ARCHITETTURE DATI

INDICE:

INTRODUZIONE AL CORSO E MODALITA DI ESAME

Riassunto degli argomenti affrontati nei corsi della triennale e introduzione:

Si è partiti con un modello relazionale e a sistema centralizzato (applicazioni stand alone, affidabile, relazionale dove solo noi siamo gli utenti), per poi spostarsi a soluzioni multi utente sempre centralizzate con risoluzioni ai problemi (esempio transizione di bonifici bancari da due conti dove si interrompe la corrente, oppure evitare la doppia registrazione su una piattaforma ad esempio di compera di biglietti del treno).

Il corso si occupa della distribuzione dei dati, questo può essere fatto con un solo modello di rappresentazione dei dati cioè il modello relazionale dando origine ad un Relational Distributed Database Management System (RDBMS) , oppure da architetture più complicate dove si usano più modelli di dati (es. soluzioni Document Base che utilizzano il modello documentale per rappresentare l'informazione e al loro interno hanno delle architetture distribuite dando vita a dei Sistemi Federati Eterogenei).

NELL'ESAME FA ANCHE UN ESERCIZIO SU ARGOMENTI TRA CUI (GUARDA LE ESERCITAZIONI PER CAPIRE CHE TIPI DI ESERCIZI PUO FARE):

- UN PICCOLO GRAFO SU CUI FARE UNA QUERY
- SCHEMA INTEGRATION
- DATA QUALITY

EMAIL PROF DELLE ESERCITAZIONI:

Dopo alcune delle vostre richieste, mando alcuni tipi di esercizi per il tema d'esame

1. Sulla frammentazione (come visto in esercitazione)

2. Sulle interrogazioni e la modellazione in MongoDB e Neo4j come riportato al seguente link

https://docs.google.com/document/d/1qEuzZSNQccu1PJTeD1PnYvir8f-4jQaJ0vXEsC4mhHc/edit?usp=sharing

SI USA SOLO UNA WEBCAM, CI DA UNA PASSWORD AL TESTO DURANTE L'ESAME, CI CHIEDE DI ACCEDERE L'AUDIO E DI INQUADRARE LE MANI E IL CORPO.

Evoluzione delle Data Technologies:

Negli anni 60 la gestione dei dati veniva organizzata con meccanismo centralizzato di dati gerarchici e meccanismi linkati dove il link rappresenta l'indirizzo di memoria dove risiede il dato, quindi per mettere in relazioni i dati era necessario indicare esplicitamente gli indirizzi di memoria dove venivano memorizzati i record.

Negli anni 70 viene introdotto il modello relazionale e iniziano a nascere le soluzioni basate sui DBMS (centralizzate e relazionali). Qui si ha una prima biforcazione dove si inizia ad affrontare il problema di distribuire i dati con parallelismo (es. i casi in cui i dati devono stare vicino a dove vengono utilizzati). In questo contesto nascono anche altre idee architetturali che sono le DB Machine; delle macchine nel cui SO è nativamente pensato un DBMS (es. i sistemi AS400 sono dei server IBM che dentro il SO ha delle primitive di DBMS di fatto è un database annidato dentro il SO).

Da qui si sviluppano nuove tecnologie come i database Distribuite che prendono un DB e lo spezzano in vari DB più piccoli, la tecnologia della Replica che nasce dall'esigenza di copiare i contenuti di un DB verso un altro nodo per garantire migliore accesso e sicurezza, nascono successivamente i DB Federate in cui ogni nodo (ossia DBMS) è indipendente dagli altri per modelli, tecnologie, vendor (es. una rete di database distribuiti relazionali uno con

PostgreSQL uno con MySQL ecc..), nascono anche le architetture Parallele che lavorano sulla scalabilità, infine si ha un'ulteriore evoluzione verso le architetture Linked e Knowledge Graph.

Tipi di organizzazione delle architetture di dati:

Non esiste una soluzione specifica per una architettura dati e si deve scegliere in base alle esigenze come ad esempio scegliere se la soluzione è centralizzata o distribuita, che tipo di modelli dati utilizzare per rappresentare le informazioni e che livello di autonomia hanno i diversi nodi (non necessariamente architetture Master/Slave).

