5.1 [Processi ETL](#)

I processi ETL sono degli algoritmi di lavorazione dei dati che verranno successivamente inseriti nel data warehouse.

I passaggi di tali processi sono:

- Extraction: dei dati dalle sorgenti utilizzate dall'azienda (es. file excel)
- Transformation: dei dati nel formato utilizzato dal database
- Loading: caricamento dei dati nel database

Tali processi vengono eseguiti la prima volta per la creazione del DW e successivamente con frequenza costante (tipicamente giornaliera)

Ad esempio, quando la segreteria del politecnico finisce di effettuare le registrazioni della giornata inserisce tutti i dati raccolti del database successivamente ad un processo di ETL.

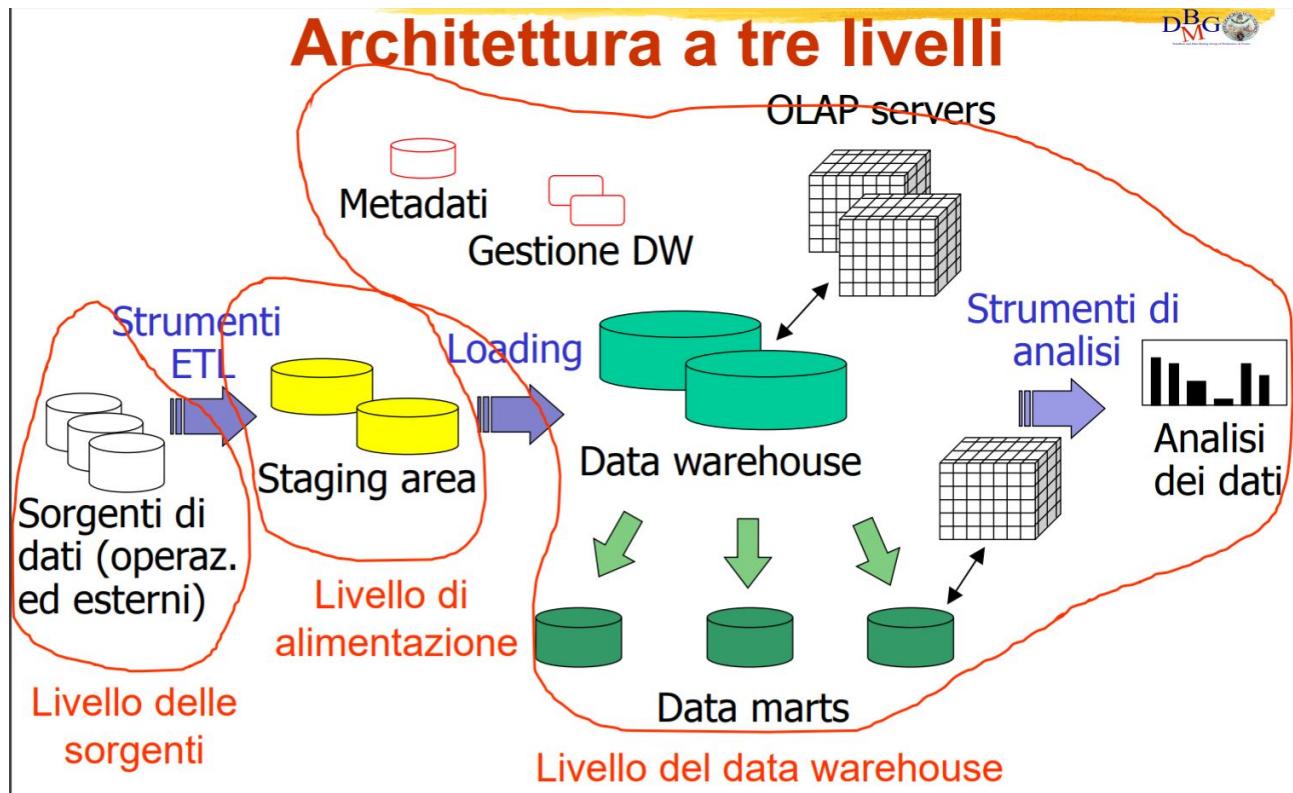
5.2 Metadati

I metadati sono dei dati utili per la trasformazione, correzione, gestione dei dati, query e per la gestione dell'impegno delle risorse all'interno del data warehouse. (esempio uso della CPU e stato delle memorie)

6. Architetture a più livelli

La progettazione di una base di dati può essere fatta a più livelli. Ad esempio, potremmo pensare di costruire un data warehouse in due livelli (livello sorgenti – livello data warehouse). Questo tipo di struttura però necessita di svolgere “al volo” gli ETL e tale operazione è difficile da effettuare.

I data warehouse moderni sono solitamente costruiti in tre o più livelli grazie alla presenza di una “staging area” chiamata ODS (Operational Data Storage) dove arrivano i dati processati da ETL e solo successivamente viene effettuato il loading nel data warehouse.



La staging area permette di separare le fasi ET da quella di Loading in modo da distribuire il carico di lavoro. Tale organizzazione aumenta la ridondanza dei dati ma permette operazioni complesse di trasformazione e pulizia dei dati.

7. Progettazione del data warehouse

7.1 Introduzione alla progettazione dei data warehouse

Prima dello sviluppo del data warehouse bisogna esaminare le necessità del cliente, spesso loro hanno delle grandi aspettative sul funzionamento del sistema che noi svilupperemo ma come spesso capita, i data warehouse non hanno uno scopo risolutivo della problematica ma mettono in luce tale ciò che necessita di attenzione.

Durante la progettazione è consigliabile inglobare nel team di sviluppo i componenti esperti dei dati dell'azienda cliente in modo da poter estrapolare delle informazioni importanti sui dati da loro (ad esempio inconsistenza su alcune tipologie di dati)

7.2 Metodi di progettazione applicativi

Esistono due procedimenti logici per la progettazione di un data warehouse:

- Top-Down: è la realizzazione di un data warehouse globale che fornisca una visione globale e completa dei dati aziendali
- Bottom-Up: è la realizzazione dei data Mart definiti su specifici settori aziendali che successivamente alimenteranno un data warehouse di dimensioni più importanti

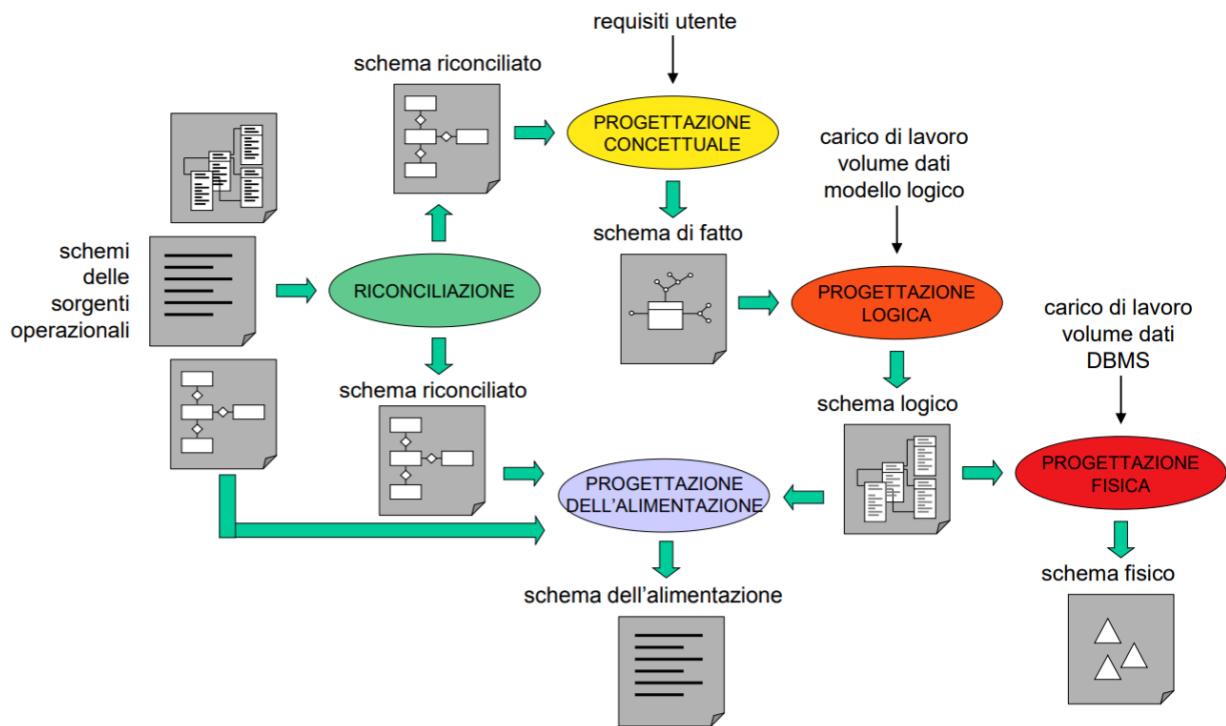
Tipicamente si preferisce una metodologia di tipo Bottom-Up perché riduce le tempistiche di consegna di un prodotto funzionante al cliente.

Un'importante fase da considerare nello sviluppo di un data warehouse è sicuramente la fase di tuning che permette il corretto funzionamento del database durante il suo ciclo di attività.

7.3 Progettazione di un data Mart

La progettazione di un data Mart funzionante si divide in sei diverse fasi:

1. Progettazione concettuale dei requisiti utente
2. Conoscenza di ciò che ho a disposizione nelle sorgenti operazionali
3. Tramite la progettazione di uno schema riconciliato costruisco successivamente uno schema di fatto che successivamente darà vita alle tabelle dei fatti
4. Progettazione logica dello schema logico
5. Progettazione ETL da schema logico e schema riconciliato
6. Progettazione fisica



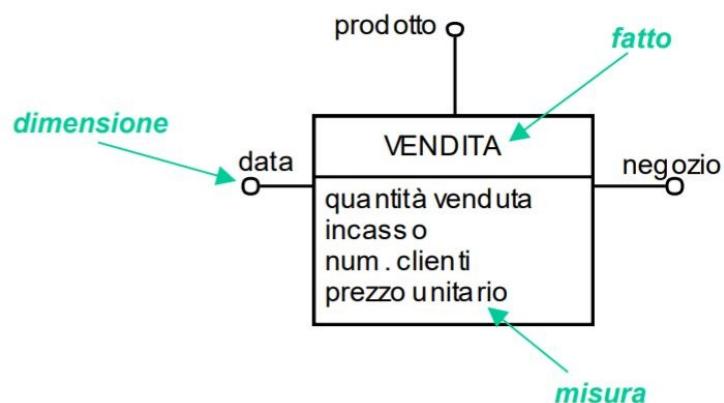
7.4 Progettazione concettuale

Prima di intraprendere il percorso di programmazione concettuale di un data mart/data warehouse bisognerà analizzare i requisiti in modo da raccogliere le esigenze di analisi dei dati che dovranno essere soddisfatte dal data mart.

L'alimentazione del data mart dovrà essere preferibilmente di poche sorgenti ma affidabili in modo da garantire più facilmente la coerenza dei dati inseriti.

Il modello utilizzato per la progettazione concettuale è il FACT MODEL ed è costituito da tre elementi base:

1. Fatto: modella un insieme di parti d'interesse, evolve nel tempo.
2. Dimensione: Descrive le coordinate di analisi di un fatto
3. Misura: descrive una proprietà numerica di un fatto

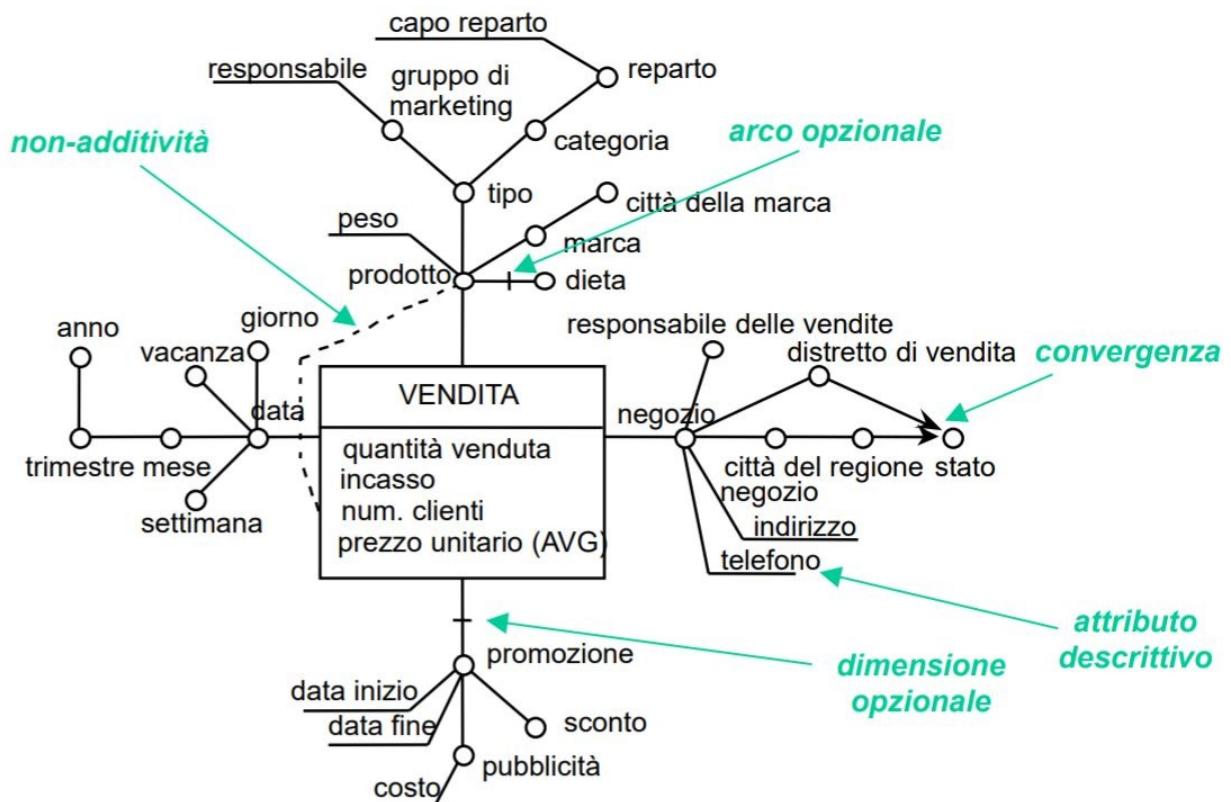


La tabella dei fatti è quindi contrassegnata da un fatto che contiene delle misure che sono le proprietà del fatto. Le dimensioni sono le coordinate variabili del fatto che variano di volta in volta.

Le dimensioni tipicamente sono rappresentate in delle gerarchie che tipicamente sono in una relazione 1:n. Se risalgo nella gerarchia allora sto “generalizzando” quindi mi perdo del dettaglio d’informazione.

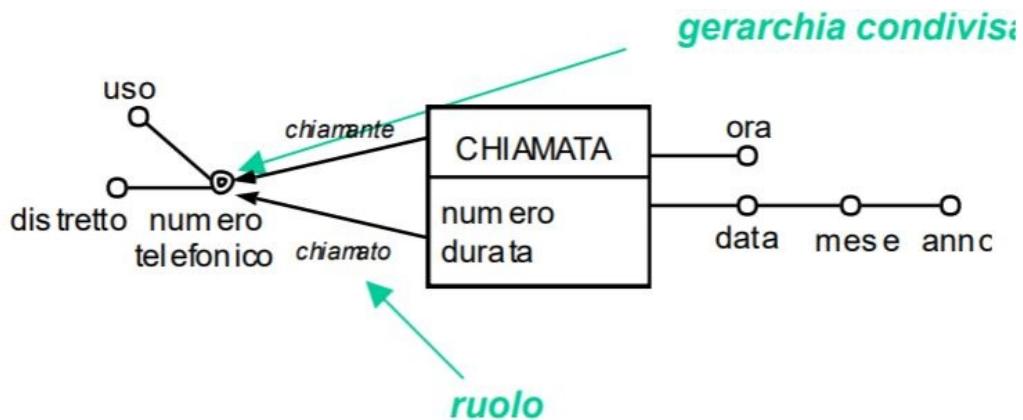


Nella stesura di un diagramma DFM per la realizzazione concettuale di un data warehouse dobbiamo tener presente di alcuni costrutti avanzati necessari per esprimere al meglio certe situazioni.



Nel diagramma possiamo notare la presenza di alcuni costrutti avanzati che sono evidenziati in color verde acqua.

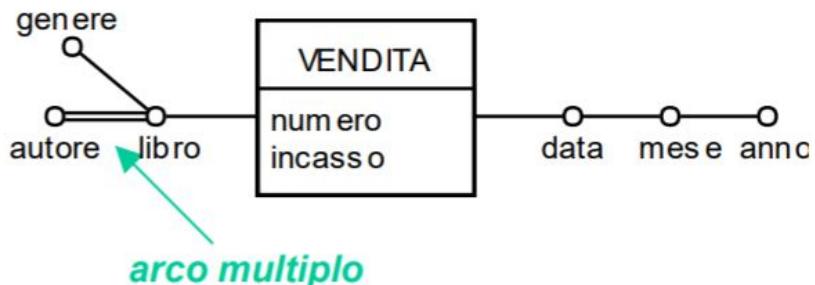
Esistono infatti degli attributi opzionali, delle dimensioni opzionali, attributi descrittivi sui quali non si fa aggregazione, la convergenza che prevede la riunione di due misure su una sola e la non additività che avviene quando non è possibile salire nella gerarchia per tutte le misure, questo perché tutte non sono aggregabili. Per comprendere la non additività posso considerare l'esempio del numero di bevande comprate lo stesso giorno. Dal numero di clienti entrati (scontrini venduti) non posso risalire la gerarchia per capire il numero di bevande comprate lo stesso giorno, questo perché uno scontrino può essere legato a più bevande ma più bevande possono essere nello stesso scontrino.



In questo diagramma possiamo notare la presenza di una gerarchia condivisa che utilizziamo per incrementare le performance di funzionamento del data warehouse. In una chiamata lo stesso numero può essere il chiamante o il chiamato, è dunque inutile fare due gerarchie per i due tipi di numeri.

Inoltre, possiamo notare la realizzazione del tempo. L'ora non è correlata alla data, il mese e quindi l'anno. Questo perché voglio poter effettuare facilmente delle ricerche in base all'ora senza dovermi "portar dietro" le informazioni sulla data. Torneremo successivamente sulla rappresentazione del tempo.

Esiste la possibilità di costruire un arco multiplo come in questo caso



Dove ogni libro può essere stato scritto da più autori e più autori possono aver scritto lo stesso libro. Spesso per problemi di spazio non si usano mai archi multipli direttamente sulla tabella dei fatti perché aumenterebbero enormemente il numero di righe totali da memorizzare nel database. Spesso per evitare queste situazioni si utilizzano dei "gruppi" come ad esempio il gruppo di diagnosi nel ricovero ospedaliero.

7.5 Aggregazione

È il processo che ci permette di calcolare delle informazioni dei dati ma di granularità meno fine rispetto ai dati di partenza. Ad esempio, calcolo del MIN, MAX, AVG, etc... su un insieme di valori.

L'obiettivo è quello di ridurre il livello di dettaglio in modo da poter estrapolare delle informazioni. Per fare ciò bisogna guardare le misure delle dimensioni (additive, non additive o misure non aggregabili)

Le misure possono essere di tre tipi differenti:

1. Misure di flusso: valutate alla fine di un certo periodo di tempo (es. importo incassato)
Sono aggregabili lungo tutti gli operatori standard
2. Misure di livello: (snapshot) valutate in uno specifico istante di tempo (es. saldo, num pezzi magazzino)
Non sono aggregabili lungo la dimensione tempo
3. Misure unitarie: esempio prezzo unitario di un prodotto
Non sono aggregabili lungo tutte le dimensioni

L'aggregazione ci permette di snellire certe operazioni, questo perché estrarre le informazioni da una mole di dati minore rispetto a quella di partenza ci permette di evitare di impegnare risorse in una ricerca fra molte tuple.

Tali liste di aggregati sono chiamate liste materializzate e i dati sono ottenibili aggregando a più livelli.

Non tutti gli operatori sono utilizzabili in più livelli di aggregazione, infatti esistono vari operatori di aggregazione:

- o Operatori distributivi: SUM, MIN, MAX. Mi permettono il calcolo di aggregati da dati a livello di dettaglio superiore
- o Operatori non distributivi: AVG. Non permettono il calcolo di aggregati a più livelli almeno che non tenga conto del relativo peso (count) che devo opportunamente memorizzare per ricalcolare il disaggregato. Servono quindi dei dati di supporto.

7.6 Rappresentazione del tempo

La variazione dei dati nel tempo si modellizza in 3 modi:

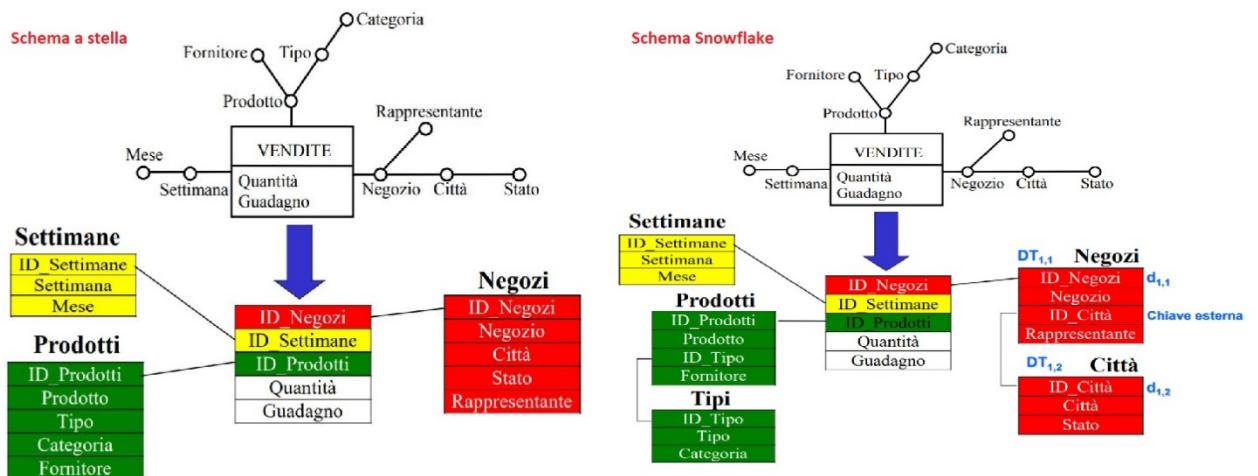
1. Tipo 1: fotografia dell'istante attuale esegue la sovrascrittura del dato con il valore attuale (NO storicitizzazione)
2. Tipo 2: memorizzo le diverse situazioni per storicitizzare lo stato. Ogni volta che cambio lo stato creo un nuovo oggetto partendo dal vecchio (che manterrò salvato) con lo stato aggiornato. Se faccio aggregazioni per lo stato trovo tutte le copie in quello stato. Esempio dello stato sociale che varia da single a sposato.
3. Tipo 3: Serve a rispondere alla domanda “cosa sarebbe successo se?”. È un modello che storicitizza l'intero evento con data d'inizio validità e fine validità. La prima tupla creata si chiama master e tutte le copie hanno il riferimento ad essa in modo da potersi

spostare più semplicemente nel tempo. In questo modo posso fare interrogazioni del tipo: “tutte quelle tuple che hanno come master...”

7.7 Progettazione logica: schema a stella o snowflake?

Nella progettazione di un diagramma DFM bisogna scegliere se effettuarlo con uno schema a stella o con un diagramma snowflake. La differenza sostanziale risiede nella generazione o meno di tabelle secondarie per la gestione dei fatti. Nello schema a stella vi è una tabella per ogni schema di fatto, per lo schema snowflake vi è invece la separazione di alcune dipendenze funzionali. Tale operazione permette nello snowflake di risparmiare della memoria una volta che il datawarehouse verrà implementato ma aumenta notevolmente la complessità d'interrogazione tramite query poiché queste devono essere molto più articolate.

Lo schema a stella però non è normalizzato.

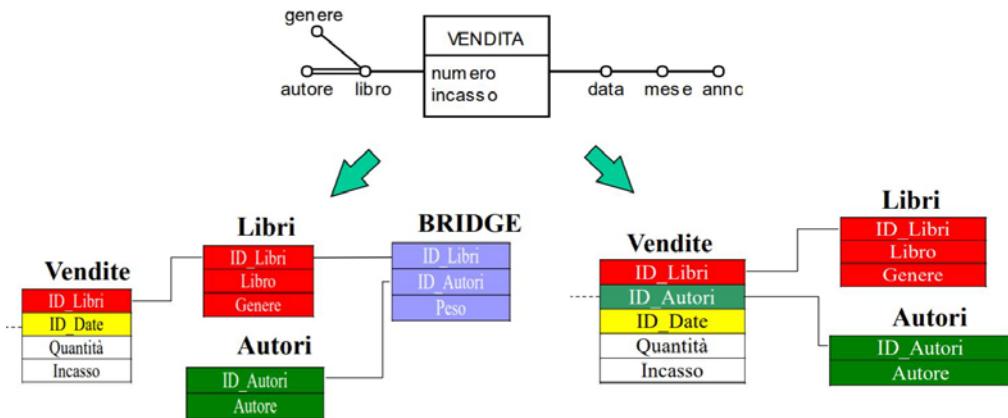


Nello schema a stella, lo schema di fatto contiene tutte le tre chiavi primarie delle tre dimensioni dell'esempio e queste chiavi sono insieme la chiave primaria della tabella dei fatti. Le “misure” sono tutti gli attributi nella tabella dei fatti oltre le chiavi primarie.

Nello schema a snowflake le tabelle sono a “matriosca”. Alla fine, il risparmio di memoria nello schema snowflake è minimo perché ciò che pesa veramente sono le tabelle dei fatti primarie.

La gestione degli archi multipli è gestita in due metodologie differenti:

1. Bridge table: si realizza una tabella aggiuntiva che fa da “ponte”. Tale tabella contiene entrambe le chiavi primarie e una variabile “peso” che può essere opzionale e che permette di pesare la partecipazione delle tuple nella relazione. Si modella così la relazione “molti a molti”
2. Push Down: Le chiavi primarie sono integrate direttamente nella tabella dei fatti.



7.8 Dimensioni degeneri

Per le dimensioni rappresentate da un solo elemento spesso si possono integrare nella tabella dei fatti ma a patto che l'attributo sia piccolo. Ad esempio, un attributo che mi afferma che la spedizione di un ordine è andata a buon fine o no.

Le altre dimensioni solitamente vengono assegnate ad un junk dimension, ovvero si raccolgono tutti gli attributi dimensionali nella stessa dimensione anche se questi non sono correlati fra loro. Conviene solamente se le combinazioni degli attributi sono di dimensioni piccole.

8. OLAP – Analisi

Per l'analisi dei dati si possono usare differenti ambienti di ricerca. Le operazioni che possono essere effettuate su un data warehouse sono:

- Roll up, drill down
- Slice and dice
- pivot di tabelle
- ordinamento

Tali operazioni possono anche essere combinate fra loro nella stessa query

❖ Roll up

Riduzione del livello di dettaglio che viene effettuata tipicamente con una group by.
Ad esempio -> group by negozio, mese

❖ Drill down

Aumento del livello di dettaglio tramite l'aggiunta di una nuova dimensione

❖ Slice and dice

Selezionare una fetta dei dati (SLICE)

Sottodimensione (DICE)

Queste operazioni non fanno aggregazione ma effettuano solamente un'operazione di selezione riducendo così il volume di dati

❖ Pivot

Cambia la struttura della visualizzazione della tabella senza variarne il livello di dettaglio

9. Funzioni OLAP - Estensioni del linguaggio SQL

Negli anni il linguaggio SQL è stato esteso per permettere delle ricerche su determinate finestre (esempio media mobile) di calcolo. Nasce così una nuova classe di funzioni aggregate (funzioni OLAP) caratterizzata da:

- Partizionamento simile ad una group by ma che non collassa le tuple rendendole tutte disponibili
- Ordinamento delle righe separate all'interno di ogni partizione (simile ad order by)
- Finestra di aggregazione che definisce il gruppo di righe su cui l'aggregato è calcolato, per ciascuna riga della partizione. Tale finestra di aggregazione prevede che nei dati non vi siano dei "buchi".

Un esempio del potenziale di tale base di dati:

Abbiamo una base dati strutturata come nell'immagine

Vendite(Città, Mese, Importo)

| Città | Mese | Importo |
|--------|------|---------|
| Milano | 7 | 110 |
| Milano | 8 | 10 |
| Milano | 9 | 70 |
| Milano | 10 | 90 |
| Milano | 11 | 35 |
| Milano | 12 | 135 |
| Torino | 7 | 70 |
| Torino | 8 | 35 |
| Torino | 9 | 80 |
| Torino | 10 | 95 |
| Torino | 11 | 50 |
| Torino | 12 | 120 |

vogliamo visualizzare, per ogni città e mese:

- Importo delle vendite
- La media rispetto al mese corrente e ai due mesi precedenti, separatamente per ogni città

L' interrogazione che risponde a queste domande sarà la seguente:

```
SELECT Città, Mese, Importo,  
       AVG(Importo) OVER (PARTITION BY Città  
                           Order BY Mese  
                           ROWS 2 PRECEDING)  
              AS MediaMobile  
FROM Vendite
```

La finestra di aggregazione può essere definita a livello fisico (ad esempio la riga corrente e le due righe precedenti) tramite l'uso della chiave ROWS o a livello logico (ad esempio il mese corrente e i due mesi precedenti) tramite l'uso della chiave RANGE.

In questo modo posso fare anche un confronto fra i dati dettagliati e i dati complessivi. Posso anche usare solo una parte di questo nuovo modo, ad esempio se voglio solamente partizionare la nostra base dati per le città. Grazie a queste tecniche aumentano anche le prestazioni delle nostre interrogazioni.

Definizione intervallo fisico

L'intervallo fisico della finestra d'aggregazione può essere stabilito in più modalità:

- Posso ad esempio selezionare la mia finestra fra un estremo inferiore e la riga corrente tramite il comando: ROWS 2 PRECEDING
- Tra un estremo inferiore ed uno superiore: ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING
- Tra l'inizio o la fine di una partizione e la riga corrente: ROWS UNBOUNDED PRECEDING (o FOLLOWING)

Il raggruppamento fisico è adatto ai dati che non hanno interruzioni nella sequenza (no buchi)

Definizione intervallo logico

Si utilizza il costrutto RANGE con la stessa sintassi dell'intervallo fisico. Bisogna però definire la distanza fra gli estremi dell'intervallo e il valore corrente sulla chiave di ordinamento. RANGE 2 MONTH PRECEDING

A differenza dell'intervallo fisico, l'ordinamento logico si adatta nel caso di dati "sparsi".

Possiamo utilizzare solamente dati di tipo numerico o data.

Un esempio di un calcolo dell'importo cumulativo delle vendite al trascorrere dei mesi

```
SELECT Città, Mese, Importo,  
       SUM(Importo) OVER (PARTITION BY Città  
                           ORDER BY Mese  
                           ROWS UNBOUNDED PRECEDING)  
              AS SommaCumul  
FROM Vendite
```

Possiamo anche utilizzare questo nuovo metodo per effettuare un confronto fra i dati dettagliati e i dati complessivi. Ad esempio, se vogliamo visualizzare l'importo totale delle vendite sul periodo completo per la città corrente.

```
SELECT Città, Mese, Importo,  
       SUM(Importo) OVER (PARTITION BY Città)  
              AS ImportoTotale  
FROM Vendite
```

Possiamo anche effettuare interrogazioni che mettano in confronto immediato i valori appena acquistati.

```
SELECT Città, Mese, Importo,  
       Importo/SUM(Importo) OVER ()  
              AS PercSuImportoTot,  
       Importo/SUM(Importo) OVER (PARTITION BY Città)  
              AS PercSuImportoCity,  
       Importo/SUM(Importo) OVER (PARTITION BY Mese)  
              AS PercSuImportoMese  
FROM Vendite
```

Possiamo anche “strizzare” i dati con una group by e poi eseguire una where per scremare i dati, infine possiamo anche fare una over per fare un calcolo di aggregati di aggregati.

```
SELECT Città, Mese, SUM(Importo) AS ImportoTotMese,  
       AVG(SUM(Importo)) OVER( PARTITION BY Città  
                               ORDER BY Mese  
                               ROWS 2 PRECEDING)  
FROM Vendite  
GROUP BY Città, Mese
```

Funzioni di Ranking

Le funzioni di ranking permettono di ordinare la posizione aggiungendo una colonna che rappresenta la “classifica” del rank. Tali funzioni sono:

- rank(): ordina la posizione lasciando intervalli vuoti successivi con buchi in caso di presenza di due valori uguali in due tuple diverse su cui si fa il rank. Esempio: 1 2 2 4 5 <- salta il 3
- densrank(): simile a rank ma nel caso di due tuple con lo stesso valore su cui si fa il rank non viene lasciato un “buco” ma l’andamento è continuo. Esempio: 1 2 2 3 4

Esempio: visualizzare per ogni città nel mese di dicembre l’importo delle vendite e la posizione nella graduatoria.

```
SELECT Città, Importo,  
       RANK() OVER(ORDER BY Importo DESC) AS Graduatoria  
FROM Vendite  
WHERE Mese = 12
```

DESC = ordine decrescente

Non ci interessa partizionare perché una città avrà solo un importo x il mese di dicembre. Ci basta una sola partizione con tutte le città su cui poi faremo la where.

Adesso vogliamo ordinare il risultato precedente, mediante la clausola Order By, in ordine alfabetico di città.

```
SELECT Città, Importo,  
       RANK() OVER(ORDER BY Importo DESC)  
  FROM Vendite  
 WHERE Mese = '12'  
 ORDER BY Città
```

Così prima creiamo una classifica con rank in base all'importo grazie alla order by interna e poi riordiniamo il tutto con una order by esterna in base all'ordine alfabetico x città.

Estensione della clausola GROUP BY

Lo standard SQL-99 ha esteso la clausola della group by per effettuare il calcolo di totali parziali "in un colpo solo". Tale estensione aumenta l'efficienza.

Le operazioni che possiamo effettuare sono 3:

- Rollup: Esegue il calcolo aggregato (ad esempio somma) e lo presenta alla fine di ogni elenco eliminando il calcolo per un attributo uno alla volta.
(Ordine degli elementi influente)
- Cube: Calcola la funzione aggregata a tutte le combinazioni possibili degli elementi contenuti (ordine degli elementi NON influente)
- Grouping Set: Esegue il calcolo della funzione aggregata solamente sul set specificato.
(Ordine degli elementi NON influente)

Viste Materializzate (viste ma MEMORIZZATE nella base dati)

Le viste materializzate sono delle tabelle di supporto (calcolati con rollup, cube, grouping set) che ci permettono di accedere velocemente ai dati aggregati. Tali viste sono definite e memorizzate nella base dati sotto forma di tabella. Al contrario delle viste virtuali (che sono volatili e generate sul momento) le viste materializzate ci permettono di operare su dati aggregati salvati in memoria. Sono sommari pre-calcolati della tabella dei fatti. Sono quindi viste d'aggregazione utilizzate per aumentare l'efficienza delle operazioni. Ciò avviene perché spesso si effettuano delle interrogazioni su dati che devono essere aggregati, è quindi da evitare il ricalcolo continuo delle viste. (come invece si dovrebbe fare con le viste virtuali)

Tutte le viste che possiamo effettuare prendono il nome di reticolo multidimensionali.

Le viste le usiamo per risalire nella gerarchia. Ad esempio, posso pensare di utilizzare una vista aggregata per risalire dalla categoria prodotto->tipo_prodotto

Il problema è che possiamo avere tantissime viste materializzate. Bisogna capire quali viste materializzate utilizzare.

Co più gerarchie si generano facilmente tantissime viste che occupano quindi molta memoria
(prende il nome di reticolo multidimensionale)

Per capire quali liste materializzate mi conviene calcolare, in base al costo e agli aggiornamenti, si usano delle funzioni di costo che mi permettono di stimare il “miglioramento” nel tempo di lettura (predizione del carico) e il “peggioramento” nel tempo di scrittura.

Dobbiamo considerare anche dei Vincoli per l’uso delle viste materializzate:

- Spazio disponibile sul disco
- Tempo a disposizione per l’aggiornamento
- Tempo di risposta
- Freschezza dei dati

Progettazione Fisica

Per accelerare le operazioni di ricerca ci sono 2 strade:

- Viste Materializzate
- Indici

Più avanti vedremo in maniera approfondita come si esegue la progettazione fisica sia per le basi di dati OLTP sia per quelle OLAP. In base alla cardinalità dei dati si scelgono infatti diversi indici.

Molti valori (OLTP) – pochi valori (OLAP)

Comunque, durante la progettazione bisogna tener conto del carico di lavoro della base dati e capire quali sono gli accessi più frequenti. L’accesso è tipicamente in sola lettura e le interrogazioni tipicamente leggono aggregati della tabella dei fatti che sono frazioni grandi della tabella dei dati.

Bisogna tener conto anche degli aggiornamenti della tabella dei fatti e quindi di tutte le strutture aggregate di supporto che devono essere “rinfrescate” continuamente.

-Per la progettazione fisica reale si guarda ciò che “è più importante”, come ad esempio le interrogazioni eseguite più frequentemente o che devono essere svolte dall’amministrazione (amministratore delegato). Comincio dunque a scegliere le strutture fisiche più adatte per quel tipo di interrogazioni in modo da accelerarle.

Il modello logico relazionale è indipendente dai dati e mi permette quindi, a posteriori, di effettuare delle fasi di tuning per ottimizzare il funzionamento del mio data warehouse.

Alimentazione dei Datawarehouse - [ETL](#)

L'alimentazione dei datawarehouse viene fatta una prima volta per il popolamento iniziale del datawarehouse e poi successivamente vengono effettuate delle alimentazioni giornaliere per gestire l'aggiornamento dei dati. Queste ultime devono avere un tempo di elaborazione molto più breve rispetto al primo caricamento.

Le fasi ETL si dividono in delle sottocategorie:

-Estrazione

- Statica: avviene al primo popolamento e fa una fotografia dei dati storici e non
- Incrementale: estrazione delle differenze per rintracciare i nuovi dati da aggiornare
Le modalità di estrazione dipendono dalla natura dei dati (storici, semi-storici, transitori)

Ci occuperemo principalmente dell'estrazione incrementale che può essere eseguita in diverse modalità:

- Modifica delle applicazioni OLTP: aumenta il carico applicativo, molto oneroso
- Uso dei file di LOG: non aumenta il carico applicativo ma se il file di log è in formato proprietario non posso accederci (preferibile se possibile)
- Definizione di Trigger: è come un campanello di allarme che esegue una parte di codice quando arriva una modifica sulla base dati salvandola opportunamente (non modifica il carico applicativo ma scrive sulla base dati)
- Timestamp: I record operazionali modificati sono marcati con il timestamp dell'ultima modifica. È oneroso e in più mi perdo le modifiche multiple nello stesso giorno per la stessa tupla perché tengo traccia solo dell'ultima modifica.

Tutte queste modifiche sono transazionali quindi si occupano di scrivere la modifica nel momento in cui questa avviene.

Le operazioni di aggiornamento sono:

- Delete
- Aggiornamento
- Inserimento

-Pulitura – Data cleaning

Sono le operazioni destinate al miglioramento della qualità dei dati. I dati solitamente sono "sporchi" per varie motivi come: errori di battitura, incongruenza nella progettazione dei campi attributi nei diversi database aziendali, errori di registrazione etc etc..

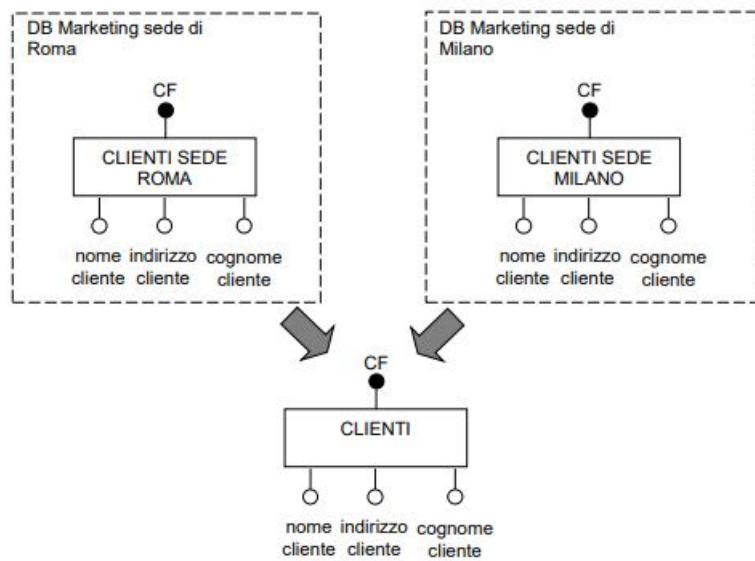
Per evitare problemi è consigliabile non utilizzare campi d'inserimento di testo libero nelle interfacce, in modo che gli utenti siano vincolati ad una selezione precisa dell'opzione. Ciò può essere implementato usando dei menù a tendina o dei box di selezione.

Per ripulire dei dati già sporchi si usano dei dizionari che rintracciano il root della parola inserita e la riconducono alla dicitura interessata (sono chiamate tecniche di fusione approssimata). Un'altra

tecnica è l'uso dei domini ragionevoli che permette di escludere l'inserimento di valori fuori da un range ragionevole. (esempio impiegato con stipendio di C. Ronaldo)

Spesso l'inconsistenza dei dati si genera quando siamo obbligati ad effettuare dei join approssimati che usiamo quando vogliamo incrociare delle tabelle tramite l'uso di attributi comuni che non sono però la chiave primaria o la chiave esterna.

Tale operazione genera però il problema del Purge/Merge che prevede l'eliminazione dei dubbi e la correzione degli errori di battitura successivamente alla generazione delle tabelle create dai Join Approssimati.



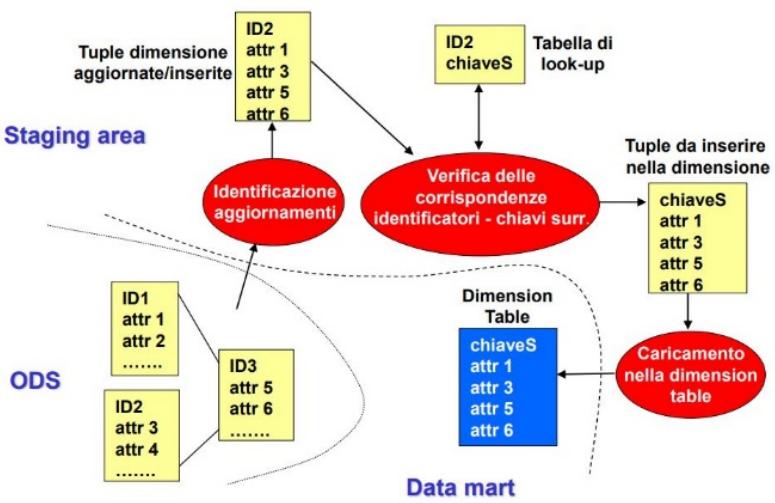
-Caricamento

In questo passo sono necessarie le proprietà ACID (proprietà transazionali dei dati).

Per mantenere l'integrità dei dati devo seguire un ordine di caricamento definito:

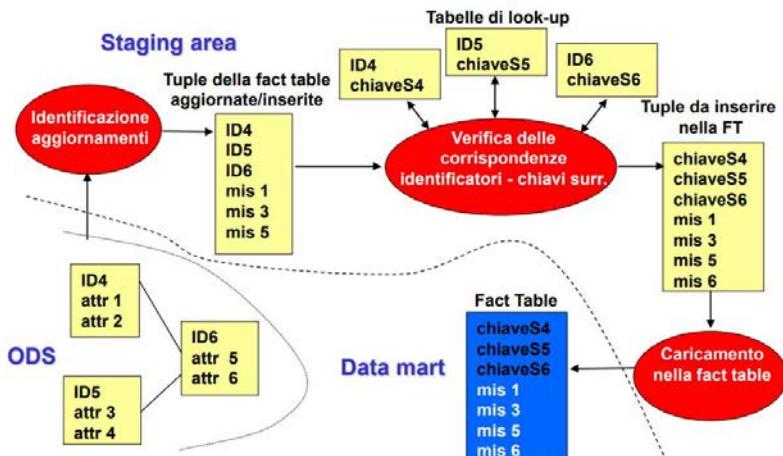
1. Dimensioni
2. Tabelle dei fatti
3. Viste materializzate e indici

Alimentazione delle dimensioni



Per l'aggiornamento delle dimensioni dobbiamo prelevare i dati dalle sorgenti, identificare gli aggiornamenti e creare una tabella relazionale adeguata. Successivamente bisognerà sostituire le chiavi primarie relazionali con dei codici identificativi (contatori) che ci permettono la gestione del tempo per storicizzare un dato. Se usassi la chiave primaria relazionale dovrei sovrascrivere il vecchio dato, in questo modo invece riesco, aggiornando il contatore, a mantenere anche il dato passato.

Alimentazione delle tabelle dei fatti



Il processo è uguale a quello per l'alimentazione delle dimensioni solamente che qui devo sostituire più chiavi primarie con più contatori.

Alimentazione delle viste materializzate

Per aggiornare le viste materializzate posso agire direttamente sui dati al livello più basso oppure posso rifarmi alle viste di livello più basso. Posso quindi effettuare un aggiornamento utilizzando i dati aggregati delle viste materializzate a livello più basso.

10. Tecniche di Analisi dei dati

Le tecniche di analisi dei dati possono essere di due tipi:

- Descrittive (esplorare i dati e conoscerli meglio): danno delle rappresentazioni che descrivono i dati in esame, ad esempio le tecniche di clustering
- Predittivi (predire il tipo di dato): usano la conoscenza di alcune variabili del problema per predire una variabile di interesse. (variabile target) L'esempio che si usa è l'etichetta del filtro antispam

Gli algoritmi che studieremo saranno di:

- Associazione
- Classificazione
- Clustering

11. Preparazione dei dati e tipi di dato

I dati possono essere di **diversi tipi**:

- Tabella: contengono diversi tipi di attributi (identificatore, booleani, categorico o nominale, numerico, etichetta di classe (può essere binario o di diverso tipo))
- Documento: sono testi, insiemi di parole non ordinate. Posso contare la frequenza delle singole parole che devono però prima essere processate riconducendo le parole al root, eliminando: articoli, maschili, femminili, preposizioni etc etc..
- Transazionali: sono dati speciali caratterizzati dalla presenza di set di oggetti. Possiamo dunque vederli come insiemi di insiemi. Le righe sono di lunghezza variabile.
- A Grafo: ultimamente nella ricerca si sta lavorando molto per i dati a grafo.
- Chemical Data: ad esempio se vogliamo descrivere la struttura delle molecole
- Ordered data: I dati sono memorizzati in maniera ordinata in modo da poter fare degli studi sull'ordine. Si usa ad esempio per studiare le sequenze delle proteine o del DNA.

Anche gli attributi possono avere tipologie differenti:

- Nominale o Categorico: ad esempio lo stato civile, possiamo fare un confronto di tipo “uguale” o “diverso”
- Ordinali: ad esempio la taglia delle magliette. Sono ordinabili grazie a questi simboli.
- Numerici: posso essere di due tipologie:
 - Intervallo: tipo le date, posso fare operazioni di differenza
 - Ratio: posso fare tutte le operazioni, ad esempio le lunghezze

Differenziamo anche gli attributi se sono continui o discreti.

Nei dati possono essere presenti delle eccezioni come dei dati mancanti. In questo caso gestisco i dati mancanti: eliminarli, ignorarli, stimare il valore mancante, rimpiazzo con una distribuzione di valori.

Tecniche per l'elaborazione dei dati - Preprocessing

Prima dell'immissione dei dati dentro il data warehouse, avvengono dei passi di preprocessing che ci permettono di elaborare i dati e renderli più gestibili.

Le tecniche che ci permettono di elaborare i dati per ridurne i volumi sono:

- Aggregazione: Combino due o più attributi in un unico attributo. Tale operazione mi permette di risalire lungo la gerarchia cambiando quindi la scala di valutazione. Ad esempio, posso valutare un fenomeno in base al mese. Successivamente potrò aggregare per anno e quindi cambiare la scala in annuale valutando la comparsa di tale fenomeno lungo il corso dell'anno.
- Riduzione del volume dei dati: Spesso è importante ridurre il volume dei dati per renderli più gestibili. Abbiamo diverse tecniche per ridurre il volume dei dati:
 - Campionamento: Riduciamo la cardinalità della tabella diminuendo il **numero di righe**
 - Feature Selection: Riduciamo **le colonne** della tabella diminuendo gli attributi considerati. In questo modo possiamo velocizzare i nostri algoritmi e inoltre, possiamo rendere le misure delle differenze più accurate durante lo studio di Clustering
 - Discretizzazione: Ridurre la **cardinalità del dominio** (ad esempio dividere per fasce d'età)

Campionamento

Questo processo riduce il numero di righe della tabella.

Il campionamento può avvenire perché il processo di campionamento costa (ISTAT) oppure perché non si riesce a gestire il volume di dati che si ha.

L'insieme campionario deve avere il più possibile le stesse proprietà dell'insieme di partenza. Il campionamento classico è quello casuale che può essere:

- con rimpiazzamento: non cambia la probabilità di estrazione dei campioni ancora da estrarre ma nel campionamento finale devo accettare di avere dei possibili duplicati. Si usa per classi molto piccole. (*prendo un campione e lo ributto dentro*)
- senza rimpiazzamento: cambia la probabilità di estrazione dei campioni ancora da estrarre
- stratificato: è il più utilizzato nell'ambito del Machine Learning. Consiste nel creare un campione che mantiene le stesse proporzioni dell'insieme campionario relativamente ad un attributo prescelto, ad esempio l'etichetta di classe

Feature Selection

Questo processo riduce il numero di colonne della tabella.

Vi sono varie tecniche che permettono di ridurre il numero di dimensioni grazie alla ricerca della combinazione lineare fra due attributi, rendendolo così un solo attributo. ([PCA](#))

Possiamo anche ridurre l'informazione ridondante eliminando gli attributi che sono ricavabili da altri attributi dentro la base dati. Anche gli identificatori sono da eliminare. Questi sono degli attributi che tipicamente disturbano il processo di analisi. Ad esempio, se devo fare uno studio degli studenti che possono fare bene al politecnico, non avrò bisogno dell'informazione della matricola per dedurre ciò che sto cercando.