Le situazioni gerarchiche o relazionali e in parte anche le soluzioni distribuite rientrano nella tipologia di organizzazione gerarchica dove c'è un centro che decide e degli attori che eseguono.

Parte delle soluzioni distribuite, le replicate e le federate rientrano nella tipologia di reti di organizzazione (Organizzazioni Networked) che collaborano tra di loro per un unico fine (es. le transazioni bancarie quando si spostano i soldi da un conto di una banca ad un altro conto appartenente ad un'altra banca, questo è possibile perché sono state creati dei circuiti che organizzano delle basi di date federate per scambiare i dati tra delle banche diverse che sono entità autonome che collaborano per effettuare uno scambio di informazioni).

Le soluzioni parallele, linked e knowledge graph sono differenti, di fatto in esse l'organizzazione è autonoma e indipendente.

Tecnologie delle architetture dati Small Data vs Big Data:

Al crescere del volume dei dati che devono essere trattati, le tecnologie del modello relazionale si sono spostate da una soluzione centralizzata verso una soluzione federate (utilizzate sempre per i small data). Al crescere ulteriormente dei dati e con l'avvento dei Big Data si sono sviluppate le architetture parallele, linked e knowledge graph. Le soluzioni relazionali non sono in grado di gestire i Big Data in maniera adeguata.

Soluzioni architetturali possibili:

Le architetture centralizzate (shared nothing) sono composte da un'unica applicazione che contatta un unico DBMS su unico DB. Esistono tuttavia diverse soluzioni come ad esempio le soluzioni Shared Disk dove tante applicazioni accedono ad un unico RDBMS che però a sua volta accede ad una grande quantità di dischi organizzati sotto forma di SAN (Storage Area Network) cioè degli array di dischi che contengono le stesse informazioni per consentire al RDBMS di bilanciare il carico scaricando i dati da un nodo piuttosto che da un altro (ad oggi questo tipo di soluzione viene adottato con successo solo da Oracle con la architettura RAC). Per le architetture estremamente parallele l'approccio cambia e si applica una architettura Shared Nothing su diversi nodi indipendenti tra di loro, in quanto ogni nodo ha le sue funzioni e i suoi dati e attraverso una tecnologia di collaborazione e di gestione dei dati per garantire il parallelismo che comporta una enorme scalabilità dei dati e alla possibilità di gestire anche volume, velocità e a livello applicativo la varietà (queste sono le 3 V fondamentali nella definizione dei Big Data).

Approcci delle architetture dati:

Ci sono due modi di intendere le architetture di dati gli approcci OLTP (Online Transaction Processing) detti transazionali e gli approcci OLAP (Online Analytical Processing) detti analitici. Gli OLTP sono processi online in cui vengono fatte delle transazioni; si tratta sostanzialmente di tante persone che fanno delle richieste semplici e ripetitive su uno schema di basi di dati (relazionale o non relazionale) es. voglio comprare un telefono e sposto i soldi dal mio conto corrente al conto corrente della baca del venditore, dove non devono capitare transazioni ambigue se si presentano dei problemi di qualsiasi tipo. Gli OLAP sono processi che prendono dati diversi e li mettono insieme per avere delle informazioni aggiuntive per la risoluzione di problemi complessi. Alcuni dei dati presi sono: l'insieme dei miei clienti (CRM), l'insieme dei pagamenti, i sistemi ERP gestionali ecc... Questi dati vengono poi immagazzinati in un Data Warehouse per poi compiere delle azioni di analisi diverse e complesse come il Reporting e il Data Mining.

RICHIAMO AI SISTEMI CENTRALIZZATI

DataBase Management System (DBMS):

Il DBMS è un componente software in grado di gestire collezioni di dati che possiedono le seguenti caratteristiche:

- GRANDI: hanno dimensioni molto maggiori della memoria centrale dei sistemi di calcolo utilizzati
- PERSISTENTI: il dato memorizzato ha un lungo periodo di vita che è indipendente dalle singole esecuzioni dei programmi che lo utilizzano
- CONDIVISE: i dati devono essere accessibili a diversi utenti/applicazioni sia in lettura che in scrittura
- AFFIDABILI: per essere affidabili devono essere resistenti rispetto ai guasti e affidabile rispetto alla sicurezza dei dati sia in lettura che in scrittura

Il DBMS è strutturato da uno o più sistemi di storage detti Basi di dati (DB) che memorizzano i dati, un componente logico (DBMS) che è un componente software, infine ci sono le applicazioni che consentono di interagire con il DBMS.

Sui database gli utenti effettuano delle interrogazioni per ottenere dei dati, degli aggiornamenti dei dati o delle transazioni.

L'amministratore della base dati (DBA) che tramite una serie di linee di comandi o interfacce grafiche si occupa della sicurezza, del controllo degli accessi e dell'ottimizzazione delle interrogazioni ecc..

DBMS Centralizzato:

L'architettura dati di un DBMS centralizzato definito da ANSI/SPARC è su tre livelli: nel primo livello abbiamo gli schemi esterni ossia delle porzioni del database messe a disposizione per le varie applicazioni, al secondo abbiamo uno schema concettuale o logico relazionale, infine esiste uno schema fisico. Lo schema fisico comprende anche la scelta di come un DBMS implementa le tabelle (un unico tabellone gigante o più tabelle, il formato in cui viene salvato ecc…)

Le principali caratteristiche di un DBMS centralizzato sono dunque:

- Unico schema logico che utilizza un'unica semantica
- Unica base di dati ovvero un insieme di record interrogati e aggiornati dagli utenti
- Unico schema fisico collegato allo schema logico che rappresenta fisicamente i dati

Da questo si può dire che non c'è nessuna forma di eterogeneità in quanto tutte le componenti sono su una singola macchina e non vi è nessuna distribuzione dei dati.

Altre caratteristiche sono:

- Unico linguaggio di interrogazione, ovvero usando un solo DBMS che utilizza una versione particolare del linguaggio SQL
- Unico sistema di gestione dell'accesso degli aggiornamenti, delle interrogazioni e dei guasti
- Unica modalità di ripristino a fronte di guasti
- Unico amministratore dei dati

Approfondimento delle caratteristiche dei DBMS:

Data la grandezza e persistenza dei dati delle basi di dati è necessario che la gestione dei dati avvenga in una memoria secondaria (HDD/SSD) e in maniera sofisticata ed efficiente. Viene introdotto dunque il concetto di Buffer (che funziona in maniera molto simile ai buffer dei SO) che è una componente software che trasporta in memoria centrale i dati presenti nella memoria secondaria tramite una logica di vicinanza (se accedo ad una riga probabilmente vorrò accedere anche ad una riga sua vicina). Mentre nei SO che quando si va a modificare un dato tale dato rimane modificato all'interno del Buffer finché il SO non decide di scaricarlo nella memoria secondaria secondo delle sue

logiche, ciò non è possibile fare all'interno dei DBMS perché essi devono garantire delle caratteristiche che consentono il funzionamento del buffer in fase di lettura in maniera uguale al buffer dei SO, ma in maniera differente per quanto riguarda la scrittura dei dati.

Una base di dati è una risorsa integrata e condivisa fra le varie applicazioni e questo pone problemi di sicurezza, di accesso sicuro/ autorizzazione (non tutti possono accedere alla lettura e/o scrittura di determinati dati) e di concorrenza per le attività multi-utente su dati condivisi. Un esempio semplice di concorrenza è quando due persone vogliono prendere un biglietto aereo e entrambi decidono di comperare il biglietto per lo stesso posto in contemporanea, in questo caso l'ultimo ad avere "scritto" quindi prenotato e pagato sarà il possessore del biglietto. Queste situazioni possono essere evitate intuitivamente evitando di incombere in situazioni di concorrenza degli accessi attraverso al serializzazione (una sola operazione alla volta), ma ciò è impossibile in un'ottica web in quanto l'efficienza ne sarebbe enormemente penalizzata. Il controllo della concorrenza permette un ragionevole compromesso introducendo ad esempio dei controlli di consistenza (es. una volta prenotato il posto non è più possibile prenotare per lo stesso posto).