Gli algoritmi che fanno estrazione delle features sono:

- Brute force: È la prova di tutte le combinazioni delle features, cercando così quelle che offrono risultato d'analisi migliori.
- Embedded: La features selection avviene all'interno dell'algoritmo di Data Mining
- Wrapper: Prova dei set di features ma, a differenza di brute force, questi vengono scelti in base a dei criteri greedy che dovrebbero selezionare le features ragionevolmente più utili.
- Features Creation: Questa tecnica NON scarta nessuna features. Consiste nel creare una feature di sintesi che mi permette di riassumere l'insieme di partenza
- Mapping data to a new space: trasformiamo i dati da uno spazio ad un altro spazio. Ad esempio la trasformata di fourier

Discretizzazione

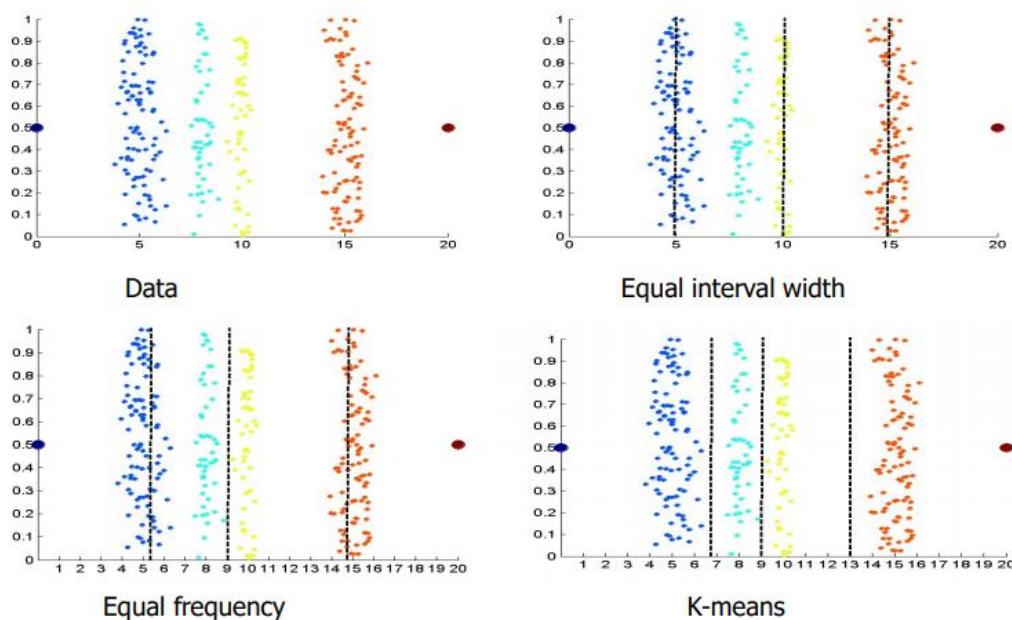
È una fase che mi permette di ridurre la cardinalità del dominio d'analisi.

Utilizzando degli insiemi, li chiameremo ‘bucket’, che conterranno i dati da analizzare.

Suddivideremo quindi i dati in bucket. Un esempio potrebbe essere la taglia di una maglietta (S,M,L) che grazie ad una suddivisione del dominio ci permette di capirne immediatamente le dimensioni. Quando comperiamo un capo d'abbigliamento non ci interessiamo infatti di grandezze con granularità assoluta ma utilizziamo la taglia per capire se il vestito potrebbe essere delle dimensioni corrette o meno.

Il principio da utilizzare è lo stesso ma bisogna scegliere opportunamente come delimitare i miei bucket. Esistono più modi per delimitare i miei intervalli di discretizzazione (bucket):

- Equal Interval Width (fasce equi spaziate): È la separazione degli intervalli di discretizzazione in egual grandezza. Il dominio di partenza viene dunque suddiviso in n intervalli equi spaziati. Tale sistema è estremamente sensibile alla presenza di outlier sugli estremi. Infatti, in presenza di una situazione simile, alcuni intervalli di integrazione potrebbero rimanere vuoi poiché un outlier influenzerebbe la posizione degli intervalli. Questo modo di partizionamento è statico, quindi è semplice effettuare l'assegnazione dei nuovi valori. (No sensibile outlier, No incrementale)
- Equal Frequency: Divido gli intervalli di discretizzazione in modo che contengano lo stesso numero di elementi. In questo modo la frequenza degli elementi all'interno di ogni bucket sarà la stessa ed eviterò il problema degli outlier. Questa tecnica ha però un difetto riguardo l'inserimento dei nuovi dati. Inserire nuovi dati cambierà la distribuzione dei valori quindi i miei bucket non saranno più ad uguale frequenza. Dovrò quindi effettuare più volte il processo di bucketizzazione per garantire una discretizzazione adeguata. (No sensibile outlier, No incrementale)
- Algoritmo di Clustering Monodimensionale: Risolve il problema della corretta separazione dei bucket in modo da garantire attributi “vicini” nello stesso bucket e attributi “lontani” in domini diversi. Tale algoritmo divide i cluster in maniera semantica in base all'attributo considerato. (No sensibile outlier, No incrementale, corretta separazione)



Come riportare i valori per il funzionamento degli algoritmi

I valori degli attributi da analizzare devono essere riportati in una certa scala in modo permettere il corretto funzionamento di alcuni algoritmi. Ad esempio, posso trasformare la scala di misura dei valori oppure posso normalizzare tali valori per ricondurli ad un certo intervallo noto (ad esempio -1, +1) per permettere di avere la stessa scala di variazione fra attributi diversi. La procedura di normalizzazione è fondamentale per il calcolo della distanza fra gli attributi. La tecnica z-score riporta i dati a media nulla e deviazione standard=1.

Misure di distanza

Siamo interessati a misurare la somiglianza o la dissimiglianza fra due attributi. Per far ciò dobbiamo tenere in considerazione la tipologia degli attributi.

| Attribute Type | Dissimilarity | Similarity |
|-------------------|---|---|
| Nominal | $d = \begin{cases} 0 & \text{if } p = q \\ 1 & \text{if } p \neq q \end{cases}$ | $s = \begin{cases} 1 & \text{if } p = q \\ 0 & \text{if } p \neq q \end{cases}$ |
| Ordinal | $d = \frac{ p-q }{n-1}$ (values mapped to integers 0 to $n-1$, where n is the number of values) | $s = 1 - \frac{ p-q }{n-1}$ |
| Interval or Ratio | $d = p - q $ | $s = -d, s = \frac{1}{1+d}$ or $s = 1 - \frac{d - \min_d}{\max_d - \min_d}$ |

Se abbiamo dei punti caratterizzati da n attributi possiamo usare tecniche differenti per calcolarne la distanza fra essi:

- Distanza Euclidea: È il calcolo della differenza della coordinata fra p e q per il k -esimo attributo, la elevo al quadrato, le sommo per tutti i k attributi e poi ne estraggo la radice quadrata. In caso di attributi con scala differente di ordini di grandezza (età, stipendio), il calcolo della distanza euclidea sarà spostato verso l'attributo più grande in modulo. Quindi devo, prima di calcolare la distanza, normalizzare i dati per confrontare più attributi, eliminando così i pesi relativi.
 - Minkowski Distance: È una generalizzazione della distanza euclidea. In base al valore di r si hanno dei casi degeneri. Quando $r \rightarrow \infty$ allora si parla di distanza suprema che consiste nel
 - Euclidean Distance
- $$dist = \sqrt{\sum_{k=1}^n (p_k - q_k)^2}$$
- Where n is the number of dimensions (attributes) and p_k and q_k are, respectively, the k^{th} attributes (components) or data objects p and q .
- Normalization is necessary, if scales differ.
 - Minkowski Distance is a generalization of Euclidean Distance
- $$dist = \left(\sum_{k=1}^n |p_k - q_k|^r \right)^{\frac{1}{r}}$$
- Where r is a parameter, n is the number of dimensions (attributes) and p_k and q_k are, respectively, the k^{th} attributes (components) of data objects p and q .

prendere il massimo valore di distanza fra i punti ovvero gli attributi più distanti fra i due punti considerati.

La distanza è sempre una grandezza positiva, simmetrica e vale la proprietà triangolare.

Per il calcolo della distanza fra variabili booleane non possiamo utilizzare il calcolo classico. Nei casi simili al test di screening, bisogna differenziare i pazienti in base al fatto che siano positivi ad un determinato test e non negativi. Quindi è più rilevante l'informazione positiva rispetto a quella negativa. In questo caso calcoliamo la distanza eliminando l'informazione non interessante (M00). Tale tecnica prende il nome di distanza di Jaccard.

Nel caso di variabili numeriche, invece che booleane, si calcola la distanza grazie al coseno dell'angolo formato fra i vettori v_1 e v_2 .

Volendo possiamo anche introdurre dei pesi se sappiamo che certi attributi sono più importanti di altri.

12. Regole di Associazione

Sono regole che permettono di estrarre delle situazioni di correlazione fra oggetti all'interno di una base dati transazionale (OLTP). Ad esempio, una base dati con gli scontrini di cassa di un supermercato.

Gli oggetti all'interno di tale base non sono ordinati, li denomineremo *Item*. Ogni transazione è un insieme di Item non ordinati.

- A collection of transactions is given
 - a transaction is a set of items
 - items in a transaction are *not ordered*
- Association rule
 - $A, B \Rightarrow C$
 - $A, B = \text{items in the rule body}$
 - $C = \text{item in the rule head}$
- The \Rightarrow means co-occurrence
 - *not* causality
- Example
 - coke, diapers \Rightarrow milk

| TID | Items |
|-----|----------------------------|
| 1 | Bread, Coke, Milk |
| 2 | Beer, Bread |
| 3 | Beer, Coke, Diapers, Milk |
| 4 | Beer, Bread, Diapers, Milk |
| 5 | Coke, Diapers, Milk |
| ... | ... |

Con queste tecniche cercheremo di capire com'è strutturata una base dati, la predizione di un'informazione verrà studiata successivamente.

Tali tecniche sono adatte, oltre allo studio market basket, anche allo studio dei dati testuali e i dati all'interno di tabelle strutturate.

Analisi dei testi

Si osserva l'occorrenza delle parole di un testo e si cerca di capire se alcune parole hanno occorrenza dipendente o meno.

Analisi dei dati all'interno di tabelle strutturate

Possiamo osservare il valore dei vari campi di una riga estratta della tabella cercando di estrapolare delle regole di occorrenza, o dei legami, fra i valori degli attributi studiati.

Definizioni

Itemset: un set che include uno o più item

k-itemset: è un itemset che contiene k elementi. K=cardinalità dell'itemset

support count: conteggio dell'occorrenza di un'itemset

Supporto: È la frequenza relativa di un dato Item / rispetto alla dimensione della base dati.
(support count/dim DB)

Itemset Frequenti: Sono itemset la cui frequenza all'interno della base dati è superiore ad una certa soglia di supporto. Lo uso per selezionare un numero di risultati gestibili.

Regola (di associazione): Un certo item-set comporta un certo item.

Misura della qualità della regola – definizioni

Supporto (della regola): è la frazione delle transazioni contenenti A e B (insieme) rispetto alla cardinalità totale del server transazionale. Dove A e B sono Itemset (non item).

$$\frac{\#\{A,B\}}{|T|}$$

- $|T|$ is the cardinality of the transactional database
- a priori probability of itemset AB
- rule frequency in the database

Confidenza: È la frequenza della testa della regola (B) nelle transazioni che contengono il corpo (A). $A \Rightarrow B$. È la misura di correlazione fra gli item. B include A!

- Confidence is the frequency of B in transactions containing A
- conditional probability of finding B having found A
- "strength" of the " \Rightarrow "

$$\frac{\text{sup}(A,B)}{\text{sup}(A)}$$

È come il calcolo della probabilità condizionata.

- From itemset {Milk, Diapers} the following rules may be derived
- Rule: Milk \Rightarrow Diapers
 - support
 $\text{sup} = \#\{\text{Milk}, \text{Diapers}\} / \#\text{trans.} = 3/5 = 60\%$
 - confidence
 $\text{conf} = \#\{\text{Milk}, \text{Diapers}\} / \#\{\text{Milk}\} = 3/4 = 75\%$

| TID | Items |
|-----|----------------------------|
| 1 | Bread, Coke, Milk |
| 2 | Beer, Bread |
| 3 | Beer, Coke, Diapers, Milk |
| 4 | Beer, Bread, Diapers, Milk |
| 5 | Coke, Diapers, Milk |

Come estraggo le regole di correlazione?

Prima devo generare gli Item-Set frequenti che soddisfano delle condizioni di soglia sul supporto e sulla confidenza e successivamente posso estrarre le regole di associazione.

Come faccio a generare gli Item-Set frequenti? Vi sono varie modalità:

- Brute-force: genero tutte le possibili combinazioni degli itemset e poi vado a controllare quali soddisfano i requisiti di confidenza richiesti. Questa tecnica è troppo dispendiosa dal punto di vista computazionale.

- Dividere il processo di estrazione: Prima generiamo le permutazioni degli Item-set frequenti e poi da questi generiamo le regole di associazione che soddisfano il vincolo di confidenza

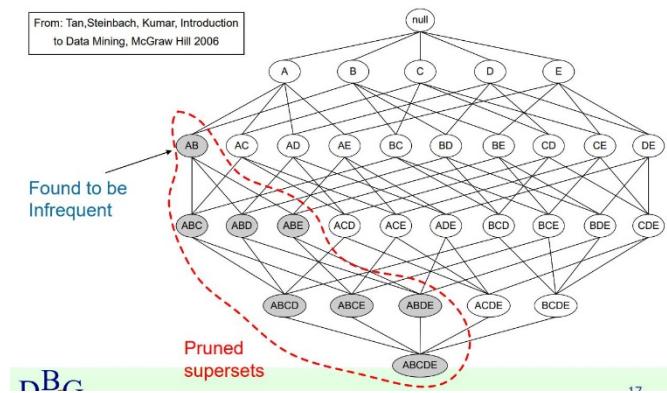
13. Algoritmi per la generazione degli Item-Set frequenti

Apriori

Devo ridurre lo spazio di ricerca dei candidati per gli item-set frequenti. Un algoritmo che ci permette di generare gli Item-Set frequenti è l'algoritmo "APRIORI".

- Funzionamento APRIORI:

- Se un item-set lungo k è frequente allora tutti i suoi sottoinsiemi devono essere frequenti
- Se un certo insieme non è frequente allora tutti i suoi soprainsiemi non saranno frequenti
- Il supporto di un item-set non può essere mai più elevato dei suoi sottoinsiemi. (supporto antimonotono)



Uso tali affermazioni per potare l'albero di ricerca dei miei item-set frequenti.

Tale algoritmo al passo K, calcola gli itemset del passo k+1 che potrebbero essere frequenti (candidati) e successivamente calcola la frequenza effettiva facendo una scansione sulla base dati per i candidati frequenti in modo da decidere se sono veramente frequenti oppure no. Quando scelgo i candidati devo potare la base dati come visto nel grafico sopra.

I passi dell'algoritmo Apriori sono:

- Join: al passo k genero tutti i candidati di lunghezza k+1
- Prune: ai candidati applico apriori guardando se contengono sottoinsiemi (k-itemset) che non sono frequenti, in caso affermativo li scarto. Esempio: AB non frequente, ABD lo scarto perché contiene AB che non è frequente quindi anche ABD non sarà frequente.
- Scan DB: scandisco la base dati per contare se effettivamente i miei candidati sono frequenti

Tale algoritmo ha però numerosi problemi legati al fatto che al livello k=2 devo creare tutti gli itemset con il prodotto cartesiano. Inoltre, devo accedere più volte alla base dati per effettuare la lettura ogni volta che voglio controllare se un dato itemset è frequente o meno.

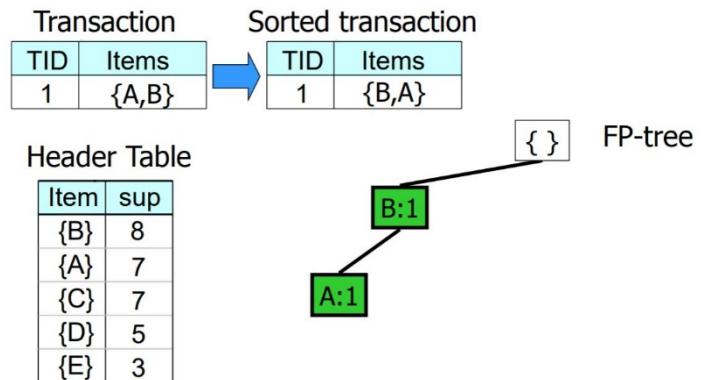
Fp-Growth

Questo algoritmo mi permette di generare gli item-set frequenti in maniera molto più efficiente rispetto all'algoritmo Apriori se il database è denso.

Per il funzionamento non si calcolano più i candidati ma si cerca di costruire in memoria una rappresentazione degli item frequenti tramite la costruzione di un albero chiamato FP-Tree con gli item frequenti nella base dati. Successivamente, grazie a delle visite ricorsive dell'albero FP Tree, l'algoritmo riesce ad estrapolare gli itemset frequenti. Servono 2 scansioni della base dati per il funzionamento dell'algoritmo.

I passi sono:

- Prendo gli itemset facendo prune degli item sotto la soglia di supporto dove possibile. (soglia supporto)
- Costruisco un Header table dei singoli item ordinati in maniera decrescente per numero di supporto
- Costruzione dell'albero ordinando gli item dentro gli itemset in ordine decrescente del loro supporto, guardando la header table. Gli item più frequenti saranno verso la root dell'albero. Ogni nodo dell'albero corrisponde ad un item e ha un contatore che indica l'occorrenza dell'item.
- Aggiungo dei puntatori per ogni item



Per estrarre gli itemset frequenti devo:

- Scandire la header table partendo dall'item con il supporto più basso
- Per ogni item i nella header table devo estrarre gli itemset frequenti che includono l'item i e gli item precedenti. Parto quindi dalla foglia e procedo verso la root dell'albero.
- Richiamo ricorsivamente il processo per ogni item i della header table.

estrarre (conditional patthern base) tutti i prezzi di transazione che contengono l'item selezionato e tutti gli item al di sopra nell'albero. Genererò quindi un altro FP-Tree partendo dall'item considerato e cercando le transazioni più frequenti.

ECLAT - Soluzioni complementari a Fp-Growth

Nel caso in cui il dataset è molto sparso possiamo utilizzare delle tecniche modificate di fp-growth che permettono una visualizzazione verticale delle occorrenze dei Transition ID. Avrò poche righe in caso di dataset sparsi. Per trovare gli Item-set frequenti farò l'intersezione delle Transition ID.

| TID | Items |
|-----|---------|
| 1 | A,B,E |
| 2 | B,C,D |
| 3 | C,E |
| 4 | A,C,D |
| 5 | A,B,C,D |
| 6 | A,E |
| 7 | A,B |
| 8 | A,B,C |
| 9 | A,C,D |
| 10 | B |

| A | B | C | D | E |
|---|----|---|---|---|
| 1 | 1 | 2 | 2 | 1 |
| 4 | 2 | 3 | 4 | 3 |
| 5 | 5 | 4 | 5 | 6 |
| 6 | 7 | 8 | 9 | |
| 7 | 8 | 9 | | |
| 8 | 10 | | | |
| 9 | | | | |

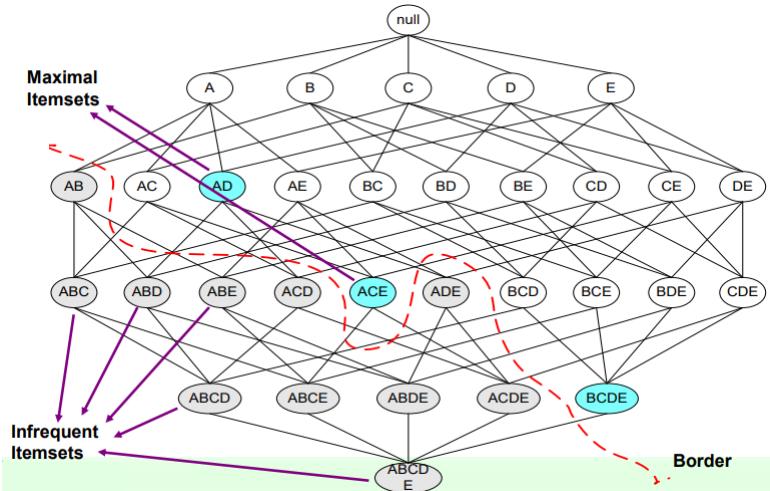
Rappresentazione Compatta

In alcuni casi, se il supporto è basso, possiamo estrarre un elevatissimo numero di ItemSet frequenti. Per prelevare informazioni riguardo le regole di associazione non servono sempre così tanti ItemSet, negli anni si è dunque cercato di estrarre solo un quantitativo di ItemSet "più rilevanti". Le tecniche utilizzate sono varie:

Itemset Frequenti Massimali

Un ItemSet frequente massimale è l'ItemSet più lungo, per quell'itemset lì, sopra la soglia di supporto. Non esisterà quindi un ItemSet soprainsieme dell'ItemSet considerato sopra la soglia di supporto.

Sono stati definiti degli algoritmi per l'estrazione degli ItemSet massimali. Questa tecnica ha però lo svantaggio che, una volta estratto un ItemSet massimale non conosco più il supporto di tutti gli altri ItemSet, anche di quelli che compongono il massimale. In questo modo sto riducendo la quantità d'informazione ma sto perdendo traccia del supporto degli ItemSet non massimali.



Closed ItemSet

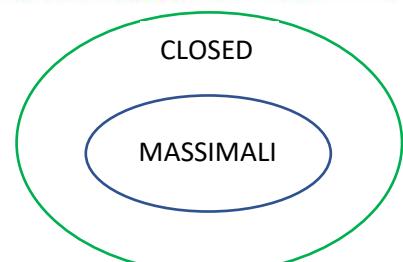
Un ItemSet *Closed* è definito come un Itemset dove nessuno dei suoi soprainsiemi ha lo stesso supporto dell'ItemSet che sto considerando. Ad esempio, se $\{B\}$ ha supporto 5, $\{A,B\}$ ha supporto 4 e $\{B,C\}$ ha supporto 3 allora $\{B\}$ sarà *closed*. Devo guardare solo gli ItemSet appena successivi, il fatto che B sia *closed* non è influenzato dal supporto di $\{B,C,D\}$.

In questo caso $\{A,B,D\}$ è *closed* perché ha supporto 3 e il suo soprainsieme è $\{A,B,C,D\}$ che però ha supporto 2. Anche $\{B,C,D\}$ ha come soprainsieme $\{A,B,C,D\}$ quindi anche lui sarà *closed*.

Tutti gli ItemSet Massimali sono anche Closed quindi i closed sono un soprainsieme dei massimali.

Grazie a delle tecniche supplementari posso ricavare il supporto di tutti gli ItemSet frequenti partendo dai *closed* ItemSet.

| itemset | sup |
|---------------|-----|
| $\{A,B,C\}$ | 2 |
| $\{A,B,D\}$ | 3 |
| $\{A,C,D\}$ | 2 |
| $\{B,C,D\}$ | 3 |
| $\{A,B,C,D\}$ | 2 |



La scelta della soglia di supporto

La soglia di supporto è la nostra leva per rendere il problema della ricerca degli ItemSet frequenti gestibile a livello computazionale. Ciò che ricerchiamo è però legato alla correlazione fra gli item il cui indice è rappresentato dalla Confidenza. Se scegliamo una soglia di supporto elevata rischiamo di perdere ItemSet a confidenza elevata ma sotto la soglia di supporto. Tali informazioni sono le più ricercate poiché una correlazione inattesa è proprio ciò che ricerchiamo durante lo studio di correlazione. Bisogna quindi scegliere un valore opportuno della soglia di supporto in modo da rendere il problema gestibile a livello computazionale ma allo stesso tempo utile per ritrovare correlazioni non scontate fra gli ItemSet.

Misure d'interesse

Nel valutare le regole di associazione trovate dobbiamo applicare delle misure che ci permettono di capire se un dato pattern (regola di associazione, ItemSet) è interessante o meno.

Le misure applicate al riguardo possono essere di tipi differenti:

- Oggettive: sono statistiche sui dati, come ad esempio il Supporto e la Confidenza
- Soggettive: sono difficili da quantificare. Un'ItemSet è interessante se è inatteso, quindi contraddice la mia aspettativa. È anche importante poter sfruttare l'informazione estratta (actionable) per chi ha commissionato l'analisi.

Noi ci concentreremo sulle misure Oggettive.

Affidabilità della Confidenza

Quando la testa della regola è molto frequente allora la confidenza non restituisce un valore attendibile. Nel caso in esempio:

- 3750 persone mangiano cereali
- 3000 persone giocano a basket
- 5000 persone totali
- 2000 mangiano cereali e giocano a basket

| | basket | not basket | total |
|-------------|--------|------------|-------|
| cereals | 2000 | 1750 | 3750 |
| not cereals | 1000 | 250 | 1250 |
| total | 3000 | 2000 | 5000 |

In questo esempio si osserva che per la regola

play basket \Rightarrow eat cereals

$$\text{supporto} = \text{testa+corpo/tutti} = 2000/5000 = 40\%$$

$$\text{confidenza} = \text{testa+corpo/corpo} = 2000/3000 = 66,7\%$$

Tale regola di correlazione sembra dunque ragionevolmente interessante poiché ha una confidenza e un supporto alto. In realtà si osserva che *eat cereals* ha un supporto molto elevato. maggiore della confidenza della regola trovata poco fa) supporto=3750/5000=75% *eat cereals*. Quindi la regola play basket \Rightarrow eat cereals non ci interessa perché la testa delle regole è molto frequente (compare dappertutto). In questi casi utilizziamo un'altra grandezza chiamata correlazione.

Correlazione (or lift)

Tale grandezza si usa quando non possiamo utilizzare la confidenza.

La correlazione mi dice quanto è diversa la probabilità congiunta di trovare A e B rispetto a quella che io posso stimare facendo l'ipotesi che A e B siano indipendenti.

$$\text{Correlation} = \frac{P(A, B)}{P(A)P(B)} = \frac{\text{conf}(r)}{\text{sup}(B)}$$

$r: A \Rightarrow B$

La correlazione ha significati differenti in base al valore che ottiene:

- Incorrelazione Correlazione = 1 :Vi è indipendenza statistica fra A e B
- Correlazione Positiva Correlazione > 1 :La probabilità di trovare A e B insieme è maggiore della probabilità di trovarli separati. Quando c'è A allora ci sta anche B .
- Correlazione Negativa Correlazione < 1 :La probabilità di trovare A e B insieme è minore della probabilità di trovarli separati. Quando c'è A allora non ci sta B .

Misure di Peso

Quando valuto gli items posso essere interessato a considerare il peso. Ad esempio, se in una regola di correlazione noto la presenza di un item (acqua minerale) potrò essere interessato a conoscere la quantità di quell'item acquistata, ovvero quante volte compare quell'item.

In questo caso si definisce un DataSet pesato dove ogni item dentro la transazione ha un peso. In questo caso bisognerà estendere le definizioni viste.

Considerare le Gerarchie

Data una gerarchia posso sfruttare l'informazione della gerarchia per estrarre delle regole associative.

Posso estrarre degli ItemSet dove gli item estratti sono generalizzati all'interno della gerarchia osservata, anche a livello variabile. In questo modo posso estrarre delle regole di correlazione anche su item non frequenti e sotto la soglia di supporto. Posso quindi reperire una conoscenza di livello più aggregato quando quella più bassa (in gerarchia) è troppo poco presente per emergere dalle associazioni.

In questi casi dobbiamo definire il supporto dell'ItemSet generalizzato che vogliamo valutare. Tale supporto possiamo definirlo chiedendoci se l'item che sto osservando nella transazione è figlio dell'ItemSet generalizzato. Ad esempio, il Vino Barbera sarà figlio di Vino che è più in alto nella gerarchia, quindi più generalizzato.

Se ho un DataSet molto sparso posso avere problemi nella scelta della soglia di supporto perché rischio di recuperare troppi ItemSet frequenti (soglia bassa) oppure posso facilmente perdermi delle informazioni di correlazione interessanti (soglia alta). In questi casi mi conviene cercare delle associazioni con una soglia di supporto generale e se non trovo ItemSet frequenti l'algoritmo sale di livello nella gerarchia finché poi non trova delle associazioni sopra la soglia di supporto. Così posso estrarre regole di livello: alto (generalizzate, item generalizzati), intermedio (fra item generalizzati e item dettagliati), basso (fra item di livello dettagliato nella gerarchia). Tipicamente

prima effettuo una ricerca delle regole di correlazione fra ItemSet generalizzati e poi faccio Drill Down cercando fra gli Item più in basso nella gerarchia, riducendo così opportunamente lo spazio del dominio di ricerca delle regole.

14. Classificazione

Gli algoritmi di classificazione servono per predire un'etichetta di classe. Partendo da un insieme di dati di addestramento detti di training (che sono già etichettati con l'etichetta di classe) gli algoritmi di classificazione imparano un modello che viene poi sfruttato per associare l'etichetta di classe a dei dati non ancora classificati (senza etichetta di classe).

Un esempio è il filtro antispam.

Posso anche usare il modello generato come descrizione delle categorie, ovvero le varie etichette di classe attribuibili. Quest'ultimo sistema è utilizzato per classificare e descrivere le categorie che sono costruite con algoritmi di Clustering.

Riassumendo gli obiettivi della classificazione:

- Attribuire l'etichetta di classe
- Descrivere le categorie

Per stimare la qualità del funzionamento del classificatore, usiamo un test set che viene usato solamente per stimare la bontà del modello.

Il test set è una parte di dati di cui io conosco l'etichetta di classe. Li faccio classificare dal mio modello e poi confronto l'etichetta di classe attribuita dal classificatore con la reale etichetta di classe che conoscevo dei dati del test set.

Vi sono vari algoritmi per gestire la classificazione dei dati. In base al problema da classificare vi sono algoritmi più adatti di altri.

- Decision trees
- Classification rules
- Association rules
- Neural Networks
- Naïve Bayes and Bayesian Networks
- k-Nearest Neighbours (k-NN)
- Support Vector Machines (SVM)

Valutazione di un algoritmo di Classificazione

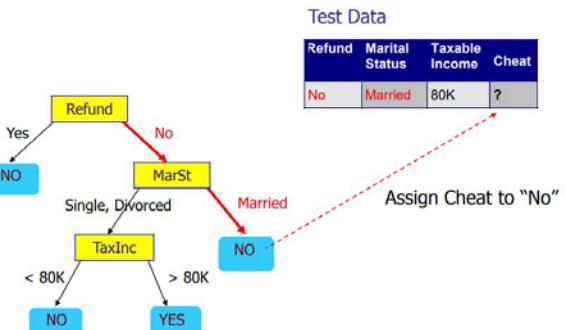
I parametri di confronto fra i classificatori sono:

- Accuratezza: qualità della predizione
- Efficienza: velocità di modellazione e di classificazione
- Scalabilità: nella gestione di DataSet sempre più grandi, anche nel numero di attributi
- Robustezza: come gestisco dati sporchi, rumorosi e mancanti
- Interpretabilità: Il modello mi permette di capire le caratteristiche delle classi? Lo capisco sulla minima istanza?

Albero di Decisione

È un metodo di classificazione basato sulla costruzione di un albero particolarizzato da nodi interni che sono associati ad un attributo e hanno un insieme di archi uscenti pari, al massimo, al numero di valori del dominio dell'attributo. Sugli archi possono esserci delle differenze che vedremo in seguito. I nodi foglia sono etichettati con un'etichetta di classe.

Partendo da un DataSet di partenza possiamo generare l'albero di decisione in tantissimi modi diversi. La struttura dell'albero cambia però l'efficienza del modello, è quindi opportuno costruirlo osservando dei valori che ci indicheranno come strutturarne la costruzione.



Per costruire l'albero di decisione ci sono molto algoritmi, noi studieremo Hunt's Algorithm.

Hunt's Algorithm

L'algoritmo di Hunt è il modo più semplice per generare un albero di decisione.

I passi per la costruzione dell'albero di decisione, a partire da un training Set, sono:

- Si osserva se la porzione di training Set contiene record appartenenti a più di una classe. In caso affermativo dovrà effettuare uno Split scegliendo un nuovo attributo su cui fare lo Split. A quel nodo associo l'attributo A su cui fare lo split (l'apertura degli archi uscenti)
- Cresce i vari archi uscenti, che possono essere da 2 a n (dove n è il numero massimo di valori nel dominio di quell'attributo).
- Ogni arco uscente avrà quindi una porzione di DataSet che soddisfa la condizione associata all'arco.
- Mi fermo quando:
 - tutto il mio training Set che arriva al nodo ha la stessa etichetta di classe. In questo caso sono in presenza di un nodo foglia che verrà etichettato con l'etichetta di classe della classe di appartenenza del training Set.
 - Posso anche fermarmi nel caso in cui ho finito gli attributi da osservare per cercare di partizionare il training set.
 - Posso fermarmi anche nel caso in cui non ho nessun valore del training Set che può intraprendere un cammino verso un determinato arco uscente, in questo caso creiamo un nodo default class che ha l'etichetta di classe più frequente all'interno del Training Set. In questo caso i dati di training non intraprenderanno mai il percorso verso la default class ma quando poi utilizzerò il modello sui data set reali potrò incontrare situazioni differenti.

Sto quindi immaginando di partizionare, nodo dopo nodo, il mio DataSet in modo da raggruppare, in base al valore assunto dagli attributi, i dati che hanno caratteristiche comuni rappresentate da una determinata etichetta di classe.

Questo algoritmo è di tipo greedy poiché la scelta dell'attributo da posizionare nel nodo avviene guardando l'ottimo locale, non garantendo quindi l'ottimo globale.

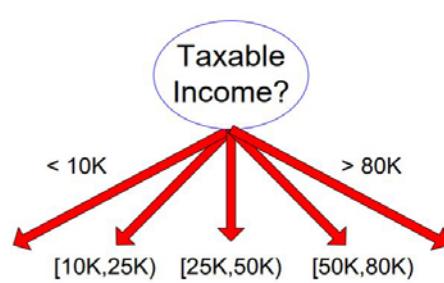
Struttura delle condizioni di Test

La struttura dell'albero è influenzata da come sono strutturate le condizioni di test di un determinato nodo. Le condizioni di test sono influenzate da:

- Tipologie di attributo
 - Nominale (categorico)
 - Ordinale (categorico con un ordine)
 - Continuo
- Da come voglio strutturare l'albero
 - Binary Split: Albero binario (forzo due uscite da ogni nodo), raggruppo una molteplicità di split diversi in un unico ramo. In questo caso devo valutare lo split migliore fra determinati attributi. Devo quindi effettuare una ricerca dell'ottimo.
 - Multi-way split: Albero arbitrario (n archi uscenti dal nodo), posso inserire un numero di archi uscenti pari ai valori assumibili nel dominio



(i) Binary split



(ii) Multi-way split

Split di un attributo continuo

Nel caso di un attributo continuo è più difficile effettuare uno split. Esistono due opzioni differenti:

- Discretizzo i valori continui creando così dei valori ordinali. Tale operazione viene effettuata prima di creare l'albero di decisione oppure può essere fatta runtime sulla partizione del DataSet “che arriva” (al nodo) direttamente al livello del nodo che deve valutare l'attributo interessato.
- Creo un albero binario cercando un valore che divide il dominio dell'attributo. In questo caso devo effettuare una ricerca del miglior valore da attribuire per effettuare uno split che mi possa partizionare il DataSet in modo corretto dal punto di vista della suddivisione dei valori delle etichette di classe. (migliore dal punto di vista qualitativo ma più dispendiosa dal punto di vista computazionale)

Scelta degli attributi su cui fare lo split

Quando devo effettuare lo split devo escludere le chiavi primarie e le chiavi candidate perché non generalizzano. Immaginando l'arrivo di un nuovo dato, questo avrà sicuramente una chiave primaria differente da quella su cui è stato costruito il modello quindi non riuscirà a classificarlo.

Vogliamo scegliere come attributi preferenziali quelli che ci generano una distribuzione dei record

sottostanti, dopo lo split, che sia omogenea sulla classe (più puri possibili). Ho bisogno di valutare la purezza di un nodo in modo da

effettuare un confronto fra gli split possibili, scegliendo così lo split con l'attributo che genera dei nodi più puri (distribuzione dei record sottostanti omogenea sulla classe).

Per valutare il grado di purezza esistono vari indici:

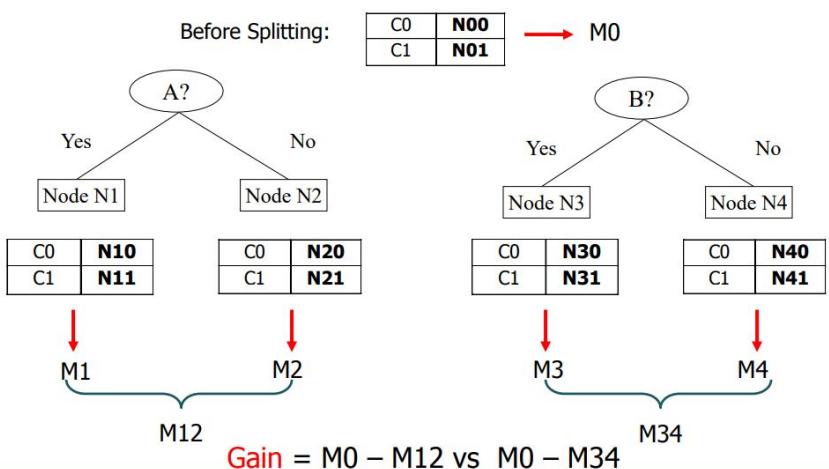
- [Gini index](#)
- [Entropia](#)
- [Misclassification error](#)

Dopo aver scelto il metodo di classificazione (albero di decisione, rete neurale etc..) devo anche scegliere anche i parametri di funzionamento (come gli indici del grado di purezza) e capire quale combinazione mi offre la performance migliore per il mio classificatore.

Come funziona il procedimento della scelta degli indici

Prima della partizione per un attributo (A) valuto la distribuzione delle classi tramite il calcolo di un indice di purezza (M0) partendo dal mio DataSet.

Eseguo lo split e valuto nuovamente la partizione delle classi per i nodi generati a valle del nodo relativo all'attributo precedente, calcolando così due nuovi indici di purezza M1 e M2.



Tale processo viene eseguito per tutti gli attributi che possiamo ancora considerare (B, C etc..) generando di passo in passo nuovi indici di purezza M3 e M4.

Dopo aver generato gli indici di purezza per tutti gli attributi possibili dovrò effettuare un confronto sugli split effettuati per ogni attributo. Genererò un indice di purezza composito, da M1 e M2 genererò M12 e da M3 e M4 genererò M34, che li racchiude tutti e due. In fine effettuiamo un confronto cercando quale split mi offre un *guadagno* maggiore, studiando la differenza fra l'indice di partenza M0 e l'indice composito M12 oppure M34.

Gini Index

Tale indice studia la frequenza relativa di ogni classe in ogni nodo.

In Gini index vale **0** quando siamo nel caso ottimale, ovvero la nostra partizione ha tutti elementi relativi ad una sola classe (situazione ideale) oppure vale **0.5** quando tutti gli elementi della partizione sono distribuiti in modo binario (caso peggiore) quindi le classi sono equilibrate.

Gini Index for a given node t

$$GINI(t) = 1 - \sum_j [p(j | t)]^2$$

$p(j | t)$ is the relative frequency of class j at node t

| | |
|-------------------|----------|
| C1 | 0 |
| C2 | 6 |
| Gini=0.000 | |

| | |
|-------------------|----------|
| C1 | 1 |
| C2 | 5 |
| Gini=0.278 | |

| | |
|-------------------|----------|
| C1 | 2 |
| C2 | 4 |
| Gini=0.444 | |

| | |
|-------------------|----------|
| C1 | 3 |
| C2 | 3 |
| Gini=0.500 | |

| | |
|----|----------|
| C1 | 0 |
| C2 | 6 |

$$P(C1) = 0/6 = 0 \quad P(C2) = 6/6 = 1$$

$$\text{Gini} = 1 - P(C1)^2 - P(C2)^2 = 1 - 0 - 1 = 0$$

| | |
|----|----------|
| C1 | 1 |
| C2 | 5 |

$$P(C1) = 1/6 \quad P(C2) = 5/6$$

$$\text{Gini} = 1 - (1/6)^2 - (5/6)^2 = 0.278$$

Valutare la bontà dello split effettuato su un nodo

Dopo aver calcolato il Gini Index per il nodo dovrò effettuare il calcolo del Gini Index totale dello split per capire la purezza reale dello split effettuato. Devo quindi valutare se quello split mi divide la partizione in maniera interessante dal punto di vista del numero di record che vengono suddivisi dallo split. Devo quindi cercare uno split che mi divida il DataSet in modo da avere multi elementi appartenenti alla stessa classe e inoltre deve garantire una buona quantità di partizione dei dati.

- Per fare ciò dobbiamo calcolare il Gini Index dei nodi

Prima calcolo il Gini index per i nodi N1 e N2 e poi calcolo il Gini Index dello split su B pesandolo in base al numero opportuno di elementi

$$\text{Gini}(N1) = 1 - (5/7)^2 - (2/7)^2 = 0.408$$

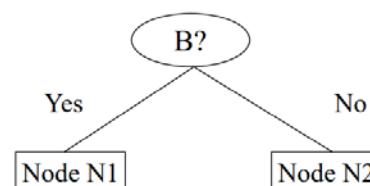
$$\text{Gini}(N2) = 1 - (1/5)^2 - (4/5)^2 = 0.32$$

where

$$n_i = \text{number of records at child } i$$

$$n = \text{number of records at node p}$$

| | Parent |
|---------------------|----------|
| C1 | 6 |
| C2 | 6 |
| Gini = 0.500 | |



| | N1 | N2 |
|---------------|----------|----------|
| C1 | 5 | 1 |
| C2 | 2 | 4 |
| Gini=? | | |

$$\text{Gini(split on B)} = \frac{7}{12} * 0.408 + \frac{5}{12} * 0.32 = 0.371$$

12 = numero di elementi totali

N1 e N2 e poi sommarli in modo pesato in base al numero di elementi che finiscono nel nodo N1 o N2 rispetto al numero di tuple totali. In questo modo abbiamo calcolato il Gini Index relativo all'intera partizione sull'attributo B.

Dovrò fare il calcolo del Gini per ogni split relativo ad ogni attributo e poi dovrò prendere quello dove guadagno di più in modo locale (algoritmi greedy), portandomi quindi il Gini più vicino al valore 0.

Gini index in base ai valori

Nel caso di attributi con più valori posso generare un nodo a più archi uscenti oppure posso generare più alberi binari diversi mettendo insieme più attributi in un unico ramo. Si osserva che nello split a più vie, il gini index è più basso o al massimo uguale alla versione binaria. Quindi questa soluzione è un lower bound dello split Two-way split.

| CarType | | | |
|---------|--------------|--------|--------|
| | Family | Sports | Luxury |
| C1 | 1 | 2 | 1 |
| C2 | 4 | 1 | 1 |
| Gini | 0.393 | | |

|

| CarType | | | |
|---------|---------------------|----------|--|
| | {Sports, Luxury} | {Family} | |
| C1 | 3 | 1 | |
| C2 | 2 | 4 | |
| Gini | 0.400 | | |

|

| CarType | | | |
|---------|--------------|---------------------|--|
| | {Sports} | {Family, Luxury} | |
| C1 | 2 | 2 | |
| C2 | 1 | 5 | |
| Gini | 0.419 | | |

Se voglio dividere in un albero binario, deve valutare le migliori permutazioni degli attributi e valutarne il gini index passo dopo passo in modo da capire quali insiemi di attributi posso racchiudere in un unico ramo.

Valutare uno split per un attributo continuo

Per valutare uno split in caso di un attributo continuo devo:

- Ordinare i valori dell'attributo di cui voglio fare lo split
 - Valutare di volta in volta un valore di split all'interno del dominio calcolando il Gini Index per ogni valore di separazione considerato. Devo quindi capire se la partizione è migliore o meno in base ai Gini Index calcolati.
 - Finito il calcolo scelgo lo split ideale in base ai gini index calcolato. Il valore calcolato è riferito all'ottimo locale per effettuare lo split e non su tutto il dataset. Quindi il valore su cui effettuare lo split è scelto in base alla porzione di Training Set che arriva a quel nodo.

| Cheat | No | No | No | Yes | Yes | Yes | No | No | No | No |
|----------------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| Taxable Income | | | | | | | | | | |
| → | 60 | 70 | 75 | 85 | 90 | 95 | 100 | 120 | 125 | 220 |
| → | 55 | 65 | 72 | 80 | 87 | 92 | 97 | 110 | 122 | 172 |
| → | <= > | <= > | <= > | <= > | <= > | <= > | <= > | <= > | <= > | <= > |
| Yes | 0 3 | 0 3 | 0 3 | 0 3 | 1 2 | 2 1 | 3 0 | 3 0 | 3 0 | 3 0 |
| No | 0 7 | 1 6 | 2 5 | 3 4 | 3 4 | 3 4 | 3 4 | 4 3 | 5 2 | 6 1 |
| Gini | 0.420 | 0.400 | 0.375 | 0.343 | 0.417 | 0.400 | 0.300 | 0.343 | 0.375 | 0.400 |

Sorted Values
Split Positions

Quando valuto lo split su 65 non devo rileggere la base dati ma, conoscendo l'etichetta di classe di 60, posso direttamente spostare una tupla da NO > 65 a NO <= 65.
Ricalcolando poi i Gini.

Entropia

È un altro modo, rispetto al gini index, per calcolare la purezza (split ideale) per un attributo in un albero decisionale.

$$Entropy(t) = -\sum_j p(j|t) \log_2 p(j|t)$$

| | |
|----|----------|
| C1 | 0 |
| C2 | 6 |

$$P(C1) = 0/6 = 0 \quad P(C2) = 6/6 = 1$$

$$\text{Entropy} = -0 \log 0 - 1 \log 1 = -0 - 0 = 0$$

Il guadagno è calcolato come la differenza fra l'entropia di partenza e l'entropia della situazione d'arrivo pesata per la frequenza dentro le varie partizioni di arrivo.

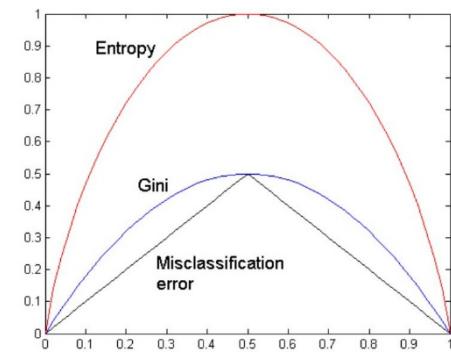
| | |
|----|----------|
| C1 | 1 |
| C2 | 5 |

$$P(C1) = 1/6 \quad P(C2) = 5/6$$

$$\text{Entropy} = -(1/6) \log_2 (1/6) - (5/6) \log_2 (5/6) = 0.65$$

L'entropia e il Gini index favoriscono una frammentazione pesante dell'albero poiché più le partizioni sono piccole e più sono pure. Per scoraggiare le partizioni pure troppo piccole si applica un fattore correttivo, chiamato SplitINFO, che serve ad evitare una frammentazione troppo accentuata dell'albero.

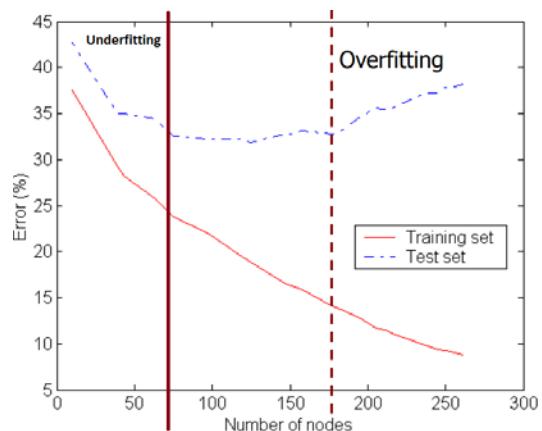
La massima disomogeneità per la classe (caso peggiore) è a valore 1 a differenza del gini index che è 0,5. Inoltre, L'entropia si sposta velocemente verso valori di caso migliore o di caso peggiore, ha infatti un andamento più rapido rispetto al Gini Index.



Condizioni di arresto. Underfitting & Overfitting

Certe volte è opportuno bloccare la crescita dell'albero di classificazione.

Per valutare la bontà del classificatore posso far classificare, una volta rimossa l'etichetta di classe, i dati del Training Set con i quali avevo costruito il modello del classificatore. Si osserva che l'errore di classificazione si riduce all'aumentare del numero di nodi all'interno del nostro modello ad albero di classificazione.



Tale procedimento però è relativo, infatti se effettuiamo lo stesso test di classificazione con un altro insieme di dati (test set) differente dai dati del training set, si nota che l'andamento del grafico relativo all'errore ha un comportamento differente.

Se nel modello di classificazione vi sono troppi nodi vi è un livello di dettaglio elevato (dettagli minimi). Tale situazione genera un aumento dell'errore durante la classificazione di nuovi insiemi di dati. Questa situazione prende il nome di Overfitting. Ciò avviene perché i dati del test set devono essere classificati in modo opportuno. Serve quindi una capacità di classificazione intermedia fra underfitting e overfitting in modo da classificare opportunamente dati differenti dal training set.

Quindi non dobbiamo generare né alberi con troppi nodi né alberi con troppi pochi nodi.

Come decido quando devo arrestarmi nella crescita dell'albero di classificazione?

L'arresto della crescita nella costruzione dell'albero di classificazione può avvenire per diversi motivi:

- Quando dentro la foglia dell'albero ho un numero di istanze minore o uguale ad una certa quantità.
- Test d'indipendenza statistica (solitamente chi-square) che serve per vedere se gli attributi rimasti sono dipendenti dall'etichetta di classe. Nel caso in cui fossero indipendenti allora non potrò più partizionare quindi arresterò il processo di costruzione.
- Posso vedere il guadagno (differenza fra i gini) e se il è troppo basso posso fermarmi

Per la configurazione dell'albero devo decidere tutti questi parametri d'arresto, scegliendo i numeri soglia che mi permetteranno di arrestare la costruzione dell'albero di decisione.

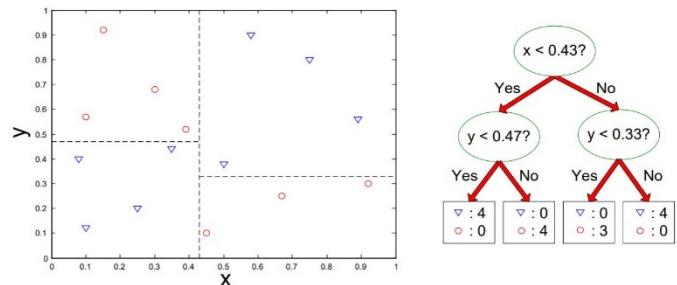
Trovare la configurazione migliore per la gestione dell'albero è molto difficile.

Vi è anche una tecnica (Post pruning) che si basa sulla costruzione totale dell'albero per poi successivamente provare a potare dei rami valutando di volta in volta la crescita dell'errore

I problemi degli alberi di decisione

Gli alberi di decisione hanno problemi quando bisogna classificare dei dati particolari. Ad esempio, la classificazione di un dato che ha un attributo nullo è problematica per la struttura ad albero di decisione. Sia per la costruzione del modello, sia per la classificazione futura.

Per effettuare la classificazione, l'albero di decisione crea delle partizioni dello spazio generando una separazione ortogonale alla variabile che sto guardando e parallela a tutte le altre. In caso di spazi dove la classificazione deve avvenire in modo obliquio non posso quindi utilizzare un albero di decisione normale per classificare i dati.



Riepilogo alberi di decisione

Pro:

Contro:

| | |
|------------------------|---|
| Veloci da costruire | Accuratezza disturbata da dati mancanti |
| Classificazione veloce | |
| Facile interpretazione | |
| Accuratezza discreta | |

Random Forest

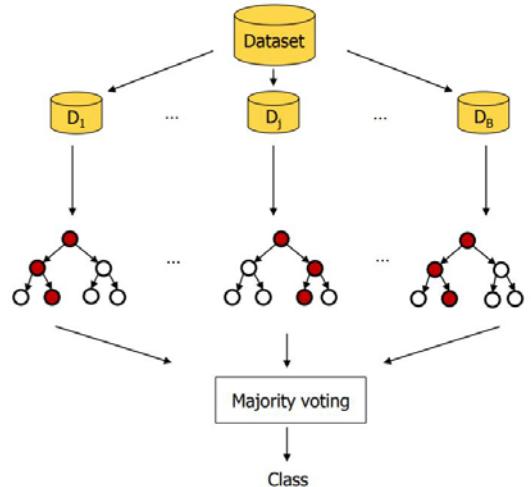
Il concetto è quello di usare più alberi di decisione invece di uno. In questo modo potrà utilizzare la classificazione di più alberi di lunghezza e forma differenti in modo da poter ottenere una classificazione più accurata a maggioranza. La decisione estrapolata sarà quindi presa a maggioranza in base alle valutazioni di più alberi di decisione. Così facendo posso arginare il fenomeno dell'overfitting, aumentare l'accuratezza e aumentare la stabilità dei parametri di valutazione. La predizione unica è quindi l'insieme delle predizioni di tutti gli alberi.