Le basi di dati vengono interrogate dagli utenti che vedono il modello logico relazionale. I dati attraverso le interrogazioni vengono prelevati dalla memoria secondaria, ma in base al modo in cui tale operazione è svolta questa può richiedere più o meno tempo. È necessario dunque eseguire una ottimizzazione delle interrogazioni in quanto se eseguo una operazione di tipo SELECT * FROM molto spesso ottengo come risultato una versione ridotta della tabella di partenza, ma se effettuo una interrogazione con un JOIN il processo si complica in quanto vengono unite più tabelle per crearne delle nuove e dunque la quantità di tempo richiesto per gestire tale operazione se non ottimizzata sarà incredibilmente maggiore rispetto al tempo necessario per effettuare una interrogazione con un semplice SELECT * FROM.

Le basi di dati devono essere affidabili ossia che anche in presenza di malfunzionamenti il sistema funzioni correttamente (es. salta la corrente mentre stai pagando la retta dell'università e poi a te risulta pagata la retta mentre all'università no). Le transazioni nelle basi di dati DEVONO essere atomiche (o risulta pagato per entrambi o non risulta pagato per nessuno) e definitive (una volta che è conclusa una transazione il risultato non viene mai dimenticato).

Per garantire tutte le caratteristiche precedenti sono state identificate un insieme di tecnologie per generare un DBMS chiamato architettura funzionale di un DBMS che vale anche nelle architetture distribuite in cui ci sono i seguenti componenti fondamentali:

- Query compiler: che prende le query scritte in SQL e le traduce utilizzando un compilatore
- Gestore di interrogazioni e aggiornamenti: si occupa di trasformare le richieste espresse in SQL in forma di comandi elementari e fare una serie di operazioni di ottimizzazione
- Gestore dei metodi d'accesso: trasforma in comandi di accesso alla memoria secondaria
- Gestore del buffer: tramite il buffer prende i dati dalla memoria secondaria e li porta nella memoria principale
- Gestore delle transazioni:
- Gestore della concorrenza: (es. gestisce le situazioni di prenotazione contemporanea)
- Gestore della affidabilità: (es. controlla che il dato salvato non si perderà mai)

In un ambiente distribuito le componenti di query compiler, DDL Compiler, gestore di interrogazioni e aggiornamenti, gestore delle transazioni e gestore della concorrenza restano invariati e le altre componenti vengono profondamente cambiate in quanto cambia il modo di distribuire i dati.

Come ottimizzare le interrogazioni:

Il primo passo per ottimizzare le interrogazioni è prendere la query e verificare se essa è sintatticamente corretta, se i nomi degli attributi sono presenti all'interno del mio schema (fase di scanning, Parsing). Per farlo nei DBMS di tipo relazionale esiste un database chiamato Data Catalog che contiene le informazioni sugli altri database (tutte le tabelle e i loro attributi), interrogando il Data Catalog dunque riesco a capire se esistono le tabelle/attributi che sto cercando

di interrogare con la query. Successivamente si costruisce un query tree astratto utilizzando l'algebra relazionale che verrà trasformato in un query plan logico contenete operazioni di algebra che ottimizzano rispetto al costo le operazioni (non faccio un SELECT dopo un JOIN se non necessario perché se mi basterebbe fare solo il SELECT fin da subito ci metterei la metà del tempo). Nel query plan logico la query viene rappresentata come un albero nel quale le foglie corrispondono alle tabelle e i nodi intermedi corrispondono alle operazioni come il JOIN e SELECT.

Esempio di esercizio di query e ottimizzazione della stessa:

- Employee [Ename, lname, ssn, bdate, address]
- Department [Dname, dnumber,mgrssn]
- Project [Pname, pnumber, plocation, dnum]
- Works-on [Essn, pno, hours]

Date le seguenti tabella trovare i Progetti localizzati a Stafford e il loro Manager con nome, indirizzo e data di nascita.

La query di soluzione è la seguente:

SELECT pnumber, dnum, lname, bdate, address

FROM Employee E, Department D, Project P

WHERE P.plocation = 'Stafford' AND P.dnum = D.dnumber AND D.mgrssn = E.snn

Le condizioni di JOIN sono messe all'interno della clausola WHERE. Il modo in cui è strutturato il FROM fa si che le tre tabelle vengano unite attraverso il prodotto cartesiano.

Eseguendo passo passo l'interrogazione avverranno i seguenti passaggi:

- Prendo la tabella P e faccio la selezione su Stafford (plocation ="Stafford")
- Il risultato lo metto in JOIN con la tabella D specificando che il numero di dipartimento deve coincidere (dnum=dnumber)
- Il risultato lo metto in JOIN con la tabella E specificando che l'ssn deve coincidere (mgrssn=ssn)
- Vado a prendere la proiezione degli attributi richiesti (pnumber, dnum, lname, bdate, address)

L'ottimizzatore dei DBMS identifica alcuni dati inutilizzati che ci portiamo dietro come ad esempio il Dname che quando facciamo il JOIN (dnum=dnumber) ci portiamo dietro ma non utilizziamo mai. Dopo avere identificato questi dati inutilizzati va a modificare il query plan logico semplificando la query selezionando tra tutti i piani logici che portano allo stesso risultato quello che ottimizza il costo di esecuzione (es. in generale conviene anticipare le SELECT rispetto ai JOIN perché così facendo i JOIN operano su tabelle di dimensioni minori rispetto alle originali).

Supponiamo ora che tutti i progetti vengano eseguiti a Stafford; a questo punto la selezione plocation="Stafford" è inutile. Il DBMS per conoscere questa informazione utilizza un altro database che si chiama Statistics che contiene per ogni attributo alcune informazioni (numerosità, minimo, massimo, ecc.). Successivamente attraverso il query plan fisico le tabelle logiche vengono trasformate in strutture fisiche e metodi di accesso alla memoria per potere andare a reperire le informazioni necessarie dalla memoria esterna. L'ottimizzazione del query plan fisico si occupa di ottimizzare la trasformazione di queste tabelle logiche riducendone il costo attraverso l'introduzione di approssimazioni in basi a euristiche (es. utilizzando degli alberi di costi).

Transazioni:

Le transazioni sono operazioni di accesso in lettura e scrittura sulla base di dati e godono di alcune proprietà molto importanti che garantiscono la corretta esecuzione in un ambiente concorrente e non affidabile (cioè per i modelli relazionali).

Le transazioni iniziano con “begin-transaction” o “start transaction” in SQL e si concludono con “end-transaction” oppure senza essere esplicitato in SQL. Tra le varie operazioni eseguibili all’interno di una transazione DEVE essere eseguito una e una sola volta uno dei seguenti comandi:

- Commit work (per terminare correttamente la transazione)
- Rollback work (per abortire la transazione)

Un sistema transazionale OLTP è in grado di definire ed eseguire transazioni per un numero alto di applicazioni concorrenti.

Segue un esempio di transazione:

```
start transaction;                    (si fa partire la transazione)
update ContoCorrente                  (si fanno le varie operazioni necessarie)
set Saldo = Saldo +10 where
NumConto = 12202;
update ContoCorrente                  (tolgo i soldi da un conto corrente e li aggiungo ad un altro)
set Saldo = Saldo -10 where
NumConto= 42177;
select Saldo into A                   (verifico se il conto che riceve il denaro ha saldo positivo)
        from ContoCorrente
        where NumConto = 42177;
if (A>=0) then commit work            (se il saldo è positivo la transazione è avvenuta e si fa commit)
        else rollback work;           (se il saldo è negativo c’è stato qualche problema e si fa rollback)
```

Nel codice appena riportato si può notare come la posizione del commit/rollback non sia importante al fine della programmazione, in quanto tutto ciò che è posto prima del commit/rollback verrà salvato/dimenticato una volta giunto a questo comando. Proprio per questo è necessario che all’interno di una transazione una istanza dei due comandi DEVE essere utilizzata. Non sono ammesse soluzioni intermedie; tutto quello che è stato modificato in caso di commit deve essere salvato per sempre, tutto quello che è stato modificato in caso di rollback deve essere dimenticato.

Le proprietà di cui godono le transazioni sono definite ACID che garantiscono Atomicità, Consistenza, Isolamento e Durata (persistenza).

L’atomicità è una proprietà che dice che una transazione è una unità atomica di elaborazione, ossia viene garantito l’esito della transazione sulla base di dati che non resterà mai in uno stato intermedio anche in presenza di errori /guasti. Questa proprietà viene garantita dal gestore della affidabilità.