Come si implementa?

Durante il Training creo più alberi di decisione. Divido il mio DataSet di partenza in gruppi di dati randomici generando quindi un albero per ognuno di essi. Gli alberi vengono quindi addestrati su insiemi di features differenti. L'etichetta di classe viene assegnata a maggioranza.

Partendo da un training Set D seleziono B volte un campione casuale con rimpiazzamento ottenendo così un SubSet di quel DataSet per ogni iterazione che faccio ($D_1, D_2, D_3, \dots, D_B$).

Oltre questa suddivisione lavoro anche sul numero di features. I SubSet di training avranno un numero di features (tipicamente) pari a \sqrt{p} dove p è il numero massimo di features del DataSet di partenza. L'estrazione delle features è casuale e viene effettuata localmente da ogni nodo dell'albero. Quindi quando i dati arrivano ad un nodo verranno estratte solo alcune delle features disponibili e il nodo lavorerà su quelle. In questo modo cerco di far considerare attributi differenti in punti diversi dell'albero, introducendo delle variazioni che mi aiutino a generalizzare l'albero.



Pro e contro del metodo Random Forest:

PRO:

Contro:

| | |
|---|--|
| Vanno meglio degli alberi di decisione | Perdo la caratteristica d'interpretabilità, non posso seguire il percorso di un dato dentro tutti gli alberi. Posso però avere una ranking list delle features più importanti e più usate da tutti gli alberi della random forest. In questo modo posso capire quale feature è più importante (attributi dominanti per assegnazione) |
| Training veloce. Faccio B iterazioni ma solo su \sqrt{p} features quindi le iterazioni sono più corte | |
| Migliore robustezza a rumore e outliers | |

Classificare con le Regole

Un altro modo per creare un classificatore è generarlo con le Regole.

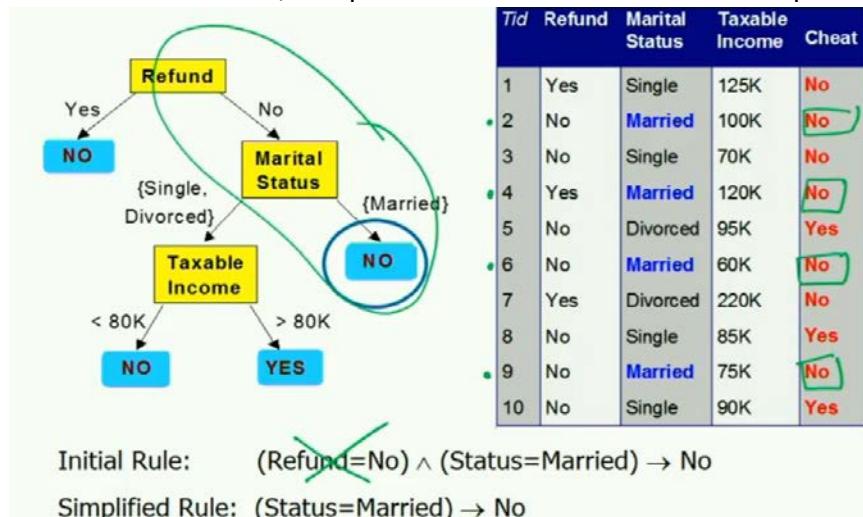
Devo studiare la condizione che sta nel corpo della regola e assegnare l'etichetta di classe che sta nella testa della regola. (*Condition*) -> *Y* Condition=ItemSet Y=etichetta di classe

Una regola copre un'istanza quando i suoi predicati sono veri su quell'istanza. Questo perché tale istanza rende veri i predicati della regola

In questo caso alcuni dati possono soddisfare più regole quindi potrebbero essere assegnate più etichette di classe. Le condizioni sulle regole che garantiscono il corretto funzionamento di un classificatore sono:

- Mutuamente esclusive: Tutte le regole non possono avere condizioni che possono essere vere contemporaneamente. Se soddisfo tale condizione solamente una regola alla volta potrà essere vera. Nell'albero ciò è sempre verificato perché un dato non può intraprendere due strade contemporaneamente.
- Esaustive: tutti i casi possibili sono coperti dalle regole. Nell'albero di decisione genero un arco uscente per ogni valore possibile dell'attributo, anche se l'attributo non è presente nella porzione di trainingSet che c'è in quel nodo lì (l'arco lo genero sempre e termino con un nodo foglia).

Quindi ogni record è coperto al massimo da una regola e c'è sempre una regola che copre ogni record. Nell'albero di decisione ciò accade per costruzione. Nelle regole non ho dei vincoli strutturali che impongono un'analisi in progressione degli attributi dei dati quindi, le regole mi permettono di commutare le condizioni associate in \wedge (and) e addirittura eliminarne delle altre se non necessarie all'attribuzione dell'etichetta di classe, semplificando così il modello. Stiamo quindi semplificando il modello di certe condizioni intrinseche nell'albero. Se effettuiamo le semplificazioni del modello perdiamo però le condizioni di mutua esclusione ed esaustività che nel modello di classificazione a regole non sono quindi più valide, a differenza dell'albero di decisione.



La semplificazione del modello

mi introduce delle problematiche (ad esempio un record soddisfa più regole) che posso gestire in più modi:

- Meccanismi di votazione e vado a maggioranza (caso random forest)
- Posso ordinare le regole in base in base ad una misura della qualità della regola, scegliendo poi la regola più in alto in questa classifica.

Quando ho un record che non innesca nessuna regola allora lo metto nella [classe di default](#) che è la classe di maggioranza, attribuendo l'etichetta di classe più frequente al record.

- | |
|---|
| R1: (Give Birth = no) \wedge (Can Fly = yes) \rightarrow Birds |
| R2: (Give Birth = no) \wedge (Live in Water = yes) \rightarrow Fishes |
| R3: (Give Birth = yes) \wedge (Blood Type = warm) \rightarrow Mammals |
| R4: (Give Birth = no) \wedge (Can Fly = no) \rightarrow Reptiles |
| R5: (Live in Water = sometimes) \rightarrow Amphibians |

| Name | Blood Type | Give Birth | Can Fly | Live in Water | Class |
|--------|------------|------------|---------|---------------|-------|
| turtle | cold | no | no | sometimes | ? |

Per costruire classificatori basati a regole ci sono due metodologie:

- [metodologie dirette](#): generano direttamente le regole
- [indirette](#): generano le regole partendo dall'albero

Pro e Contro dei classificatori basati su Regole e classificatori basati sull'albero di decisione:

PRO:

CONTRO:

| | |
|---|--|
| Comprensibili come un albero di decisione | |
| Tutte le regole che identificano una classe rappresentano un modello descrittivo della classe | |
| Velocità di costruzione del modello come quello dell'albero | |
| Processo di classificazione veloce, simile a quello dell'albero | |
| Qualità del risultato paragonabile a quella dell'albero di decisione, tipicamente un po' meglio | |

Classificatori fatti con le regole, classificazione associativa

È un modello di classificazione basato sull'uso delle regole. Devo generare delle regole fra tutti gli item a mia disposizione che nella testa della regola hanno un'etichetta di classe. Successivamente devo ordinare queste regole in base alla [confidenza](#) o il [lift](#), che mi danno il legame con l'etichetta di classe, e successivamente in base al supporto o alla lunghezza della regola.

Una parte delle regole trovate viene eliminata tramite un processo chiamato [Database Coverage](#) in cui per ogni record del training set vado a vedere quale regola copre il record selezionato e successivamente elimino tutti i record del training set che sono coperti da quella regola, inserendo

così la regola in esame nel modello di classificazione. Questa funzione genera un SubSet di regole fra tutte quelle disponibili.

Pro e Contro della classificazione associativa:

I classificatori associativi hanno un'accuratezza migliore degli alberi e dei normali modelli basati su regole perché osservo congiuntamente la probabilità di occorrenza di tutti gli elementi della regola (co-occorrenza degli elementi), al contrario dell'albero di decisione dove invece dovevo scegliere in modo greedy quale attributo osservare su cui effettuare lo split scegliendo così l'ottimo locale.

In caso di dati mancanti non vi sono problemi, al contrario dell'albero. Posso scegliere la default class.

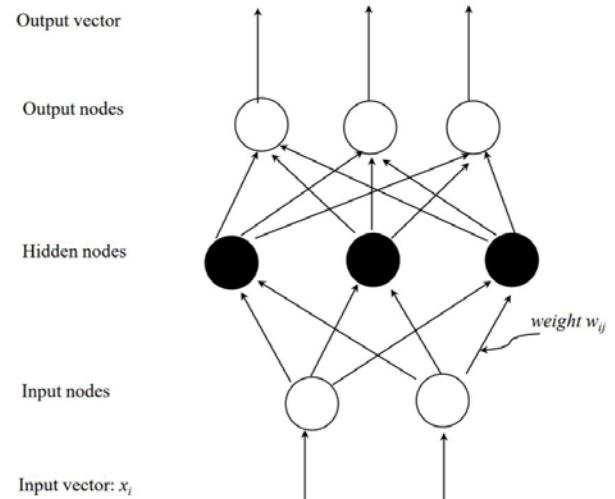
Il problema è dato dal tempo di generazione del modello che cresce enormemente sul numero di attributi da considerare (colonne dei nostri dati). Il tempo di costruzione del modello è quindi superiore rispetto all'albero decisionale che utilizza delle tecniche greedy. Il supporto si usa per regolare il tempo di generazione del modello.

Reti Neurali

Sin dagli anni 80 si è cercato di emulare il funzionamento del cervello umano. Gli elementi base sono i neuroni (centri di calcolo) e le sinapsi (interconnessioni fra i neuroni).

Similmente le reti neurali sono composte da più strati che eseguono input, calcolo, output:

- **nodi di ingresso:** ogni nodo d'ingresso è collegato a tutti i **nodi nascosti** facendo loro da "passacarte", normalizzando i valori dell'ingresso riducendoli ad un dominio noto (-1, 1 oppure 0, 1). In una rete neurale vi sono un numero di nodi d'ingresso pari a:
 - al numero di attributi, se l'attributo è continuo
 - al numero di valori possibili per l'attributo, se l'attributo è discreto (categorico). Questo perché ogni nodo converte un ingresso in un'uscita booleana (0-1), dovendola quindi effettuare per tutti i valori possibili dell'attributo.
- **Nodi di uscita:** sono pari al numero di etichette di classe, tranne nel caso binario dove avrà solo un nodo in uscita che mi afferma se il dato appartiene, con una certa probabilità, alla classe 0 oppure alla classe 1. Se ho n classi avrà n nodi d'uscita e ogni nodo d'uscita segnala l'appartenenza ad una classe di un record attribuendo anche una percentuale probabilità d'appartenenza di quel record alla classe identificata dal nodo.
- **Nodi nascosti:** Sono i nodi centrali che possono essere divisi in più strati. Sono i nodi che eseguono realmente i calcoli, attribuendo l'ingresso ad una determinata etichetta di classe. Facendo sempre una considerazione statistica sulla percentuale probabile.

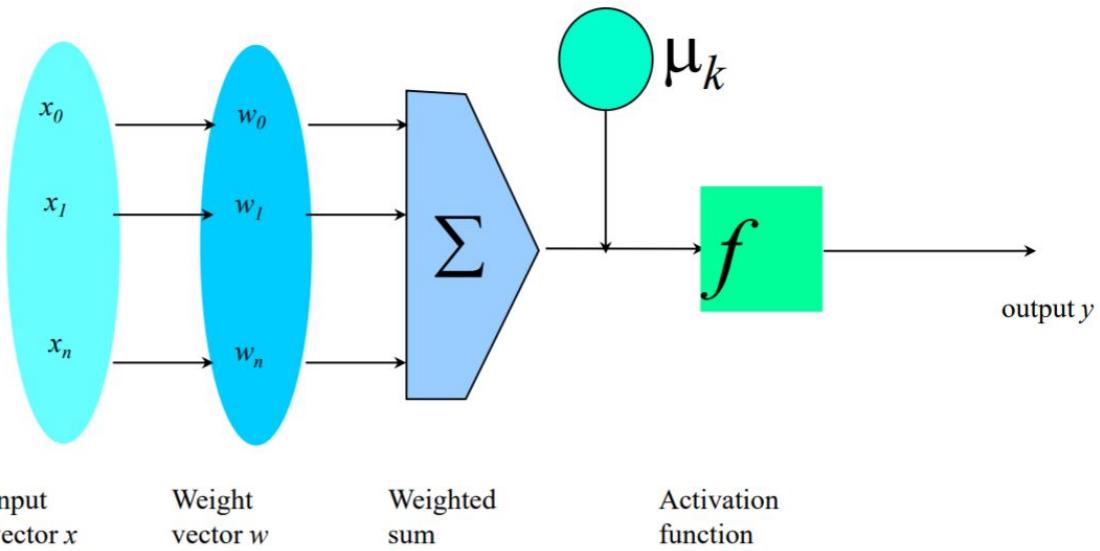


Struttura di un neurone (nodo)

Il nodo della rete neurale svolge diversi calcoli per trasformare gli ingressi in uscite.

Principalmente:

- Riceve in ingresso un vettore degli ingressi che rappresenta l'ingresso di tutti i nodi a monte del nodo in esame
- Effettua il prodotto cartesiano del vettore in ingresso con un vettore dei pesi. Il prodotto cartesiano è svolto moltiplicando ogni singolo elemento del vettore d'ingresso con il relativo peso, successivamente viene effettuata una somma fra tutti i risultati ottenuti. Vettore d'ingresso e vettore dei pesi hanno la stessa dimensione perché ogni peso ha un relativo ingresso del vettore degli ingressi.
- Viene sommato un valore di offset al risultato ottenuto
- Il valore ora ottenuto è espresso in una scala arbitraria che varia da $-\infty$ a $+\infty$. Prima di emettere in uscita tale valore dovrà normalizzare il risultato all'interno di una scala di variazione (ad esempio 0, +1) tramite l'applicazione di una funzione di attivazione. Solitamente viene utilizzata una funzione sigmoidea oppure una tangente iperbolica.



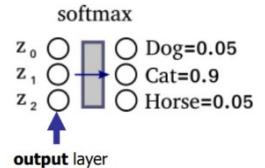
Tale operazione viene svolta da tutti i nodi d'uscita e per tutti i nodi degli strati nascosti.

Funzioni di attivazione

L'architettura interna del neurone è basata sull'applicazione del prodotto scalare a cui poi somma un offset. Esistono funzioni di attivazione di diverse tipologie:

- Nelle reti tradizionali si usa una funzione di attivazione con forma a sigmoidea che normalizza l'uscita fra 0 e +1
- La Tangente Iperbolica è simile ad una sigmoidea ma normalizza l'uscita fra -1 e +1
- Lo step non si usa perché ha un andamento troppo "brusco" e la derivata non è definita in $x=0$ e la derivata è 0 in tutti gli altri punti. Siccome il meccanismo del calcolo dell'errore è basato sul calcolo della derivata allora non usiamo lo step.
- Nelle reti convolutive (classificazione d'immagini) si usa la ReLU (Rectified Linear Unit). Questa funzione non satura, ovvero non ha un asintoto orizzontale ed evita il fenomeno del vanishing gradient. Il gradiente è il modo che permette di calcolare la derivata degli errori per poi propagarle all'indietro, quindi se ho tantissimi strati può capitare che dopo un po' di back propagation la derivata non abbia più dei valori interessanti. Tale funzione di attivazione evita dunque il problema del vanishing gradient.
- Softmax è una funzione che si utilizza nei neuroni d'uscita che opera contemporaneamente su tutti i neuroni di output garantendo che la somma dell'output sia pari ad 1.

Funzione sigmoidea e tangente iperbolica introducono una componente non lineare nel sistema che è desiderata. Sono usate nelle FFNNs (Feed Forward Neural Networks)



Come avviene la costruzione di una rete neurale?

Non vi è un algoritmo generale per la costruzione di una rete neurale. I passi fondamentali sono però:

- Alla partenza la rete ha assegnati dei valori casuali ai pesi e agli offset di tutta la rete. Poi avremo un seed (seme) che sarà un altro parametro che useremo per generare i valori dei pesi.
- Prendo un Training Set e un record alla volta lo faccio passare all'interno della mia rete neurale. Una volta all'output faccio un confronto fra la predizione ricevuta da quel record e la vera etichetta di classe che io conoscevo (essendo un record del training set).
- Calcolo l'errore che è la predizione vera e quella effettuata dal classificatore nel suo stato attuale
- Prendo l'errore calcolato e lo propago all'indietro correggendo i valori dei pesi e degli offset in modo da correggere quell'errore lì
- Tale processo viene effettuato per tutti i record del training set
- Ripeto il procedimento più volte per tutto il training set

Tale operazione devo svolgerla finché non ho soddisfatto alcuni criteri di convergenza:

- L'errore fra la predizione della rete e il valore reale del record del training set è sotto una certa soglia minima
- Gli offset e i pesi non variano più tanto quando propago a ritroso il mio errore, quindi sono in un punto di minimo
- Quando non ho più voglia di proseguire con l'addestramento della rete, questo perché tale processo richiede molto tempo e quindi è possibile considerare il caso in cui voglia arrestarlo dopo un determinato numero di iterazioni d'addestramento. Un'iterazione sul training set completo viene chiamata "epocha", posso quindi fissare un numero massimo di epocha

PRO e CONTRO delle reti neurali:

PRO:

CONTRO:

| | |
|---|---|
| Accuratezza molto elevata, specialmente per determinati problemi | Tempo di training molto lento. Difficile interpretazione dei parametri d'esecuzione. Difficile capire le configurazioni giuste dei parametri d'esecuzione |
| Robuste a rumore e eccezioni, buona gestione degli outliers perché essendo pochi non forniscono un contributo rilevante nel ricalcolo dei pesi. Con l'albero di decisione non posso tornare indietro, una volta deciso su cui fare lo split | Il modello prodotto non è interpretabile quindi non è possibile identificare cosa ha scaturito l'assegnazione di una determinata etichetta di classe |
| Posso usarli per fare predizioni di classe discrete o continue | |
| Processo di classificazione rapido | |

Classificatori Bayesiani

I classificatori Bayesiani calcolano la probabilità condizionata di trovare un'etichetta di classe C dato un record da classificare attribuendo quindi l'etichetta al record con probabilità più elevata.

$$P(C|X) = P(X|C) * P(C)/P(X)$$

C=etichetta di classe

X=vettore di variabili che voglio classificare, ho k attributi da X_1 a X_k

$P(C)$ è la frequenza della classe C nel training Set.

$P(X)$ è la probabilità del record considerato

Per avere un classificatore Bayesiano devo calcolare la probabilità di ogni classe dato il record X e attribuire ad X l'etichetta della classe con probabilità maggiore.

Il problema che tutte queste probabilità non sono calcolabili, effettuiamo quindi delle semplificazioni che ci permettono di procedere con il calcolo:

- Per ognuna delle classi diverse considerate, la $P(X)$ è costante perché è la probabilità del record. Tale termine non serve quindi per calcolare la probabilità d'assegnazione di una classe al record perché è una costante uguale per tutte le classi, posso quindi eliminarlo.
$$P(C|X) = P(X|C) \cdot P(C) / P(X)$$
- Considero gli attributi e i loro valori in modo statisticamente indipendente fra loro. Tale semplificazione è una forzatura che ci permette di separare la probabilità condizionata di ordine k ad una probabilità condizionata data dal prodotto delle probabilità condizionate di ordine 1. Non sempre infatti gli attributi sono statisticamente indipendenti fra loro, ad esempio la provincia e la regione sono sicuramente dipendenti. Così facendo posso però effettuare il calcolo. Tale semplificazione è molto grossolana, al tal punto che i classificatori che implementano queste soluzioni vengono chiamati Naive bayes

$$P(x_1, \dots, x_k | C) = P(x_1 | C) P(x_2 | C) \dots P(x_k | C)$$

▪ statistical independence of attributes x_1, \dots, x_k

▪ not always true

▪ model quality may be affected

| Outlook | Temperature | Humidity | Windy | Class |
|----------|-------------|----------|-------|-------|
| sunny | hot | high | false | N |
| sunny | hot | high | true | N |
| overcast | hot | high | false | P |
| rain | mild | high | false | P |
| rain | cool | normal | false | P |
| rain | cool | normal | true | N |
| overcast | cool | normal | true | P |
| sunny | mild | high | false | N |
| sunny | cool | normal | false | P |
| rain | mild | normal | false | P |
| sunny | mild | normal | true | P |
| overcast | mild | high | true | P |
| overcast | hot | normal | false | P |
| rain | mild | high | true | N |

| outlook | |
|------------------------------|----------------------------|
| $P(\text{sunny} p) = 2/9$ | $P(\text{sunny} n) = 3/5$ |
| $P(\text{overcast} p) = 4/9$ | $P(\text{overcast} n) = 0$ |
| $P(\text{rain} p) = 3/9$ | $P(\text{rain} n) = 2/5$ |
| temperature | |
| $P(\text{hot} p) = 2/9$ | $P(\text{hot} n) = 2/5$ |
| $P(\text{mild} p) = 4/9$ | $P(\text{mild} n) = 2/5$ |
| $P(\text{cool} p) = 3/9$ | $P(\text{cool} n) = 1/5$ |
| humidity | |
| $P(\text{high} p) = 3/9$ | $P(\text{high} n) = 4/5$ |
| $P(\text{normal} p) = 6/9$ | $P(\text{normal} n) = 2/5$ |
| windy | |
| $P(\text{true} p) = 3/9$ | $P(\text{true} n) = 3/5$ |
| $P(\text{false} p) = 6/9$ | $P(\text{false} n) = 2/5$ |

$$P(p) = 9/14$$

$$P(n) = 5/14$$

- Data to be labeled
 $X = \langle \text{rain, hot, high, false} \rangle$
- For class p **Per classe positiva**

$$P(X|p) \cdot P(p) = P(\text{rain}|p) \cdot P(\text{hot}|p) \cdot P(\text{high}|p) \cdot P(\text{false}|p) \cdot P(p) = 3/9 \cdot 2/9 \cdot 3/9 \cdot 6/9 \cdot 9/14 = 0.010582$$
- For class n **Per classe negativa**

$$P(X|n) \cdot P(n) = P(\text{rain}|n) \cdot P(\text{hot}|n) \cdot P(\text{high}|n) \cdot P(\text{false}|n) \cdot P(n) = 2/5 \cdot 2/5 \cdot 4/5 \cdot 2/5 \cdot 5/14 = 0.018286$$

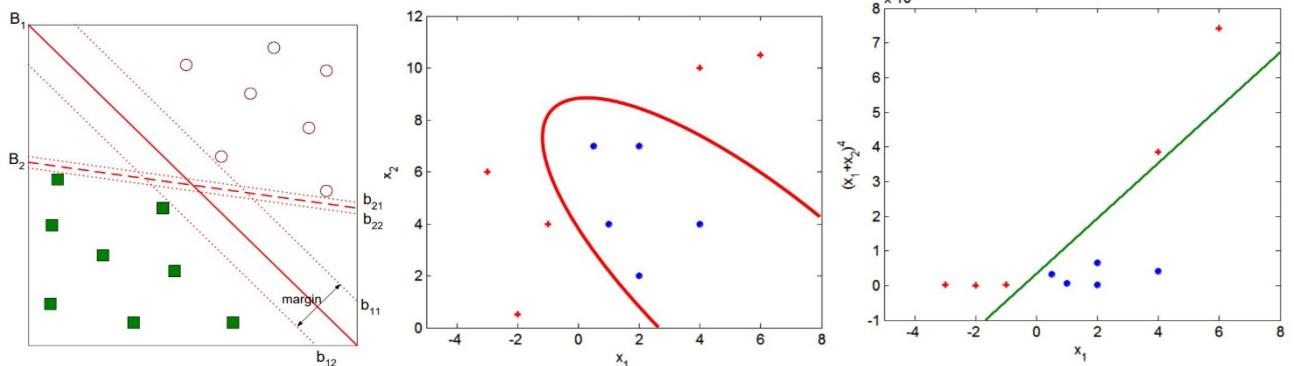
Si è tentato di superare l'ipotesi di indipendenza statistica, tramite l'uso di tabelle che descrivono la dipendenza fra alcuni attributi dal punto di vista della dipendenza statistica.

Nonostante la semplificazione d'indipendenza statistica, i classificatori Bayesiani sono molto utilizzati poiché sono incrementali in modo semplice, ovvero riesco facilmente ad aggiornare il modello (basta cambiare i numeri delle probabilità) in caso d'inserimento di un nuovo record che possiede una nuova etichetta di classe e, inoltre, di effettuare la modifica senza possedere il datraining set. Posso sfruttare le statistiche già calcolate nel modello e utilizzare quelle per aggiornarlo permettendogli così di accettare la nuova etichetta di classe. Nessun altro modello è in grado di incorporare nuove informazioni come il modello bayesiano di classificazione.

Questo metodo, a differenza dei precedenti, permette di incorporare nuove informazioni senza dover ricalcolare tutto da capo.

Support Vector Machines - SVM

È una tecnica nata negli anni 90 e mi permette di individuare un iperpiano ad n dimensioni che massimizza il margine, ovvero la distanza fra i punti più vicini di due classi differenti. In caso di problemi dove la separazione lineare non è adatta possiamo utilizzare kernel non lineari che applicano una trasformazione dello spazio, in questo modo possiamo poi applicare una separazione fra i punti interessati.



Questo metodo si adatta molto bene a problemi non lineari (come le reti neurali) ma l'interpretabilità del modello è praticamente nulla. Le SVM sono più stabili delle reti neurali alla configurazione dei dati.

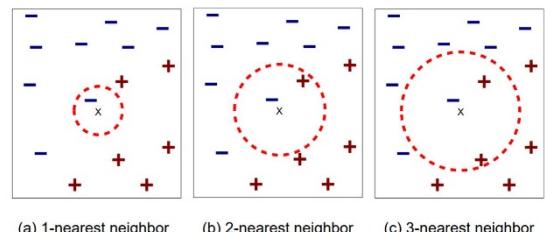
K-Nearest Neighbour

Tutti gli altri metodi si tengono una rappresentazione astratta del Training Set che noi abbiamo chiamato modello. La tecnica K-Nearest neighbour non genera il modello. Quando mi arriva un nuovo oggetto da classificare devo andare a ricercare i k record più vicini (simili) all'oggetto che devo classificare all'interno del Training Set.

Dobbiamo quindi:

- Memorizzare tutti i record del Training Set
- Utilizzare una misura di distanza per misurare la distanza dal punto nuovo a tutti i record del training set
- Assegno l'etichetta di classe con un meccanismo di votazione (maggioranza). Il punto prende quindi l'etichetta di classe più frequente fra i k punti a lui più vicini.

Il parametro k di soglia deve essere scelto in modo opportuno. Se k=1 allora il problema si riconduce in un problema conosciuto nell'informatica chiamato diagramma di Voronoi.



Quando calcoliamo la distanza fra il nuovo punto e i k punti più vicini ad esso dobbiamo valutare la votazione dei punti in modo normalizzato secondo la distanza al quadrato ($1/d^2$). In questo modo il voto dei punti più vicini darà un contributo maggiore rispetto ai punti più distanti dal nostro record. La distanza è calcolata secondo la formula della distanza euclidea. Le distanze devono essere normalizzate. Nel caso di grandi volumi di dati questo approccio non funziona tanto bene.

La scelta di k è molto importante:

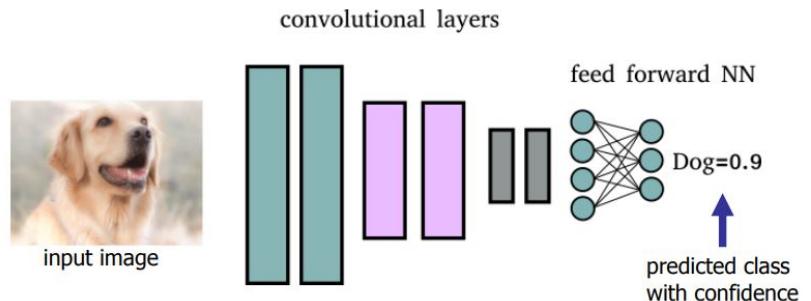
- Se k è piccolo potrei adattarmi a dei valori che sono rumorosi
- Se k è grande potrei includere punti di altre classi

Questo è un metodo incrementale in modo semplice che riesce ad aggiungere nei nuovi record etichettati, proprio come i classificatori Bayesiani.

Reti convolutive

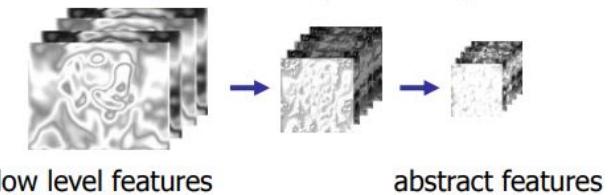
Una rete convolutiva estraie delle caratteristiche da un'immagine e le usa per assegnare delle etichette di classe sull'immagine. L'immagine è processata da punti di vista differenti, quindi avviene un'estrazione di features diverse. Le etichette di classe verranno attribuite in base ad una certa probabilità plausibile.

Le reti convolutive sono quindi un'estensione delle reti neurali [Feed Forward](#), abbiamo quindi l'inserimento di alcuni layer convolutori prima della rete feed forward.



A tutte le etichette di classe viene attribuita una probabilità e quelle con probabilità sotto ad una certa soglia vengono scartate poiché non interessanti.

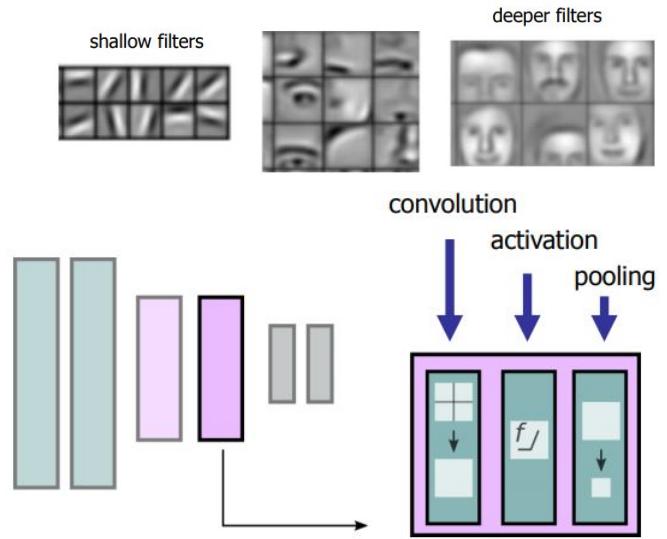
Ogni strato lavora su un certo insieme di features, andando avanti nella rete riconoscerò dettagli sempre più generali, riconoscendo così delle features più generali.



Com'è fatto uno strato convolutivo

Sono composti da tre parti:

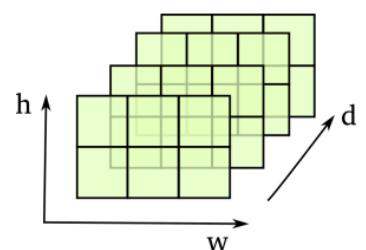
- Convoluzione: vi sono n filtri scorrevoli che traslano sull'immagine e estraggono dei pezzi e ogni filtro guarda analizza delle features diverse (linea separazione, superfici diverse etc..). Ognuno degli n filtri genererà una nuova immagine.
- Attivazione: modifica il risultato della convoluzione, uso la ReLU
- Pooling: riduce la dimensione del problema che sto processando



Le reti neurali sono basate sul concetto di [Tensore](#).

I Tensori sono vettori ad n dimensioni (rango = num. dimensioni, forma=numero elementi dentro) in cui rappresento le informazioni d'interesse.

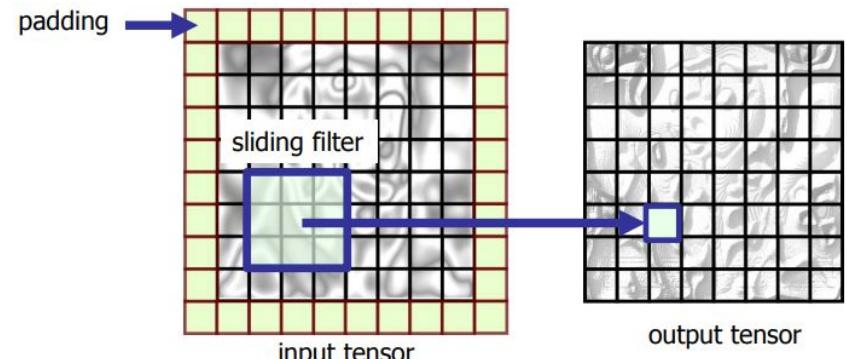
Per rappresentare le immagini usiamo dei tensori di rango 3. La profondità serve per rappresentare i colori.



Strato convolutivo

Processa le immagini che sono rappresentati sotto forma di tensori. Il processo è fatto generando dei tensori di output partendo da tensori di input.

La trasformazione è effettuata tramite dei filtri che esaltano certe componenti dell'immagine e ne attenuano altre. I filtri lavorano su piccole matrici partendo dalla grande matrice del tensore, successivamente questi filtri "scorrono" sul tensore d'ingresso concentrandosi su alcuni tratti dell'immagine.

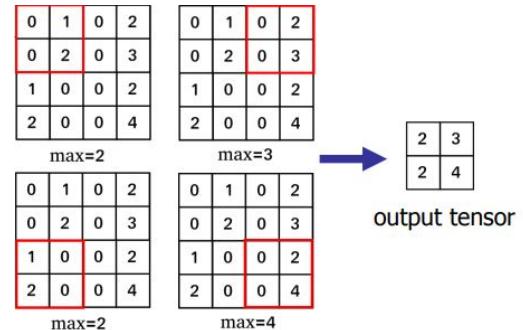


I tensori d'ingresso e d'uscita sono delle stesse dimensioni, devo quindi applicare un bordo di padding per trattare i pixel di bordo, altrimenti non potrei applicare il filtro.

I filtri contengono i pesi della rete neurale.

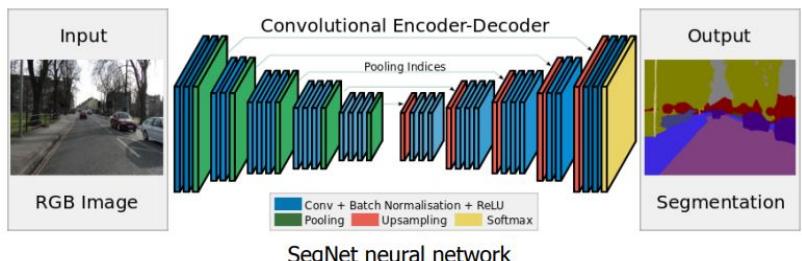
Pooling

È un filtro che trasforma un tensore in ingresso in un tensore in uscita raggruppando più dati insieme. In questo modo possiamo generare un solo campione per gli elementi della dimensione del filtro. Uno dei filtri più utilizzati è il maxpool che seleziona il valore massimo degli elementi dentro al filtro.



Semantic segmentation CNNs

Certe volte occorre la capacità di assegnare i pixel appartenenti ad oggetti. Assegnando l'etichetta di classe ai singoli pixel e non agli oggetti.



Voglio quindi riconoscere degli oggetti in maniera non generale, assegnando l'etichetta ad un livello di dettaglio più profondo.

Queste reti sono composte da un insieme di filtri convolutivi che fanno:

- Convoluzione
- Attivazione (normalizzazione ReLu)
- Pooling

Dopodiché si fa il processo al contrario ricostruendo l'immagine di partenza. In questo modo genero più campioni partendo da un campione solo. In fine classifico ogni pixel dell'immagine con una softmax.

Questo tipo di reti può riconoscere forme più particolari.

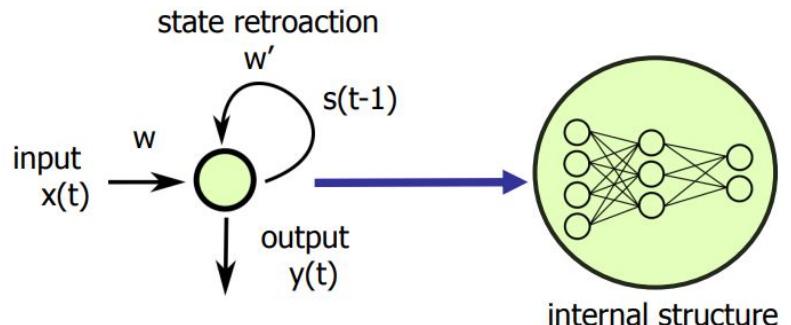
Reti neurali ricorrenti, un altro tipo di reti

Sono delle reti neurali che sono in grado di tener conto dell'ordine degli elementi (in cui compaiono gli eventi). Possono processare dati sequenziali. Sono delle reti neurali classiche che hanno un loop che torna indietro e riporta lo stato precedente.

Queste reti vengono utilizzate nelle serie temporali, nella traduzione automatica e nel riconoscimento del parlato.

I nodi hanno quindi memoria dello stato attuale e dello stato precedente al tempo $t-1$. Ho dei pesi sull'ingresso al tempo t e sullo stato al tempo $t-1$ che contribuirà all'uscita.

Tali reti hanno però la memoria solo allo stato $t-1$. Vogliamo implementare delle reti che hanno memoria anche agli stati precedenti, come una memoria a lungo termine.



LSTM (long short term memory)

Oggi esistono delle LSTM che sono come dei circuiti logici in software che implementano una memoria a lungo termine tramite dei gate che permettono di processare o meno un'informazione.

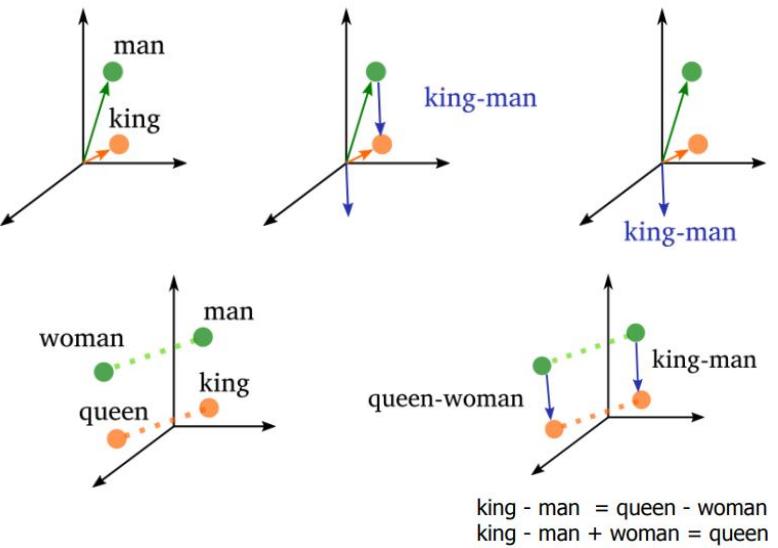
Autoencoders

Sono reti neurali che permettono di processare, tramite i filtri visti prima, immagini rumorose in modo da poterle ricostruire espandendole prive di rumore.

Word Embedding

Tale sistema associa a delle parole una rappresentazione n-dimensionale della parola stessa. In questo modo possiamo calcolare la semantica dalle parole in ingresso.

Tale processo è interessante per l'elaborazione dei testi. Le parole sono rappresentate come vettori su cui poi possiamo applicare delle operazioni fra vettori, ad esempio la differenza. In questo modo possiamo catturare la semantica delle parole.

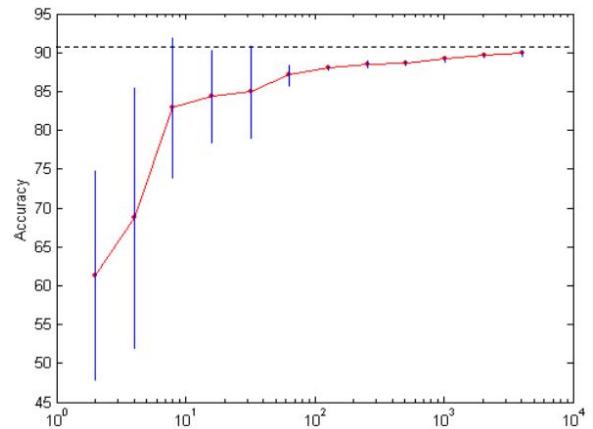


Come valutare un modello di classificazione

La valutazione riguardo l'accuratezza di un modello di classificazione viene effettuata stimando l'accuratezza grazie a dei dati di test set. Possiamo dunque effettuare una stima del classificatore osservando il funzionamento sul modello reale.

La variabilità della stima varia in base alle dimensioni del campione. Si osserva dalla curva di apprendimento che se aumentiamo le dimensioni del campione la valutazione, la qualità del classificatore diventa più stabile anche dal punto di vista della varianza. Se le dimensioni del campione sono piccole avrò una varianza incredibilmente più elevata.

Come si procede?



Devo separare i dati disponibile in due campioni: uno di training e uno di test.

Il campionamento è stratificato sulle etichette di classe ovvero avrò le stesse proporzioni di appartenenza alle classi sia nel campione di test che in quello di training.

Nel caso in cui i miei dati hanno una classe di minoranza (molto piccola) posso pensare di fare il campionamento con rimpiazzamento (pesco il mio campione e lo rimetto dentro).

I metodi di partizionamento che permettono di dividere i dati di partenza nei due sottoinsiemi (training, test) sono:

- Holdout - Partizionamento fisso: questo tipo di partizionamento è basato sul tener da parte una parte di dati da usare come test set (1/3) e una parte come training set (2/3). Questo processo è più veloce ma meno accurato nel generare la stima del classificatore.
- Cross Validation: prendo i miei dati e li divido in k partizioni diverse, poi prendo $k-1$ partizioni e le uso per il training, l'altra per il test. Poi a rotazione lo rifaccio con un'altra partizione, questa cosa viene fatta k volte. In questo modo ciascuna delle partizioni passerà con il ruolo di test set. Questo metodo di partizionamento studia tutti i record del DataSet, almeno una volta come TestSet. Tale processo è meno veloce ma la stima è più affidabile rispetto a quella che faccio con l'holdout. Qui genero k stime in k classificatori diversi.
- Leave-one-out: si usa un cross validation per $k=n$. Solo un record resta fuori nel TestSet e tutti gli altri vanno del TrainingSet. Si usa per database veramente piccoli, è infatti un caso degenere.

Tale processo serve solo per ottenere la stima della bontà del classificatore. In realtà il classificatore che poi consegnerò sarà solo uno e sarà quello prodotto con i dati di TrainingSet.

Quali metriche uso per valutare la qualità del classificatore

La metrica classica di valutazione è l'accuratezza.

Il risultato di una classificazione viene rappresentato in una matrice di confusione che indica quanti record sono stati etichettati correttamente per ogni classe e quanti invece no. Da una parte visualizzo le predizioni e dall'altra la classe reale dei record. Fuori dalla diagonale ho gli errori, ovvero gli oggetti etichettati in modo sbagliato.

■ binary classifier

| | | PREDICTED CLASS | |
|--------------|-----------|-----------------|----------|
| | | Class=Yes | Class>No |
| ACTUAL CLASS | Class=Yes | a | b |
| | Class>No | c | d |
| | | | |

a: TP (true positive)
b: FN (false negative)
c: FP (false positive)
d: TN (true negative)

Le metriche usate per la valutazione della qualità di un classificatore sono:

- Accuratezza: numero di oggetti correttamente assegnati ad una classe / numero totale di oggetti

processati. Nella nostra matrice di confusione sarà $(a+d)/(a+b+c+d)$.

L'accuratezza ha qualche difetto:

- se abbiamo delle classi sbilanciate non effettua una valutazione affidabile poiché privilegia la classe di default. Non è una misura affidabile sulla qualità della classificazione per una classe minoritaria
- Non mi consente di distinguere fra errori di diversa importanza. Ad esempio, in uno screening medico l'errore commesso sulla classificazione per un falso negativo (malato contrassegnato come in salute) è più rilevante rispetto ad un falso positivo. Importanza di classificazione diversa per classi diverse.

- Richiamo: Ci permette di valutare la qualità della

classificazione sulla singola classe. Separatamente per la classe valuta il numero di oggetti classificati correttamente alla classe C rispetto al numero totale di appartenenti realmente alla classe C. "I malati classificati rispetto ai malati totali". I correttamente assegnati / tutti quelli che dovevo assegnare

- Precisione: Ci permette di valutare la qualità della

classificazione sulla singola classe. Valuta il numero di oggetti correttamente assegnati alla classe C fratto il numero di oggetti assegnati alla classe C dal classificatore. I correttamente assegnati / gli assegnati

$$\text{Recall}(r) = \frac{\text{Number of objects correctly assigned to C}}{\text{Number of objects belonging to C}}$$

$$\text{Precision (p)} = \frac{a}{a + c}$$

$$\text{Recall (r)} = \frac{a}{a + b}$$

$$\text{F - measure (F)} = \frac{2rp}{r + p} = \frac{2a}{2a + b + c}$$

$$\text{F - measure (F)} = \frac{2rp}{r + p}$$

Richiamo e Precisione non possono essere massimizzate contemporaneamente quindi si esegue la media armonica delle due. Tale media armonica prende il nome di F-measure. È più vicina al più basso dei due valori.

Curva ROC

Inventata per calcolare l'errore di detect dei radar è ora usata anche nel nostro ambito.

Sull'asse x abbiamo il tasso di falsi positivi

$$FPR = FP / (FP + TN)$$

Sull'asse y abbiamo il tasso dei veri positivi

$$TPR = TP / (TP + FN)$$

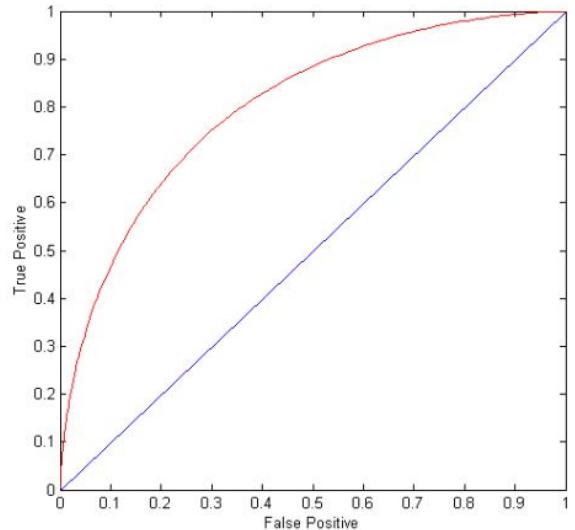
Nel punto 0,0 tutto è finito nella classe negativa

Nel punto 1,1 tutto è finito nella classe positiva

Il caso ideale è quando passo da 1 in alto a sinistra, quindi non commetto mai errore di classificazione.

Per costruire questo diagramma ho bisogno della probabilità di appartenenza alla classe positiva. Ad esempio, i classificatori bayesiani o le reti neurali ci forniscono tali informazioni.

La bontà della classificazione si valuta misurando l'area sotto la curva ROC. L'area sotto la curva si chiama AUROC (Area under ROC)



15. Clustering

È un'analisi di tipo esplorativo che cerca di capire come sono strutturati i dati. Si cerca di raggruppare oggetti simili fra loro separandoli da oggetti differenti. Il clustering mi serve per estrarre delle informazioni oppure per riassumere una collezione di documenti dentro lo stesso cluster.

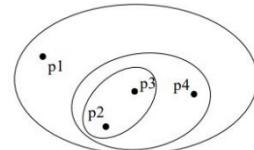
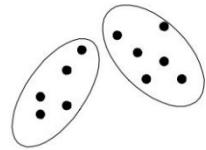
Per effettuare tali operazioni ci occorre misurare la distanza fra i punti e comprendere, grazie a tale misura, se i punti appartengono o meno allo stesso cluster.

Il clustering è un set di cluster, ovvero un insieme di insiemi.

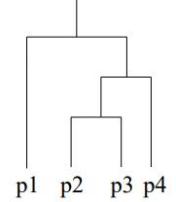
Un problema rilevante della fase di clustering è conoscere il numero di cluster che voglio generare. Vi sono alcuni algoritmi che necessitano in input tale informazione.

Posso generare i cluster in due modi:

- **Modo Partizionale:** I cluster non sono sovrapposti fra loro e ogni oggetto appartiene esattamente ad un cluster
- **Modo Gerarchico:** Costruisco i cluster nidificati in modo gerarchico. Il punto di partenza sono tutti gli elementi separati e il punto di arrivo è un unico cluster che contiene tutti i punti.



I cluster in modo gerarchico sono rappresentati da un Dendrogramma. L'altezza del dendrogramma indica la distanza in cui applico l'accorpamento. Tagliando il dendrogramma ad una certa altezza posso visualizzare quanti cluster ho a quel livello.



In alcuni casi vorrei non assegnare alcuni punti ad un cluster, ad esempio i punti di rumore, alcuni algoritmi sono in grado di eliminare i dati di rumore, altri invece no.

Tipi di Cluster:

Well – Separated Clusters

Questo è il caso ideale. Tutti i cluster sono ben separati e hanno delle forme globulari che rendono semplice la divisione.

Center Based

Spesso capita che i cluster nella realtà non sono ben separati, vi sono quindi più metodi che mi permettono di costruire tipi di cluster differenti, basati su intuizioni differenti:

- Uso dei Centroidi: mi permette di assegnare dei punti ad un cluster osservando a quale centroide sono più vicini. Il centroide è un punto virtuale nello spazio
- Uso dei Medoidi: è il punto più vicino al centroide esistente fisicamente, questi metodi sono più resistenti al rumore o agli outlier

Contiguity-Based

Sono algoritmi di clustering che si basano sulla vicinanza dei punti. Il cluster si diffonde osservando la prossimità dei punti, anche in modo transitivo. Posso “inseguire” delle forme più complesse di cluster.

Density-Based

Posso definire un cluster anche studiando un’area densa dove ho tanti punti costruendogli un cluster attorno. Questo metodo è difficile da configurare ma funziona molto bene.

Conceptual Clusters

Ci sono anche dei metodi basati su delle proprietà funzionali di qualche tipo. Si basano quindi sulla rappresentazione di un concetto

Metodo Partizionale

K-means

È un algoritmo estremamente semplice.

Ad ogni cluster è associato un centroide che è il punto medio (baricentro). Ad ogni iterazione dell'algoritmo il punto è associato al cluster con baricentro più vicino.

Con questo algoritmo devo sapere a priori il numero di clustering sul quale effettuare la suddivisione. Devo quindi fornire il parametro di configurazione k che indica quanti cluster avrà.

Algoritmo:

- Scelgo a caso la posizione dei k centroidi in k posizioni dello spazio
- Misura la distanza di ogni punto da ogni k centroide
- Assegno ogni punto al centroide più vicino
- Una volta assegnati tutti i punti ricalcolo la posizione dei centroidi
- Con i nuovi centroidi ripeto il processo
- Mi fermo quando i centroidi non si spostano più, quando i punti non cambiano più cluster oppure posso vare delle valutazioni sull'errore

"Calcolo la distanza di ogni punto dai centroidi e assegno il punto al centroide più vicino, successivamente ricalcolo la posizione dei centroidi. Con i nuovi centroidi ricalcolo il processo"

Problemi del k-means:

- I punti iniziali dei centroidi sono scelti a caso. La convergenza dell'algoritmo è influenzata dalla posizione iniziale dei centroidi.
- Può capitare che a causa della scelta casuale dei centroidi, uno di essi si posizionerà in mezzo al punto di due cluster non spostandosi più. La scelta dei punti iniziali ha un impatto importante sulla corretta suddivisione dei punti nei cluster.

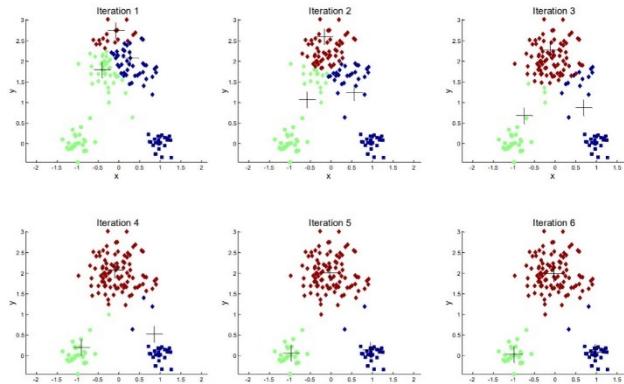
Tale algoritmo generalmente converge rapidamente trovando l'ottimo locale.

La complessità è lineare per il numero di punti, questo algoritmo viene definito come quick and dirty perché fornisce un risultato velocemente ma non sempre valido.

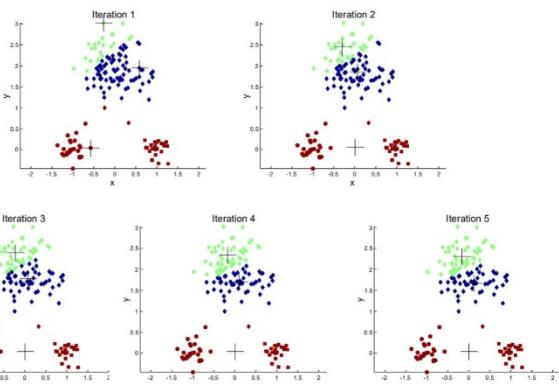
Complexity is $O(n * K * I * d)$

- n = number of points, K = number of clusters, I = number of iterations, d = number of attributes

Optimal k-means



Wrong k-means



Come valutare il risultato dell'algoritmo di clustering

Per valutare la qualità di un cluster uso una misura di errore chiamata Sum of Squared Error (SSE). Se voglio valutare la qualità globale posso considerare la doppio sommatoria comando l'SSE di ogni cluster, se voglio valutare la qualità del singolo cluster uso una sola sommatoria.

$$SSE = \sum_{i=1}^K \sum_{x \in C_i} dist^2(m_i, x)$$

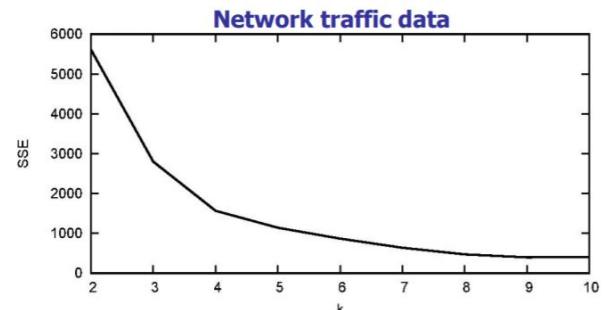
SSE misura la distanza fra un punto considerato e il suo centroide. Tale operazione viene calcolata per tutti i punti.

Quando devo scegliere fra più risultati di clustering posso selezionare quello che ha l'SSE più basso e molto probabilmente sarà quello di qualità migliore. (dipende dalla forma dei cluster)

La misura dell'SSE diminuisce al crescere in K. (aumentare dei cluster)

Come scelgo il valore di k?

Per scegliere il valore di k devo guardare la curva dell'SSE al variare di k. Fissato un k devo quindi provare varie configurazioni iniziali di partenza dei centroidi e poi scegliere quella che mi fornisce un SSE migliore. Infine posso plottare tutti i valori SSE ottenuti al variare di k costruendo così un grafico.



Per la scelta di k utilizzeremo l'approccio del ginocchio che consiste nello scegliere il valore di k in base al punto dove la curva SSE ha una pendenza che si smorza, ovvero un andamento smorzato. Nell'esempio k=5 potrebbe essere un buon valore di k.

La curva SSE può avere dei saltelli nei casi dove il centroide posizionato casualmente non abbia dei punti vicini. Quel centroide lì rimane un cluster vuoto, in questi casi SSE peggiora.

Come rimediare al problema della posizione del centroide

Il problema si genera quando un centroide si posiziona fra due cluster, non potendo più distinguergli.

Per rimediare al problema dei centroidi si possono tentare diversi approcci:

- Posso provare ad effettuare varie iterazioni sperando di trovare la configurazione ideale
- Utilizzo un algoritmo gerarchico (più accurato ma più lento -> complessità quasi n^3) per generare i centroidi su un campione di dati, successivamente utilizzerò questi centroidi nell'algoritmo k-means su tutti i dati a disposizione. Uso quindi un algoritmo gerarchico come start per il k-means
- Genero più centroidi tentando di aggregare i cluster che dovrebbero stare insieme
- Bisecting K-means: è un altro algoritmo che cerca di risolvere il problema del posizionamento dei centroidi ma è più lento del k-means. È una specie di ibrido fra il gerarchico e il k-means

Cluster vuoto

Per risolvere il problema dei cluster vuoti posso sostituire l'assegnazione del cluster vuoto con quella di un altro centroide. Cerco il punto a distanza più alta (maggiore influenza SSE) e lo uso come nuovo centroide.

Posso prendere il cluster con l'SSE più alto (cluster più frammentato), selezionarne un punto e definirne un centroide.

Metodo ibrido Partizionale-Gerarchico

Bisecting K-means

Questo algoritmo produce un clustering ibrido fra partizionale e gerarchici. Procede come segue:

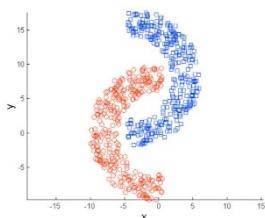
- Parte definendo $k=2$
- Esegue il k-means con $k=2$
- Prende i due cluster generati e li posiziona nella *lista dei cluster*.
- Prende il cluster con SSE peggiore ed esegue k-means su quel cluster lì con $k=2$, spaccando il cluster interessato in altri due cluster. Abbiamo ora 3 cluster.
- Prosegue così finché non raggiungo il valore di k impostato (come nel k-means), iterando così k volte

Questo algoritmo elimina il problema dei cluster vuoti e anche il problema del posizionamento randomico di tutti i centroidi.

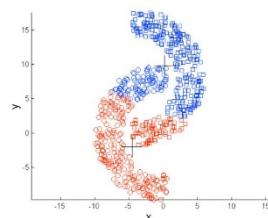
Considerazione sul k-means

L'algoritmo k-means non funziona in molti casi:

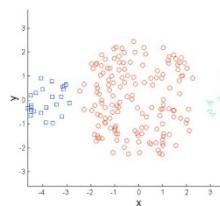
- Cluster di dimensioni diverse
- Cluster di forme diverse
- Cluster con densità diverse
- Cluster con forme non globulari (strane)
- Gli outlier finiscono dentro un cluster sporcandomi la definizione di cluster. Gli outlier vanno quindi trattati ed eliminati in fase di preprocessing.



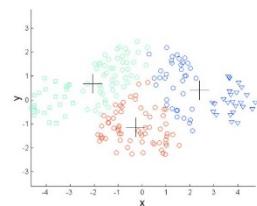
Original Points



K-means (2 Clusters)



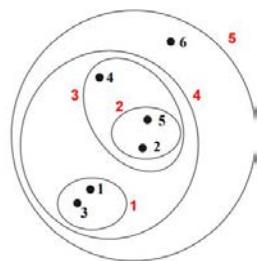
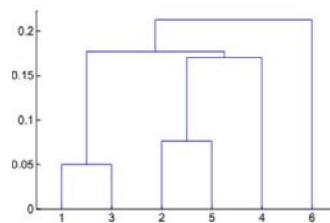
Original Points



K-means (3 Clusters)

Metodo Gerarchico

Il clustering gerarchico è costituito da un insieme di cluster che sono nidificati uno dentro l'altro. È rappresentato da un dendrogramma che permette di visualizzare la gerarchia dei cluster annidati grazie ad una rappresentazione della distanza (asse y) su cui viene effettuata "l'agglomerazione" dei punti.



La distanza sull'asse y mi indica a quale distanza i punti sono agglomerati.

I metodi di clustering gerarchico possono essere di tipo:

- Agglomerativo: parto da un cluster con un singolo elemento e nella situazione finale avrò un cluster che racchiude tutti gli elementi
- Divisivo: parto da un cluster che contiene tutti gli elementi e lo divido fino a formare dei cluster che contengono un solo elemento

Vantaggi e svantaggi per clustering gerarchico:

- Vantaggi: Non devo definire il numero n di cluster che devo tirar fuori. Posso anche valutare ad un certo livello di distanza, tagliando orizzontalmente il dendrogramma, quanti cluster sono formati ad un certo livello di distanza.
- Difetti: Devo prendere la decisione ottima (ottimo locale) sull'agglomerazione dei punti. Tale operazione non è reversibile quindi non posso fare backtrack e vedere una possibile decisione sbagliata al raggiungimento dell'ottimo globale.

La misura di distanza del k means non è più sufficiente perché dovremmo generalizzare la distanza fra punti per arrivare alla distanza fra due cluster oppure fra un punto e un cluster. Bisognerà quindi cercare delle nuove tecniche per misurare la distanza.

Il clustering gerarchico non viene usato sempre perché è molto dispendioso a livello di tempo (la complessità è dell'ordine di n^3 dove n è il numero di punti). Inoltre devo memorizzare la matrice di prossimità che occupa in memoria uno spazio pari a n^2 . La qualità è alta ma il processo è lento. Posso fare il gerarchico quando ho pochi punti oppure come pilota su un campione per il k-means.

Algoritmi di cluster agglomerativi

Il funzionamento degli algoritmi di cluster agglomerativi è basato su alcuni passi fondamentali:

- Calcolo della matrice delle distanze (prossimità) che riporta tutte le distanze fra i punti esistenti. **Sulla diagonale (matrice delle distanze) ci sono tutti 0.** Nel caso di una **matrice di somiglianza** ci sarebbe stato **1**. La matrice è anche simmetrica
- Consideriamo che ogni punto sia un cluster, dobbiamo quindi cercare i punti più vicini fra loro
- Uniamo i punti più vicini e ricalcoliamo la matrice delle distanze
- Ripetiamo il tutto finché non otteniamo un unico cluster che rappresenta tutti i punti.

| | p1 | p2 | p3 | p4 | p5 | . |
|----|----|----|----|----|----|---|
| p1 | | | | | | |
| p2 | | | | | | |
| p3 | | | | | | |
| p4 | | | | | | |
| p5 | | | | | | |
| . | | | | | | |

Proximity Matrix

Il costo di questo algoritmo è quindi il continuo ricalcolo della matrice delle distanze.

Parto da una matrice che contiene le distanze fra punti, man mano si riduce e ottengo una matrice di distanze fra cluster.

Quando dobbiamo unire due punti dovrò eliminare le corrispondenti righe e colonne dalla matrice delle distanze e generare una nuova riga e colonna per il cluster che ingloba i punti di partenza.

Come definire la distanza fra punti e cluster oppure fra cluster e cluster

Vogliamo misurare la distanza fra oggetti che sono a più dimensioni.

Vogliamo definire un concetto di stanza fra punti (distanza o somiglianza) e poi estenderlo al concetto di distanza fra cluster.

I modi per definire la distanza fra cluster sono vari:

- MIN: la distanza minima fra i punti più vicini appartenenti a due cluster diversi
- MAX: la distanza fra i punti più lontani di due cluster differenti
- Group Average: Posso calcolare la distanza media fra tutti i punti di due cluster
- Distance Between Centroids: è la distanza fra i centroidi definiti per i due cluster in esame
- Ward's Method: Basato sul controllo dell'SSE

Il primo step della generazione di un cluster gerarchico è sempre uguale, quando devo considerare dei punti. Sfruttiamo infatti il MIN. Successivamente uso le misure di distanze viste sopra.

Che effetto fa usare una misura invece che un'altra?

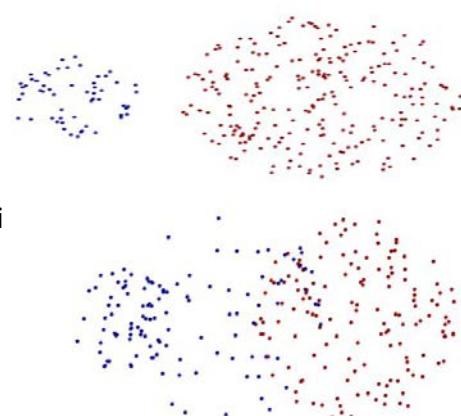
In base alla distribuzione dei dati certe misure di distanza mi permettono di ottenere delle separazioni corrette o meno dei cluster

MIN

Partendo dalla matrice di similarità prendo la distanza minima quindi la somiglianza massima. Posso anche partire dalla matrice delle distanze e prendere i punti con distanza minore quindi valore più piccolo.

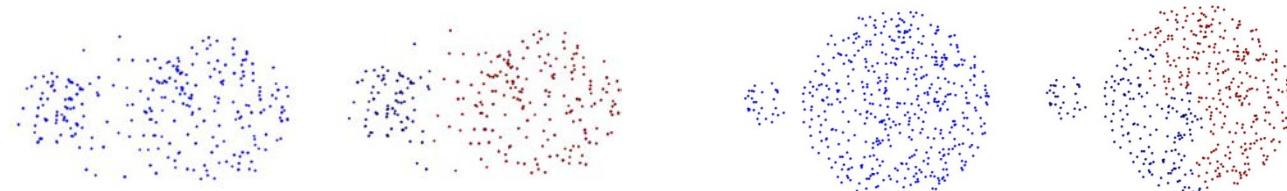
Utilizzando MIN come misura di distanza posso separare correttamente due cluster che hanno cardinalità differente. A differenza di ciò che accadeva nel k-means.

Se ho dei punti di rumore o outlier, min li usa come dei punti di collegamento (ponticelli) per la generazione del cluster, generando così un errore nella separazione fra i cluster. Tale misura è quindi sensibile a rumore o outliers.



MAX

Questo tipo di distanza è meno sensibile a rumore e outliers ma non riesce a riconoscere cluster di dimensioni differenti.



Group Average

Si basa sul calcolo della distanza media fra tutti i punti appartenenti ad un cluster e tutti quelli appartenenti ad un altro. Questo tipo di tecnica è un compromesso fra MIN e MAX.

Questo tipo di tecnica non è tanto suscettibile al rumore però genera cluster di forme tipicamente globulari

Ward's method

Questa misura di distanza ragiona in base all'SSE (indice di qualità dei cluster) agglomerando per primi i cluster che aumentano di meno l'errore. Tale metodo si usa spesso per trovare i centroidi del k-means.

Metodo basato sulla misura di densità e prossimità

DBSCAN (1995)

Algoritmo basato su un paradigma diverso -> misura di densità e di prossimità

Posso individuare cluster di tipologie differenti. Bisognerà configurarlo settando eps e minpoints.

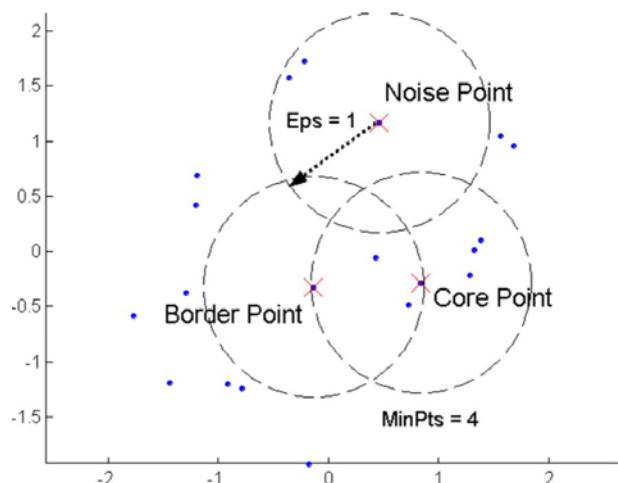
Dobbiamo definire la nozione di densità di un'area. Per farlo ci occorrono due dimensioni:

- Raggio di uno sferoide (chiamato eps)
- Centro dello sferoide

Per ogni punto che vogliamo "clusterizzare" dobbiamo quindi costruire uno sferoide d'influenza che ha centro nel punto interessato e raggio impostato come parametro.

All'interno dello sferoide dobbiamo contare quanti punti ricadono e usiamo un parametro soglia (minpoints oppure MinPts) per distinguere tre punti differenti:

- Core Point: Un punto è un core point se all'interno dello sferoide di raggio eps contiene minpoints punti. Ovvero l'area del suo sferoide è un'area densa. I punti core sono i punti al centro di un'area densa.
- Border Point: Non ha minpoints punti nella sua sfera d'influenza ma lui è contenuto nella sfera d'influenza di un core point. Quindi la sua non è un'area densa ma lui è contenuto nell'area densa di un core point.
- Noise Point: Non hanno minpoints punti nel loro sferoide e non sono nell'area densa di un altro punto.



L'algoritmo riesce ad etichettare i punti rumorosi. Questa è un'ottima feature che gli altri algoritmi di clustering non riescono ad avere.

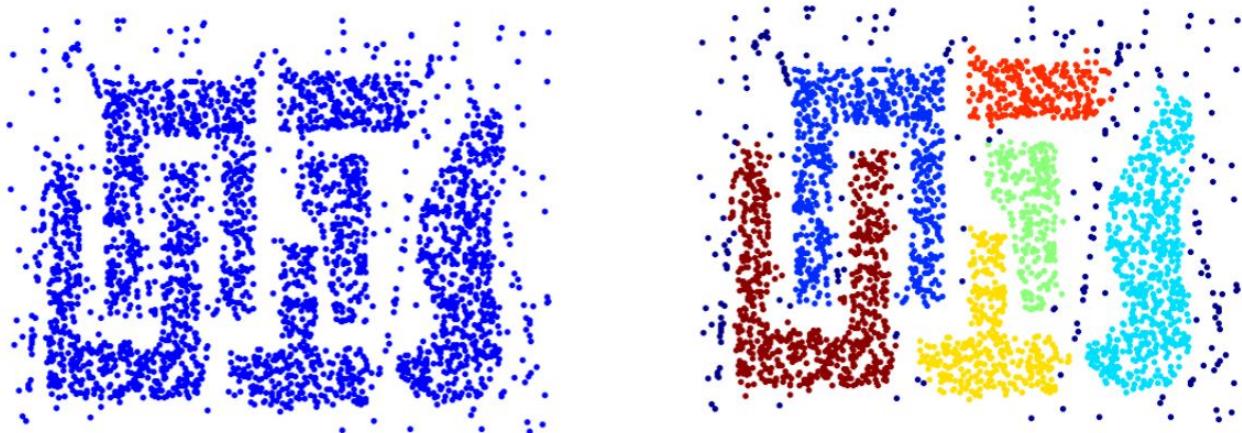
Esecuzione dell'algoritmo:

- etichetto tutti i punti come core, border o noise
- elimino i punti noise
- prendo un punto core, se ha già un'etichetta allora fa già parte di un cluster altrimenti devo creare un nuovo cluster assegnandogli una nuova etichetta
- osservo i punti nelle sue vicinanze e se questi punti non hanno un'etichetta allora li devo etichettare con l'etichetta del cluster appena creato
- prendo i nuovi punti appena etichettati e, se sono anche loro dei punti core, con il criterio di prossimità incomincio ad etichettare tutti i punti vicini con l'etichetta di quel cluster

Uso i concetti di densità e prossimità per inglobare tutti i punti vicini del cluster.

Tale procedimento è svolto per tutti i punti core, facendo espandere a macchia d'olio il cluster. In questo modo posso generare cluster con forme anche molto diverse da quelle globulari

I parametri critici da impostare sono epsilon (raggio) e minpoints.



PRO e Contro del DBSCAN:

PRO:

| | |
|--|---|
| estremamente resistente al rumore, i noise points sono tutti messi in un cluster a parte | Non individuo i cluster a densità variabile. In questi casi però potremmo eseguire l'algoritmo più volte modificando passo dopo passo il valore di densità, selezionando così i cluster in base alla loro densità |
| Nessun problema di forme e cardinalità dei cluster | Non risolve il problema dei dati a dimensionalità elevata |

CONTRO:

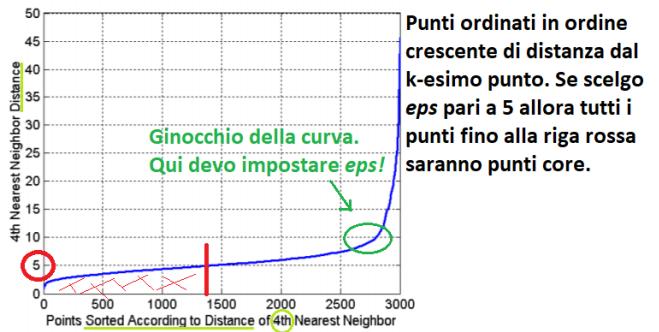
Approfondimento sui cluster a densità differente

Se ho dei cluster con valori di densità molto differenti posso avere dei problemi riguardo la generazione dei cluster. In questi casi posso utilizzare dei valori di densità differenti per tirar fuori prima i cluster più densi e poi quelli meno densi. Scegliendo opportunamente i parametri e modificandoli opportunamente di volta in volta.

Come posso scegliere il valore ottimale di *eps* e di *minpoints*?

Per scegliere il valore di *eps* e di *minpoint* devo cercare di capire l'andamento del clustering al variare dei parametri. Si usa il seguente metodo:

- Calcolo le distanze di tutti i punti da ogni altro punto (matrice di prossimità)
- Traccio un grafico dove fisso un valore di minpoints e per ogni punto traccio la distanza fra lui e il k-esimo punto più vicino. (nel grafico riportato k=4)
- Ordiniamo i punti in ordine crescente di distanza dal k-esimo punto



I punti a destra del ginocchio della curva saranno sicuramente dei punti rumorosi quindi devo selezionare un *eps* di distanza prima che la curva s'impenni in modo da escludere i punti rumorosi dai cluster.

Usiamo questo metodo perché lo studio di tutte le combinazioni possibili di *eps* e *minpoints* è troppo oneroso da svolgere.

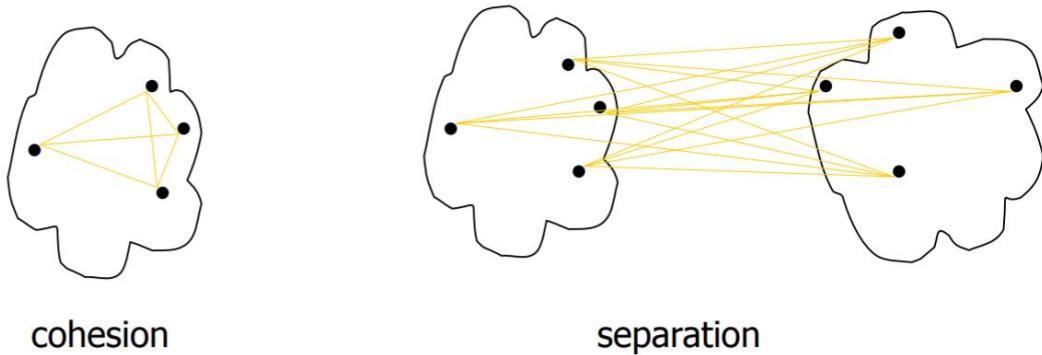
Quando le densità dei dati sono differenti, la nostra curva potrebbe avere una forma a gradini con più ginocchi. In quel caso scegliamo proprio i valori dei ginocchi per trovare gli *eps* che ci permetteranno di estrarre i cluster in base alle loro densità.

In questo modo posso scegliere il valore di *eps* adeguato, il valore di *minpoints* sarà uguale a *k*.

Misure di qualità del clustering

La qualità dell'algoritmo di clustering può essere valutata in due modi diversi:

- Posso usare degli indici “di tipo esterno” per valutare le etichette dei dati assegnate dal clustering con quelle che io conoscevo già prima. Questo posso farlo se ho già le etichette e sto inventando un algoritmo di clustering. Solitamente non ho le etichette di classe quindi questo metodo è poco utilizzabile.
- Indici interni: non hanno conoscenza delle etichette di clustering dei dati all'interno, uso quindi degli indici per valutare l'omogeneità dei punti. In particolare, valuto:
 - coesione: valutare quanto sono coesi i punti all'interno di un cluster. Possiamo misurare la distanza di un punto da tutti gli altri punti appartenenti allo stesso cluster e poi calcolare valori come la media o la somma.
 - separazione: quando sono distanziati fra loro i punti in due cluster diversi
 - SSE: funziona solo per forme globulari dei cluster



Silhouette

È un metodo, molto oneroso dal punto di vista computazionale, che valuta sia la coesione che la separazione. Spesso usiamo proprio questo indice interno per valutare la qualità del cluster.

Il funzionamento di questo indice è il seguente:

- Per ogni punto valuto due misure a e b . La misura $a(i)$ calcola la dissimiglianza, ovvero la distanza dall'i-esimo punto a tutti i punti dello stesso cluster e poi calcola la media. La misura $b(i)$ calcola la separazione dell'i-esimo punto, ovvero il valore medio della distanza del punto i da tutti gli altri punti appartenenti a tutti gli altri cluster.

$$\text{La silhouette di un punto è definita come } s(i) = \frac{b(i)-a(i)}{\max\{a(i), b(i)\}}$$

Questo indica varia fra -1 ed 1. Un buon algoritmo di clustering avrà Silhouette tendente ad 1, nel caso opposto si avrà invece un algoritmo di clustering di pessima qualità. Tale grandezza indica però la qualità per ogni singolo punto, per valutare la qualità del singolo cluster posso calcolare la media delle silhouettes di tutti i punti del cluster esaminato. Posso anche calcolare la silhouette globale fra tutti i punti del cluster.

Posso usare la silhouette a granularità diversa considerando così la qualità dei singoli elementi.

16. Triggers – Active Database System

È una modalità differente di gestione dei dati, definita Attiva.

Il DBMS può essere:

- Passivo: aspetta una richiesta per poi rispondere
- Attivo: in questo caso il DBMS è in grado di reagire ad un evento che avviene dall'esterno, facendo delle operazioni in più rispetto a quelle richieste passivamente.
Questi sistemi fanno monitoraggio degli eventi e reagiscono alle azioni d'interesse.

La differenza principale è che le regole scattano automaticamente all'accadere dell'evento.

Un'applicazione esterna dovrebbe invece intervenire esternamente in polling dopo una scansione dell'intera base dati.

Tali eventi attivi in un DBMS hanno una forma simile ad una regola. Sono composti da:

- Innesco: sono gli eventi da monitorare che portano all'esecuzione della regola
- Condizioni: sono le condizioni di attivazione della regola. Può essere opzionale e se non la specifico allora la condizione è sempre vera.
- Azioni: sono le azioni da svolgere, ovvero la sequenza di istruzioni SQL che devo seguire al verificarsi di un evento. Può anche essere una procedura applicativa

L'evento innesca un'azione in base a determinate condizioni.

Il motore relazionale monitora gli eventi e sceglie le regole le cui condizioni sono verificate dall'evento accaduto.

Gli eventi sono azioni di codifica della base dati, le condizioni sono predicati sulla base dati. In base al vero o falso della condizione (può essere anche opzionale) eseguo l'azione.

Supponiamo di voler costruire una regola per la gestione di un magazzino che scatena un'azione di riordino se la quantità di un prodotto scende al di sotto di una certa soglia. I parametri che osserveremo per la costruzione della regola sono:

- Eventi:
 - quando ho un **update** della quantità di un prodotto x
 - quando ho **l'inserimento** di un nuovo prodotto nella base dati e devo ancora scriverne la quantità
- Condizione:
 - La quantità del prodotto x è scesa sotto la soglia
 - Non ho già riordinato per il prodotto x. Perché se ho già emesso un ordine e sono quindi già sottosoglia, non dovrò rimetterlo in caso di un'altra vendita
- Azione: sarà l'emissione dell'ordine, possibilmente dopo un'approvazione umana in un caso reale

Per cosa sono usati i trigger?

- Per mantenere i vincoli d'integrità
- Azioni di memorizzazione delle modifiche aggiornando una seconda tabella a fronte della modifica di una prima tabella di riferimento. Tale operazione ci permette di mantenere consistenti i dati contenuti nella nostra base dati
- Gestione degli eventi di notifica per fare alerting

I triggers sono definiti utilizzando l'SQL (con funzione CREATE TRIGGER).

Noi studieremo la semantica d'esecuzione dei trigger in SQL. Ogni produttore una sintassi proprietaria che differisce un po' dallo standard.

Struttura di un Trigger

Un trigger è sensibile ad eventi di modifica dei dati che operano su una sola tabella alla volta.

Le azioni che possono scatenare un trigger sono:

- Insert
- Delete
- Update

Le condizioni sono un predicato SQL

Le azioni possono essere:

- Sequenza di istruzioni SQL
- Blocchi di codice programmati in altri linguaggi, ad esempio Java

Esecuzione di un trigger

Un trigger viene eseguito quando avviene l'evento e la condizione è verificata. In questo caso eseguo l'azione.

Ci sono diversi modi per eseguire il trigger:

- Prima dell'azione che lo innesca
- Dopo l'azione che lo innesca
- Prima del commit (questa condizione è stata scartata perché è troppo complessa da mantenere a livello di esecuzione)

L'esecuzione del trigger può essere di due tipi differenti:

- Tuple (or row level) -> il trigger viene eseguito una volta per ogni riga che innesca il trigger
- Statement level -> esegue il trigger una volta sola per tutta l'istruzione

Quindi abbiamo il modo di attivazione (before o after) e il livello di granularità, a livello di singola tupla, in cui eseguo l'istruzione (row level oppure statement level).

Sintassi di un Trigger Oracle

```
CREATE TRIGGER      TriggerName
    Mode Event {OR Event}
    ON TargetTable
    [[REFERENCING ReferenceName]
    FOR EACH ROW
    [WHEN Predicate]]
    PL/SQL Block
```

Mode:

- BEFORE: esegue il trigger prima dell'evento
- AFTER: esegue il trigger dopo l'evento
- INSTEAD OF: esegue il codice del trigger invece dell'evento (NON SI USA)

La maggior parte dei trigger sono di tipo AFTER. La semantica BEFORE si usa solo in rari casi dove è necessaria. Tipicamente per le Business Rules la semantica scelta è AFTER.

Event:

- INSERT: l'evento scatenante è l'inserimento di un nuovo dato
- DELETE: l'evento scatenante è la rimozione di un dato
- UPDATE [OF ColumnName]: l'evento scatenante è l'aggiornamento di un dato, posso anche specificare su quale singola colonna devo osservare l'aggiornamento

Posso scrivere più eventi con {OR Event}

TargetTable:

- È il nome della tabella su cui devo specificare gli eventi

FOR EACH ROW:

È opzionale. Se la scrivo voglio eseguire il trigger tupla per tupla. Se non la scrivo allora significa che la semantica è di tipo statement ovvero per ogni istruzione completa.

Quando faccio FOR EACH ROW posso vedere lo stato vecchio o lo stato nuovo della tupla che sta innescando il trigger. Per fare ciò devo usare le parole chiavi **OLD**, e **NEW**, seguite dal nome della colonna che voglio identificare. Posso poi mettere delle condizioni sia dentro la parte condizionale sia dentro l'azione che usano queste variabili. Nell'esempio della gestione del magazzino posso visualizzare OLD.QuantitàDisponibile oppure NEW.QuantitàDisponibile.

REFERENCING:

È opzionale. Si usa se voglio rinominare OLD. e NEW. attribuendogli dei nomi differenti

WHEN predicate:

È la condizione che **posso specificare solo se ho usato FOR EACH ROW**. Se la semantica è di tipo statement allora non posso specificare la condizione nella parte condizionale ma devo posizionarla dentro il trigger. In questo blocco condizionale posso utilizzare le variabili OLD. e NEW.

PL/SQL Block:

Qui inserisco il codice SQL che svolgerà delle azioni. Qui genererò delle azioni che generano un errore, come un Alert.

Limiti -> no istruzioni transazionali. Posso però sollevare un'eccezione.

Flusso d'esecuzione dei Trigger in Oracle

Quando Oracle deve eseguire un'istruzione transazionale osserva se tale istruzione innesca dei trigger. In caso affermativa eseguirà i trigger secondo quest'ordine:

- Tutti i trigger Before statement
- Tutti i trigger Before Row
- Modifico la base dati
- Controllo i vincoli d'integrità sulla tupla modificata
- Eseguo i trigger After Row
- Controllo i vincoli d'integrità di tabella
- Eseguo i trigger After statement

L'esecuzione dei trigger non è deterministica.

Errori in Oracle

L'esecuzione in cascata dei trigger in Oracle è consentita fino ad un massimo di 32 inneschi, dopodiché viene bloccata. I trigger possono innescarsi a vicenda, creando anche un'ipotetica situazione di loop d'innesto infinito.

Quando viene rilevato un errore:

- Faccio *rollback* di tutte le operazioni effettuate dai triggers
- Faccio *rollback* dell'intera transazione che ha innescato il trigger

In caso di errore durante l'esecuzione di un trigger Oracle si farà rollback ripristinando lo stato prima dell'esecuzione del trigger e prima dell'esecuzione dell'istruzione transazionale che ha generato l'errore.

I trigger si possono innescare anche a vicenda perché applicano delle modifiche della base dati. Per evitare la formazione di loop d'innesto fra i trigger, i DBMS bloccano il numero di inneschi ricorsivi ad un numero pari a 32. I Trigger Oracle non scatenano un errore in questo caso (DB2 sì).

Concetto di tabella Mutante

È la tabella modificata dall'istruzione della transazione che innesca il trigger. **NON può essere né letta né scritta dall'istruzione che innesca il trigger ROW**. Un trigger ROW può vedere solo la tupla che lo ha innescato, non posso quindi poi effettuare operazioni di SELECT su altri campi della tabella.

Un trigger di tipo **STATEMENT** può modificare la tabella mutante. Posso quindi usare la SELECT su altri campi della tabella.

Nell'esempio proposto il valore di soglia per il riordino di un prodotto deve essere contenuto nelle tabelle, e non nel trigger, perché così posso distribuire la soglia di riordino in modo differente per ogni prodotto.

Trigger in DB2

È una variante rispetto ai trigger SQL ma è più vicina allo standard.

```
CREATE TRIGGER      TriggerName  
    Mode Event  
    ON TargetTable  
    [REFERENCING ReferenceName]  
    FOR EACH Level  
    WHEN Predicate  
    Procedural SQL Statements
```

Mode:

- Before
- After

Event:

- Insert
- Delete
- Update

Posso scrivere un solo evento per ogni trigger, non sposo quindi mettere trigger con più eventi scatenanti insieme ma, in questo caso, dovrò creare due trigger differenti. **Ogni trigger DB2 è sensibile ad un solo evento d'innesto**

Level:

- ROW
- STATEMENT

In base a come vogliamo eseguire la procedura del trigger scriveremo FOR EACH ROW oppure FOR EACH STATEMENT

WHEN:

è la clausola che introduce la condizione che può essere specificata sia nei trigger ROW che nei trigger STATEMENT

Le variabili OLD e NEW possono essere accadute per i ROW triggers. Per i trigger di tipo STATEMENT posso accedere all'intera tabella vecchia o all'intera tabella nuova usando OLD_TABLE oppure NEW_TABLE.

In DB2 i Trigger BEFORE non possono eseguire modifiche della base dati (tranne le tuple che hanno scatenato il trigger), non potendo innescare altri trigger. In ORACLE invece non vi erano differenze fra i trigger BEFORE o AFTER, infatti entrambi potevano applicare delle modifiche sulla base dati. Così facendo evitiamo di poter scatenare dei loop di trigger BEFORE in modo ricorsivo. I trigger BEFORE si usano principalmente per mantenere i vincoli d'integrità.

I trigger in DB2 possono modificare le variabili di stato e le tuple di transizione.

Flusso d'esecuzione dei trigger in DB2

A differenza di Oracle, non vi è un flusso d'esecuzione ben definito. I Trigger possono essere eseguiti quindi in modo intercalato fra STATEMENT e ROW.

Si usa un ordine d'esecuzione che è legato all'ordine di generazione dei trigger nella base dati. Possiamo dunque scrivere i trigger in modo da deciderne la posizione d'esecuzione.

L'esecuzione dei trigger in DB2 è quindi deterministica.

L'esecuzione di un trigger, all'arrivo di una transazione T che contiene un'istruzione scatenante S che genera l'evento E, è indicata dai seguenti passi:

- Sospendo l'esecuzione della transazione T, lo stato viene salvato nello stack
 - Calcolo i nuovi valori dopo l'istruzione S e prima dell'istruzione S (OLD_TABLE e NEW_TABLE)
 - Eseguiamo i trigger BEFORE su E
 - Aggiorno la base dati (siccome i trigger before non scrivono la base dati in modo da poter aggiornare altri trigger allora sono sicuro che non si innescheranno trigger ulteriori)
 - Verifico i vincoli d'integrità. In questo punto posso innescare altri trigger in modo ricorsivo perché il rispetto dei vincoli d'integrità potrà portarmi a nuove INSERT, UPDATE o DELETE)
 - Eseguo tutti i trigger AFTER (anche qui in ordine deterministico, indipendentemente da ROW o STATEMENT). Questi trigger possono scrivere nella base dati quindi possono innescare altri trigger in modo ricorsivo.
 - Viene restituito il controllo alla transazione T che ripristina dallo stack lo stato d'esecuzione.
- Passiamo all'istruzione successiva della transazione.

Errori in DB2

Un errore in DB2 viene segnalato quando:

- vi è un innesco di molti trigger in cascata, rischiando di formare un loop infinito
- vi è un errore interno all'esecuzione di un trigger

Quando viene segnalato un errore:

- Viene effettuato un rollback delle operazioni effettuate dai trigger e dall'istruzione che ha scatenato il trigger

In caso d'innesco di molti trigger in cascata il DBMS bloccherà l'esecuzione segnalando un errore al programma chiamante. In questo modo evitiamo la formazione di loop infiniti d'innesco dei trigger.

In caso di errore durante l'esecuzione di un trigger DB2 si farà *rollback* ripristinando lo stato prima dell'esecuzione del trigger e prima dell'esecuzione dell'istruzione transazionale che ha generato l'errore. Proprio come nella gestione degli errori per i trigger Oracle solo che qui abbiamo una semantica leggermente differente che permette di effettuare il rollback direttamente dalla transazione.

Confronto fra i trigger Oracle e i trigger DB2

| | ORACLE | DB2 |
|---|----------------------|------------------------------|
| Riferimento a OLD_TABLE e NEW_TABLE nei trigger STATEMENT | NO | SI |
| Clausola WHEN nei trigger STATEMENT | NO | SI |
| Ordine d'esecuzione fra trigger ROW e STATEMENT | specificato | Arbitrario |
| Ordine d'esecuzione dei trigger | Non specificato | Ordine di creazione |
| Più di un evento permesso nel trigger | SI | NO |
| Accesso proibito alla Tabella Mutante | Si per i trigger ROW | NO |
| Uso della semantica instead of | SI | NO |
| Scrittura della base dati da parte dei trigger before | SI | NO, Solo vincoli d'integrità |

Progettazione dei Trigger

La definizione di un trigger è generalmente semplice. Le componenti che dobbiamo considerare sono:

- Semantica d'esecuzione (before/after)
- Evento
- Condizione (opzionale)
- Azione

Dobbiamo considerare l'eventuale mutua interazione fra i trigger che potrebbe generare dei problemi d'esecuzione. Ad esempio, potrei avere un innesco ciclico dei trigger che potrebbe ridursi in un loop infinito.

Le proprietà che i trigger devono rispettare sono:

- Terminazione: In uno stato arbitrario della base dati o per una transazione utente, l'esecuzione del trigger deve terminare in uno stato finale. (anche lo stato di *abort*)
- Confluenza: vogliamo che l'esecuzione termini in uno stato finale unico indipendentemente da come vengono eseguiti i trigger. Quindi l'ordine d'esecuzione di questi non influenza lo stato finale d'arrivo.

La terminazione è comunque garantita dal limite massimo d'inneschi in cascata consentiti dal DBMS. In realtà possiamo accorgerci prima dell'eventuale formazione di loop d'innesco rappresentando un grafo d'innesco (triggerring graph) che ci permette di effettuare un'analisi di ciclicità sui cicli d'innesco dei trigger.



Utilizzo dei Trigger per la gestione dei vincoli d'integrità

Spesso i trigger vengono usati per gestire i vincoli d'integrità, anche complessi. I passi per la stesura di un trigger per la gestione dei vincoli d'integrità sono:

- Scriviamo il predicato SQL corrispondente al vincolo d'integrità che vogliamo garantire. Questo vincolo può essere anche complesso. Potrà essere la condizione per il trigger, se complesso non possiamo usarlo come condizione perché non riusciamo a scriverla nella condizione ma dovremmo spezzarla nel blocco di azione.
- Capire a quali eventi è sensibile il vincolo considerato. (Update, Insert, Delete)
- Decidere cosa fare. Il trigger potrebbe cercare di fare *rollback* oppure tentare di riparare il vincolo d'integrità

Tipicamente sono trigger di tipo before

Manutenzione delle viste materializzate utilizzando i Trigger

Oggi i DBMS aggiornano automaticamente le viste materializzate ma queste possono essere gestite anche dai trigger. Una vista materializzata è una tabella che viene definita tramite un'interrogazione sulla base dati che serve ad accelerare l'esecuzione di altre interrogazioni. Le viste materializzate sono salvate in memoria, guadagnando però in tempo d'esecuzione.

I valori nelle viste materializzate devono essere allineati con la realtà, utilizziamo i trigger per aggiornare quindi i dati nelle viste materializzate.

Un esempio:

| | |
|----------------------------------|----------------------|
| S(SiD , SName, DCId) | Tabella studenti |
| DC(DCId , DCName) | Tabella Corsi |
| ES(DCId , TotalStudents) | Vista Materializzata |

Al primo riempimento devo popolare la vista materializzata utilizzando il seguente codice SQL:

```
SELECT DCId, SUM(*) AS TotalStudents  
FROM S  
GROUP BY DCId;
```

successivamente potrò utilizzare un trigger per l'aggiornamento della vista materializzata ES. Dobbiamo considerare che la vista materializzata deve essere aggiornata nel caso di:

- Insert di un nuovo studente nella Tabella Studenti
 - In caso d'inserimento di un nuovo studente in un DCId già esistente nella tabella ES dovrò fare un +1 del numero di studenti nella vista materializzata
 - In caso d'inserimento di un nuovo studente in un DCId non ancora esistente dovrò creare un nuovo corso e impostare il TotalStudents per quel corso ad 1.
- Delete di un nuovo studente nella Tabella Studenti
 - Dovrò aggiornare il TotalStudents per il relativo DCId facendo un -1 per il relativo corso
- Update dell'attributo DCId nella Tabella Studenti
 - Se DCId aggiornato è un DCId già esistente allora dovrò fare un +1 per il nuovo DCId e un -1 per il vecchio DCId
 - Se DCId aggiornato è un DCId non ancora esistente allora dovrò inserire il nuovo DCId e impostare il TotalStudents ad 1. Dovrò anche decrementare TotalStudents del vecchio DCId facendo un -1.

Bisognerà quindi scrivere 3 triggers differenti che gestiscano: Insert, Delete e Update.

Trigger esempio Insert per la gestione della vista materializzata

```
CREATE TRIGGER InsertNewStudent
AFTER INSERT ON S
FOR EACH ROW
DECLARE
N number;
BEGIN
SELECT Count(*) INTO N
FROM ES
WHERE DCId = :NEW.DCId      ;se il codice in ES = nuovo codice corso
;inserito in S
If(N<>0) then            ;se ho una tupla N!=0 quindi faccio +1
Update ES                 ;aggiorno ES facendo +1 a TotalStudents
Set TotalStudents = TotalStudents+1; ;faccio +1
Where DCId = :New.DCId     ;nella tupla interessata
Else
Insert into ES(DCId, TotalStudents)
Values(:NEW.DCId, 1)        ;inserisco i nuovi valori mettendo
;TotalStudents = 1
End if;
END;
```

Trigger esempio delete per la gestione della vista materializzata

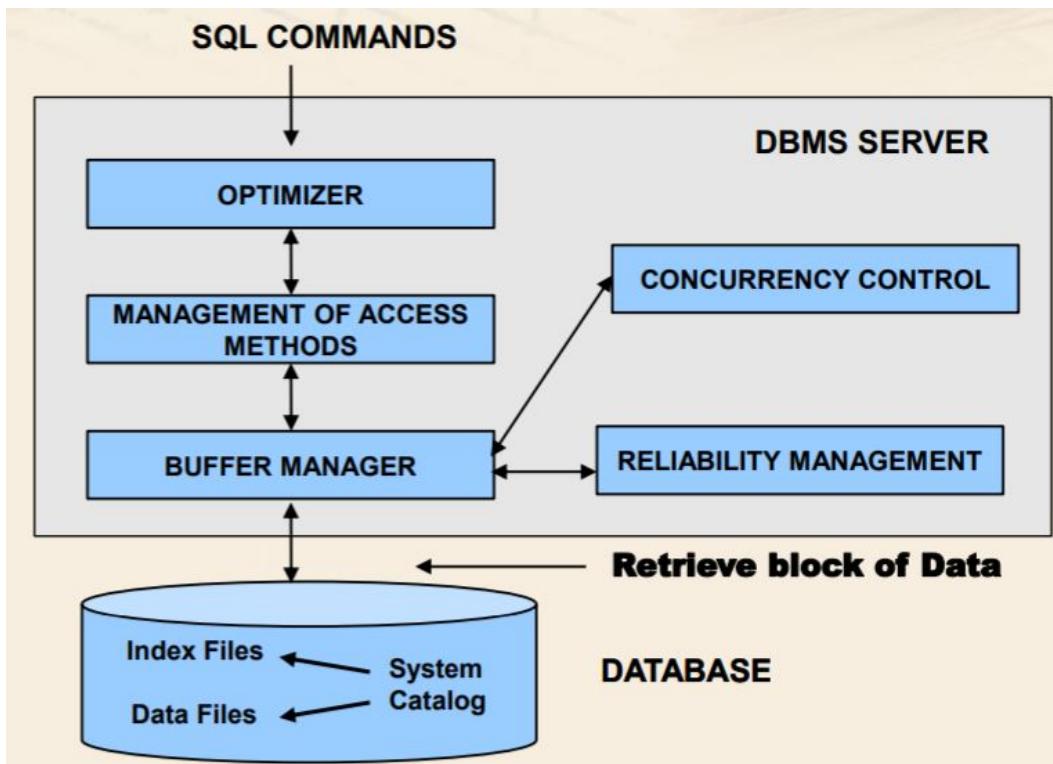
```
CREATE TRIGGER DeleteStudent
AFTER DELETE ON S
FOR EACH ROW
DECLARE
N number;
BEGIN
SELECT TotalStudents INTO N
FROM ES
WHERE DCId = :OLD.DCId
If(N = 1) then
DELETE FROM ES
WHERE DCId = :OLD.DCId;
Else
UPDATE ES
SET TotalStudents = TotalStudents -1
WHERE DCId = :OLD.DCId;
end if;
END;
```

Trigger esempio update per la gestione della vista materializzata

```
CREATE TRIGGER UpdateStudent
AFTER UPDATE DCId ON S
FOR EACH ROW
DECLARE
    N number;
BEGIN
    SELECT TotalStudents INTO N
    FROM ES
    WHERE DCId = :OLD.DCId      ;guardo il corso dove stava lo studente
    If(N = 1) then
        DELETE FROM ES
        Where DCId = :OLD.DCId
    Else
        UPDATE ES
        SET TotalStudents = TotalStudents -1
        Where DCId = :OLD.DCId
    End if;
    SELECT Count(*) INTO N
    FROM ES
    WHERE DCId = :NEW.DCId      ;guardo il corso dove andrà lo studente
    if(N<>0) then
        UPDATE ES
        SET TotalStudents = TotalStudents +1
        WHERE DCId = :NEW.DCId
    Else
        INSERT INTO ES(DCId, TotalStudents)
        VALUES(:NEW.DCId, 1)
    End if;
END
```

17. Struttura dei DBMS

In questa sezione studieremo il funzionamento interno dei DBMS. In particolare, l'architettura del DBMS che risiede al di sotto delle interrogazioni SQL che l'utente effettua.



L'ultima parte è oggi gestita direttamente dal sistema operativo. In precedenza, il DBMS formattava il disco nel formato utilizzato per la memorizzazione del file. Oggi invece tale compito è affidato al gestore dei file system.

Ciò che analizzeremo sarà dunque il contenuto della scatola azzurra.

Ottimizzatore

L'ottimizzatore riceve in ingresso le interrogazioni SQL ed esegue le seguenti operazioni:

- Effettua un'analisi sintattica dell'istruzione controllandone la correttezza e semantica cercando di capire cosa vuole fare quell'istruzione
- Traduce l'istruzione in una rappresentazione interna vicina all'algebra relazionale (perché è procedurale, ovvero indica l'ordine delle istruzioni da seguire)

Questo blocco garantisce l'indipendenza dei dati. Ovvero: non dobbiamo riscrivere un'interrogazione nel caso in cui cambiamo il modo in cui sono memorizzati fisicamente i dati nel nostro database. In questo modo posso modificare lo strato fisico (gli indici) senza dover riprogettare le applicazioni.

Management of Access Methods – Gestore d'accesso

Cerca di comprendere come fare ad accedere ai dati. Conosce gli indici che sono degli acceleratori di prestazioni. L'ottimizzatore dichiara di accedere ad un relativo dato usando un particolare indice. Il Gestore d'accesso conosce com'è fatto quell'indice e lo usa in modo adeguato.

Vi sono infatti indici basati su varie strutture dati (alberi, hash table etc etc..) e a seconda del tipo di indice devo accedere alla struttura fisica in modo differente.

Il gestore d'accesso conosce come sono organizzati i dati in base al tipo di indice che l'ottimizzatore gli ha detto di utilizzare.

Buffer Manager

È il componente che effettua l'accesso fisico al disco.

Fornisce le pagine che servono al Gestore d'accesso.

Tale componente ha un ruolo molto importante perché gestisce la memoria (RAM) del processo, non chiedendo altra memoria al sistema operativo.

Questo blocco è in comunicazione con il [Gestore d'Accesso Concorrente](#) e con il [Gestore d'Affidabilità](#).

Concurrency Control – Gestione Accesso Concorrente

Il Buffer Manager si occupa anche dell'accesso concorrente di più interrogazioni SQL alla stessa pagina. Ad esempio, quando tutti vogliono accedere alla stessa pagina di una materia di un corso di laurea. Sia in lettura che in Scrittura.

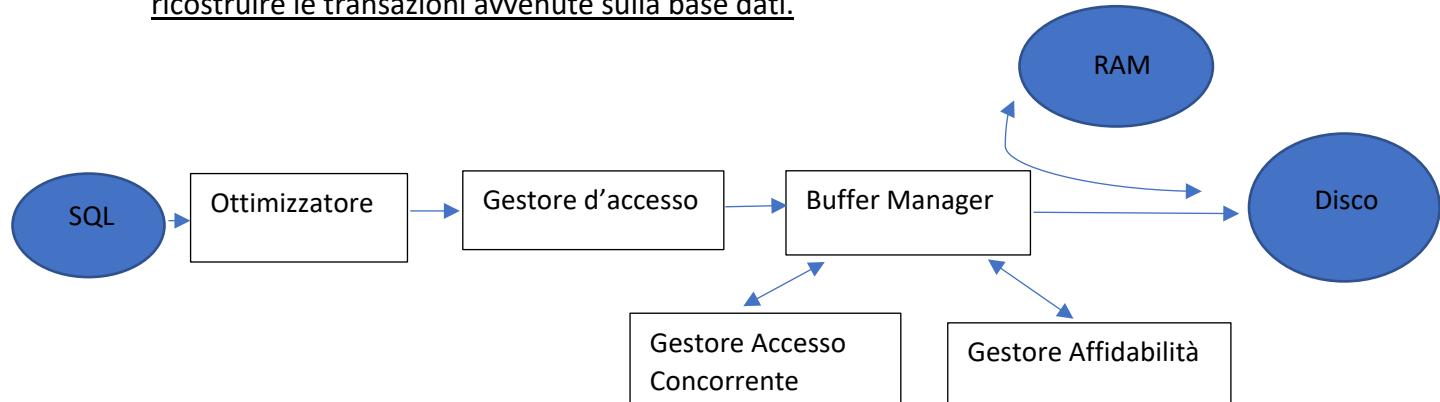
L'accesso in scrittura è più difficile da gestire rispetto a quello in lettura perché devo cercare di garantire l'integrità dei dati assicurandosi che le scritture non interferiscano fra di loro.

Reliability Management - Gestione Affidabilità

Garantisce che, durante la scrittura delle transazioni, la base dati non perda dei contenuti. Ovvero che le transazioni che terminano vengano correttamente scritte nella base dati.

Nell'eventualità di un guasto alla base dati, tale componente è in grado di ripristinare uno stato precedente al guasto dei dati.

Il funzionamento di tale componente è basato sull'uso di alcuni file di log che permettono di ricostruire le transazioni avvenute sulla base dati.



La Transazione

È un'unità logica di lavoro che viene eseguita da un'applicazione in esecuzione. Contiene una o più istruzioni SQL che vengono eseguite in sequenza. Tali istruzioni possono eseguire delle letture oppure delle scritture.

Le proprietà di queste transazioni sono:

- Correttezza
- Affidabilità
- Isolamento: la transazione deve eseguire come se fosse unica, senza nessun'altra transazione in parallelo. Quando accediamo ad un servizio in un sistema, all'utente deve sembrare di ricevere un servizio come se fosse l'unico a richiederlo anche se nel sistema avvengono più interrogazioni in modo concorrente.

Per garantire il corretto funzionamento delle transazioni dobbiamo garantire l'esecuzione completa di tutte le transazioni all'interno di un unico blocco. Abbiamo quindi dei delimitatori delle transazioni:

- Begin Transaction: non viene mai usato perché le transazioni vengono eseguite implicitamente quando ho un programma che accede ai dati.
- Transaction end: può finire in due modi.
 - Commit: La transazione va a buon fine. Scrivo "Commit;" alla fine del blocco di transazione e quando questo verrà eseguito, il DBMS mi darà "l'ok" allora mi garantirà che quei dati non sono andati persi.
La scrittura fisica sul disco è gestita successivamente dal DBMS, non avviene al momento del commit.
Consistenza Base Dati garantita modificata.
 - Rollback: In caso di errori durante l'esecuzione della transazione all'interno del blocco, l'istruzione di terminazione Rollback mi permetterà di ripristinare lo stato della base dati alla situazione precedente la transazione che ha avuto l'errore. Eliminando così le modifiche apportate dalla base dati dalle transazioni.
Il gestore dell'affidabilità tiene traccia delle transazioni effettuate per poi ripristinare lo stato in caso di errore.
Consistenza Base Dati garantita intoccata.

Tipicamente il 99.9% delle transazioni finisce in commit.

Il Rollback può essere chiesto da una transazione che va in errore (Suicidio) oppure dal sistema (Omicidio), ad esempio quando il sistema blocca l'esecuzione ricorsiva di una catena di trigger poiché troppo lunga.

Le proprietà che garantiscono il corretto funzionamento delle transazioni sono riassunte nelle proprietà ACID.

- **Atomicity**
- **Consistency**
- **Isolation**
- **Durability**

Atomicity

Non posso dividere una transazione in sotto pezzi, l'eseguo per intero (esempio bonifico) o non la eseguo affatto (rollback).

Gli stati intermedi non vengono mai memorizzati in modo permanente nella base dati.

Le operazioni che utilizzeremo per garantire l'atomicità sono:

- Undo: Permette di disfare il lavoro effettuato fino a quel punto all'interno della base dati
- Redo: Riesegue il lavoro che la transazione aveva svolto che poteva non essere stato scritto fisicamente sul disco a causa di un eventuale errore. Se ho ricevuto l'ok ad un commit allora ho scritto in un log le operazioni di modifica della transazione eseguita e, in caso di errore, posso effettuare la scrittura sul disco garantendo la corretta consistenza dei dati all'interno della base dati.

Consistency

È legata alla correttezza che è garantita dai vincoli d'integrità.

Quando avviene una violazione il sistema ha due opzioni:

- Annulla l'operazione invalidante
- Ricostruisce uno stato nuovo e corretto della base dati (`raise_application_error`)

Isolation

L'esecuzione di una transazione è indipendente da tutte le altre transazioni che stanno operando sugli stessi dati. ([GESTORE ACCESSO CONCORRENTE](#))

Le richieste ricevute vengono ordinate in maniera opportuna in modo da garantire l'accesso concorrente. L'esecuzione dei programmi è interlacciata tramite uno scheduling gestito dal [gestore dell'accesso concorrente](#).

Durability - Persistenza

È legata all'affidabilità dei dati. ([GESTORE AFFIDABILITÀ](#))

Le transazioni che hanno fatto commit non verranno mai perse, anche in caso di errore. Riuscirò quindi a mantenere un certo grado di affidabilità dei dati.

Il funzionamento è basato sull'esistenza dei file di log.

Buffer Manager

Tale componente ha l'incarico di prelevare i dati dalla memoria di secondo livello (disco o a stato solido) e posizionarli in memoria centrale per essere elaborato dal sistema.

Effettua quindi il trasferimento di dati.

Il Buffer Manager gestisce anche l'allocazione dinamica della memoria del processo che non viene quindi più gestita dal Sistema Operativo.

La gestione del buffer manager è un punto chiave per le performance di un DBMS.

Buffer

È lo spazio di memoria che viene allocato quando viene avviato il processo padre per la gestione del Database. Questa memoria viene condivisa dalle transazioni (processi) in esecuzione che chiedono l'accesso ai dati. La memoria allocata viene dunque usata per i dati e non per il programma.

All'interno del buffer la memoria è organizzata in pagine e queste pagine sono legate come dimensioni alla grandezza massima della pagina che può essere trasportata dal sistema operativo. Le pagine nel buffer hanno grandezza pari ad un multiplo della grandezza massima trasportabile dal sistema operativo.

Alcune delle funzioni base per la gestione del buffer sono uguali a quelle già utilizzate per la gestione delle memorie all'interno dei sistemi operativi:

- Principio della località dei dati: I dati che sono stati utilizzati di recente è probabile che vengano nuovamente utilizzati.
- Legge 20-80: Il 20% dei dati è tipicamente letto dall'80% delle applicazioni

Oltre al blocco dei dati scambiato con il Sistema Operativo, dentro al Buffer vengono contenute delle informazioni ausiliarie per la gestione fisica dei dati:

- Informazione della posizione di ogni pagina su disco. Ciò è necessario nel caso in cui io voglia scrivere su disco la pagina. In questo caso dovrò infatti riposizionarla nel posto opportuno. In caso di modifica di una tupla verrà scambiata l'intera pagina contenente la tupla modificata fra memoria primaria e disco. Devo quindi sapere qual è il file e la posizione all'interno del file dove posizionare il blocco.
- Variabili di stato. Mi segnalano cosa sta accadendo a quella pagina.
 - Count: il numero di transazioni che in quel momento stanno accedendo alla pagina
 - Dirty bit: Afferma se la pagina è stata modificata da qualche transazione

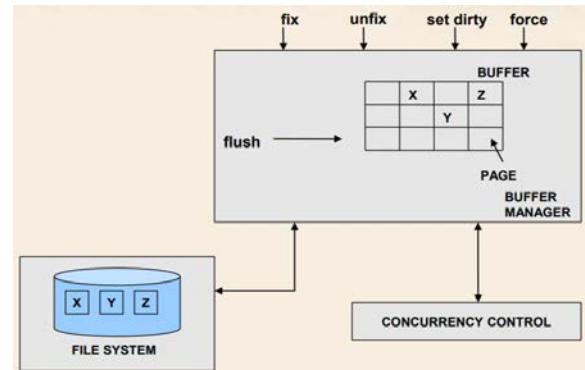
Il Buffer Manager mette a disposizione del sistema alcune primitive che permettono di operare per impartire dei comandi al buffer manager. (ad esempio *fix*)

Il Buffer Manager esegue quindi le seguenti operazioni:

- Carica la pagina in memoria (o verifica che la pagina sia già caricata)
- Chiede al gestore dell'accesso concorrente se può rendere disponibile la pagina alla transazione che la richiede

Le primitive che ci permettono di operare con il Buffer Manager sono:

- [Fix](#)
- [Unfix](#)
- [Force](#)
- [Set dirty](#)
- [Flush](#)



Primitiva FIX

Serve per chiedere l'accesso ad una determinata pagina sul disco. La transazione conosce già quale pagina le serve grazie al blocco “gestore dei metodi d'accesso”.

Il funzionamento per il richiamo di una pagina è il seguente:

- La transazione chiede l'accesso ad una pagina che è sul disco
- Buffer Manager preleva la pagina dal disco e la carica in memoria (Buffer)
- Se il sistema di accesso concorrente dà l'ok il puntatore alla pagina nel Buffer viene restituito alla transazione
- Il contatore del numero di transazioni che sta accedendo a quella pagina viene incrementato di uno rispetto al valore precedente

La primitiva FIX esegue le seguenti operazioni:

- Monitora se la pagina richiesta dalla transazione è già nel Buffer
- Se è già nel Buffer allora ho finito altrimenti, se la pagina non è in memoria devo:
 - Cercare un posto libero in memoria (Buffer) dove allocare la pagina. Possiamo avere due casi:
 - Se vi è una pagina libera allora salvo in memoria principale la pagina d'interesse
 - Se non vi è la pagina libera posso andare a vedere se vi sono delle pagine non libere ma che hanno count=0 ovvero che le transazioni che stavano lavorando in quella pagina l'hanno già rilasciata (unfix). Anche se il [gestore dell'accesso concorrente](#) non ha ancora "mollato" quella pagina.
Le pagine non libere con count=0 vengono chiamate "vittime" e possono essere eliminate. Queste pagine potrebbero però avere il Dirty bit=1 quindi dovrò prima scriverle sul disco e solo successivamente potrò sovrascrivere in memoria principale con la nuova pagina che voglio salvare. Tale metodo di scrittura prende il nome di *modo sincrono* (il sistema operativo scrive in quel momento e non quando vuole lui).

Primitiva UNFIX

Quando una transazione utilizza la primitiva di unfix rilascerà la pagina utilizzata. In questo caso il count per quella pagina viene decrementato di uno.

Primitiva SET DIRTY

Quando una pagina è stata aggiornata, la transazione che ha effettuato la modifica chiama la primitiva SET DIRTY impostando il flag Dirty a 1. In questo modo la pagina modificata verrà caricata su disco.

Primitiva FORCE

Può essere utilizzata dalle transazioni e richiede il trasferimento sincrono della pagina sul disco.
"Questa pagina deve essere scaricata sul disco adesso. Aspetto finché la pagina non viene scritta".
La transazione che richiede la Force viene sospesa finché il sistema operativo non effettua la scrittura e ritorna il comando al Buffer Manager.

Primitiva FLUSH

Questa primitiva viene richiamata quando ci sono dei cicli liberi di CPU. Grazie a questa primitiva il Buffer Manager scarica delle pagine dalla memoria principale alla memoria su disco.
Sfruttando così i momenti dove il Buffer Manager non è già impegnato.

Politica di Gestione delle Pagine

Vi sono diverse politiche di gestione delle pagine che permettono di diversificare le performance. Tipicamente una politica “per bene” andrà più lenta di una politica meno “per bene”.

Vari tipi di politica di gestione delle pagine:

- Steal: Permette al Buffer Manager, in caso di necessità di memoria, di scaricare le pagine con count=0 dalla memoria principale anche se sono ancora bloccate da una transazione in corso d'esecuzione perché il gestore dell'accesso concorrente non le ha ancora rilasciate. Questa tecnica è più rischiosa.
Quando vogliamo fare steal di una pagina con dirty_bit=1 che non è ancora stata rilasciata, bisognerà scriverla sul disco. Se non è stata ancora rilasciata allora la transazione che la ha modificata non ha ancora effettuato il *commit*. Stiamo quindi scrivendo su disco una pagina la cui transazione potrebbe finire ancora in *abort*. In caso di *abort* sarà più difficile effettuare il rollback poiché dovrò ricaricare la pagina in memoria.
- No Steal: Il Buffer manager non può prendere le pagine con count=0 ma deve aspettare che queste diventino libere. Il Buffer Manager in questo caso dovrà mettere in attesa più transazioni. Questa tecnica è più “lenta” della precedente ma è più “per bene”
- Force: Le pagine attive di una data transazione, ovvero le pagine che la transazione ha modificato e quindi hanno dirty_bit=1, vengono scritte in modo sincrono su disco appena la transazione fa il commit. Il controllo alla transazione viene restituito al commit dopo che il Sistema Operativo ha effettuato la scrittura della pagina.
- No Force: Le pagine sono scritte sul disco in maniera asincrona. Le pagine modificate rimangono nel buffer e poi vengono scritte in modo asincrono usando la primitiva FLUSH. In questo caso devo essere sicuro che anche in caso di eventuali guasti riuscirò a scrivere la pagina sul disco. Anche in caso di perdita dei dati in memoria principale riuscirò comunque a scrivere le pagine modificate su disco grazie alla primitiva Redo.

Le politiche di gestione delle pagine tipicamente utilizzate sono Steal e NoForce insieme.

Il File System e il Buffer Manager sono a stretto contatto poiché il Buffer Manager utilizza i servizi offerti dal File System. Le operazioni di accesso alla memoria sono:

- Lettura
 - Posso chiedere di leggere un blocco
 - Posso chiedere di leggere una sequenza di N blocchi contigui
- Scrittura
 - Posso scrivere un blocco singolo
 - Posso scrivere una sequenza di N blocchi contigui

Gestore dei Metodi d'Accesso

Le pagine che vengono memorizzate sul disco possono essere memorizzate in formati diversi. Il Gestore dei Metodi d'Accesso conosce come sono strutturati i dati all'interno della pagina a seconda della tipologia di pagina che sta leggendo. Vi sono infatti svariati formati per le pagine dei dati e per le pagine degli indici.

Tale componente riceve dall'ottimizzatore un particolare percorso d'accesso (che può avvenire usando l'indice o direttamente accesso ai dati) e in caso d'uso di indice il gestore dei metodi d'accesso ne capisce la tipologia e lo usa adeguatamente per accedere ai dati.

Metodo d'accesso

Capisce quali sono i blocchi che devono essere caricati dalla memoria e li richiede al Buffer Manager. Il metodo d'accesso conosce la struttura fisica della pagina ed è in grado di reperire tuple o informazioni dentro la singola pagina.

Struttura della Pagina

La pagina è composta da una parte che è l'informazione utile e da un'altra parte che sono altre informazioni utili per reperire i dati. Quest'ultima parte è scritta dal Metodo d'accesso. Un altro piccolo pezzo della pagina è scritto dal sistema operativo.

Le tuple hanno una dimensione variabile, ad esempio le stringhe sono salvate in maniera dinamica allocando la dimensione opportuna per il contenimento dei caratteri interessati.

Devo anche gestire dati di dimensioni oltre quelli di una pagina.

Strutture d'accesso

Servono a garantire l'accesso ai dati dalle istruzioni SQL.

Le strutture fisiche d'accesso descrivono come i dati sono memorizzati nel disco.

Nei sistemi relazionali abbiamo strutture d'accesso, quindi metodologie, differenti per memorizzare:

- Dati (Tabelle)
 - Strutture Sequenziali
 - Tabelle di Hash
- Indici (acceleratori)
 - Strutture ad Albero (B-Tree, B⁺-Tree)
 - Indici hash non clusterizzati
 - Bitmap index

Memorizzazione Tabelle - Strutture Sequenziali

Le tuple sono memorizzate in ordine sequenziale. Gli ordinamenti interni delle tuple possono essere di due tipologie differenti:

- Heap file: tuple ordinate in ordine di arrivo
- Ordered sequential structure: ordinata su uno o più campi chiave

Heap File

Le tuple sono ordinate nell'ordine d'inserimento. La scrittura avviene quindi sempre in "append". Il sistema riempie le pagine in modo ordinato. Prima riempie una pagina, la chiude e poi ne apre un'altra iniziando a riempire anche questa.

Anche se le scritture su disco posso avvenire sia in modo sincrono che asincrono, sono dunque indipendenti da questo processo.

-PRO e CONTRO del metodo heap File:

PRO:

CONTRO:

| | |
|----------------------|--|
| Lettura molto veloce | quando faccio una delete marco la tupla che deve essere cancellata ma così lascio un buco nella pagina perché ormai è chiusa. |
| | Quando faccio update, se la tupla aggiornata ha la stessa dimensione della vecchia tupla potrò metterla senza problemi. In caso contrario non potrò inserirla. |

Tutte queste problematiche si hanno poiché le tuple hanno dimensione variabile. Tale metodo d'ordinamento spreca più memoria ma è più veloce.

Ordered Sequential Structure

Le tuple all'interno della pagina sono ordinate in base ad una chiave d'ordinamento.

Tale chiave d'ordinamento prende il nome di sort key e può riguardare più attributi. Posso quindi effettuare un ordinamento su più attributi.

Le strutture ordinate sequenziali sono più efficienti quando devo effettuare delle operazioni di Join o di OrderBy.

Oggi però non usiamo strutture ordinate sequenziali poiché quando devo effettuare delle operazioni di insert o di delete è complicato mantenere l'ordinamento delle tuple all'interno delle pagine. La memorizzazione delle pagine avveniva dunque lasciando una percentuale di spazio libero prevista per le insert future.

Questo tipo di ordinamento delle tuple all'interno delle pagine viene usato spesso con i B⁺-Tree.

Strutture ad Albero – Balance Tree – B-Tree e B⁺-Tree

Sono strutture di memorizzazione dei dati utilizzate per gli indici.

L'obiettivo delle strutture ad albero è quello di consentirmi un accesso "quasi" diretto ai dati basandosi sull'uso di una chiave.

Vogliamo quindi evitare una scansione della base dati per la ricerca di uno specifico dato.

La struttura ad albero non mi vincola la posizione delle tuple all'interno della Base Dati. Le strutture ad heap o le strutture sequenziali sono invece vincolanti per la posizione delle tuple.

L'albero è strutturato in modo che:

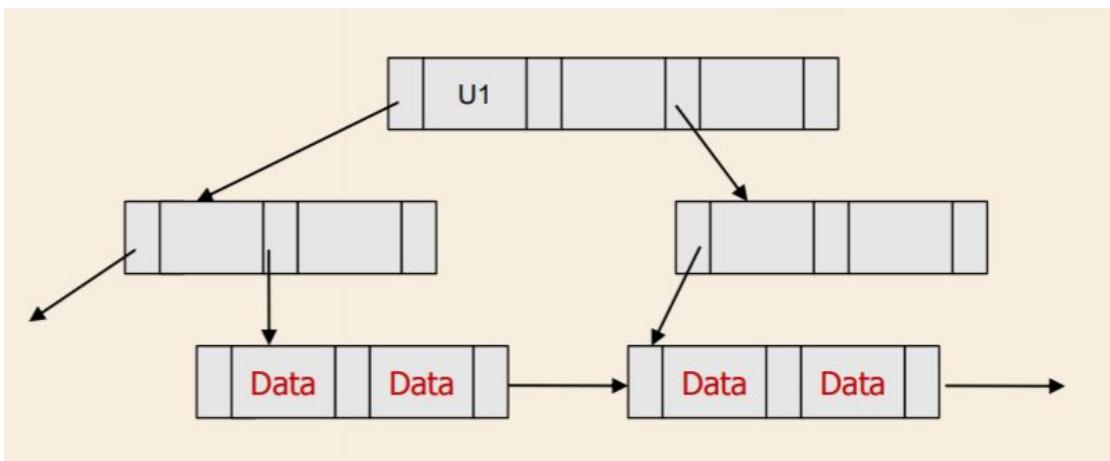
- **Clustered** (key-sequence): Le tuple sono contenute direttamente nei nodi foglia dell'albero. Tale soluzione è preferibile rispetto alle strutture sequenziali perché mantengo un ordinamento dei dati ma ho un accesso migliore rispetto al sequenziale.
- **Unclustered** (indice secondario): I nodi foglia dell'albero contengono i puntatori per il raggiungimento del dato. Ecco perché la posizione delle tuple non è vincolata dalla struttura dell'albero. Sfruttando infatti i puntatori posso scambiare l'ordine delle tuple senza perdere il loro riferimento.

Gli alberi usati per le strutture di memorizzazione sono:

- **B-Tree**: Per arrivare ai dati (nodi foglia) posso passare solamente dalla radice dell'albero.
- **B⁺-Tree**: Per raggiungere i dati posso passare dalla radice dell'albero come nel B-Tree ma in più posso navigare sequenzialmente la parte dati. Tutti i nodi foglia dell'albero sono concatenati, con dei puntatori, in una sequenza di ordinamento che è la chiave dell'albero. In questo modo posso evitare l'ordinamento fisico delle tuple e mantenere un ordinamento sulla chiave direttamente sulle foglie dell'albero per raggiungere le tuple interessate.

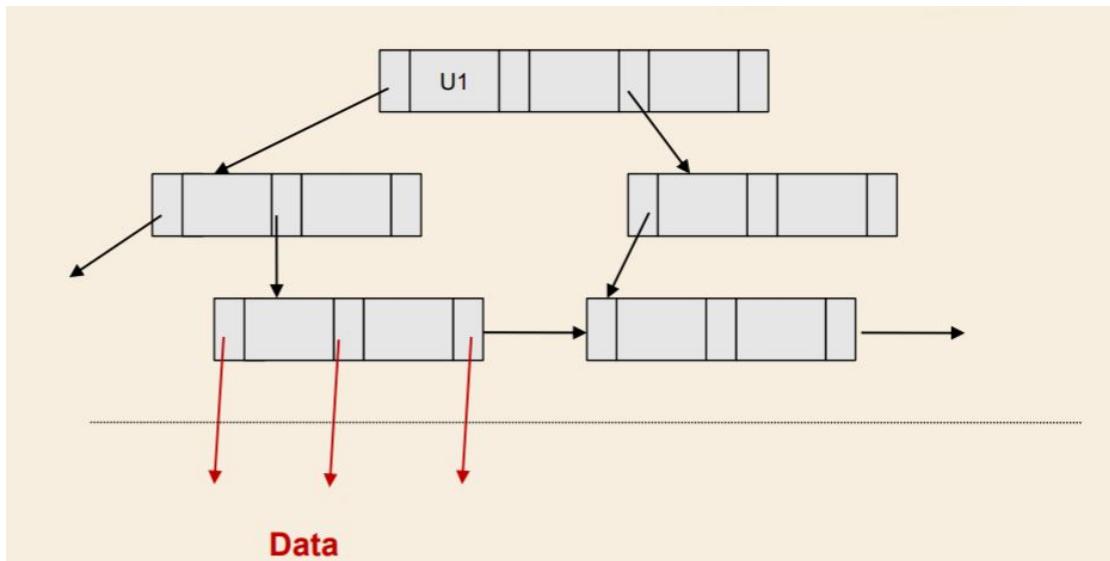
Per mantenere una velocità d'accesso ai dati costante (indipendente dai valori cercati), gli alberi che memorizzano le informazioni sui dati sono costruiti in modo **balanced**. Questo perché una struttura differente, come un albero binario, potrebbe generare un albero sbilanciato con conseguenti tempi d'accesso differenti ai dati.

Esempio di B⁺-Tree Clustered



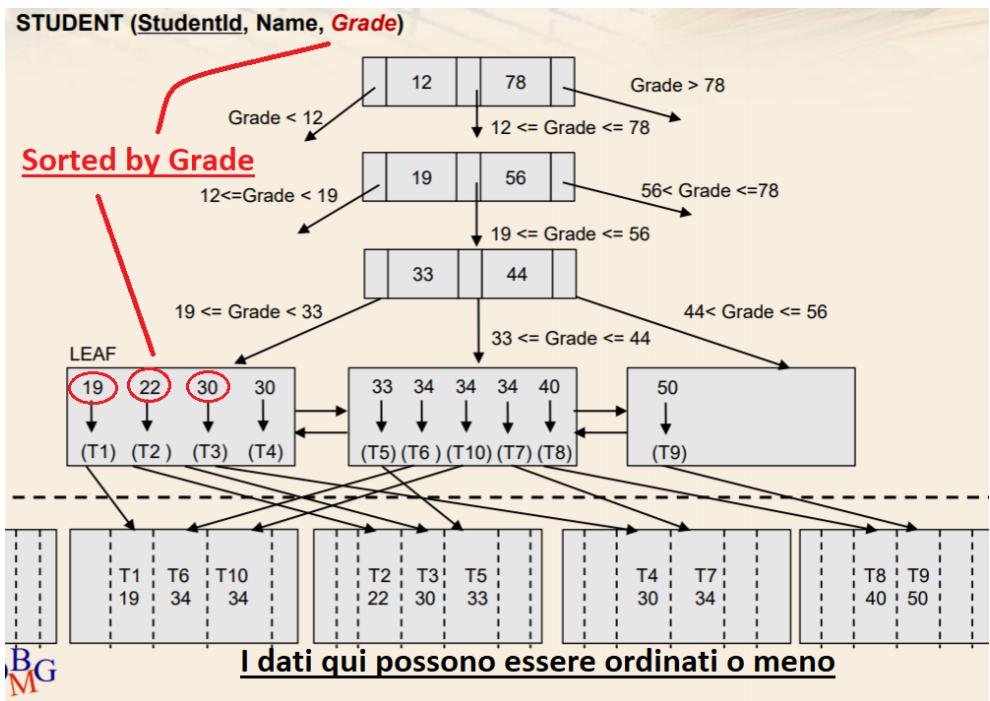
Come possiamo vedere, possiamo navigare all'interno dei nodi foglia grazie a dei puntatori. I nodi foglia contengono direttamente i dati (tipo Clustered). In questo modo posso anche aumentare direttamente lo spazio del nodo foglia spaccandolo in due nel caso di una update per una generica tupla. Tale soluzione è quindi più versatile rispetto ad un [Heap File](#). Tali soluzioni sono spesso utilizzate per memorizzare i dati secondo la chiave primaria.

Esempio di B⁺-Tree Unclustered

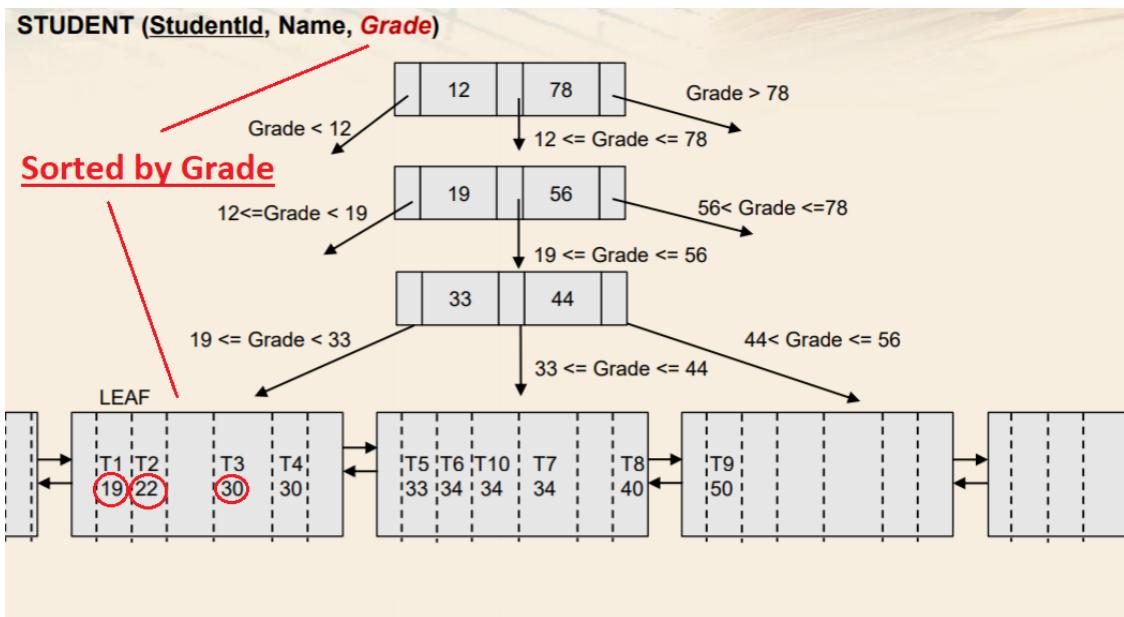


In questo caso i nodi foglia contengono dei puntatori fisici ai blocchi di dati che sono collocati in altre strutture dati (Heap File, Strutture sequenziali ordinate). In questo modo posso recuperare le tuple interessate indipendentemente dalla loro posizione fisica all'interno del disco.

B⁺-Tree Unclustered



B⁺-Tree Clustered



L'accesso ai dati per la loro lettura applica un ritardo nel metodo unclustered perché dovrà ritrovare il dato utilizzando l'accesso tramite il puntatore fisico.

PRO e CONTRO degli Alberi

PRO

CONTRO

| | |
|--|--|
| Ottimi per le query d'intervallo | Insert possono creare problemi perché se le dimensioni della foglia non permettono l'insert devo spaccare in due il nodo. Se il padre della foglia da spaccare è pieno devo spaccare in due anche lui. Tale situazione si può ripercuotere fino al nodo di root, in questo caso dovrò spaccare in due anche il root generando un nuovo nodo di root. Ribilanciamento alberi operazione costosa |
| Buoni per letture sequenziali in un certo ordine | Eliminare delle informazioni può essere complicato perché potrei dover effettuare il merge fra più nodi per risparmiare spazio. Dovendo propagare questo merge a ritroso fino al root dell'albero. |

Hash Structure

Tali strutture permettono l'accesso ai dati tramite uno o più campi chiave.

Questi meccanismi applicano una funzione (funzione di *hash*) a quel campo chiave.

Per il funzionamento si suppone di dividere la struttura fisica di memorizzazione in B blocchi.

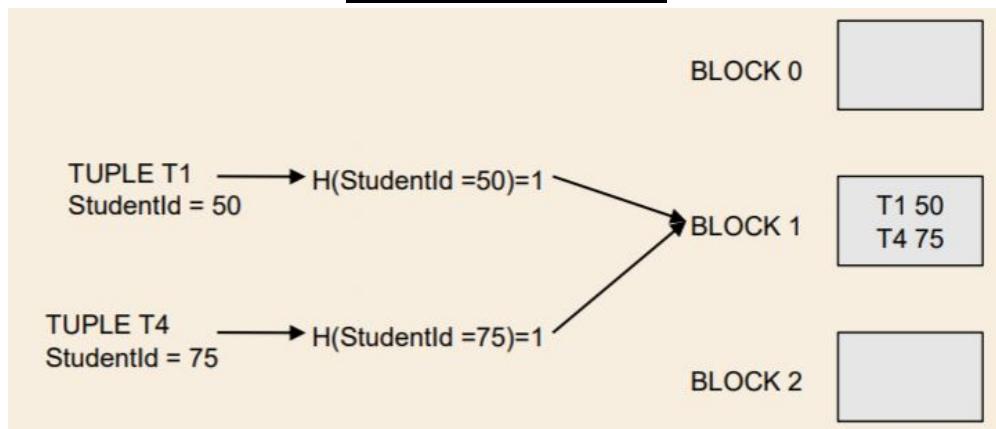
La funzione di *hash* restituisce un valore compreso fra 0 e $B-1$ quando viene applicata al campo chiave che è stato scelto per definire la struttura, indicandomi così dov'è collocato il blocco che sto cercando.

Bisogna dimensionare opportunamente lo spazio disponibile per questi B blocchi in modo da lasciare del margine per eventuali futuri inserimenti. Tipicamente questi blocchi non sono mai riempiti completamente.

Una volta trovato il blocco dov'è posizionato il mio dato dovrò poi cercarlo all'interno del blocco stesso.

Tale sistema mi fornisce un indice che indica la posizione del blocco.

Hash Structure Clustered



PRO e CONTRO delle strutture Hash:

PRO

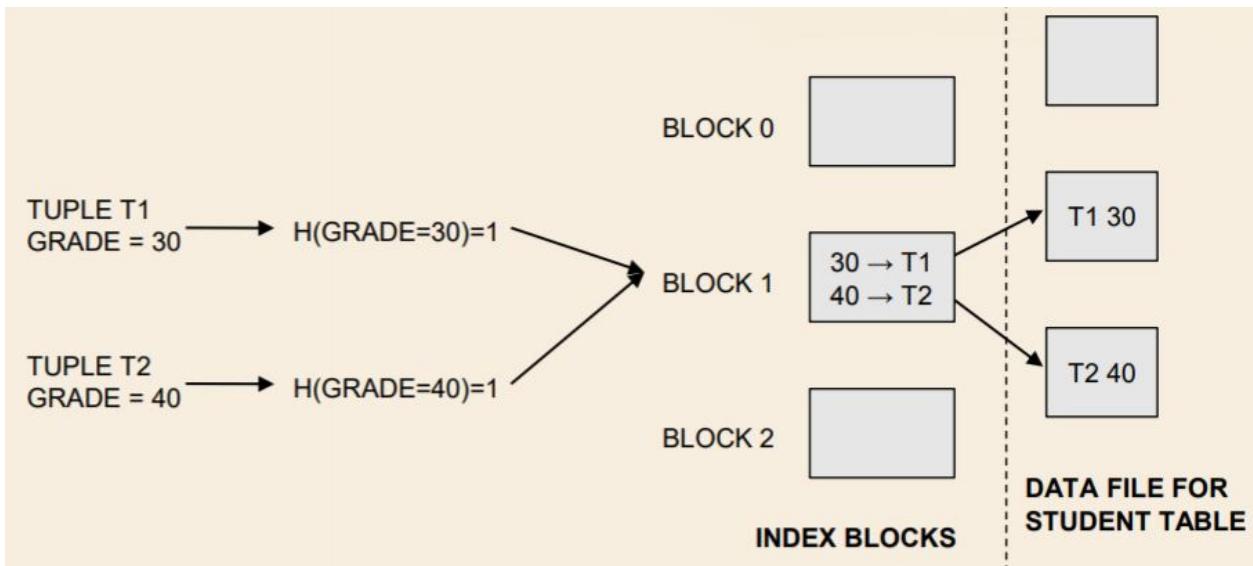
CONTRO

| | |
|---|--|
| Non necessita un ordinamento dei dati all'interno del disco | se volessi fare delle <u>interrogazioni d'intervallo</u> , che selezionano un range di valori possibili, dovrei potenzialmente leggere tutti i blocchi che mi interessano. In questo caso dovrei caricare molti blocchi in memoria principale. |
| Efficiente nelle query che usano la chiave come predicato | Le collisioni possono causare il riempimento di un blocco della mia struttura dati in caso di insert |

Hash Index Unclustered

Fornisce un accesso diretto, utilizzando un'opportuna funzione di hash, al blocco che contiene l'indirizzo alla tupla cercata.

A differenza del caso clustered, i blocchi non contengono la tupla fisica ma il puntatore alla tupla fisica cercata.



Bitmap Index

Il funzionamento delle Bitmap Index è basato sulla realizzazione di una matrice di bit che è una struttura separata dai dati. In questo modo non abbiamo nessun vincolo sulla posizione delle tuple all'interno del file.

Bitmap Index sfrutta la posizione della riga che vogliamo cercare (IDs – ROW IDentifiers).

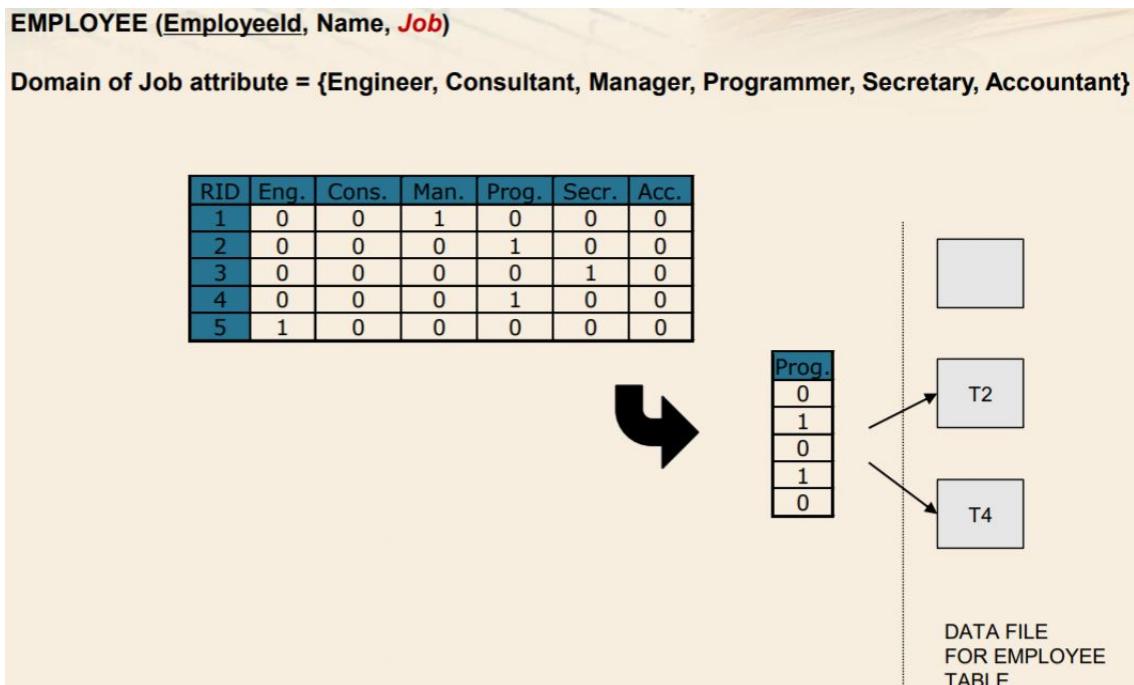
Questa tabella ha una riga per ogni tupla della mia base dati e poi ha una colonna per ogni campo dell'attributo che voglio indicizzare. Se quindi ho un attributo con 5 valori allora avrò 5 colonne nella mia matrice di bit.

Ogni intersezione riga-colonna memorizza solamente 1 bit. Il significato di tale bit è:

- Bit **0**: quella riga non ha quel valore per l'attributo indicizzato
- Bit **1**: quella riga, per l'attributo indicizzato, ha quel valore

EMPLOYEE (EmployeeId, Name, Job)

Domain of Job attribute = {Engineer, Consultant, Manager, Programmer, Secretary, Accountant}



Nell'esempio se voglio tutti gli impiegati che come Job fanno il programmatore dovrò prelevare la colonna "Prog." e in base alla posizione dei bit ad 1 capirò quale Row Identifiers sarà quello che identifica la tupla interessata.

Vantaggi e Svantaggi degli indici Bitmap

PRO

CONTRO

| | |
|---|---|
| Molto efficiente quando ho dei predicati con delle espressioni booleane | Indice adatto soltanto se il dominio dell'attributo è limitato. Non devo indicizzare sulla chiave primaria (tipo matricola) perché altrimenti otterrei un indice diverso per ogni riga e non sarebbe significativo per questo metodo. |
| Appropriato per domini con cardinalità limitata | Non idoneo per indicizzare valori continui. Posso valutare di effettuare la discretizzazione |

Ottimizzatore - Query Optimization

Vogliamo capire come fa il sistema a tradurre una query SQL in un'azione di reperimento dei dati all'interno della nostra struttura dati. Vogliamo quindi decidere come eseguire un'interrogazione SQL.

SQL è un linguaggio dichiarativo. Un'interrogazione SQL afferma di voler richiedere le proprietà dei dati che voglio reperire ma non dice nulla sul come voglio fare per reperirli. Infatti, le condizioni nella *where* sono commutabili come ordine.

Sarà quindi il sistema a scegliere l'ordine di esecuzione delle istruzioni e l'ordine di lettura delle tabelle.

L'arma vincente dell'SQL esteso è 'l'indipendenza dei dati' ovvero l'indipendenza fra le istruzioni SQL e il modo in cui sono memorizzati fisicamente i dati all'interno della mia base dati. In questo modo non devo riscrivere il parco applicativo nel caso in cui io decida di cambiare il modo in cui sono memorizzati i dati all'interno del mio database. (ad esempio, quando cambio l'indicizzazione di una tabella da un B-Tree ad un Hash Index)

L'ottimizzatore genera automaticamente un piano d'esecuzione che decide come andare a reperire i dati. Il piano d'esecuzione specifica quale indice utilizzare (B-Tree, Hash Index, Bitmap etc..), quale attributo osservare per prima scegliendo il suo indice opportuno e quale ordine di Join eseguire.

Prima dei modelli relazionali il piano d'esecuzione veniva scritto da chi generava il programma ed era quindi compito del programma scegliere come recuperare i dati.

Grazie ai modelli relazionali riusciamo quindi a dividere le competenze fra programma e gestore dei dati.

Il piano d'esecuzione generato dal sistema relazionale è generalmente più efficiente di quello generato da un programmatore. Ciò accade perché:

- È in grado di esplorare un numero di possibilità maggiore, rispetto all'umano medio, per ricercare il metodo migliore ricercare i dati. Tali soluzioni non trovano sempre l'ottimo globale ma sicuramente un buon ottimo locale
- L'ottimizzatore può osservare com'è fatta dinamicamente la base dati. Può quindi studiare la distribuzione statistica delle tuple e scegliere il piano migliore da utilizzare osservando queste statistiche
- La scrittura del codice che implementa il piano dati è generalmente effettuata da sviluppatori che da anni ricercano in tale settore. Le soluzioni proposte sono quindi le migliori, difficilmente replicabili da uno sviluppatore comune se dovesse reimplementare, partendo da zero, tutta la parte di gestione dei dati per ogni applicazione.
- Se i miei dati cambiano la distribuzione nel tempo posso richiedere all'ottimizzatore di ricompilare il piano d'esecuzione si riadatta alle variazioni che ci sono state nella distribuzione dei dati

Parser – Analisi degli errori

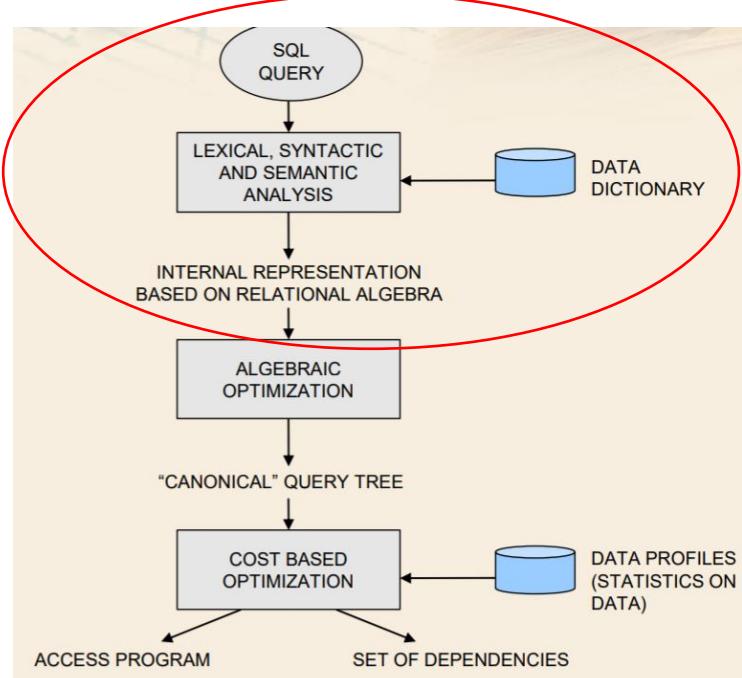
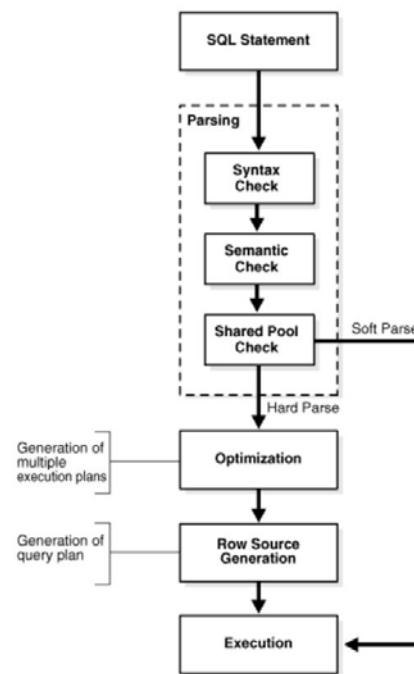
Le operazioni eseguite dal blocco parser dell'ottimizzatore sono:

- Analisi Lessicale: controlla che le parole chiavi del linguaggio siano state scritte in modo corretto. Ad esempio, SELECT che per errore viene scritta SELCT
- Analisi Sintattica: controlla che non vi siano errori sintattici all'interno della nostra interrogazione. Ad esempio, non posso scrivere SELECT senza una conseguente FROM
- Analisi Semantica: gli errori semanticci che possono essere riconosciuti sono quelli quando mi sbaglio a referenziare oggetti della base dati. Ad esempio, faccio una SELECT di un campo che non esiste nella tabella che ho messo nella FROM. Per fare questi controlli, l'ottimizzatore deve poter accedere al dizionario semantico dei dati del sistema

Il risultato d'uscita del parser è una rappresentazione interna espressa in algebra relazionale estesa.

Usiamo l'algebra relazionale poiché è procedurale ovvero l'ordine di lettura delle tabelle è definito. Possiamo quindi specificare l'ordine d'esecuzione dei vari operatori.

Inoltre, l'algebra relazionale ci permette di spostare alcuni operatori grazie a delle opportune proprietà. In questo modo possiamo ricercare la forma migliore dal punto di vista dell'esecuzione.



Successivamente a questo passo avviene l'ottimizzazione vera e propria. I blocchi che effettuano l'ottimizzazione possiamo separarli in due:

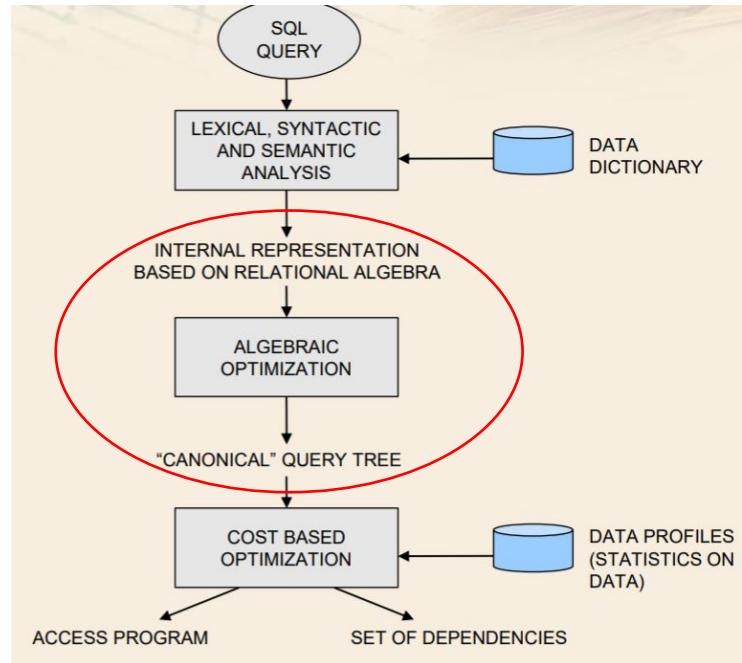
- Ottimizzazione Algebrica
- Ottimizzazione basata sui Costi

Ottimizzazione Algebrica

In questa fase utilizziamo le proprietà dell'algebra relazionale per cambiare l'ordine degli operatori relazionali.

Vi sono delle operazioni che sono considerate sempre “benefiche”, ad esempio conviene anticipare le selezioni.

Tale operazione è eseguita al fine di produrre un albero canonico anche se stesse istruzioni vengono eseguite in maniera diversa. Normalmente questo passo viene effettuato senza osservare la distribuzione dei dati.



Ottimizzazione basata sui costi

Questo modulo sceglie il piano d'esecuzione, ovvero quali attributi andare a guardare per prima e quali indici usare per determinati attributi.

Quest'operazione avviene secondo alcune fasi:

- Va a vedere nella base dati come sono strutturati i dati e che distribuzione hanno
- Prende delle decisioni basate sulla propria conoscenza delle modalità di realizzazione degli operatori e sulla conoscenza acquisita sulla distribuzione dei dati all'interno della base dati.

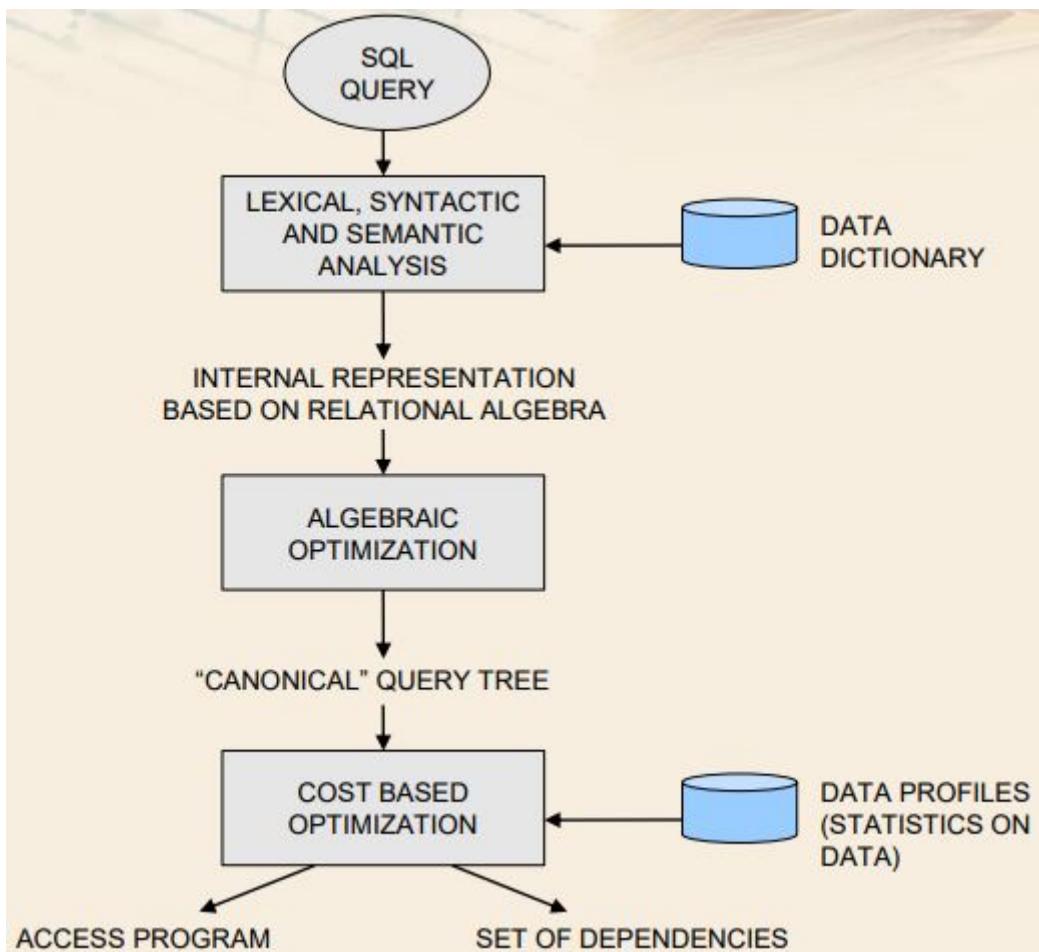
In questo modo sceglierò il metodo d'accesso per ogni tabella e poi dovrò scegliere un metodo di come eseguire gli operatori selezionati in base a quali operatori devo svolgere. Ad esempio, devo decidere come effettuare un'operazione di Join.

Tutte queste operazioni sono fatte avendo dei pesi di costo per ogni operazione da effettuare sulla base dati. In questo modo il sistema valuta il costo per ogni soluzione esplorata, scegliendo così la soluzione ottima fra quelle esplorate.

Un'altra informazione necessaria alla produzione del piano d'esecuzione, sono l'insieme delle dipendenze ovvero le condizioni di validità del piano d'esecuzione che è stato generato. Ad esempio, il sistema deve accorgersi di quali indici posso usare per reperire un certo dato se non sono tutti disponibili.

L'output dell'ottimizzazione basata sui costi è un programma in formato eseguibile per accedere ai dati del DBMS in modo appropriato. Oltre al programma si generano l'insieme del set di dipendenze. In tale modo il sistema può accorgersi di eventuali modifiche alle strutture di accesso ai dati (modifiche indici) che violano le dipendenze.

In caso di violazione di dipendenze, il mio piano d'esecuzione non sarà più valido e quindi dovrò generare un nuovo piano d'esecuzione.



Modalità d'esecuzione delle interrogazioni SQL

Vi sono differenti modalità d'esecuzione delle interrogazioni SQL:

- Compile and go: Genero il piano d'esecuzione, lancio l'interrogazione SQL e poi elimino il piano d'esecuzione. Il set di dipendenze non viene generato. Questo caso avviene quando non devo ripetere continuamente le stesse istruzioni SQL.
- Compile and store: Genero il piano d'esecuzione e lo memorizzo nella base dati insieme al suo set di dipendenze. Tale meccanismo viene utilizzato quando dovrei ripetere la stessa interrogazione SQL.

In questo modo genero il piano d'esecuzione una volta sola, evitando così di dover ricompilare continuamente il tutto.

Ad esempio, il piano d'esecuzione per la prenotazione esami all'interno del portale della didattica non viene ricompilato continuamente.

In questo caso devo controllare le dipendenze in caso di cambiamenti nella struttura fisica.

Ottimizzazione Algebrica – analisi del processo

Dal punto di vista concettuale vogliamo produrre un albero canonico d'interrogazione. Le trasformazioni sono di equivalenza ovvero, le espressioni relazionali producono lo stesso risultato indipendentemente dallo stato della base dati.

Le trasformazioni interessanti sono:

- Quelle che riducono la dimensione dello stadio intermedio che dobbiamo memorizzare. Conviene dunque anticipare le selezioni prima di effettuare i Join.
- Prepariamo la struttura dell'albero al fine di migliorare le performance per le interrogazioni utili.

Proprietà di trasformazione

-Atomizzazione della selezione – preparazione (riduzione mole di dati)

Due selezioni in AND posso scomporle in N operazioni più semplici attuate in cascata.

$$\sigma_{F1 \wedge F2}(E) \equiv \sigma_{F2}(\sigma_{F1}(E)) \equiv \sigma_{F1}(\sigma_{F2}(E))$$

-Proiezione in cascata - preparazione (riduzione mole di dati)

Faccio la proiezione su un insieme più grande di X e poi faccio la proiezione di X su quell'insieme. Mi serve per spostare la proiezione in giro nell'albero.

-Anticipo della Selezione – efficienza

Per ridurre il volume dei dati posso fare prima l'operazione di selezione e poi quella di join

$$\pi_X(E) \equiv \pi_X(\pi_{X,Y}(E))$$

-Anticipare le proiezioni rispetto al join

Faccio una selezione prima di effettuare il join riducendo il numero di colonne delle mie tabelle. Tengo solo le colonne interessanti e la colonna degli attributi su cui farò il Join. In questo caso J.

$$\pi_L(E_1 \bowtie_p E_2) \equiv \pi_L((\pi_{L1,J}(E_1)) \bowtie_p (\pi_{L2,J}(E_2)))$$

- L1 = L - Schema(E₂)
- L2 = L - Schema(E₁)
- J = set of attributes needed to evaluate join predicate p

-Derivazione di Join da prodotto Cartesiano

Se ho una selezione e poi un prodotto cartesiano posso trasformarlo in un'operazione di Join. Il predicato F deve riguardare attributi che riguardano il predicato di join.

$$\sigma_F(E_1 \times E_2) \equiv E_1 \bowtie_F E_2$$

-Distribuzione della selezione rispetto all'unione

Facciamo la selezione e successivamente l'unione.

$$\sigma_F(E_1 \cup E_2) \equiv (\sigma_F(E_1)) \cup (\sigma_F(E_2))$$

-Distribuzione della selezione rispetto alla differenza

Come il caso precedente ma con la differenza invece che l'unione.

$$\begin{aligned} \sigma_F(E_1 - E_2) &\equiv (\sigma_F(E_1)) - (\sigma_F(E_2)) \\ &\equiv (\sigma_F(E_1)) - E_2 \end{aligned}$$

-Distribuzione della proiezione rispetto all'unione

Posso prima applicare la proiezione e poi l'unione

$$\pi_X(E_1 \cup E_2) \equiv (\pi_X(E_1)) \cup (\pi_X(E_2))$$

NON posso distribuire la proiezione rispetto alla differenza. Vale solo se X contiene la chiave primaria dell'espressione E1 ed E2. La rimozione dei duplicati avviene in modo diverso

$$\cancel{\pi_X(E_1 - E_2) \equiv (\pi_X(E_1)) - (\pi_X(E_2))}$$

-Trasformare un AND di predici in un'operazione d'intersezione e un OR di predici in un'operazione di unione

$$\begin{aligned} \sigma_{F1 \vee F2}(E) &\equiv (\sigma_{F1}(E)) \cup (\sigma_{F2}(E)) \\ \sigma_{F1 \wedge F2}(E) &\equiv (\sigma_{F1}(E)) \cap (\sigma_{F2}(E)) \end{aligned}$$

-Distribuzione del Join rispetto all'unione

Faccio prima il join e successivamente l'unione

$$E \bowtie (E_1 \cup E_2) \equiv (E \bowtie E_1) \cup (E \bowtie E_2)$$

Tutti gli operatori algebrici sono commutativi e distributivi, eccetto la differenza

Ottimizzazione basata sui Costi – analisi del processo

Il processo di ottimizzazione basata sui costi applica ulteriori modifiche rispetto a quelle dell'ottimizzazione algebrica e poi decide qual è l'esecuzione migliore.

Sceglie ad esempio quale ordine di join è migliore da applicare.

I ragionamenti dell'ottimizzatore basato sui costi iniziano dal dizionario dei dati.

Il dizionario dei dati contiene informazioni descrittive di come sono fatti i dati all'interno della base dati. Sono quindi descrizioni statistiche sui dati.

Il risultato dell'ottimizzatore basato sui costi è il piano d'esecuzione all'interno del DBMS prescelto. La scelta di ottimizzazione è quindi basata sul profilo statistico dei dati che stiamo osservando sul DBMS.

Se il piano d'esecuzione viene salvato all'interno della base dati, memorizzerà anche il set di dipendenze in modo da sapere se devo ricalcolare il piano d'esecuzione in caso di una modifica alla struttura della base dati.

L'ottimizzazione basata sui costi è quindi fondata sulla conoscenza della distribuzione dei dati all'interno della base dati che è rappresentata nei profili dei dati (dizionari dei dati).

Questi non sono altro che rappresentazioni statistiche della distribuzione dei dati (sia per le tabelle che contengono i dati che per i risultati intermedi di memorizzazione dei dati, come ad esempio i risultati delle operazioni di join).

L'ottimizzatore basato sui costi conosce delle formule approssimate che permettono di stimare il tempo d'esecuzione di una certa operazione. I modelli approssimati sono proprietari di ogni DBMS.

Profili dei dati

Sono descrizioni quantitative dei dati, riguardano quindi le informazioni sulle tabelle e sulle colonne delle tabelle.

Effettueremo anche una stima sulle tabelle temporanee memorizzate durante le operazioni intermedie.

Le informazioni quantitative che caratterizzano i dati sono relative alle tabelle e alle colonne:

- Cardinalità delle tuple all'interno della tabella
- Dimensione media in byte della tupla all'interno della tabella T
- Dimensione media per ogni attributo all'interno della tabella T
- Numero di valori distinti per ogni attributo. Questo parametro è molto importante perché così posso distinguere, nel caso vi siano molti valori distinti per quell'attributo, una chiave candidata da un attributo normale. Questo valore ci indica quanti gruppi diversi avremo in caso di una group by.
- Intervallo di variazione minimo e massimo per un dato attributo. Questo valore è relativo ai dati studiati in quel momento sulla base dati, potrebbe quindi cambiare in futuro.

Tutte queste informazioni fotografano l'informazione statistica della base dati in un preciso istante temporale.

Siccome il calcolo dei profili dei dati è un'operazione costosa, questi non sono sempre aggiornati all'ultima transizione. Non riflettono quindi sempre la situazione della base dati.

La modifica dei profili viene effettuata ogni volta che questa viene richiesta (UPDATE STATISTICS). I profili dei dati vengono ricalcolati quando noto un calo di performance oppure quando ho del tempo di IDLE, in modo da impegnare il sistema quando ha poco carico da svolgere.

Operatori di accesso alla base dati

L'accesso ai dati può essere fatto in diverse modalità:

- Sequential Scan: Esegue l'accesso sequenziale a tutte le tuple della tabella. Il sistema leggerà quindi tutta la tabella in memoria. Il DBMS può applicare delle operazioni durante il sequential scan come proiezione, selezione, sorting e operazioni di modifica delle tabelle come Insert, Update e delete.
Tale operazione è generalmente lenta e la useremo solo in casi particolari
- Sort: Il memory sort non sempre è attuabile interamente in memoria. Se dati troppo grandi devo fare swopping con il disco.
- Accesso tramite indice: In base al tipo di predicato che vogliamo sfrutteremo alcuni indici (B⁺-Tree, Hash, Bitmap) invece che altri per il reperimento dei dati:
 - Se abbiamo un'uguaglianza di tipo attributo=valore ($A_i=v$) allora possiamo usare tutti e tre gli indici.
 - Se abbiamo un predicato d'intervallo (<, >, compreso etc) dobbiamo utilizzare un B⁺-Tree perché è a differenza delle Bitmap e delle strutture Hash ha l'ordinamento fisico del dato.
 - Quando il predicato ha una selettività limitata bisogna valutare se vale la pena usare un indice oppure no.
Quando devo leggere una frazione grande di una tabella certe volte conviene leggerla tutta utilizzando un sequential scan.
Altre volte si possono usare anche indici bitmap.

Operatori di accesso ai dati di più predici

Nel caso di più predici in AND, il sistema valuta l'attributo più selettivo e prende per primo quello lì, sperabilmente mediante un indice carica i dati in memoria usando quel predicato e successivamente tutti gli altri predici li valuta sui dati in memoria non usando più gli indici.

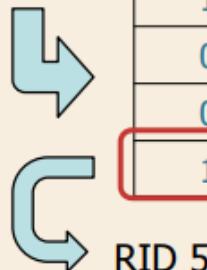
Nel caso in cui ho un predicato che ha un indice B-Tree e uno Hash, l'ottimizzatore può prelevare le informazioni riguardanti solo i ROWIDs che rispettano quei predici, fare l'intersezione di questi ROWIDs e successivamente prelevare i dati per caricarli in memoria principale.

Se ho degli indici bitmap posso effettuare quest'operazione in modo più efficiente perché i vettori delle bitmap sono vettori posizionali.

La risoluzione dei predici in AND tramite l'uso di indici bitmap è così efficiente che certe volte i sistemi costruiscono al volo l'indice Bitmap anche se l'indice non è disponibile e successivamente risolvono il predicato in AND tramite la bitmap appena creata.

Un predicato in AND è l'intersezione delle colonne delle bitmap che rispettano i singoli predici.

| RID | Gender | Exempt | Region | Donne Piemontesi Esenti da tasse | | |
|-----|--------|--------|----------|----------------------------------|--------|----------|
| | | | | Gender | Exempt | Piemonte |
| 1 | M | Y | Piemonte | 0 | 1 | 1 |
| 2 | F | Y | Liguria | 1 | 1 | 0 |
| 3 | M | N | Puglia | 0 | 0 | 0 |
| 4 | M | N | Sicilia | 0 | 0 | 0 |
| 5 | F | Y | Piemonte | 1 | 1 | 1 |



The diagram illustrates a table scan process. A large blue arrow points downwards from the original table to a smaller, simplified version below it. This smaller version contains only three columns: Gender, Exempt, and Piemonte. The row corresponding to RID 5 is highlighted with a red box. The value '1' in the Piemonte column of this row is also highlighted with a red circle.

Nel caso in cui il predicato è una disgiunzione (OR) non riesco ad ottimizzarlo come nel caso AND.

Le situazioni sono:

- Table scan se ho solo una parte degli attributi indicizzati
- Indice solo quando tutti gli attributi sono indicizzati

Join

L'operazione di Join contribuisce a garantire l'indipendenza dei dati.

L'operazione di join avviene fisicamente grazie al join dei puntatori che indicizzano i dati. In questo modo il posizionamento reale dei dati sarà indipendente dal metodo d'accesso.

La dimensione della tabella risultato dell'operazione di join è tipicamente più grande della tabella più piccola di partenza su cui ho effettuato il join.

Siccome la tabella intermedia dell'operazione di join dovrà essere memorizzata in memoria, il sistema dovrà essere in grado di stimare la dimensione di tale tabella in modo da poter allocare uno spazio utile.

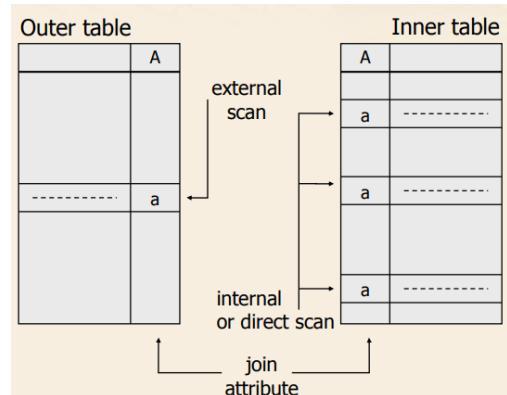
Esistono differenti algoritmi per effettuare l'operazione di Join:

- Nested loop: ancora utilizzato
- Merge scan join: quasi mai utilizzato
- Hash join: molto utilizzato
- Bitmapped join: utilizzato soprattutto nei DataWarehouse

Nested loop – Brute Force

L'operazione di Nested Loop effettua due loop nidificati per eseguire il Join.

Per ogni riga della tabella esterna facciamo una scansione della tabella interna cercando tutte le tuple della tabella interna che si abbinano alla tupla della tabella esterna secondo l'attributo di Join.



Il Nested Loop si usa ancora perché nel caso in cui la tabella interna abbia una dimensione piccola, potrò salvarla in memoria principale rendendo così l'operazione veloce. Il rallentamento delle operazioni è infatti causato principalmente dall'accesso alla memoria secondaria (disco).

Inoltre, se la tabella interna è indicizzata sull'attributo di Join e se devo effettuare un join che non prevede l'accesso agli attributi delle tuple (caso "IN") dove devo controllare solo se la tupla esiste o meno posso salvare in memoria principale solo gli indici e poi effettuare un controllo sull'esistenza tramite questi.

L'esecuzione dell'algoritmo Nested Loop **non è simmetrica**. Il costo d'esecuzione potrebbe variare a seconda di quale tabella come interna o esterna. Posso infatti avere indici solo sul predicato di una o dell'altra tabella. La tabella esterna viene letta una sola volta mentre la tabella interna viene letta potenzialmente tante volte, fare quindi A join B può avere un costo diverso da B join A.

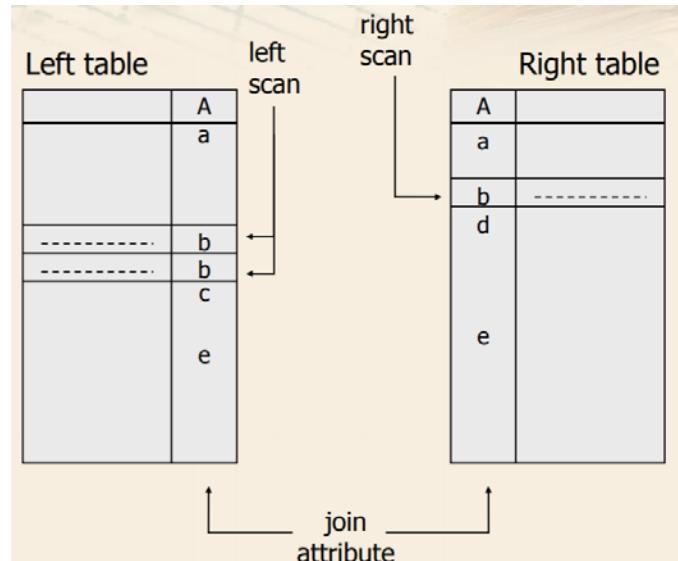
Gli ottimizzatori valutano entrambi i modi di fare il join al fine di capire qual è l'esecuzione più efficiente per realizzare l'operazione di Join.

Merge Scan

Merge Scan è un algoritmo che implementa l'operazione di join basato sull'ordinamento fisico delle tuple sull'attributo di join.

Entrambe le tabelle vengono scandite una volta sola e le tuple di entrambe le tabelle su cui vogliamo effettuare il join vengono messe insieme in un'unica tabella risultato, procedendo così in maniera armonica.

L'algoritmo **Merge Scan è simmetrico** e il costo d'esecuzione rimane invariato poiché entrambe le tabelle sono processate una volta sola.



A differenza del Nested Loop pagherò il costo dell'ordinamento fisico delle tuple.

Questa tecnica è stata molto utilizzata in passato per effettuare l'operazione di join fra tabelle grandi. Oggi è stata soppiantata dall'algoritmo di Hash Join.

Hash Join

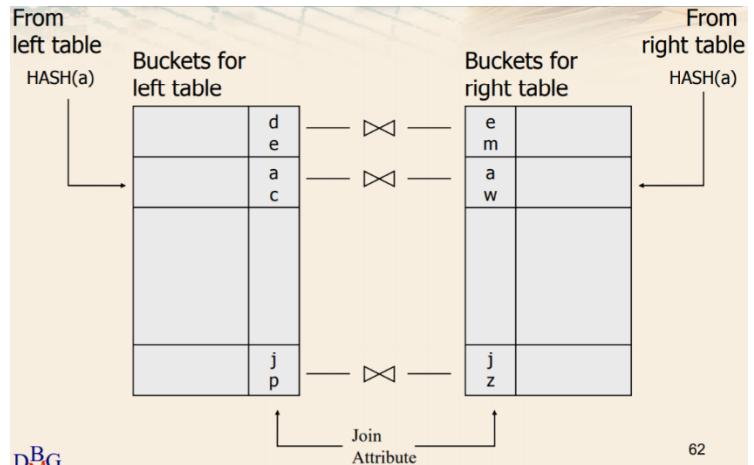
Tramite l'esecuzione di un'opportuna funzione di Hash, che utilizza come chiave l'attributo di join, collochiamo le tuple con lo stesso attributo all'interno dello stesso bucket.

Lo stesso procedimento è eseguito per entrambe le tabelle.

In fine eseguito l'operazione di Join fra i bucket interessati (stesso attributo di join) di entrambe le tabelle.

Siccome nello stesso bucket potrò avere delle collisioni dovrò selezionare solamente la porzione interessante del bucket analizzato (tramite algoritmo di sort) e poi effettuare il join delle porzioni di bucket tramite un altro algoritmo, ad esempio Merge Join.

Questa tecnica offre delle performance superiori rispetto alle altre ed è la più utilizzata all'interno dei database transazionali.



L'algoritmo di **Hash Join è simmetrico**.

Bitmapped Join Index

Gli indici di Join non vengono utilizzati poiché le loro dimensioni sono confrontabili con la tabella dei fatti. L'unico indice di Join che oggi utilizziamo è proprio la Bitmapped Join Index.

L'algoritmo di Bitmapped Join Index serve per effettuare l'operazione di join fra due tabelle (A, B). La tabella bitmap rappresenta la "precomputazione" di un join ed è costruita come segue:

- Ha una colonna per ogni riga della tabella A
- Ha una riga per ogni riga della tabella B
- Valore 0 indica che la riga J e la colonna I non vanno in Join quindi le tuple identificate non possono effettuare l'operazione di Join
- Valore 1 indica che la riga J e la colonna I vanno in Join quindi le tuple identificate posso effettuare l'operazione di Join

Le righe e le colonne della matrice rappresenteranno quindi una specifica tupla all'interno della rispettiva tabella.

In questo modo posso trovare una rappresentazione delle associazioni fra le tuple delle due tabelle su cui posso effettuare il Join. Senza memorizzare i puntatori ma solamente dei bit in modo più compatto.

Un indice di questo tipo è tipicamente lento in aggiornamento poiché dovrò ricalcolare tutto il vettore di bit. Per tale motivo un indice del genere non viene utilizzato se la base dati è di tipo transazionale. (caso vendite)

Utilizziamo questa soluzione quando la nostra base dati ha una struttura a stella e la tabella principale ha una dimensione importante.

Esempio di funzionamento gli indici bitmapped:

Vogliamo definirci un indice bitmapped fra esame-corso e un indice bitmapped per l'attributo anno di corso, combinandoli di un'unica interrogazione e producendo un risultato senza accedere alla tabella dei fatti.

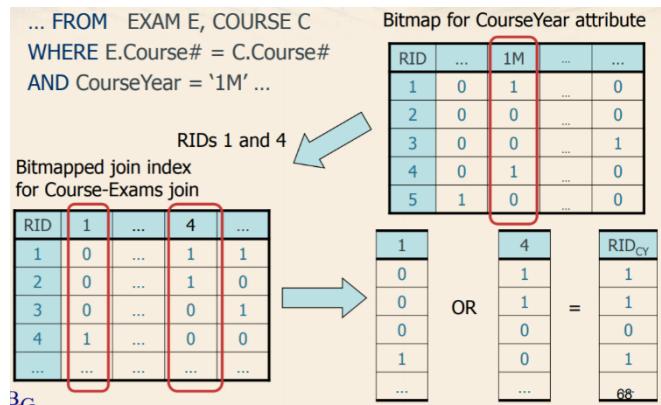
Seleziono la colonna relativa all'anno di corso interessato e la uso per capire quali colonne devo prelevare dell'indice bitmapped per il join fra le tabelle Course-Exams.

In questo modo prelevo gli studenti numero 1 e numero 4 (con CourseYear = '1M' entrambi 1° anno magistrale) vado ad osservare i loro esami nella bitmap di join fra course-exams.

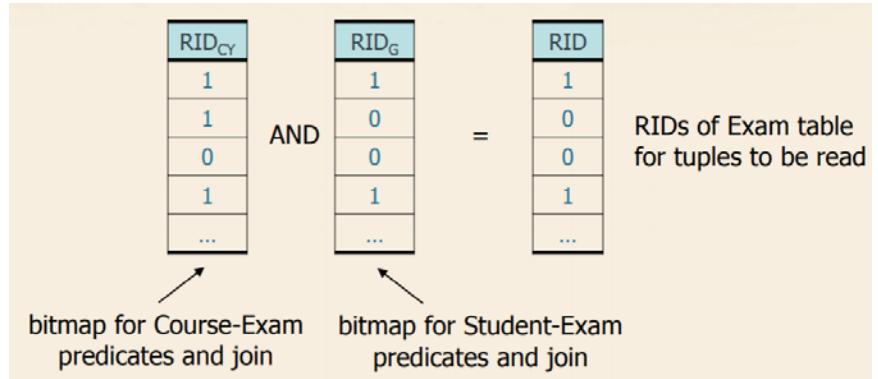
Prelevo le colonne da quest'ultima bitmap e le metto in OR ottenendo così una colonna che rappresenta gli esami degli studenti del primo anno di magistrale.

- STUDENT (Reg#, SName, Gender)
- COURSE (Course#, CName, CourseYear)
- EXAM (Reg#, Course#, Date, Grade)

```
SELECT AVG (Grade)
FROM STUDENT S, EXAM E, COURSE C
WHERE E.Reg# = S.Reg#
AND E.Course# = C.Course#
AND CourseYear = '1M'
AND Gender = 'M';
```



Possiamo effettuare la stessa operazione anche sul predicato Gender = 'M' e ottenere in questo caso la colonna risultante che rappresenta gli studenti Maschi che hanno sostenuto esami.



Infine dovrò estrapolare una colonna risultato fra le colonne di partenza. Per fare ciò dovrò mettere in AND le colonne poiché la condizione deve rispettare sia il predicato sul Gender che il predicato su CourseYear. In questo modo garantiamo che le condizioni siano vere entrambe.

Group By

L'operazione di Group By all'interno dei sistemi relazionali può essere svolta in modi differenti:

- Eseguo un'operazione di sort sulla chiave dove voglio fare il group by, in questo modo avrò tutte le tuple con la stessa chiave contigue.
- Applico un chiave di Hash riconducendo tutte le tuple di cui voglio fare il group by nello stesso bucket. Poi applico il sort in modo locale all'interno del bucket per separare eventuali collisioni. Su numeri grandi questo metodo è più vantaggioso.
- Posso usare le [viste materializzate](#) per effettuare l'operazione id group by.

Valutazione dei costi d'esecuzione

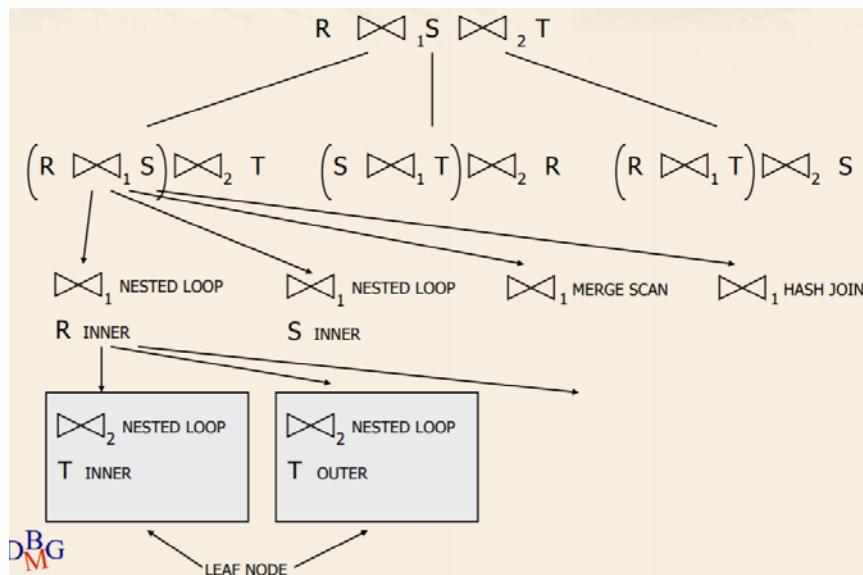
La valutazione dei costi è basata sul calcolo di alcune formule approssimate che permettono al blocco d'ottimizzazione basato sui costi di calcolare la qualità di un modello d'esecuzione rispetto ad un altro.

I parametri osservati sono:

- Modalità di lettura dei dati da disco (fullscan, indice)
- Ordine d'esecuzione degli operatori (anticipare le select etc..)
- Modalità d'esecuzione degli operatori (nel join ad esempio posso scegliere fra nested loop, merge join, hash join, bitmapped index join)

Questo tipo di ottimizzatori costruisce un **albero di alternative** costruito come segue:

- Ogni nodo corrisponde ad una decisione su una variabile del problema
- Ogni foglia rappresenta un piano d'esecuzione con il suo costo



Ogni nodo foglia contiene l'ordine d'esecuzione delle istruzioni, in modo in cui ciascun operatore deve essere eseguito e il modo in cui i dati devono essere letti.

I vari piani d'esecuzione, rappresentati dai nodi foglia, vengono successivamente valutati in modo così da selezionare quello a costo minimo.

I costi vengono valutati osservando il numero di operazioni di Input/Output necessarie e il numero di cicli di CPU. In generale il costo dell'I/O è molto più alto rispetto ai cicli di CPU.

Inoltre, se ho un sistema distribuito, devo considerare anche il costo speso nel trasferimento dell'informazione all'interno della rete.

La ricerca del piano d'esecuzione migliore studiando i costi è svolta principalmente quando vogliamo effettuare un'esecuzione del tipo compile and store poiché si presume che quel tipo di piano d'esecuzione sarà lanciato più di una volta quindi sarà importante trovare una soluzione più vicina possibile all'ottimo globale.

Quando vogliamo eseguire un'istruzione in modalità compile and go non sarà vantaggioso spendere molto tempo per la ricerca ottimale del piano d'esecuzione poiché questo verrà

eliminato una volta finita l'interrogazione richiesta. In questo caso utilizzeremo quindi delle soglie d'arresto basate sullo studio del tempo di ricerca del piano d'esecuzione migliore confrontato con il tempo di ricerca d'esecuzione dell'interrogazione incriminata.

Gestore Accesso Concorrente

Tale componente ci permette di eseguire accessi concorrenti alle tabelle della nostra base dati garantendo il corretto funzionamento e l'integrità dei dati modificati durante le transazioni.

Possiamo avere due tipi differenti d'accesso alla base dati:

- Read(x): accesso in lettura dell'oggetto x. $r(x)$
- Write(x): accesso in scrittura dell'oggetto. $w(x)$

Scheduler

Dentro il blocco del gestore d'accesso concorrente vi è un oggetto chiamato *Scheduler*.

Lo *Scheduler* decide quando una certa richiesta di lettura o scrittura di un dato può essere soddisfatta, garantendo così un ordinamento corretto delle transazioni in accesso.

In assenza di un blocco di Scheduler si rischierebbe di incorrere in errori come:

- Lost Update:

| | Transaction T ₁ | Transaction T ₂ |
|--------|------------------------------|--|
| time ↓ | bot $r_1(x)$ $x = x+1$ | $x=2$ $x=3$ |
| | $w_1(x)$ commit | $x=3$ |
| | | bot $r_2(x)$ $x=x+1$ $w_2(x)$ commit |
| | | $x=2$ $x=3$ $x=3$ |

In questo caso il valore scritto di x dovrebbe essere $x=4$ dopo l'esecuzione di entrambe le transazioni.

Nell'esempio l'esecuzione della transazione T₂ viene persa e il valore scritto in memoria è pari a 3.

- Dirty Read:

| | Transaction T ₁ | Transaction T ₂ |
|--------|--|--|
| time ↓ | bot $r_1(x)$ $x = x+1$ $w_1(x)$ | $x=2$ $x=3$ $x=3$ |
| | abort | |
| | | bot $r_2(x)$ $x=x+1$ $w_2(x)$ commit |
| | | $x=3$ $x=4$ $x=4$ |

Il valore letto dalla transazione T₂ è un dato temporaneo della transazione T₁. Quando questa farà *abort* scriverà in memoria il valore OLD di X, ovvero $x=2$. In questo modo perderemo l'aggiornamento effettuato dalla transazione T₂ anche se questa ha effettuato il *commit*.

- Inconsistent read:

| | Transaction T ₁ | Transaction T ₂ |
|--------|----------------------------|--|
| time ↓ | bot $r_1(x)$ | $x=2$ |
| | $r_1(x)$ commit | $x=3$ |
| | | bot $r_2(x)$ $x=x+1$ $w_2(x)$ commit |
| | | $x=2$ $x=3$ $x=3$ |

Il valore di X letto durante l'esecuzione della transazione T₁ è inconsistente nei due casi presenti poiché viene modificato da un'altra transazione durante la sua esecuzione. Situazione simile avviene quando comperiamo un oggetto ma al momento del pagamento è già esaurito nonostante fosse nel carrello. Il dato è bloccato solamente alla fine.

- Ghost Update (tipo a):

| Transaction T ₁ | | Transaction T ₂ | |
|----------------------------|-------|----------------------------|------------|
| bot | | bot | |
| r ₁ (x) | x=400 | r ₂ (y) | y=300 |
| r ₁ (y) | y=300 | y = y -100 | y=200 |
| | | r ₂ (z) | z=300 |
| | | z = z + 100 | z=400 |
| | | w ₂ (y) | y=200 |
| | | w ₂ (z) | z=400 |
| | | commit | |
| time ↓ | | r ₁ (z) | z=400 |
| | | total = x + y + z | total=1100 |
| | | commit | |

Un aggiornamento prodotto da un'altra transazione si è “infilato” durante l'esecuzione della transazione T₁ generando così un'inconsistenza dei valori letti dalla prima transazione. In T₁ ho il vecchio valore di y ma il nuovo valore di z.

Il nuovo valore di z è prodotto da T₂ che viene eseguita durante T₁

- Ghost Update (tipo b):

| Transaction T ₁ | | Transaction T ₂ | |
|---|--|---------------------------------------|--|
| bot | | bot | |
| read the salary of all employees in department x and compute AVG salary | | insert a new employee in department x | |
| | | commit | |
| read the salary of all employees in department x and compute AVG salary | | | |
| commit | | | |
| time ↓ | | | |

In questo caso il calcolo dell'aggregato è sbagliato poiché vi è una modifica della tabella da cui leggo i dati per il calcolo.

La lettura è inconsistente a causa di un'insert.

A causa di questi problemi abbiamo bisogno di uno scheduler, ovvero un modulo per la gestione dell'accesso concorrente che garantisca il corretto funzionamento per l'accesso alla base dati.

- La transazione è una sequenza di operazioni d'accesso in lettura e in scrittura caratterizzate dallo stesso TID (Transaction Identifier).

Nelle nostre rappresentazioni le operazioni della stessa transazione saranno accomunate dallo stesso TID che verrà rappresentato come pedice.

Ad esempio: $r_1(x) w_1(x) w_1(y)$

sono rispettivamente:

lettura di x della transazione 1, scrittura di x della transazione 1, scrittura di y della transazione 1

- Lo schedule è una sequenza di operazioni di read e write che vengono fatte da transazioni che accedono in modo concorrente.

Ad esempio: $r_1(x) w_2(x) w_1(y)$

sono rispettivamente:

lettura di x della transazione 1, scrittura di x della transazione 2, scrittura di y della transazione 1

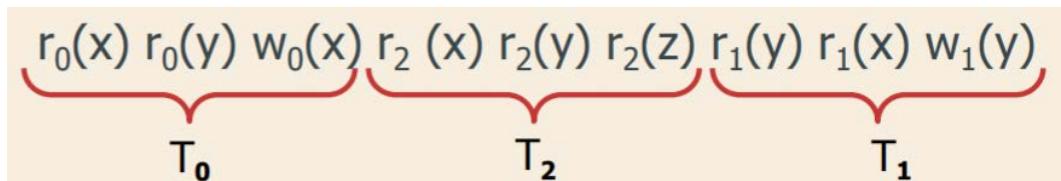
Durante l'arrivo delle transazioni, lo scheduler non conosce l'esito delle transazioni (commit/abort). Il problema è quindi abbastanza complicato, non basta solamente capire quali schedule sono permessi e quali no ma bisognerebbe considerare eventuali *rollback*.

Al momento faremo un'ipotesi di proiezione sul commit. Considereremo quindi che tutte le transazioni facciano sempre il commit. Non considereremo quindi l'anomali Dirty Read.

Tipi di Scheduler – Scheduler Seriale

Lo scheduler seriale processa in ordine tutte le transazioni della stessa transazione.

La gestione delle transazioni avviene in maniera seriale e non interlacciata.



In questo caso non si verifica nessun problema (Lost Update, Dirty Read, Inconsistent, Ghost Update) ma lo scheduler è poco performante.

L'obiettivo che vogliamo raggiungere è utilizzare uno scheduler non seriale ma che offre un comportamento, riguardo la gestione dei problemi, uguale a quello di uno scheduler seriale.

Uno scheduler non seriale che si comporta come uno seriale prende il nome di scheduler serializzabile. Pur eseguendo le transazioni in modo interlacciato, migliorando così le performance.

Classi di equivalenza

Durante gli anni si sono ricercate differenti classi di equivalenza che hanno come obiettivo quello di riconoscere quali sono gli schedule serializzabili, ovvero quelli che possono essere accettati.

Le classi di equivalenza sono:

- View equivalence
- Conflict equivalence
- 2 phase locking
- Timestamp equivalence

Ognuno di questi metodi riconosce un insieme più o meno grande di schedule che sono accettati. Più grande è l'insieme di schedule accettati e più è complessa l'operazione di riconoscimento quindi maggiore sarà il tempo di ricerca degli schedule serializzabili.

View equivalence - VSR

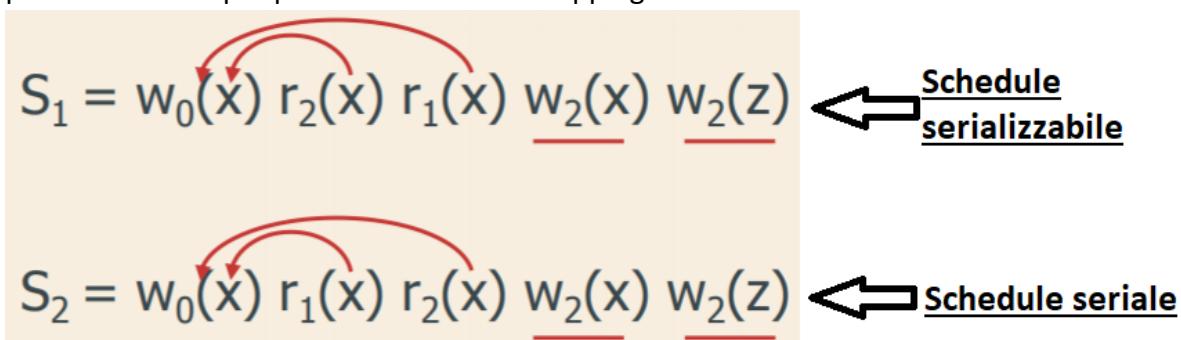
Per il funzionamento di questo scheduler dobbiamo definire due insiemi di schedule:

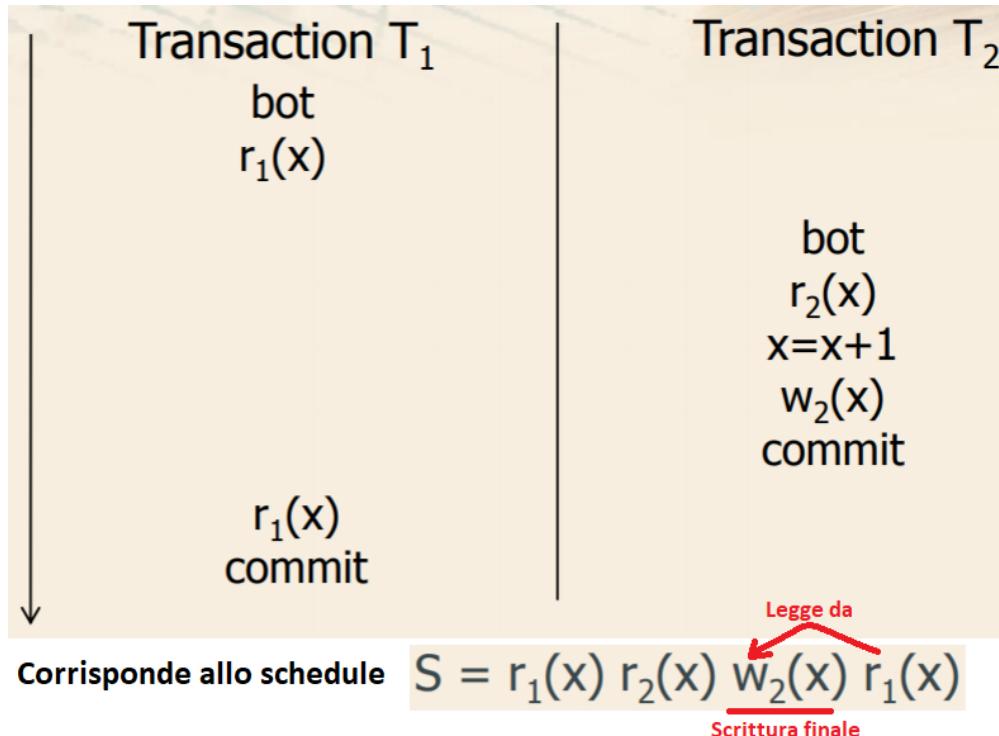
- Legge da: controlla per ogni operazione di lettura della transazione da quale operazione di scrittura legge. Se ho quindi una scrittura che precede la lettura dell'oggetto 'x', quella lettura leggerà dalla scrittura che è stata eseguita da un'altra transazione.
È l'insieme di tutte le letture che vengono effettuate da scritture che precedono la lettura in cui la transazione è diversa.
Sono tutte le letture dello stesso oggetto che è stato scritto da transazioni diverse.
 $w_i(x)$ precede $r_k(x)$ con $i \neq k$.
- Scritture finali: Sono le ultime scritture di un oggetto nello schedule che ho a disposizione.

Due schedule si definiscono **View equivalenti** se hanno gli stessi insiemi di "Legge da" e di "Scritture finali". Nel caso in cui uno schedule sia View equivalente con uno schedule seriale potremo affermare che tale schedule sarà serializzabile.

Nel caso in cui uno schedule non sarà View equivalente con uno schedule seriale verrà rifiutato dallo scheduler.

L'operazione di View Equivalence per il riconoscimento degli schedule serializzabili ha una complessità NP, dobbiamo quindi utilizzare dei metodi approssimativi per la ricerca degli schedule serializzabili. View Equivalence spesso scarta degli schedule, anche se sono serializzabili, poiché il tempo d'esecuzione per processarli sarebbe troppo grande.





I due possibili
schedule seriali sono:

$S1 = r1(x) \ r1(x) \ r2(x) \ \underline{w2(x)}$

$S2 = r2(x) \ \underline{w2(x)} \ r1(x) \ r1(x)$

$S1$ non ha nessun "legge da".

$S2$ ha stessa "scrittura finale" ma differenti "legge da"

S non è quindi serializzabile

L'algoritmo di view equivalence è funzionale ma dal punto di vista computazionale ha un costo troppo elevato. Quando dobbiamo trovare lo schedule giusto fra tutti quelli possibili di tutte le transazioni in arrivo, il problema diventa ingestibile.

Le soluzioni proposte al problema sono:

- Sono quindi state proposte delle tecniche per il riconoscimento degli schedule serializzabili che sono più conservative ed accettabili a livello computazionale. Devo pagare però il costo di scartare degli schedule che sarebbero serializzabili, esplorando quindi uno spazio di ricerca più piccolo.
Gli schedule che considererò serializzabili saranno un sottoinsieme fra tutti gli schedule possibili.
- Cerchiamo l'equivalenza degli schedule seriali basata sul concetto di conflitto. [Conflict Equivalent]

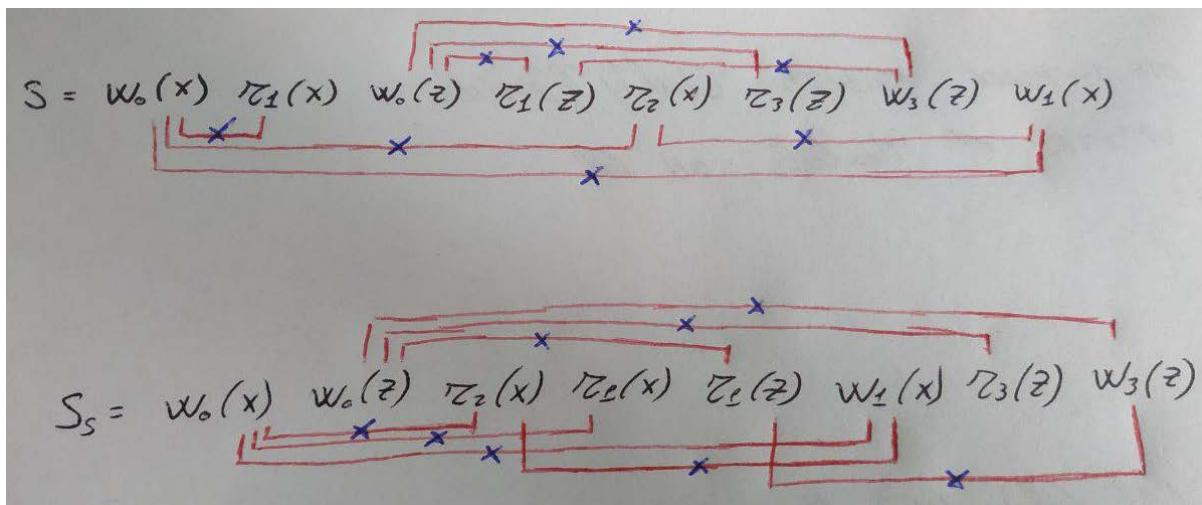
Conflict Equivalent - CSR

La ricerca degli schedule serializzabili avviene grazie al concetto di conflitto.

Un conflitto è rappresentato da due azioni A_i e A_j che operano sullo stesso oggetto dove almeno una delle due è una scrittura.

- $R_i(x) W_j(x)$
- $W_i(x) R_j(x)$
- $W_i(x) W_j(x)$

Secondo la nozione di conflict equivalenza, uno schedule è serializzabile quando è conflit equivalente ad uno schedule seriale.



Nell'esempio S è uno schedule serializzabile perché ha lo stesso insieme di conflitti d'equivalenza dello schedule S_s che è uno schedule seriale.

Il metodo per scrivere i conflitti d'equivalenza è:

- Parto dalle write
- Collego a tutte le read o write con pedice diverso per lo stesso oggetto

Lo studio della conflit equivalenza avviene tramite l'analisi di un grafo chiamato **grafo dei conflitti** che rappresenta lo schedule delle transazioni in arrivo.

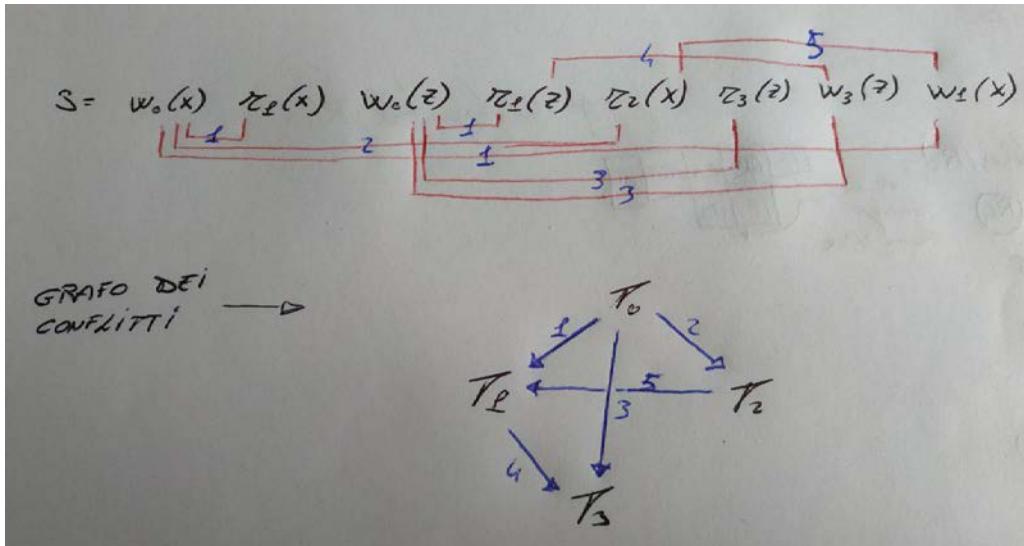
Nel caso in cui il grafo studiato presentasse delle ciclicità allora si può affermare che lo schedule studiato non sarà serializzabile.

Il grafo dei conflitti è costruito come segue:

- Si crea un nodo nel grafo per ogni transazione
- Si crea un arco orientato fra le due transazioni (T_i e T_j) se esistono due azioni (a_i e a_j) che sono in conflitto nell'ordine indicato dall'arco del grafo.

La complessità della verifica di ciclicità all'interno del grafo dei conflitti è lineare all'interno del grafo.

In questo modo possiamo confrontare la serializzabilità di uno schedule senza confrontarlo con tutti i possibili schedule seriali.



Il grafo dei conflitti dell'esempio è serializzabile.

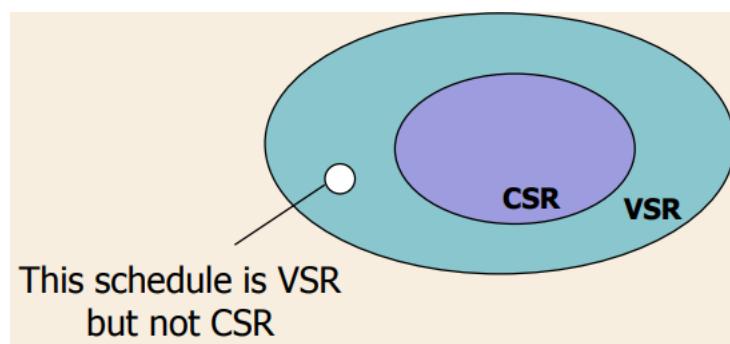
Nel caso reale questo metodo per il riconoscimento degli schedule serializzabili non è applicabile poiché la mole di transazioni al secondo è enormemente elevata e difficile da gestire con la rappresentazione del grafo dei conflitti.

Dobbiamo quindi ricercare un metodo ancora più restrittivo per la ricerca degli schedule serializzabili.

Nell'immagine possiamo osservare come uno schedule *conflict equivalent* offra una soluzione che è in grado di selezionare solo un sottoinsieme fra tutti gli schedule serializzabili disponibili.

Lo scheduler *View equivalent* è ingrado di riconoscere come serializzabili più schedule.

Nonostante ciò anche lo scheduler *conflict equivalent* non è computazionalmente ammissibile.



2 Phase Locking – 2PL

Questo sistema permette di implementare uno scheduler in grado di rintracciare gli schedule serializzabili con un tempo computazionale accettabile.

Il funzionamento è basato sul concetto di lock.

Un lock è il blocco di una risorsa che può prevenire l'accesso ad altre azioni su quella risorsa.

Le operazioni concesse sono:

- Read lock (R-Lock): Lock in lettura
- Write lock (W-Lock): Lock in scrittura
- Unlock: Rilascio del lock precedentemente acquisito

Il funzionamento del lock è basato sui seguenti passi:

- Ogni azione di lettura deve essere preceduta dall'operazione R-Lock
- Ogni azione di scrittura deve essere preceduta dall'operazione W-Lock
- Quando ho finito di scrivere o di leggere rilascerò il lock che avevo acquisito

Proprietà delle operazioni di Lock:

- R-Lock è un lock condiviso. Chiunque può leggere i dati, posso quindi condividere il lock in lettura fra più transazioni.
- W-Lock è un lock di tipo esclusivo. Il lock in scrittura su un dato è esclusivo per un'unica transazione.
- Nel caso in cui, durante l'esecuzione di una transazione, mi servisse un lock in scrittura posso chiedere un lock escalation

In questo modo il nostro scheduler deve eseguire delle operazioni di lock manager, decidendo così quali operazioni di lock e unlock deve eseguire e quali invece no.

Per fare ciò, il lock manager deve avere nozioni riguardo lo stato della base dati e informazioni riguardo i lock attivi (transazione, dati interessati).

La decisione di quali operazioni eseguire e quali no apre due possibili scenari:

- Lock concesso: la transazione che riceve quel lock ottiene il controllo della risorsa che viene quindi bloccata.
La transazione rilascerà la pagina quando finirà di farci accesso.
- Lock rifiutato: Tale situazione avviene quando il lock per l'accesso a quella pagina lo ha già un'altra transazione e non è *compatibile*.
La transazione che non riceve il lock della pagina interessata, viene messa in attesa e verrà servita quando il lock interessato verrà nuovamente reso disponibile dalla transizione che lo rilascerà.

La concessione di un lock da parte del lock manager avviene tramite l'osservazione di una tabella di lock (lock table). La tabella di lock afferma se la transazione ha diritto o meno di accedere a quella risorsa.

Il lock manager utilizza anche una per riconoscere se le tabelle di lock sono fra loro compatibili.

Esempio di tabella dei conflitti:

| <u>REQUEST</u> | <u>RESOURSE STATE</u> | | |
|----------------|-----------------------|---------------|---------------|
| <u>R-LOCK</u> | Free | R-Locked | W-Locked |
| <u>W-LOCK</u> | Ok-> R_locked | Ok-> R_locked | No-> W_locked |
| <u>UNCLOCK</u> | Ok-> W_locked | No-> R_locked | No-> W_locked |
| | Error | Ok-> Dipende* | Ok-> Free |

Gli stati dopo la freccia rappresentano lo stato finale dopo l'esecuzione della request.

*Quando voglio fare unlock di un lock che è in stato R-Locked lo stato finale dopo l'unlock dipende poiché:

- Se lo stato di R-lock è per un'unica transazione allora dopo unclock lo stato finale sarà Free
- Se lo stato di R-lock per quell'oggetto è di più transazioni allora (R-lock è un lock condiviso) dopo unlock lo stato finale potrà essere ancora R_locked

Per tener traccia di quante transazioni stanno utilizzando il lock in lettura di una determinata risorsa, il sistema utilizza un contatore.

Abbiamo quindi un contatore per ogni risorsa il cui valore ci indica quante transazioni stanno facendo uso di quella risorsa.

La risorsa è libera quando il contatore di R-Lock per quella determinata risorsa è pari a 0.

La lock table:

è una tabella contenuta in memoria principale e per ogni oggetto contiene le seguenti informazioni:

- Stato dell'oggetto: libero, R_locked, W_locked
- Numero di transazioni in attesa per quell'oggetto

Nei sistemi commerciali il locking si usa con la modalità **2-Phase Locking**.

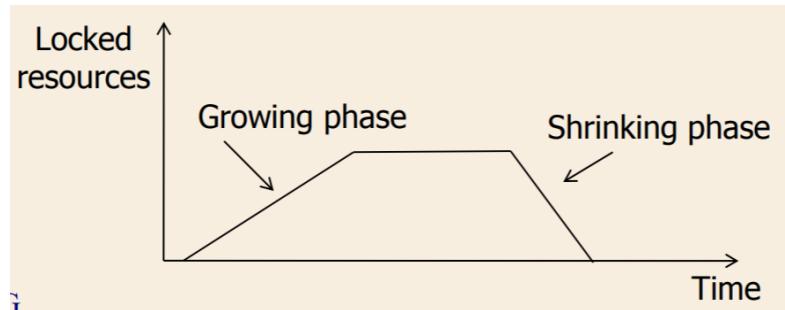
Il 2-phase locking è caratterizzato da due fasi specifiche:

- **Fase crescente** (Growing phase): Tutti i lock vengono acquisiti in maniera progressiva
- **Fase decrescente** (Shrinking phase): rilascio il lock

Nel 2-Phase Locking avrò rigidamente prima la fase crescente e poi la fase decrescente.

Dovrò prima effettuare il lock necessari e poi rilasciarli. Quando inizio il rilascio non potrò chiedere altri nuovi lock.

Quando una transazione inizia la fase di Shrinking non potrà effettuare una fase di Growing.

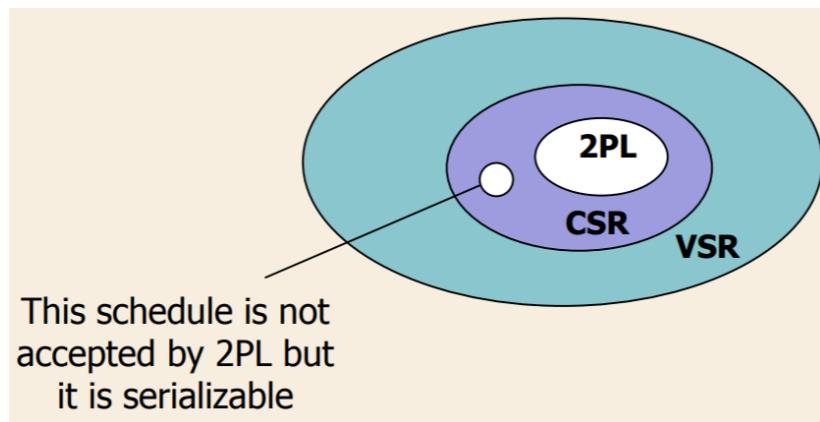


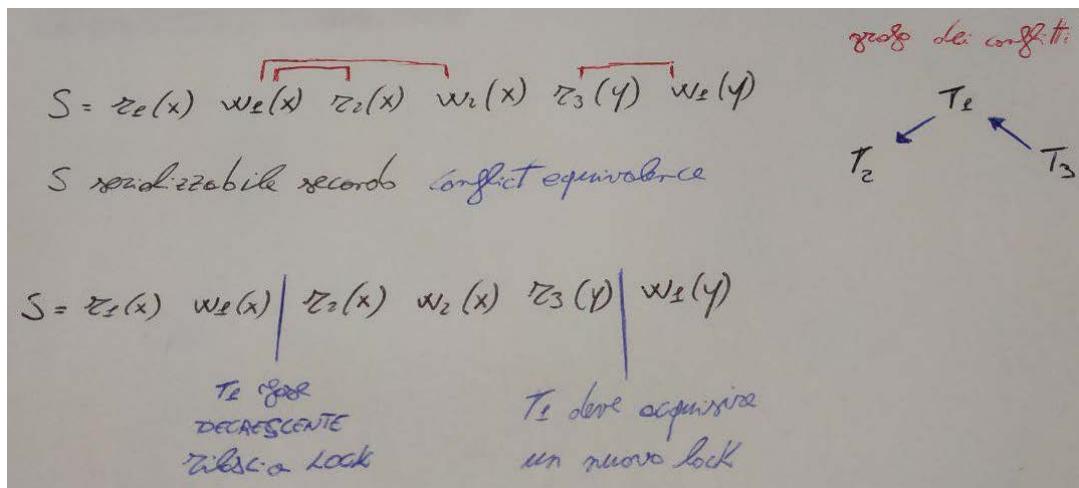
Il 2-Phase Locking garantisce la serializzabilità degli schedule in arrivo.

Il costo di tale scheduler è che l'insieme degli schedule serializzabili riconosciuti è inferiore agli scheduler visti precedentemente (View equivalente, Conflict equivalente).

A differenza dei precedenti questa soluzione è implementabile nei DBMS.

Oggi il 2-Phase Locking è uno scheduler utilizzato per il riconoscimento degli schedule serializzabili nei sistemi dove bisogna gestire l'accesso concorrente ai dati.





Nell'esempio riportato possiamo notare come lo schedule S sia serializzabile secondo lo scheduler conflict equivalence.

Anche il grafo dei conflitti, non presentando cicli, dimostra che S sia serializzabile.

Nell'ultima riga possiamo osservare invece che lo schedule S non è riconosciuto come serializzabile dallo scheduler 2-Phase Locking.

Dopo la fase decrescente, infatti, la transazione 1 dovrebbe richiedere una nuova fase crescente per effettuare $w_1(y)$, facendo così un W-Lock. Tale operazione non è permessa poiché la transazione 1 ha già effettuato una fase decrescente.

In questo caso lo scheduler 2-Phase Locking cambierà lo schedule S mettendo in attesa le transazioni in modo da riportarsi ad una condizione, per lui, serializzabile.

Negando il lock peggiorerà le prestazioni dello schedule.

Esempio Ghost Update (a) con 2-Phase Locking:

| Transactions | | Resources | | |
|-------------------------|-------------------------|-----------|------|-----------|
| T ₁ | T ₂ | x | y | z |
| bot | | free | free | free |
| r_lock ₁ (x) | | | | 1: read |
| r ₁ (x) | | | | |
| | bot | | | 2: read |
| | r_lock ₂ (y) | | | |
| | r ₂ (y) | | | 1,2: read |
| r_lock ₁ (y) | | | | |
| r ₁ (y) | | | | 2: read |
| | r_lock ₂ (z) | | | |
| | r ₂ (z) | | | 1,2: read |
| r_lock ₁ (z) | | | | |
| r ₁ (z) | | | | |
| | wait | | | |

| Transactions | | Resources | | |
|-------------------------|-------------------------|-----------|------|----------|
| T ₁ | T ₂ | x | y | z |
| commit | | | | |
| unlock ₁ (x) | | free | | |
| unlock ₁ (y) | | | | 2: write |
| | wait | | | |
| | w ₂ (y) | | | |
| | w_lock ₂ (z) | | | |
| | wait | | | 2: write |
| unlock ₁ (z) | | | | |
| | w ₂ (z) | | | |
| | commit | | | |
| | unlock ₂ (y) | | free | |
| | unlock ₂ (z) | | | free |

Grazie allo scheduler 2-Phase Locking il problema del [Ghost Update \(a\)](#) viene risolto poiché metto in attesa le operazioni di write rendendo serializzabile lo schedule.

Passi per la scrittura del seguente schema:

- Ogni azione è preceduta da un lock
- Unclock sempre dopo il commit della transazione
- Mando in WAIT le reads dopo il loro lock se la risorsa di cui voglio fare la read non è free oppure impegnata da una transazione diversa da quella che vuole effettuare la read.

Strict 2 Phase Locking

Consiste nell'effettuare, come nell'esempio precedente, le operazioni di unlock delle risorse solamente dopo il commit o il rollback della transazione.

Se forzo il rilascio dei lock solamente dopo che la transazione sia conclusa (commit/rollback), manterrò sicuramente consistenti i dati all'interno della base dati ma non potrò applicare, al livello del Buffer Manager, una politica di gestione delle pagine di tipo Steal. (leggi FIX)

Il rilascio della pagina sarà effettuato solamente dopo che le transazioni avranno rilasciato i lock, quindi le pagine non saranno più accedute da nessuna transazione. (UNFIX)

Il rilascio della pagina sarà effettuato solamente quando questa sarà stabile, ovvero quando le transazioni avranno finito di lavorarci (commit/rollback).

Grazie allo Strict 2 Phase Locking possiamo eliminare l'ipotesi di commit proiezione.

Il numero di schedule serializzabili riconosciuti sarà però ancora minore del 2 Phase Locking normale.

Grazie allo Strict 2 Phase Locking riusciamo a trattare correttamente anche l'anomalia Dirty Read.

Primitive Lock Manager

Le primitive utilizzabili dal Lock Manager sono:

- R-Lock($T, x, ErrorCode, TimeOut$)
- W-Lock($T, x, ErrorCode, TimeOut$)
- Unlock(T, x)

I parametri utilizzati dalle primitive sono:

- T -> Transazione che richiede il lock
- x -> Risorsa su cui vogliamo effettuare il lock
- ErrorCode -> codice di errore che può eventualmente essere ritornato dalla primitiva se la richiesta di lock non è andata a buon fine. Assume valori "Ok" oppure "Not Ok"
- TimeOut -> Tempo massimo che la transazione può attendere in WAIT prima di decadere. Nel caso scada il TimeOut verrà restituito un messaggio di errore.

Tecniche per la gestione del Locking

Se le azioni in arrivo dalle transazioni rispettano le condizioni nella tabella dei conflitti la gestione del locking sarà regolare.

Se invece una richiesta non può essere soddisfata, poiché la risorsa è bloccata, metterò in attesa la transazione che attenderà di poter accedere alla risorsa. (come nell'esempio del Ghost Update a)

Le transazioni in attesa vengono posizionate in una coda che può essere ordinata in base ad una certa priorità verso alcune transazioni.

La probabilità di avere un conflitto è pari a $(K \cdot M) / N$

Probability of a conflict $\approx (K \cdot M) / N$

- K is the number of active transactions
- M is the average number of objects accessed by a transaction
- N is the number of objects in the database

Quando il TimeOut di un'azione scade, la transazione viene rimossa dalla coda e viene ritornato un messaggio d'errore.

Allo scadere del TimeOut la transazione potrebbe decidere di effettuare un rollback oppure provare una nuova richiesta, evitando così di rilasciare tutti i lock precedenti.

Locking Gerarchico

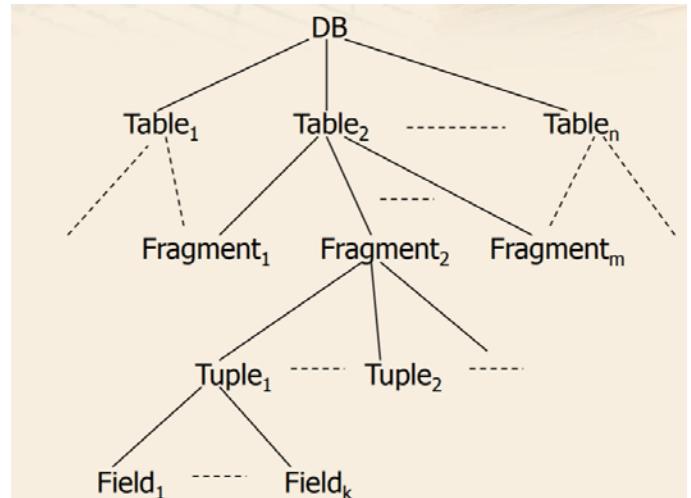
Consiste nel considerare porzioni differenti della base dati su cui effettuare l'operazione di lock.

Invece che effettuare il lock solo su alcune tuple, certe volte potrebbero interessarmi intere tabelle o tutte le tuple che soddisfano un determinato predicato.

In questo modo posso acquisire dei lock a diversi livelli di granularità all'interno della base dati.

I livelli di granularità su cui ho interesse ad effettuare il lock sono:

- Tabella
- Frammento (gruppo di tuple):
 - Livello fisico -> Una pagina
 - Livello logico -> tutte le tuple che soddisfano un dato predicato
- Singole tuple
- Singoli campi di una tupla



Grazie al locking gerarchico una transazione può effettuare il lock al livello della gerarchia più utile per il suo svolgimento.

Se la transazione dovrà effettuare aggiornamenti massicci potrebbe bloccare un'intera tabella, se invece la transazione esegue un aggiornamento puntuale potrebbe bloccare solamente un frammento o una singola tupla.

Full Table Scan: lock dell'intera tabella

Se accedo con indice posso fare solo il lock di tupla.

Primitive Locking Gerarchico

Per il corretto funzionamento del locking gerarchico dobbiamo estendere il numero di primitive del 2 Phase Locking:

- Shared Lock (SL): è il lock in lettura
- eXclusive Lock (XL): è il lock in scrittura
- Intention of Shared Lock (ISL): La transazione avverte il sistema che è intenzionata a bloccare una porzione dell'oggetto x che gerarchicamente è più in basso. (pagina, tupla o campo)
Possiamo quindi bloccare in modo condiviso un discendente del nodo che stiamo osservando
- Intention of eXclusive Lock (IXL): Analogamente a ISL, possiamo bloccare in modo esclusivo una porzione del nodo x .
- Shared lock and Intention of eXclusive Lock (SIXL): Blocco un oggetto in modo shared e avverto che successivamente farò una modifica ad un sottoinsieme di quel blocco che gerarchicamente sarà più in basso.
Ad esempio: leggo una tabella e poi vorrò modificare solo alcune tuple (bloccando le tuple con XL).

L'ordine con cui vengono richiesti e rilasciati i lock è il seguente:

- I lock vengono sempre richiesti partendo dall'oggetto più grande (la radice)
 - I lock vengono rilasciati partendo dal nodo più piccolo che è stato bloccato
 - Nella gerarchia devo avere un blocco di tipo ISL (o IXL) su una tabella se voglio avere un blocco SL o ISL in una pagina. Devo quindi prima avere un blocco sull'oggetto che gerarchicamente sta più in alto se voglio applicare un blocco ad un oggetto a granularità più piccola.
- Quindi: per richiedere un SL o un ISL per un dato nodo, la transazione deve avere un ISL (o IXL) sul nodo padre.
- Per richiedere un XL, IXL o SIXL su un dato nodo, la transazione deve già detenere un IXL o un SIXL per il nodo padre.

Quando richiedo il lock riscendo nella gerarchia e quando rilascio il lock risalgo nella gerarchia.

In questo caso la **tabella dei conflitti** ha una struttura più complicata:

| <u>REQUEST</u> | <u>RESOURCE</u> | <u>STATE</u> |
|----------------|---------------------|-------------------|
| <u>ISL</u> | <u>ISL</u> Ok | <u>IXL</u> Ok* |
| <u>IXL</u> | <u>IXL</u> Ok* | <u>SL</u> No |
| <u>SL</u> | <u>SL</u> Ok | <u>SIXL</u> No |
| <u>SIXL</u> | <u>SIXL</u> Ok** | <u>XL</u> No |
| <u>XL</u> | <u>XL</u> No | <u>IXL</u> No |

*vorremmo poter leggere/scrivere una porzione di dati differenti dello stesso oggetto

**vorremmo leggere la stessa porzione di dati ma fare eXclusive lock per scriverne un'altra.

La gerarchia dei lock è: **XL->SIXL->SL\IXL->ISL**

Selezione della granularità del Lock

Per ridurre la probabilità dei conflitti è opportuno differenziare diversi livelli di granularità dei lock. Per una transazione puntuale bisognerà effettuare il lock su oggetti puntuali garantendo così il massimo throughput delle transazioni.

Per lock massicci invece conviene effettuare il lock dell'intera tabella.

Predicate Locking

Tale sistema ci permette di superare il problema del [Ghost Update \(tipo b\)](#).

Il funzionamento è basato sul [bloccare tutte le tuple che soddisfano un certo predicato](#) sulla base dati. L'eventuale inserimento di una tupla che soddisfa il predicato del lock non verrà eseguito finché non si effettuerà il commit della transazione che ha applicato il lock.

Questo lock, a differenza dei precedenti, è di tipo logico e non fisico.

Per implementare tale meccanismo si effettua il lock di tutte le foglie dell'indice che soddisfano il predicato. In questo modo il sistema non potrà aggiornare l'indice in caso di insert.

Locking e SQL – Livello d'isolamento

In SQL le transazioni possono essere dichiarate in due modi differenti:

- [Read-Write](#)
- [Read Only](#): in questo caso non potranno essere effettuate modifiche alla base dati

Livello d'isolamento: Descrive l'interazione di una transazione con altre transazioni (in esecuzione nello stesso momento) che vogliono accedere agli stessi dati.

Il livello d'isolamento può essere impostato usando delle opportune istruzioni SQL.

I Livelli d'isolamento disponibili sono:

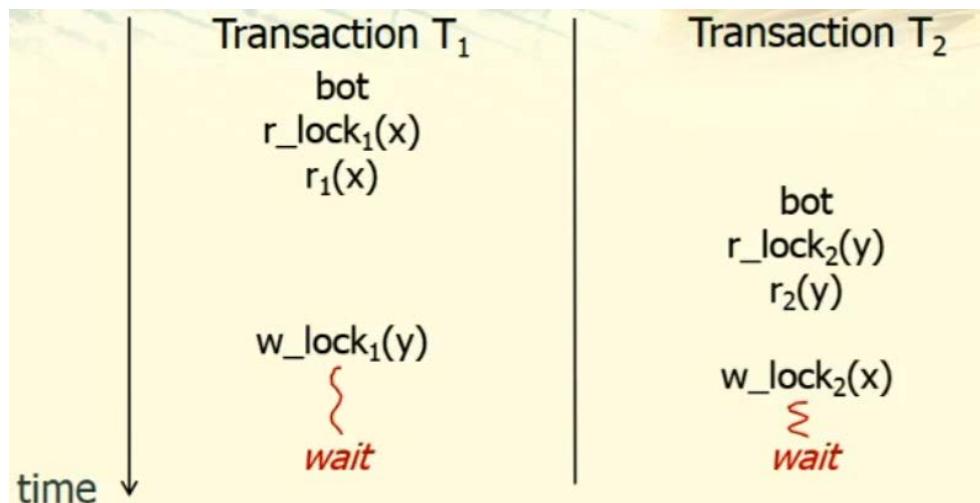
- [SERIALIZABLE](#): Se una transazione è serializable allora sarà [Strict 2 Phase Locking](#) e [Predicate Locking](#), riuscirà quindi a coprire tutte le anomalie compreso il [Ghost Update \(tipo b\)](#)
- [REPEATABLE READ](#): Utilizza lo [Strict 2 Phase Locking](#) ma non ha il [Predicate locking](#), coprirà dunque tutte le anomalie tranne il [Ghost Update \(tipo b\)](#). Non posso dunque leggere due volte lo stesso aggregato in modo allineato.
- [READ COMMITTED](#): Consente la lettura di dati che appartengono a transazioni che hanno già fatto il commit. Quando leggo l'oggetto x, il lock associato viene subito rilasciato senza attendere la fine della transazione. La lettura ripetuta dello stesso oggetto non è dunque garantita allineata.
Read Committed mi garantisce una protezione contro [Dirty Read](#).
- [READ UNCOMMITTED](#): Non abbiamo la protezione su nessun tipo di anomalia, anche dalla Dirty Read. Leggo l'oggetto senza chiedere il lock. Utilizzato solo per transazioni Read Only. Utilizzato spesso per effettuare delle statistiche sulla base dati.

In SQL il tipo di transazione e il livello d'isolamento si impostano con il comando SET TRANSACTION.

Deadlock

È un problema noto nell'informatica.

Avviene quando due transazioni richiedono delle risorse bloccate dall'altra transazione. In questo caso l'attesa per la risorsa è potenzialmente infinita.



Soluzioni al problema del DeadLock

- Timeout: Ad ogni transazione è infatti associato un timeout che permette di evitare situazioni di deadlock. Scaduto il timeout la transazione non attenderà più. Il tempo del timeout non deve essere né troppo lungo né troppo corto.
Timeout lungo -> crea tempi d'attesa lunghi
Timeout corto -> sovraccarica il sistema e inoltre rischia di killare la transazione quando il tempo di attesa può essere dovuto ai tempi d'esecuzione fisiologici del sistema e non alla presenza di un reale deadlock. Tale sistema si usa nei database centralizzati.
- 2 Phase Locking Pessimistico che consiste nell' avere tutti i lock a disposizione prima di far partire la transazione.
Tale sistema non è sempre attuabile perché dovrei poter bloccare troppe risorse in caso di più flussi condizionati all'interno della stessa transazione.
- Timestamp: basati sul tempo di avvio della transazione. Solamente le transazioni più vecchie potranno attendere un lock e mettersi in coda. Tale sistema può però generare casi di overkill, terminando delle transazioni più vecchie che potevano essere eseguite senza generare deadlock.
- Grafo delle attese: I nodi sono transazioni e gli archi rappresentano una condizione di attesa fra due transazioni. Quando il grafo presenta dei cicli allora siamo certi della presenza di un deadlock. Bisognerà killare una delle due transazioni. Il mantenimento del grafo è costoso, si usa tipicamente nei sistemi distribuiti.



Gestore dell'affidabilità

Tale componente garantisce la [Durability](#) (persistenza) e [l'Atomicità](#) delle transazioni.

I comandi transazionali che implementano il gestore dell'affidabilità sono:

- B – Begin Transaction: sancisce l'inizio di una transazione. Può essere implicito
- C – Commit Work: conclusione positiva della transazione
- A – Rollback/Abort: conclusione negativa della transazione

In caso di guasto il gestore dell'affidabilità può effettuare il riavvio del sistema grazie a due primitive:

- Warm Restart / Ripristino a Caldo: viene utilizzato quando il guasto interessa la memoria principale
- Cold Restart / Ripristino a freddo: viene utilizzato quando ho dei problemi sui mezzi di memorizzazione di massa

Il gestore dell'affidabilità utilizza delle **informazioni ridondanti**, contenute opportunamente dentro un file di log, per garantire uno stato affidabile della base dati dopo un eventuale ripristino dovuto ad un guasto.

Un file di log descrive, come un registro, tutte le operazioni di modifica che avvengono all'interno della base dati.

Assumiamo il file di log memorizzato all'interno di una memoria stabile ovvero fuori dal rischio di eventuali guasti. La memoria stabile è implementata grazie alla ridondanza. Il file di log è tipicamente memorizzato in RAID su più dischi.

Le operazioni da effettuare periodicamente sulla base dati sono:

- [Checkpoint](#)
- [Dump](#)

Struttura di un Log File

Il Log file ci permette di ricostruire le transazioni avvenute. All'interno di un log file vi sono due tipi di record:

- [Record di Transazione](#)
- [Record di Sistema](#)

Il log è scritto in *append* con l'esecuzione delle transazioni all'interno della base dati. All'interno del log visualizzo l'ordine delle transazioni per come vengono intercalate nel Lock Manager. Riuscirò dunque a visualizzare l'esatto ordine delle read/write fra le varie transazioni.

Record di Transazione

Delimitano la transazione. I delimitatori sono:

- Begin $B(T)$: Indica l'inizio della transazione T
- Commit $C(T)$: Indica il commit della transazione T
- Abort/Rollback $A(T)$: Indica l'abort della transazione T

Le azioni che la transazione può eseguire sulla base dati sono:

- Insert $I(T, O, AS)$
- Delete $D(T, O, BS)$
- Update $U(T, O, BS, AS)$

Nel *log file* non teniamo traccia delle SELECT poiché non modificano lo stato della base dati.

T: identificativo della transazione

O: oggetto che viene modificato (RowID)

AS: After State. È lo stato della porzione di base dati (oggetto O) dopo la modifica

BS: Before State. È lo stato della porzione di base dati (oggetto O) prima della modifica

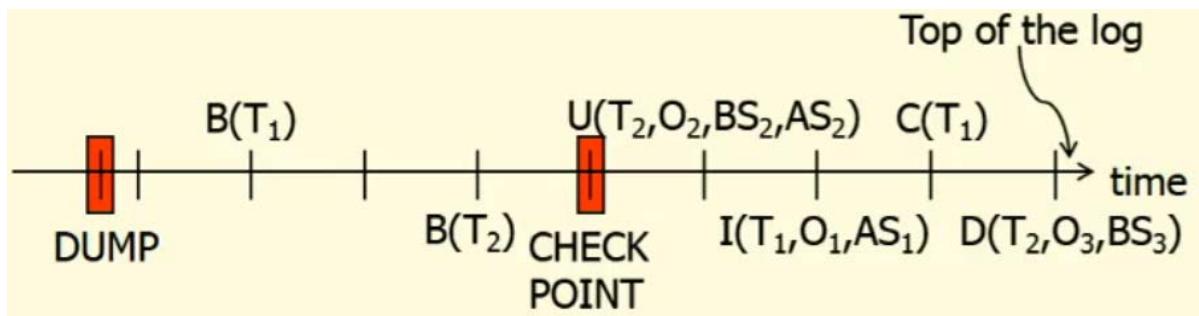
Tutte queste informazioni vengono memorizzate nel *Log File*.

Record di Sistema

Sono record che registrano delle operazioni svolte dal sistema riguardo il salvataggio dei dati su disco.

Le operazioni permesse sono due:

- Dump: è la copia fisica della base dati su un altro supporto. È un backup.
- Checkpoint: Durante il checkpoint si memorizza su disco l'elenco delle transazioni attive e non attive. Per transazioni attive intendiamo quelle che non hanno ancora eseguito il commit.



Azioni eseguite in caso di UNDO o REDO

▷ **Undo** of an action on an object O

| Action | Undo action |
|----------|----------------------------------|
| insert O | delete O |
| update O | write the before state (BS) of O |
| delete O | write the before state (BS) of O |

▷ **Redo** of an action on an object O

| Action | Redo action |
|----------|---------------------------------|
| insert O | write the after state (AS) of O |
| update O | write the after state (AS) of O |
| delete O | delete O |

Per l'UNDO devo rimettere nella base dati il Before State. Per il REDO devo invece utilizzare l'After State.

Proprietà d'Idempotenza

La base dati potrebbe andare in crash durante il processo di ripristino.

Per garantire il corretto funzionamento della base dati anche in questo caso devo sottostare alla proprietà d'idempotenza.

La proprietà d'idempotenza afferma che il risultato delle operazioni multiple di REDO e UNDO è sempre uguale alla singola operazione. Eventuali riaggiornamenti della base dati dovuti ad UNDO o REDO non saranno quindi ripetuti in modo da creare un disallineamento dei dati.

Per fare ciò devo memorizzare una copia fisica della porzione di base dati nel Before State e nell'After State.

Checkpoint

Il record di Checkpoint è utile per accelerare il processo di recovery.

L'operazione di checkpoint è effettuata automaticamente dal DBMS. Consiste nella scrittura sincrona ([force](#)) su disco tutte le pagine dati relative alle transazioni completate.

Vengono dunque scaricate su disco tutte le modifiche che sono state eseguite da transazioni che sono state correttamente completate dal sistema.

Durante il checkpoint viene anche stilata una lista delle transazioni attive, ovvero che non sono ancora concluse.

Quindi le azioni del checkpoint sono:

- Registro gli identificatori delle transazioni attive e le blocca finché il checkpoint non è finito
- Scrivo su disco tutte le pagine delle transazioni terminate (sia con commit che con abort) in maniera sincrona. Le altre attività sono messe in attesa finché la scrittura non è terminata
- Quando la scrittura delle pagine è terminata viene scritto nel log un record di checkpoint che contiene l'elenco di tutte le transazioni attive al momento del checkpoint
- Successivamente il blocco del log viene scritto su disco (in memoria stabile) in modo sincrono

Il checkpoint è un punto di sincronismo che mi garantisce transazioni persistenti fino a quel momento. In caso di ripristino non dovrò quindi considerare le transazioni che sono state rese persistenti dal checkpoint.

Il checkpoint mi permette, in caso di ripristino, di dover rielaborare tutte le modifiche avvenute sulla base dati fino all'ultimo Dump. Dovrò quindi rielaborare solo le modifiche avvenute sulla base dati postume al checkpoint.

Il checkpoint ci permette dunque di ridurre il numero di transazioni da rielaborare in caso di ripristino.

DUMP

L'operazione di DUMP crea una copia fisica della base dati.

Tale operazione viene eseguita quando il sistema è offline.

Possiamo eseguire due tipi di dump:

- Incrementale: tipicamente eseguito durante le sere della settimana
- Totale: tipicamente eseguito durante il weekend

Quando creo una copia fisica della base dati scriverò nel file di log un record di DUMP che indicherà il momento e la struttura fisica su cui è stato memorizzato il dump.

Come va scritto il Log File

Il file di log può essere scritto in due modalità differenti:

- Write Ahead Log – WAL: Prima scrivo il log (memoria stabile) con lo stato vecchio della base dati e poi eseguo la modifica nella base dati.
Dunque scrivo prima il Before State nel Log e successivamente aggiorno la base dati.
In questa modalità scrivo sempre prima il Log e poi la pagina della base dati.
Scrivo il *Before State* nel log prima di effettuare la modifica sulla base dati. Per fare UNDO
- Commit Precedence: Prima di effettuare il commit devo scrivere nel log lo stato After della base dati.
Scrivo l'After State sul log prima di effettuare il commit della transazione. In questo caso sarò in grado di effettuare il REDO dell'istruzione in caso di guasto.
Scrivo prima nel log l'*After State* della base dati e poi, dopo il record di commit, scrivo la Base Dati. Per fare REDO

Quando devo scrivere il log allora:

- Per ogni modifica della base dati scriverò prima il Before State nel Log (per fare UNDO)
- Prima di effettuare il Commit della transazione scriverò l'After state nel Log (per fare REDO)

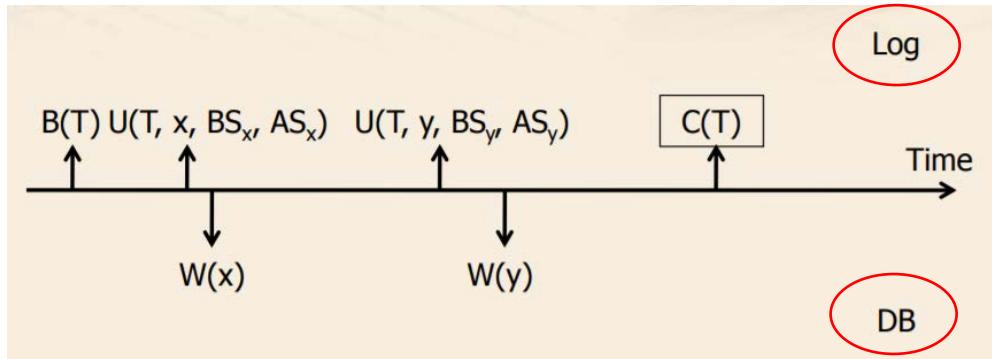
Il Before State e l'After State sono scritti in modo sincrono (force) sulla base dati tranne nel caso di Abort/rollback dove la scrittura avviene in modo asincrono.

Record di Commit

Il record di commit è il discriminante, presente nel Log File, che indica se una transazione dovrà essere rieseguita o meno in caso di ripristino della base dati.

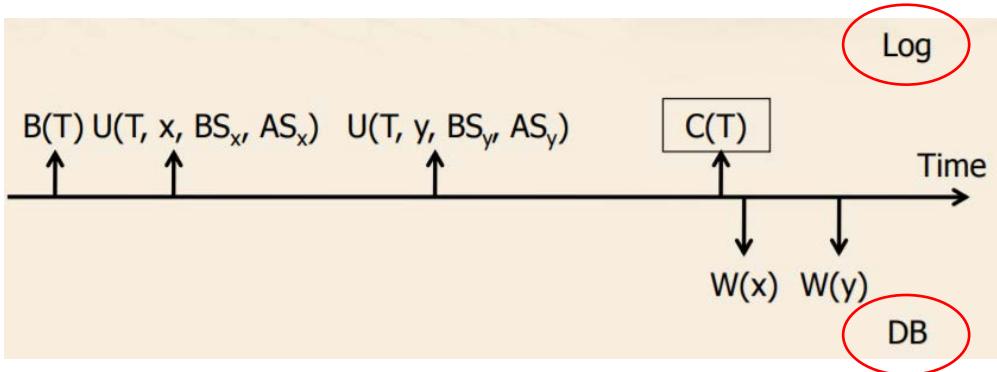
In caso di presenza del record di commit dovrò allora propagare le modifiche effettuate dalla transazione sulla base dati.

In assenza del record di commit dovrò disfare le modifiche effettuate con delle opportune istruzioni di UNDO.



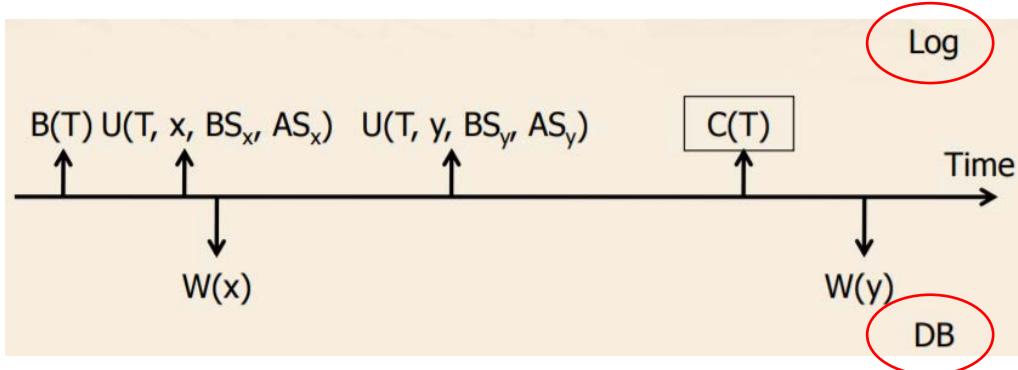
In questo caso la scrittura nel Log avviene prima di effettuare il Commit della transazione quindi sono interessato a svolgere delle operazioni di eventuale UNDO in caso di errore.

Caso [Write Ahead Log - WAL](#)



Qui invece eseguo le operazioni di modifica nel Database dopo l'operazione di Commit.
Potrò quindi effettuare delle operazioni di REDO in caso di guasto.

Caso [Commit Precedence](#)



Nei sistemi reali la scrittura nel Database può avvenire sia prima che dopo il Commit. Questo schema di scrittura potrebbe implementare una politica di tipo [steal no force](#).

In questi casi la scrittura del Log implementerà sia una politica [Write Ahead Log](#) che una [Commit Precedence](#), garantendoci così di fare sia UNDO che REDO di qualsiasi transazione.

Come avviene la scrittura de Log

Siccome la scrittura del file di Log ha un certo costo si cerca di limitarne l'impatto utilizzando delle rappresentazioni compatte, parallelizzare le scritture ed effettuare dei commit di gruppo per più transazioni insieme bufferizzando le transazioni da trascrivere nel log. (in questo modo limito il numero di force che devo effettuare)

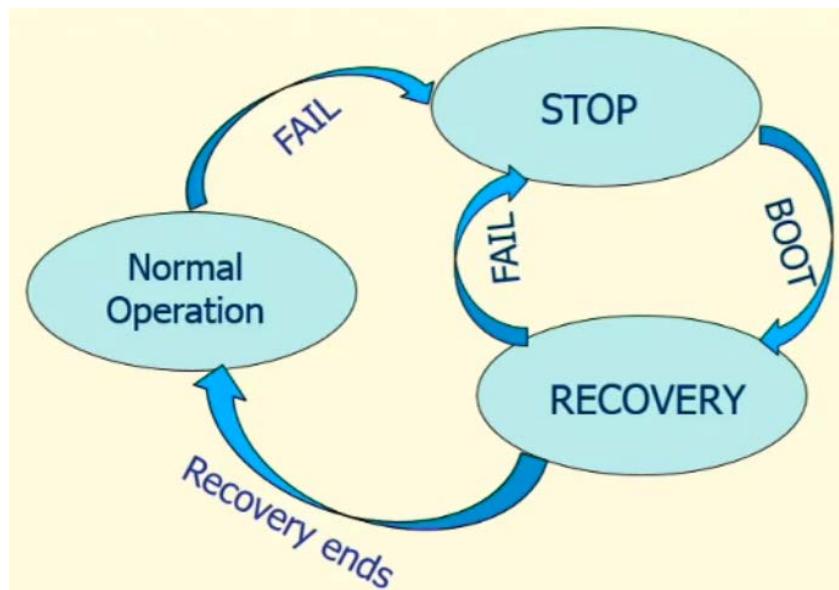
Recovery Management

La gestione dei guasti è differente al variare del tipo di problema:

- **System Failure:** È la perdita dei dati presenti in memoria principale del sistema che esegue il DBMS. Può avvenire a causa di malfunzionamenti software o problemi legati all'alimentazione.
In questo caso i dati nei dischi (memoria secondaria) rimangono integri a causa delle [force](#) eseguite.
Il Log file non è perso poiché è trascritto in memoria stabile.
- **Media Failure:** Sono i guasti fisici dei dischi su cui sono contenuti i dati della Base Dati. Anche se nei sistemi reali i dati sono salvati in modo ridondante, il ripristino della Base Dati è permesso effettuando delle operazioni di riscrittura delle informazioni perdute osservando le modifiche presenti nel Log File.

Fail-Stop model

Il processo di ripristino di un sistema è descritto dall'evoluzione di un guasto.



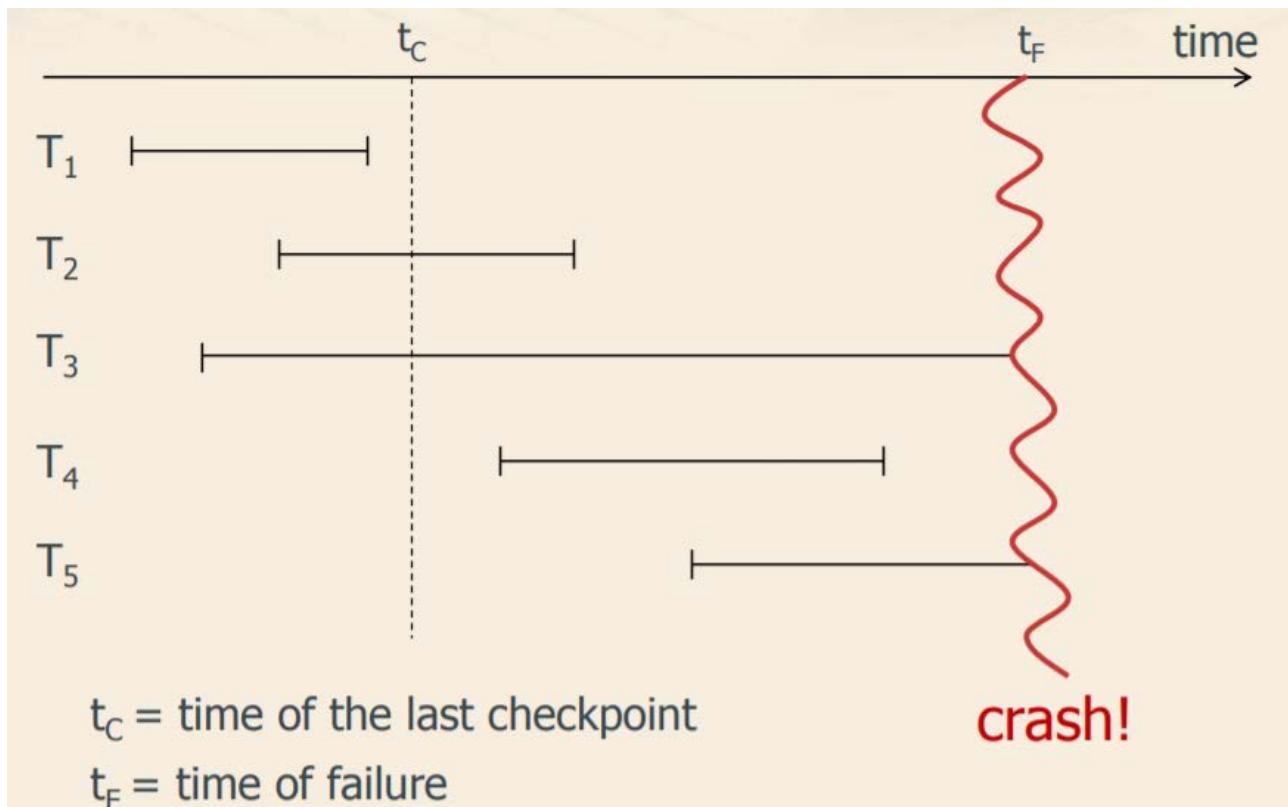
La fase di Recovery può subire ulteriori *failure* durante la sua esecuzione, ecco perché abbiamo bisogno della [proprietà d'idempotenza](#) delle operazioni di UNDO e REDO.

Recovery

Il Recovery di un sistema si differenzia in due tipi:

- **Warm Restart:** Effettuato quando si perde il contenuto della memoria. Tale processo è relativamente veloce ed in alcuni casi effettuabile in pochi secondi
- **Cold Restart:** Effettuato quando bisogna ripristinare le informazioni contenute in un disco che è andato perso. Il processo di Cold Restart è tipicamente lungo

Warm Restart



Quando avviene un guasto, le transazioni devono essere trattate diversamente in base alla casistica.

Nello schema riportato:

- T_1 non dovrà essere trattata poiché con l'operazione di checkpoint siamo sicuri che la base dati sia stata modificata in modo force con le azioni della transazione.
- T_2 dovrà essere trattata differentemente poiché vorrei poter effettuare un REDO delle azioni.
- T_3 vorrei poter effettuare un UNDO delle azioni poiché al momento del guasto la transazione era ancora attiva
- T_4 non la vedo al checkpoint ma devo effettuare un'operazione di REDO perché si è conclusa con un commit prima del guasto
- T_5 non la vedo al checkpoint ma devo effettuare un'operazione di UNDO per eliminare le modifiche che ha apportato

Più è vicino il checkpoint dal momento del guasto e meno transazioni avrò da esaminare su cui fare UNDO o REDO.

Maggiore è il tempo che intercorre fra t_c (tempo checkpoint) e t_f (tempo failure) e maggiori saranno le transazioni che dovrò trattare.

Più frequente sarà il checkpoint e meno lungo sarà il warm restart.

Algoritmo per il ripristino in caso di Warm Restart

- a. Leggo il File di Log a ritroso fino all'ultimo checkpoint e, una volta trovato, credo due liste:
 - a. UNDO = {tutte le transazioni attive fino al checkpoint}
 - b. REDO = {} (vuota)
- b. Leggo in avanti il Log partendo dal record di checkpoint e quando trovo un record di commit sposto la relativa transazione dalla lista di UNDO a quella di REDO. In caso di record di Abort non dovrò effettuare nessuna azione.
- c. Leggo il Log all'indietro ed eseguo le operazioni di UNDO per tutte le azioni transazionali delle transazioni che sono nella lista di UNDO. Tale processo si arresta al record di BEGIN della transazione "più vecchia" nella lista di UNDO.
- d. Leggo il Log in avanti partendo dal record di Begin della transazione "più vecchia" presente nella lista REDO

Applicazione per l'esercizio:

- a. Individuo il CheckPoint
- b. In UNDO metto le transazioni presenti nel CheckPoint. In REDO la lista è vuota
- c. Leggo in avanti da CheckPoint. Ogni volta che trovo un Commit sposto la transazione da UNDO a REDO
- d. Leggo all'indietro il Log File. Ogni volta che trovo un'azione di una transazione nella lista di UNDO faccio l'undo di quella transazione.
Mi fermo al record di BEGIN della transazione più vecchia nella lista degli UNDO
- e. Leggo in avanti il file ed eseguo il REDO delle azioni delle transazioni presenti nella lista di REDO.
Parto a leggere dal record di BEGIN della transazione più vecchia presente nella lista di REDO.

Undo transactions in UNDO = $\{T_2, T_3\}$

- a) DELETE O_6
- b) INSERT $O_3 = B_7$
- c) $O_3 = B_5$
- d) $O_2 = B_3$
- e) $O_1 = B_1$

Redo transactions in REDO = $\{T_4, T_5\}$

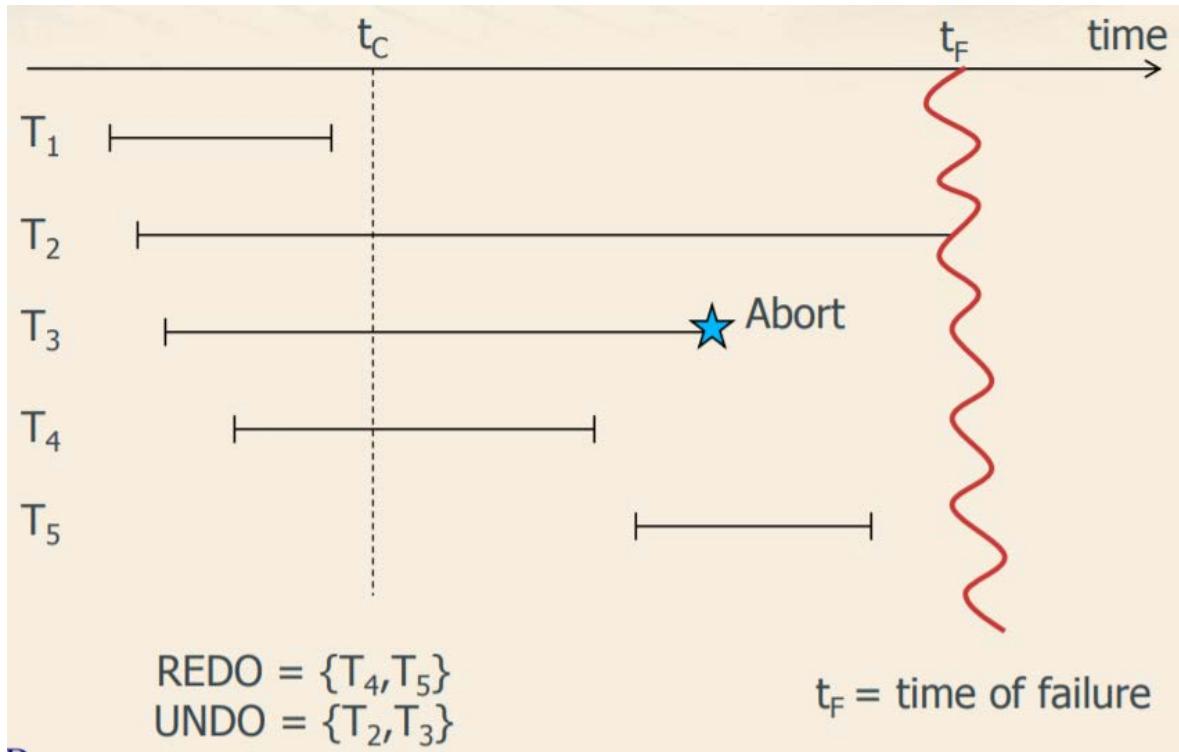
- a) $O_3 = A_4$
- b) $O_4 = A_6$

B(T_1) B(T_2) U(T_2, O_1, B_1, A_1) I(T_1, O_2, A_2) B(T_3)
C(T_1) B(T_4) U(T_3, O_2, B_3, A_3) U(T_4, O_3, B_4, A_4)
CK(T_2, T_3, T_4) C(T_4) B(T_5) U(T_3, O_3, B_5, A_5)
U(T_5, O_4, B_6, A_6) D(T_3, O_3, B_7) A(T_3) C(T_5)
I(T_2, O_6, A_8)

Read the log forward

| Operation | UNDO | REDO |
|------------|---------------------|----------------|
| CK | $\{T_2, T_3, T_4\}$ | { } |
| C(T_4) | $\{T_2, T_3\}$ | $\{T_4\}$ |
| B(T_5) | $\{T_2, T_3, T_5\}$ | $\{T_4\}$ |
| A(T_3) | $\{T_2, T_3, T_5\}$ | $\{T_4\}$ |
| C(T_5) | $\{T_2, T_3\}$ | $\{T_4, T_5\}$ |

Final lists



Cold Restart

È effettuato quando ho un media failure sulla base dati.

Per l'esecuzione di un Cold Restart devo prelevare l'ultimo DUMP e da quello ripristinare una copia su cui effettuare il riavvio.

Successivamente dovrò rieseguire, leggendo il file di Log, tutte le transazioni avvenute dall'ultimo DUMP.

In fine applico un Warm Restart per ripristinare lo stato delle ultime transazioni, mantenendo così allineata la memoria principale.

Vi sono delle versioni migliorate che tengono conto solo delle transazioni che hanno effettuato il commit. Per effettuare ciò dovrò però leggere una prima volta il Log File per identificare tutte le transazioni che hanno effettuato il commit e una seconda volta per applicare il Warm Restart a partire dal DUMP. Tale versione non sempre è la più ottimizzata poiché implica una lettura multipla del Log File.

18. Database Distribuiti

Per architettura distribuita intendiamo dati o operazioni di calcolo su macchine diverse.

Da un'architettura distribuita cerchiamo un **miglioramento delle prestazioni** (da vari punti di vista) o una **maggiore disponibilità dei dati** che implica una maggiore affidabilità in caso di guasto.
(immaginando più copie degli stessi dati su più macchine differenti)

L'architettura distribuita può essere implementata in più modi:

- **Client/Server**: Il client gestisce l'applicazione che comunica con il server dov'è implementata la Base Dati
- **DataBase distribuiti**: I nodi del sistema hanno un'istanza autonoma della base dati ma in grado di cooperare in maniera opportuna a livelli diversi. In questa configurazione sarà più difficile mantenere le proprietà delle transazioni (ad es. **Atomicità**)
- **Data replication**: riguarda le repliche dei dati presenti nella base Dati centrale. Si utilizza un nodo *slave* che è sincronizzato con i dati presenti nel DataBase centrale.
In questa versione posso usare in lettura le informazioni presenti nel nodo *slave* per effettuare delle interrogazioni
- **Architetture parallele**: Hanno come obiettivo quello delle performance. Avviene sfruttando delle macchine multiprocessore.
- **Data Warehouse**: Sono interpretabili come dei DataBase distribuiti poiché vengono alimentati da sorgenti OLTP e sono finalizzati all'esecuzione delle interrogazioni in maniera efficiente.

Proprietà rilevanti dei DataBase distribuiti

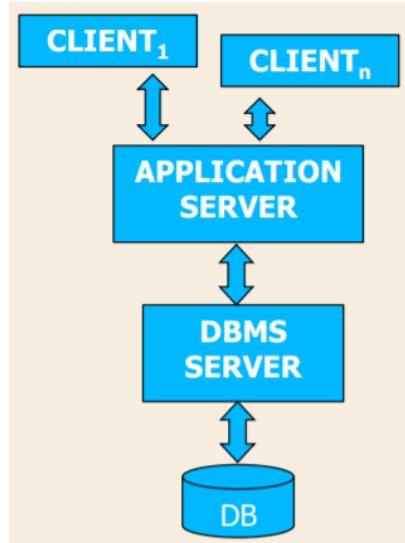
- **Portabilità**: L'SQL standard è portabile fra più sistemi senza molte difficoltà
- **Interoperabilità**: Server differenti posizionati su nodi diversi devono poter "dialogare" fra di loro. Vi sono dei protocolli che normano lo scambio d'informazioni fra le architetture parallele. Ad esempio, **ODBC** viene utilizzato nell'architettura Client/Server insieme a **X-Open-DTP** che permette di effettuare un commit di una transazione su più nodi diversi.
Ad esempio, un bonifico da una banca verso un'altra, rappresenta un commit distribuito.

Architettura Client/Server

Può essere implementata in più strati:

- Tier 2: Abbiamo due strati, uno client e uno DBMS server. La logica applicativa può essere implementata sul Client o sul Server.
- Tier 3: È una logica a tre strati.
Il Client implementa un'interfaccia molto leggera (tipicamente browser) e si interfaccia con un Application Server che implementa la logica applicativa.
L'Application Server si interfaccia con il DBMS Server che comunica con il DataBase.

Un'architettura Client/Server a 3 livelli mi permette, spostando la logica applicativa dal Client all'Application Server, di evitare dei problemi di Legacy legati ad eventuali aggiornamenti del Software. Basterà infatti aggiornare solamente l'Application Server e non tutti i client.



Le modalità d'esecuzione delle interrogazioni SQL all'interno di un sistema distribuito può essere di due tipi:

- Compile & Go: Prevede l'esecuzione dinamica dell'interrogazione SQL che viene inviata dal Client al Server. Il Client invierà al server una query SQL e questo risponderà opportunamente. Il risultato di tale interrogazione verrà poi visualizzato sull'interfaccia del Client. In questa modalità l'interrogazione è eliminata dopo la sua esecuzione insieme al suo piano d'esecuzione.
- Compile & Store: Prevede l'immagazzinamento del piano d'esecuzione all'interno del server, il quale potrà essere rieseguito anche se con parametri differenti per future interrogazioni.
Tale modalità è più vantaggiosa in caso di interrogazioni frequenti, in questo modo non dovrò rigenerare il piano d'esecuzione continuamente.

Sistemi di DataBase Distribuiti

Un database distribuito si ha quando ho un client che vuole eseguire delle interrogazioni su più DBMS diversi.

In questi casi si parla di **Autonomia Locale** ovvero: “Ogni DBMS operativo su una macchina è autonomo. Esegirà quindi operazioni di accesso alle pagine, gestione accesso concorrente etc etc in modo indipendente”. Eventualmente i DBMS saranno in grado di coordinarsi fra loro.

Perché vorrei poter distribuire i DBMS su più nodi diversi?

- **Localizzazione:** Distribuzione dei dati e delle applicazioni in posizioni diverse. Ad esempio, geograficamente
- **Disponibilità:** Se distribuisco i dati su più DBMS, in caso di guasto riduco l'impatto del guasto ad una sola porzione dei dati invece che propagarla sull'intero blocco. Tale operazione avviene invece in caso di guasto catastrofico per un DBMS centralizzato. In caso di ridondanza posso anche aumentare l'affidabilità del sistema
- **Scalabilità:** Aumento delle prestazioni

Come distribuisco una Base Dati

Posso separare una tabella **frammentandola**. Ho diversi modi per frammentare una tabella:

- **Orizzontale:** Posso dividere le tuple di una tabella secondo un predicato. Ad esempio, posso separare la tabella studenti del Politecnico in una tabella di Ingegneri e una di Architetti.
La frammentazione Orizzontale avviene applicando una **selezione** per un predicato P alle tuple. I frammenti formati da questa operazione **non sono sovrapposti**.
La tabella di partenza viene prodotta effettuando **l'unione dei frammenti**.
- **Verticale:** Posso separare una tabella in modo verticale spezzando le tuple. Posso memorizzare una parte delle informazioni di quelle tuple in una tabella e una parte in un'altra. Ad esempio, posso memorizzare l'anagrafica di un cliente dalle informazioni principali.
La selezione dei frammenti verticali avviene tramite l'operazione di **proiezione**.
Per effettuare tale operazione **devo replicare la chiave primaria** (posso replicare anche altri attributi) in modo da effettuare eventualmente delle operazioni di Join.
- **Approccio Misto:** Posso separare una tabella effettuando degli approcci misti fra orizzontale e verticale.

Parti di queste tabelle sono memorizzate in parti differenti di un sistema

Proprietà di un processo di frammentazione

- **Completezza:** Mi aspetto una frammentazione completa senza perdita di alcune parti
- **Correttezza:** L'interezza della base dati deve poter essere ricostruita partendo dai frammenti

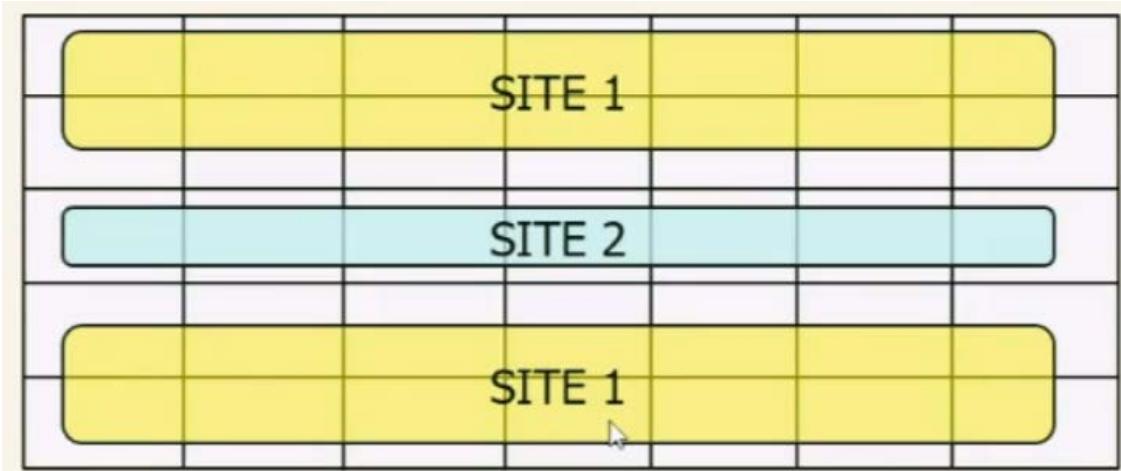
Progettazione di Basi di Dati distribuite

La progettazione di una base dati comincia dalla **definizione dei Frammenti** i quali verranno poi tipicamente collocati su server diversi, spesso sparsi in posizioni diverse geograficamente.

La tabella frammentata può dunque essere ricostruita in maniera corretta e completa attraverso tutti i frammenti.

Lo **schema di allocazione** definisce il posizionamento dei frammenti all'interno dei server nel mio sistema distribuito.

Lo stesso frammento può anche essere memorizzato in più server in modo da creare della ridondanza dei dati.



Esempio di frammentazione non ridondante



Esempio di frammentazione ridondante

In caso di ridondanza devo poter effettuare eventuali aggiornamenti su entrambi i siti che contengono lo stesso frammento in modo da mantenere l'allineamento dei dati.

Livello di Trasparenza

Indica cosa è possibile visualizzare se il DataBase è distribuito. In caso di livello di trasparenza massimo l'utente non si accorge che la base dati è distribuita.

Quando il livello di trasparenza diminuisce riconosco invece solo i frammenti presenti nel mio sito della base dati.

I livelli di trasparenza sono:

- Trasparenza di frammentazione: In questo caso il sistema non si accorge che la base dati è frammentata su più server. Le interrogazioni possono essere eseguite senza nessuna problematica.

Il sistema sottostante si occuperà di reperire il frammento corretto dal server che lo deterrà.

L'utente non si accorge che sta operando su un database distribuito

- Trasparenza di allocazione: L'applicazione conosce l'esistenza dei frammenti ma non si occupa della loro allocazione che è gestita automatico dal DBMS.

L'interrogazione deve dunque esplorare i singoli frammenti alla ricerca del frammento opportuno da cui prelevare l'informazione.

Posso anche mandare la stessa interrogazione in serie a tutti i nodi.

Il nodo che conterrà quel frammento mi risponderà opportunamente.

- Trasparenza di linguaggio: In questo caso l'utente deve scrivere il programma che conosce il frammento interessato e anche la sua allocazione fisica su un dato nodo (server).

Devo quindi eseguire il lavoro che dovrebbe eseguire il DBMS.

Questo livello di trasparenza implica un lavoro maggiore da parte dell'utente, riducendo così anche la portabilità.

Tale livello di trasparenza adatta l'SQL standard come mezzo di interrogazioni per tutti i server. Vi è anche un altro livello di trasparenza che impone all'utente l'esecuzione dell'interrogazione nel linguaggio (dialetto) del singolo nodo che voglio interrogare.

```
SELECT SName  
FROM S1  
WHERE S# = :CODE  
IF(NOT FOUND)  
    SELECT SName  
    FROM S2  
    WHERE S# = :CODE
```

```
SELECT SName  
FROM S1@xxx.torino.it  
WHERE S# = :CODE  
IF(NOT FOUND)  
    SELECT SName  
    FROM S2@xxx.roma1.it  
    WHERE S# = :CODE
```

La trasparenza indica dunque quale componente è gestita direttamente dal sistema ed io, utente, non devo manipolare direttamente.

Ad esempio:

- Trasparenza di frammento -> L'utente non vede i frammenti. Vede l'intera tabella
- Trasparenza di allocazione -> L'utente non vede la presenza dei nodi ma deve effettuare interrogazioni conoscendo i vari frammenti
- Trasparenza di linguaggio -> L'utente non deve conoscere i "dialetti" dei vari nodi, potrà così utilizzare l'SQL standard.

Transaction Classification

Discuteremo riguardo i problemi, il comportamento delle transazioni e come dobbiamo agire per poter risolvere questi problemi.

I tipi d'interrogazione che il mio sistema distribuito può processare sono differenti a seconda dell'implementazione che applico:

- Remote Request (richiesta remota): Posso eseguire interrogazioni di sola lettura verso un singolo server remoto
- Remote Transaction (transazione remota): Posso eseguire delle transazioni (sia scrittura che lettura) verso un singolo server remoto
- Distribuite Transaction (transazione distribuita): Posso eseguire qualsiasi comando SQL verso un singolo server remoto ma la transazione può includere comandi che vengono indirizzati a più server diversi.

In questo caso devo permettere l'atomicità della transazione a livello globale su tutti i server del database distribuito.

Per fare ciò si usa il meccanismo del 2 Phase Commit

- Distributed Request (richiesta distribuita): In questo caso ogni comando della transazione può fare riferimento a dati presenti in più server diversi.

Questa versione è la più complicata da sviluppare e implementa il livello di trasparenza di frammentazione

Esempio:

Ho una tabella Account su cui è stato eseguita una frammentazione orizzontale sul campo Acc# (che è la chiave primaria). Un frammento di quella tabella è memorizzato nel *nodo_1* e un altro nel *nodo_2*.

La seguente transazione è di tipo Distributed Request poiché la tabella su cui faccio l'accesso è "Account". Sarà dunque il sistema ad occuparsi del rintracciamento dei vari frammenti all'interno del database distribuito. L'ottimizzatore conosce i frammenti e i nodi dove sono collocati (trasparenza di frammentazione)

Se la transazione fosse stata di tipo Distributed Transaction allora nella Update avrei dovuto inserire il nome del frammento che conteneva la tupla in questione A₁ o A₂. In questo caso il livello di trasparenza sarebbe stato: *trasparenza di allocazione*.

Se entrambe le transazioni erano riferite allo stesso frammento (es. A₁) allora era una Remote Transaction. (perché avevo anche un'operazione di scrittura anche se accedevo allo stesso server)

The following table is given
• Account (Acc#, Name, Balance)
Fragments and allocation schema

| Horizontal fragmentation | Allocation schema |
|---|-------------------|
| $A_1 = \sigma_{acc\# < 10000} Account$ | Node 1 |
| $A_2 = \sigma_{acc\# \geq 10000} Account$ | Node 2 |

BoT (Beginning of transaction)
UPDATE Account
SET Balance = Balance - 100
WHERE Acc# = 3000

UPDATE Account
SET Balance = Balance + 100
WHERE Acc# = 13000
EoT (End of transaction)

Distributed DBMS Technology

Per garantire le proprietà ACID delle transazioni sono state delle opportune metodiche:

- Atomicity: dobbiamo coordinare l'esecuzione di una transazione distribuita. Per fare ciò si adopera l'algoritmo di 2 Phase Commit
- Consistency: i sistemi attuali la gestiscono personalmente in maniera locale. La gestione della consistenza, ovvero del rispetto dei vincoli d'integrità avviene dunque a livello locale.
- Isolation: è garantito se viene usato lo Strict 2 Phase Commit insieme al 2 Phase Commit
- Durability: è garantito apportando delle modifiche opportune alla procedura di recovery in modo da tener conto che adesso il commit è di tipo 2 Phase Commit

Altre problematiche dell'esecuzione di un'interrogazione su un sistema distribuito sono legate all'ottimizzazione. La query dovrà infatti essere suddivisa in porzioni, ognuna delle quali verrà eseguita in un dato nodo del sistema. Queste porzioni potrebbero però essere interacciate e dipendenti.

In questi casi bisogna considerare anche il costo del trasferimento delle informazioni fra i nodi.

Ci concentreremo sulle metodologie applicate al mantenimento della proprietà di Atomicità. Tutti i nodi che partecipano alla transazione distribuita dovranno effettuare la stessa decisione riguardo il commit o il rollback della transazione.

I problemi che vogliamo risolvere sono:

- Un nodo può smettere di funzionare durante l'esecuzione di una transazione
- Guasti legati alla rete che impediscono la comunicazione fra i nodi
- La rete si divide in due sottoreti rendendo isolati delle porzioni di nodi

2 Phase Commit

Mi permette di gestire il commit di una transazione che viene eseguita su più di un nodo.

L'obiettivo del protocollo è quello di risolvere tutte le problematiche legate all'esecuzione di una transazione in un sistema distribuito.

Il funzionamento della seguente procedura è descritto dal parallelismo con un matrimonio.

Durante un matrimonio, un funzionario chiederà ai due coniugi se vorranno sposare l'altro coniuge e solo dopo che entrambi avranno accettato (senza possibilità alcuna di ripensamento da parte di almeno uno dei due coniugi), l'unione dei due sarà ufficializzata.

I punti fondamentali per l'esecuzione di tale algoritmo sono:

- Domando a tutti i nodi partecipanti alla transazione
- Risposta dei nodi vincolante

Per l'esecuzione del 2 Phase Commit vi sarà:

- Transaction Manager (TM): è il coordinatore del 2 Phase Commit. Svolge il ruolo del funzionario nell'esempio del matrimonio. Il ruolo di Transaction Manager è intrapreso da un nodo qualsiasi, anche uno dei partecipanti alla transazione (resource managers)
- Resource Managers (RM): sono i nodi che partecipano alla transazione e gestiscono i dati

Implementazione:

Il TM (Transaction Manager) e l'RM (Resource Manager) hanno dei Log File separati e indipendenti.

Le informazioni registrate nel log del Transaction Manager saranno però differenti dispetto a quelle del Resource Managers

Le informazioni registrate all'interno del Log File del Transaction Manager sono mappate nei seguenti record:

- Prepare: viene scritto all'inizio del coordinamento della transazione distribuita. Tale record contiene l'informazione di tutti i Resource Managers che partecipano alla transazione (NodeID + ProcessID)
- Global Commit/Abort: la scrittura di questo record definisce se tutti i nodi partecipanti alla transazione hanno effettuato il Commit o l'Abort. Indica quindi il Commit o l'Abort a livello globale.
In questo caso avviene il rilascio dei Lock delle risorse da parte di tutti i nodi partecipanti alla transazione.
- Complete: Viene scritto quando il protocollo ha finito la propria esecuzione

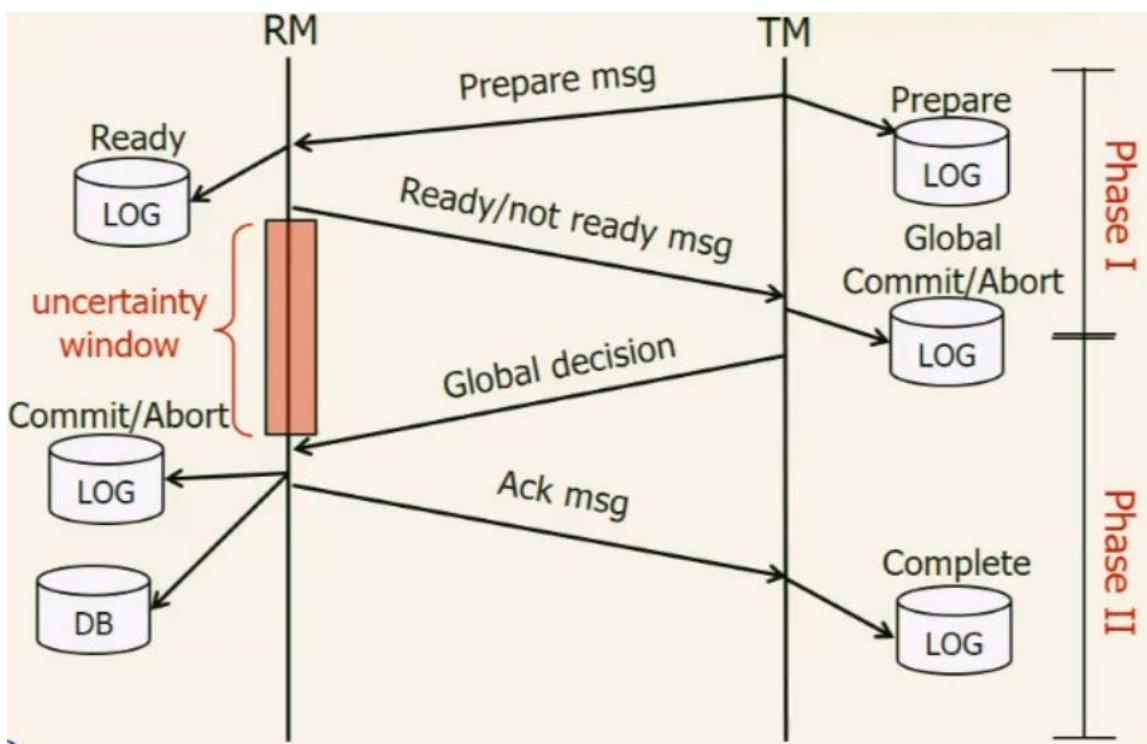
All'interno del Log File del Resource Manager dobbiamo inserire i seguenti record:

- Ready: Indica se il Resource Manager è in grado di eseguire il Commit della transazione e se intende farlo.
È il "sì" di uno dei due coniugi partecipanti al matrimonio.
In questo caso la decisione, una volta presa, non potrà essere cambiata.
Quando un nodo del Resource Manager scrive un record di Ready nel proprio Log File, si definisce come nodo in uno "*stato affidabile*".
Le risorse sono ancora Locked e vengono rilasciate solamente dopo il rilascio Globale e non locale.

Descrizione del funzionamento dell'algoritmo 2 Phase Commit

Il funzionamento del 2 Phase Commit è descritto dai seguenti passi:

- Il TM invia un messaggio di Prepare a tutti i RM partecipanti alla transazione.
Il TM scrive un record di Prepare all'interno del proprio Log File.
Il TM setta un timeout
- Ogni singolo RM che riceve la richiesta di Prepare esegue la sua porzione di transazione e se può fare commit scrive nel proprio Log File un record di Ready.
Ogni RM invia al TM un messaggio Ready/Not_Ready
- Il TM se riceverà Ready messages da tutti gli RM allora scriverà nel proprio Log File un record di Global Commit, altrimenti un record di Global Abort.
TM invierà un messaggio di Global Commit/Abort a tutti i nodi partecipanti.
Nel caso in un scada il timeout allora significa che uno dei RM non è riuscito a fornire una risposta a causa di un guasto. In questo caso la transazione va in Global Abort.
- Il TM invia l'esito della transazione ad ogni RM tramite un messaggio di tipo *Global Decision*. Adesso tutti gli RM faranno Commit o eventualmente Abort.
- Ogni RM invierà un ACK al TM e dopo che il TM avrà ricevuto tutti gli ACK scriverà nel proprio Log File un record di *complete*.
- Se il TM non riceve tutti gli ACK allo scadere del timeout, si reimposta un nuovo timeout e si inviano nuovamente dei messaggi di Global Decision a tutti i nodi il cui ACK non era stato ricevuto



Più la **finestra d'incertezza** è grande e maggiore sarà il tempo in cui l'RM manterrà le risorse della transazione bloccate senza permetterne l'accesso ad altre transazioni.

Tale situazione è una conseguenza dello [Strict 2 Phase Locking](#).

La finestra d'incertezza deve essere più piccola possibile.

Guasto di un Resource Manager, come lo gestisco?

La [procedura di Warm Restart](#) deve essere ridefinita per il corretto funzionamento.

Il Log File del Resource Manager conterrà infatti il record di Ready se la transazione era pronta ed eseguire il commit e il record di Global Commit nel caso in cui è stato eseguito realmente il commit della transazione da parte di tutti i nodi interessati.

Dobbiamo quindi modificare il funzionamento dell'algoritmo di Warm Restart inserendo una nuova lista in fase di Recovery. Tale lista prenderà il nome di **READY list** che contiene gli identificatori di tutte le transazioni che hanno scritto il record di Ready.

Leggendo il Log File dal record di checkpoint, quando incontrerò un record di Global Commit sposterò l'identificativo della transazione dal READY list alla REDO list se invece incontrerò un record di Global Abort dovrò spostare l'identificativo dal READY List alla UNDO List.

Nel caso in cui, alla fine del processo, sarà rimasto almeno un identificativo all'interno della READY list significa che il sistema ha avuto un guasto dopo aver scritto il record di Ready ma prima di ricevere il messaggio di Global Commit/Abort.

Il sistema dovrà dunque effettuare un'operazione di Recovery distribuito chiedendo al Transaction Manager l'esito della transazione in questione.

Guasto di un Transaction Manager, come lo gestisco?

Un Transaction Manager si può guastare durante l'invio o la ricezione dei messaggi:

- **Prepare** (in uscita)
 - **Ready** (in ingresso)
 - **Global Decision** (in uscita)

Fase I

Fase II

La gestione della recovery lato Transaction Manager è differente:

- Se nel Log File del Transaction manager l'ultimo record trovato è un Prepare allora la transazione va in Abort e viene inviato un messaggio di Global Abort a tutti i nodi partecipanti.
 - Se nel Log File del Transaction Manager l'ultimo record trovato è una Global Decision allora bisogna rifare la seconda fase inviando a tutti gli RM la Global Decision

Se il guasto è nella prima fase -> annulla la transazione

Se il guasto è nella seconda fase -> ripeto l'operazione un numero arbitrario di volte

Guasti di Rete, come li gestisco?

Se ho la perdita di pacchetti della prima fase (Prepare o Ready) la transazione andrà in Abort.

Se ho la perdita dei pacchetti nella seconda fase (Global decision, ACK) allora questi verranno rinviati allo scadere del timeout fino alla convergenza dell'algoritmo.

X-Open-DTP

È un protocollo standard che serve per coordinare l'esecuzione delle transazioni distribuite.
Questo prodotto può interfacciarsi con diversi DBMS anche se fra loro diversi (Ad esempio, posso avere Oracle in un nodo e SQL Server)

Lo scenario operativo di questo prodotto prevede:

- Un Client
- Un Transaction Manager (TM)
- Più Resource Managers (RM)

Il protocollo definisce un'interfaccia fra Client e Transaction Manager (**TM Interface**) e un'interfaccia fra il TM e gli RM (**XA Interface**). Tutti i server devono quindi disporre della XA Interface.

Il protocollo definisce due ottimizzazioni dell'algoritmo [2 Phase Commit](#)

19. Progettazione Fisica

Il decorso di progettazione di un database è definito da alcuni step fondamentali:

- Parto dai requisiti di partenza
- Producò uno schema concettuale (ER oppure DFM) in una fase di design concettuale
- Producò uno schema logico in una fase di design logico
- Producò uno schema fisico per l'implementazione fisica del database

L'obiettivo della progettazione fisica è generare un sistema che abbia delle buone prestazioni.

Dobbiamo ricordarci che le applicazioni non sono influenzate da come sono memorizzati i dati all'interno del sistema.

L'ottimizzatore, per aumentare le performance, usa le strutture fisiche accessorie (indice) come degli acceleratori. È importante selezionare delle adeguate strutture fisiche accessorie e non inserirne di inutili poiché queste richiedono del tempo considerevole di aggiornamento.

Durante la progettazione fisica bisogna considerare quale prodotto relazionale utilizzeremo durante l'implementazione (programma DBMS). Questa considerazione è necessaria poiché ogni modello relazionale utilizza strategie differenti d'implementazione. Ad esempio, vi sono differenze nel metodo di memorizzazione dei dati oppure nei metodi d'accesso ai dati (non tutti gli indici possono essere implementati da tutti i sistemi). Nel nostro caso studieremo Oracle.

Gli strumenti in ingresso per la produzione dello schema fisico sono:

- Input:
 - schema logico della base dati
 - caratteristiche del DBMS di partenza, conoscenza degl'indici utilizzabili dal nostro sistema
 - Carico applicativo:
 - ovvero conoscenza delle interrogazioni più importanti con la loro frequenza d'esecuzione
 - conoscenza delle istruzioni d'aggiornamento più importanti con la loro frequenza
 - conoscenza dei vincoli di performance, quindi tempo massimo d'esecuzione per quell'istruzione

Le conoscenze sul carico applicativo possono essere ragionevolmente calcolate per una base dati che implementa un sistema OLTP poiché in questo caso il carico di lavoro è di tipo strutturato e ripetitivo.

Nel caso delle basi dati che implementano dei sistemi OLAP non sarà semplice prevedere il carico di lavoro poiché l'esecuzione delle query "ad-hoc" non possiede nessuna informazione storicizzata rendendo quindi il carico di lavoro difficile da prevedere.

I risultati in uscita sono:

- Schema fisico delle tabelle
- Affermare la dimensione del blocco iniziale da allocare in memoria in modo da capire quando spazio devo allocare quando devo riallocare un nuovo blocco di dati in memoria principale
- Dimensione Buffer
- Dimensione pagine
- Organizzazione fisica dei dati:
 - Heap: configurazione compatta di rappresentazione dei dati, i blocchi saranno tutti pieni
 - Ordinato: file ordinato poiché tipicamente uso indici clustered. Posso scegliere un solo indice clustered all'interno della mia base dati poiché questo vincola la posizione fisica dei dati. Tutti gli altri indici dovranno essere di tipo unclustered.
 - Particolari configurazioni che precalcolano un'operazione di join
- Carico di lavoro:
 - Tabelle accedute durante l'interrogazione
 - Attributi utilizzati per selezioni e join
 - Selettività delle selezioni
 - Per ogni aggiornamento devo osservare gli attributi e le tabelle coinvolte nei predicati di selezione
 - Tipo di aggiornamento da eseguire

I costruttori dei DBMS propongono delle configurazioni di default poiché i parametri da settare sono difficili da comprendere.

Le strutture dati da scegliere sono dunque:

- Selezione dei metodi di memorizzazione fisica delle tabelle
 - Selezione della struttura dati da utilizzare (heap oppure clustered)
- Selezione degli indici da utilizzare
 - Attributi da indicizzare (uso Hash oppure B⁺-Tree e tipo clustered o unclustered in base al tipo di attributo)

Certe volte bisogna anche valutare se vogliamo applicare dei cambiamenti allo schema logico relazionale in modo da accelerare l'esecuzione delle istruzioni SQL pur separando alcune dipendenze funzionali. (snowflake)

Oppure se vogliamo memorizzare il database in più partizioni differenti dei dati in modo da ridurre il carico di lavoro sul controller del disco, parallelizzando così il carico.

La progettazione fisica, al contrario della progettazione logica e concettuale, non ha un procedimento standard d'esecuzione. Buona parte di questo passo di realizzazione è basata sull'esperienza del tecnico che la implementa.

Lo studio delle performance è infatti di rilevante importanza e possiamo agire sul miglioramento dell'implementazione fisica grazie a delle fasi di database tuning. Sempre rimanendo trasparenti al livello applicativo, senza modificare il parco applicativo.

Tecniche applicative di progettazione fisica

Chiave Primaria

La chiave primaria è spesso utilizzata per operazioni di selezione e Join.

È quindi ragionevole creare un indice per la chiave primaria.

Quest'indice può essere di tipo clustered o unclustered. Se ci interessa l'ordinamento fisico dei dati su un altro attributo allora creerò un indice unclustered per la chiave primaria, altrimenti potrò generarlo di tipo clustered. (ricorda: posso fare un indice clustered su 1 solo attributo perché mi condiziona l'ordinamento fisico dei dati)

L'implementazione dell'indice sulla chiave primaria ci permette di controllare il vincolo di univocità della chiave primaria rapidamente poiché, presa una nuova ipotetica chiave primaria posso ripercorrere velocemente l'indice ad albero e controllare o meno la presenza del nodo foglia. Nel caso in cui avrò un nodo foglia significherà che la chiave primaria è già presente nella mia tabella quindi non potrò inserire la tupla con chiave primaria duplicata. Se invece non avrò la presenza del nodo foglia candidato allora la tupla con la chiave primaria che voglio inserire sarà sicuramente univoca.

Predicati più comuni

Controllo i predicati più comuni e rilevanti e aggiungo gli indici sugli attributi utilizzati da quei predici. L'inserimento dell'indice viene effettuato confrontando il costo d'esecuzione del piano d'esecuzione con e senza indice. Alcune volte il costo d'esecuzione non varia anche inserendo un nuovo indice.

Devo anche considerare gli effetti collaterali della realizzazione degl'indici. In caso di aggiornamento dovrò infatti aggiornare tutti gli indici e tale operazione è generalmente costosa. Tale operazione avviene all'interno della transazione.

Gli indici possono avere dei volumi confrontabili anche con quelli della tabella dei fatti. L'introduzione di nuovi indici deve quindi essere giustificata con un opportuno aumento delle performance.

Dimensioni tabelle

Non indicizziamo mai le tabelle di piccole dimensioni. Tipicamente fino ad un migliaio di tuple (10^3) consideriamo una tabella come se di piccole dimensioni.

Se l'attributo ha un dominio con una cardinalità bassa non conviene utilizzare un indice per quell'attributo. Questo poiché se l'attributo ha una selettività bassa spesso ci conviene eseguire uno scan dell'intero blocco invece che accedere tramite indici. (caso sistemi OLTP)

Nei Data Warehouse possiamo sfruttare la presenza degli indici Bitmap e quindi in questo caso potrebbe aver senso utilizzare un indice perché rende più veloce l'operazione d'intersezione delle condizioni.

Tipo di attributo

In base al tipo di attributo su cui vogliamo applicare dei predicati avremo differenti indici da utilizzare:

- Se l'attributo è intero potrò voler effettuare delle query d'intervallo quindi utilizzerò un indice di tipo **B⁺-Tree**
- Se l'attributo invece non è di tipo intero, tipicamente effettuerò delle query d'uguaglianza quindi l'indice da preferire sarà di tipo **Hash**, potrei utilizzare anche un indice di tipo **B⁺-Tree** ma avrò delle performance peggiori.

Se vogliamo accelerare le interrogazioni possiamo pensare di utilizzare un indice di tipo clustered in modo da sfruttare la contiguità fisica.

Predicati in AND

Se abbiamo dei predicati in AND posso pensare di definire un indice composto di più attributi. Per gl'indici composti l'ordine è importante quindi se ad esempio definirò un indice sugli attributi (cognome, nome) potrò usarlo per le ricerche sui predicati di (cognome) e di (cognome, nome) ma non per i predicati solo di (nome). Posso quindi entrare soltanto dal primo campo dell'ordinamento. Quell'indice soddisfa quindi le ricerche per un sottoinsieme di attributi e non per tutte le combinazioni possibili di quegli attributi.

La manutenzione di un indice composto è molto più costosa rispetto alla manutenzione che sarà prevista per un indice su un singolo attributo.

Operazioni di Join

Se voglio accelerare un'operazione di join effettuata con un algoritmo Nested Loop, vorrei poter inserire un indice nella tabella interna in modo da velocizzare l'operazione (sempre che non sia piccola altrimenti mi conviene salvarla in memoria principale ed effettuare un sequential scan).

Se vogliamo utilizzare un algoritmo di join di tipo Merge Scan ci converrebbe utilizzare un indice di tipo B⁺-Tree possibilmente clustered, in modo da sfruttare l'ordinamento fisico dei dati.

Group By

Per l'operazione di GroupBy posso utilizzare:

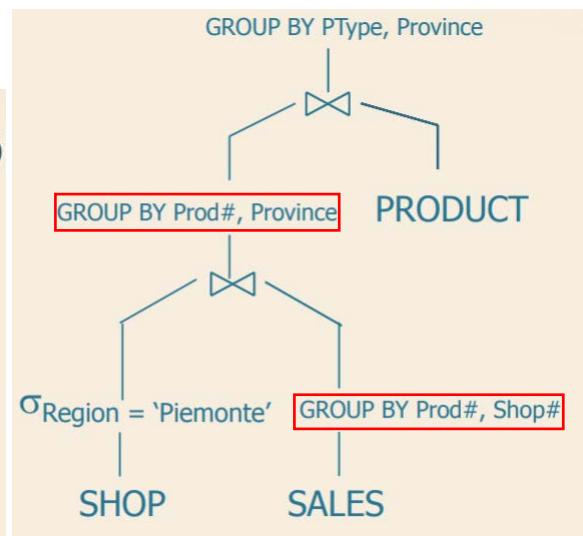
- Indici Hash o B-Tree sugli attributi di raggruppamento
- Push Down del Group By

Push Down del Group By

Un ricercatore Microsoft ha ideato un modo per ridurre la quantità di dati che dobbiamo portarci dietro negli stadi intermedi durante le operazioni di Join. Tale tecnica è basata sull'introduzione di alcune Group By.

L'operazione di Group By viene utilizzata per ridurre enormemente la dimensione dei dati prima di effettuare un'operazione di Join. Grazie alle Group By possiamo infatti eliminare la dipendenza da un attributo "strizzando" la tabella dei dati di cui vogliamo fare il Join.

| Tables |
|--|
| PRODUCT (<u>Prod#</u> , PName, PType, PCategory) |
| SHOP (<u>Shop#</u> , City, Province, Region, State) |
| SALES (<u>Prod#</u> , <u>Shop#</u> , <u>Date</u> , Qty) |
| SQL query |
| <pre>SELECT PType, Province, SUM (Qty) FROM Sales S, Shop SH, Product P WHERE S.Shop# = SH.Shop# AND S.Prod# = P.Prod# AND Region = 'Piemonte' GROUP BY PType, Province;</pre> |



Hints – Direttive applicative per l'ottimizzatore

Alcuni sistemi posso obbligare l'ottimizzatore ad impostare un certo piano d'esecuzione. Possiamo quindi scrivere dei suggerimenti (hints) che impongono certi funzionamenti all'ottimizzatore. L'utilizzo degli hints può portare dei benefici sulle performance d'esecuzione ma elimina il fattore dell'indipendenza dei dati poiché il livello applicativo prenderà così il controllo sulla struttura dei dati.

Conviene quindi evitare l'uso degli hints.

Esempi sul funzionamento del Piano di Controllo

Di seguito visualizzeremo degli esempi per comprendere il funzionamento del piano di controllo sviluppato dal sistema durante la fase d'ottimizzazione. Cercheremo dunque di capire le dinamiche delle scelte effettuate per la produzione del piano di controllo.

Esempio 1:

EMP (Emp#, EName, Dept#, Salary, Age, Hobby)
DEPT (Dept#, DName, Mgr)

- In EMP
Dept# FOREIGN KEY REFERENCES DEPT.Dept#
- In DEPT
Mgr FOREIGN KEY REFERENCES EMP.Emp#

tabella DEPT fa riferimento all'attributo Emp# nella tabella EMP.

Le tabelle che utilizzeremo per gli esempi sono le seguenti. L'attributo Dept# (num_dipartimento) della tabella EMP (impiegato) fa riferimento all'attributo Dept# della tabella DEPT (dipartimento). Mgr nella

SQL query

```
SELECT *  
FROM EMP  
WHERE Salary/12 = 1500;
```

Per la seguente query potrei pensare d'inserire un indice sull'attributo Salary per accelerare la velocità d'esecuzione. In questo tipo di indice vorrei poter svolgere delle interrogazioni con dei predicati d'intervallo quindi è opportuno impostare un indice di tipo B⁺-Tree.

Una volta generato il piano d'esecuzione possiamo osservare se l'indice generato viene usato o meno. L'indice potrebbe non venire usato per vari motivi:

- L'espressione aritmetica impedisce l'uso dell'indice. In questo caso posso riscrivere l'espressione aritmetica -> WHERE Salary = 18000;
- Se anche in questo caso l'indice non viene utilizzato allora posso ipotizzare che la frequenza di quel valore di stipendio sia molto frequente, in questo caso l'ottimizzatore potrebbe scegliere di eseguire un full table scan poiché risulterebbe più veloce invece di andare a leggere le tuple pagina per pagina utilizzando continuamente l'indice per l'accesso.

Esempio 2:

Supponiamo che la tabella Employee memorizzi circa 30 tuple per ogni pagina.

Immaginiamo la cardinalità della tabella dipartimenti abbia cardinalità:

- a) $\text{Card(DEPT)} = 50$
- b) $\text{Card(DEPT)} = 2000$

Vogliamo accedere ad un codice di dipartimento dato del dipartimento all'interno della tabella Employee. Vale la pena definire un indice secondario su Employee.Dept# in modo da accedere più velocemente alle tuple?

Ragionamento:

a) Il numero di tuple per ogni blocco è pari circa a 30, se avrò $\text{Card(DEPT)} = 50$ è abbastanza probabile che quasi tutti i dipartimenti Dept# siano presenti in ogni blocco della tabella Employee. In questo caso posso quindi evitare l'inserimento di un indice. È quindi abbastanza probabile che 20-30 dipartimenti siano presenti nel mio blocco di 30 della tabella Employee.

Accederò dunque all'intero blocco ricercando (full scan) ricercando la tupla d'interesse.

b) Nel caso in cui $\text{Card(DEPT)} = 2000$ dentro il mio blocco avrò al massimo 30 dipartimenti diversi dei 2000 possibili. In questo caso leggere un blocco invece che un altro è estremamente conveniente poiché potrei avere dei blocchi totalmente privi del Dept# che sto cercando.

Nel caso b) riusciamo quindi a caricare in memoria principale solamente i blocchi che sicuramente conterranno l'attributo che soddisfa il predicato.

Esempio 3:

```
SELECT EName, Mgr  
FROM  EMP E, DEPT D  
WHERE E.Dept# = D.Dept#  
AND DName = 'Toys';
```

Per la seguente query posso utilizzare un indice di tipo Hash Index sulla tabella Department sull'attributo DName, in modo da velocizzare l'operazione di selezione. Scelgo hash index poiché DName non potrà mai essere interessato da prediciati d'intervallo.

Per accelerare l'operazione di Join potrei utilizzare un indice Hash oppure un indice di tipo Nested Loop. Per accelerare l'operazione di join Nested Loop potrei creare un indice sulla Inner Table per l'attributo di join (Dept#).

In questo caso la *outer table* sarebbe DEPT perché accedo solo alle tuple che rispettano la condizione su DName grazie all'indice hash creato precedentemente e per ognuna di esse faccio il *probing* su EMP che sarà indicizzato con un indice su Dept#.

Esempio 4:

```
SELECT EName, Mgr  
FROM EMP E, DEPT D  
WHERE E.Dept# = D.Dept#  
AND DName = 'Toys'  
AND Age=25;
```

In questo caso abbiamo due condizioni di selezione messe in AND.
Bisognerà osservare quale dei due attributi sarà più selettivo, decidendo così se “entrare” da Toys oppure da Age.

Anche l’operazione di join subirebbe delle modifiche nel caso di Nested Loop. Se Age è l’attributo più

selettivo allora la Tabella EMP, che contiene Age, diventerebbe l’*Outer Table* e la tabella DEPT la *Inner Table*.

Nella *Outer table* sceglierò sempre la tabella con l’attributo più selettivo in modo da ridurre il numero di *fail* quando cercherò di fare match con la tupla della *Inner Table*. In questo modo avrò anche meno cicli da effettuare.

Esempio 5:

```
SELECT EName, Mgr  
FROM EMP E, DEPT D  
WHERE E.Dept# = D.Dept#  
AND Salary BETWEEN 10000 AND 12000  
AND Hobby='Tennis';
```

In questo caso tipicamente la selezione più selettiva sarà quella su Hobby. Le selezioni per intervallo sono probabilmente meno selettive di quelle d’uguaglianza.

Potrei valutare anche di fare due indici, un Hash

su Hobby e un B+-Tree su Salary. Anche se l’ottimizzatore potrebbe non utilizzarli entrambi.

Nel caso del join con Nested loop: **EMP Outer Table perché la sto leggendo tramite un indice che non è la chiave di join.** *Inner Table* è DEPT e per dipartimento devo decidere se indicizzare per l’attributo Dept#. Tale attributo è però la chiave primaria quindi potrebbe già essere indicizzato. Tipicamente i sistemi indicizzano la chiave primaria
Se la tabella DEPT è molto piccola non serve indicizzarla.

Esempio 6:

```
SELECT Dept#, Count(*)  
FROM EMP  
WHERE Age>20  
GROUP BY Dept#
```

In questa query vado a vedere il numero di impiegati con età maggiore di 20 per ogni dipartimento. Raggruppando quindi per numero dipartimento.

La condizione di selezione sull'attributo 'Age' ha un grado di selettività bassissimo quindi non è utile fare un indice sull'attributo Age.

Per aumentare le performance delle GROUP BY potrei pensare di utilizzare un indice di tipo B⁺-Tree **clustered**. In questo modo imporrò un ordinamento fisico dei dati, migliorando le performance della GROUP BY.

Se ho già un indice clustered non potrei creare uno su Dept# perché di clustered posso averne solo 1 alla volta. L'applicazione di un indice secondario (unclustered) su Dept# non ci garantisce che le performance della GroupBy vengano migliorate poiché rischierai di fare troppi accessi in lettura per caricare i blocchi dalla memoria secondaria. Leggendo i blocchi in modo scomposto potrei dover ricaricare più volte lo stesso blocco in memoria primaria. Non è vantaggioso utilizzare un indice unclustered su Dept#.

Esempio 7:

```
SELECT Dept#, COUNT(*)  
FROM EMP  
GROUP BY Dept#
```

Quest'interrogazione è uguale a quella dell'esempio 6 ma senza la selezione su Age. In questo caso devo quindi contare il numero di tuple con lo stesso Dept#.

A differenza dell'esempio 6, per fare ciò mi conviene utilizzare un indice di tipo **unclustered** poiché a me

basta contare il numero di puntatori all'interno della foglia del mio B⁺-Tree.

La foglia racchiude tutti i Dept# uguali se metto un indice su Depth#.

In questo caso non ho bisogno di accedere alla tabella perché non devo effettuare nessuna operazione di selezione nella WHERE.

La situazione dell'esempio viene definita come "**indice coprente**".

Esempio 8:

```
SELECT Mgr  
FROM DEPT, EMP  
WHERE DEPT.Dept#=EMP.Dept#
```

tabella EMP per il Dept# che prelevo dalla tabella DEPT.

L'indice su EMP.Depth# mi permette già di vedere se la tupla in questione c'è o non c'è e questo mi basta per eseguire la query.

Anche in questo caso l'indice è coprente. Se uso un indice unclustered

Esempio 9:

```
SELECT AVG(Salary)  
FROM EMP  
WHERE Age = 25  
AND Salary BETWEEN 3000 AND 5000
```

Nell'esempio seguente non ho bisogno di leggere l'informazione contenuta nella tabella EMP. Posso mettere un indice unclustered sull'attributo Depth# della tabella EMP e vedere se esiste un valore nella

In questo caso converrebbe inserire un indice su Age perché è l'attributo più selettivo. Mi converrebbe però avere un indice che su Salary in modo da accedere velocemente a tale attributo durante il calcolo della media.

Conviene dunque inserire un indice composto <Age, Salary> ricordando che:

- L'ordine di definizione degl'indici è importante, metto prima Age perché è più selettivo. Potrò usare quest'indice con Age, Salary oppure solo per accedere ad Age. Non a Salary.
- Un indice composto richiede un tempo d'aggiornamento considerevole che verrà speso durante l'esecuzione della transizione.

Esercizi di progettazione

Hint

Hint suggerisce all'ottimizzatore di Oracle cosa fare. Possiamo ad esempio suggerire di usare un indice o uno specifico algoritmo per effettuare il Join.

Possiamo quindi influenzare l'ottimizzatore poiché il progettista potrebbe conoscere bene la distribuzione dei dati.

Utilizzare Hint comporta la perdita dell'indipendenza dei dati.

Hint vincola l'ottimizzatore a fare delle scelte, il sistema non potrà cambiare il piano d'esecuzione nel caso di modifiche alla distribuzione dei dati.

Gli Hint possono essere inseriti nelle operazioni di SELECT, UPDATE or DELETE.

La notazione per inserire un Hint è

simile alla scrittura di un commento in C solo che dopo il primo * bisognerà inserire + senza inserire spazi.

```
SELECT /*+ Hint1 Hint2 Hint3 */ columnName  
FROM tableName  
WHERE conditions [...]
```

Tipologie di Hint

Possiamo definire diverse tipologie di Hint:

- Approccio di ottimizzazione da utilizzare: OPTIMIZER_MODE
 - ALL_ROWS: ottimizza un blocco per migliorare il *throughput*
 - FIRST_ROWS(n): ottimizza una singola query SQL in modo da ottenere il *fast response*
- Access Path da scegliere. Ovvero se usare un determinato indice oppure fare un full scan
 - FULL(table): accede alla tabella facendo un full scan
 - INDEX(table indexNames): posso usare un indice o più indici per la stessa tabella
 - NO_INDEX(table indexNames): non usare quell'indice o quegl'indici
 - INDEX_COMBINE(table indexNames): usa congiuntamente quell'indice o quegl'indici. Ci serve per imporre di usare bitmap
 - INDEX_FFS(table indexNames): usare fast access, ovvero multiblocco
 - NO_INDEX_FFS(table indexNames): proibisco di usare la lettura multiblocco
- Ordine di Join
 - ORDERED: usa l'ordine delle tabelle nella FROM per fare il Join
 - LEADING(tab_1, tab_2): fornisco l'ordine d'importanza delle tabelle. Se uso gli alias (a, b) devo usarli anche nell'hint.
- Modalità da usare per il join
 - USE_NL(table1, table2, ...): usa Nested Loop per fare join fra queste tabelle
 - NO_USE_NL(...): non usare Nested Loop fra queste tabelle
 - Le stesse funzioni ci sono con MERGE e HASH.

```
SELECT /*+ ORDERED */ *  
FROM emp e, dept d  
WHERE d.deptno = e.deptno  
| 1 | NESTED LOOPS | 50012 | 3125K| 168 (48)| 00:00:03 |  
| 2 | ACCESS FULL | EMP | 50111 | 2202K| 88 (4)| 00:00:02 |  
| 3 | BY INDEX ROWID| DEPT | 1 | 19 | 1 (0)| 00:00:01 |  
* 4 | INDEX UNIQUE | SYS... | 1 | 0 (0)| 00:00:01 |  
■ Emp is the outer table  
■ Dept is the inner table
```

Regole di Buon Senso

- Non devo indicizzare una tabella piccola. Una tabella è piccola se l'ordine di grandezza è al massimo 10^3 . Cardinalità $\leq 10^3$ allora NO indice
 - No indice
- Tabella media/grande se cardinalità $\geq 10^4$
 - valuto se serve indice
- Valutazione selettività predicato
 - Molto selettivo se leggo $\leq \frac{1}{10}$ dei dati. In questo caso posso valutare se usare indice.
Ne seleziono pochi con quel predicato
 - Poco selettivo se leggo $\geq \frac{1}{10}$ dei dati. Non uso indice. Valutare anche altre condizioni tipo Group By). Ne seleziono molti con quel predicato

Procedimento per lo svolgimento degli esercizi d'ottimizzazione

- Analisi delle tabelle (ipotizzando distribuzione uniforme)
 - Osservo le statistiche di:
 - Cardinalità (T tabella)
 - Min/Max su un attributo della tabella
 - # valori distinti (numero di valori distinti)
 - Cardinalità della selezione del predicato p applicato alla tabella T. $\text{Card}(\sigma_p, T)$
Questa indicazione è simile a quella fornita da un istogramma
- Leggere l'interrogazione è capire com'è fatta
- Trasformare l'istruzione SQL in un *query tree* in algebra relazionale. Se abbiamo due selezioni in AND ci conviene separare le due selezioni in due nodi diversi.
- Valutare la cardinalità del risultato intermedio (nodi intermedi) e del nodo finale
- Definisco il piano d'esecuzione senza strutture accessorie
- Posso aggiungere strutture accessorie per accelerare l'esecuzione? Valuto più alternative per migliorare le performance. Per ogni possibile indice valuto:
 - Selettività
 - Utile per GB
 - [coprente]
 - Primario (clustered) / secondario (unclustered)

Ottimizzazione - Esercizio 1

Fast full index scan multiblocco no ordine mi vincola cosa fare dopo. Se non ho ordine devo fare group by hash o sort

Full index scan, leggo indice con letture normali. In questo caso i dati sono già ordinati, posso fare group by no sort

Gli istogrammi servono al sistema per capire se una condizione di selezione è selettiva oppure no. In questo modo possono decidere se usare un indice oppure no.

Quando dominio di variazione ha num bucket inferiore del numero di valori possibili del dominio devo usare height balanced histogram perché ho più valori distinti di quanti sono i bucket. Quando invece il numero di valori distinti è uguale o minore del numero di bucket che ho a disposizione posso usare il frequency histograms che va a mettere in ogni bucket 1 singolo valore del dominio e va a contare quante tuple ci sono per quel valore lì.