La consistenza fa si che la transazione rispetti i vincoli di integrità, ossia se lo stato iniziale del database prima della transazione è corretto allora lo deve essere anche nello stato finale dunque dopo che la transazione è stata effettuata (es.nel caso in cui ci siano delle violazioni all’interno di una transazione questa deve essere annullata tramite un comando di rollback). Questa proprietà viene garantita dal gestore delle transazioni.

L'isolamento (o concorrenza) fa si che indipendentemente da quante operazioni in un determinato momento vogliono leggere/scrivere su uno stesso dato è garantito il controllo di concorrenza ai dati. L'esecuzione concorrente di una collezione di transazioni deve dunque produrre un risultato che si potrebbe ottenere con una esecuzione sequenziale. Questo garantisce non solo che ad esempio qualcuno compri il mio stesso biglietto aereo, ma evita anche degli effetti a cascata (tramite il rollback a cascata) come ad esempio se si versa una quantità di denaro non disponibile su un conto e la stessa quantità di denaro venga prelevata dal possessore del conto. Questa proprietà viene garantita dal gestore della concorrenza.

La durabilità garantisce che il dato venga memorizzato per sempre al concludersi correttamente di una transazione con il comando commit. Questo implica la presenza di una componente nel DBMS che garantisca l'affidabilità (Recovery Manager). Questa proprietà viene garantita dal gestore della affidabilità.

Gestore della concorrenza:

Il gestore della concorrenza garantisce la possibilità di eseguire quasi in parallelo più operazioni. Per spiegare come bisogna introdurre il concetto di Schedule che è un insieme di transazioni che è generato dall'ordine di arrivo delle transazioni. Uno schedule è detto seriale se una transazione inizia dopo che la transazione precedente si è conclusa del tutto. Esempio di schedule seriale: T1 trova posto, T1 alloca posto, T2 trova posto, T2 alloca posto. Esempio di schedule non seriale: T1 trova posto, T2 trova posto, T1 alloca posto, T2 alloca posto. Per evitare problemi nel caso in cui ci siano schedule non seriali il gestore della concorrenza deve garantire che l'insieme di transazioni concorrenti produce lo stesso risultato che produrrebbe una esecuzione sequenziale (seriale) delle stesse transazioni; questo concetto viene definito serializzabilità e gli schedule che godono di tale proprietà sono definiti schedule serializzabili.

Nei DBMS in realtà non è implementato un algoritmo per controllare la concorrenza ma un meccanismo tale per cui vengono prodotti degli schedule serializzabili detto Protocollo 2PL. Tale protocollo introduce le operazioni di utilizzo e rilascio di una risorsa in quanto deve essere controllato l'accesso ad una risorsa:

- Lock (esclusivo) che chiede la risorsa in modo esclusivo
- Unlock che rilascia la risorsa

Quando viene richiesta una risorsa il gestore della concorrenza deve autorizzare l'accesso esclusivo a tale risorsa altrimenti viene vietato l'accesso (nel caso in cui sia già stata assegnata). Le transazioni vengono dunque trasformate aggiungendo i comandi di lock e vengono create le tabelle di lock che memorizzano le assegnazioni delle risorse.

Oltre alle due regole di acquisizione e rilascio delle risorse il protocollo 2PL possiede una terza regola che stabilisce che gli unlock debbano essere eseguiti solo quando tutte le risorse disponibili sono state prese. Questo vuol dire che una transazione al suo avvio richiede tutte le risorse di cui necessita, esegue le sue operazioni, infine rilascia le risorse dopo l'esecuzione del comando commit.

SISTEMI DISTRIBUITI

ARCHITETTURE DI BASI DI DATI DISTRIBUITE ETEROGENEE AUTONOME

Le architetture centralizzate prevedono diverse applicazioni che accedono ad un unico DBMS che gestisce uno o più dischi. Sul DBMS le applicazioni inviano in maniera parallela richieste di interrogazioni, aggiornamenti e transazioni. L'architettura di un sistema DB distribuito invece dispone di tante basi di dati locali, diverse applicazioni su ogni nodo di elaborazione e gli utenti che come sempre si configurano con le applicazioni. Questo tipo di architettura viene definita come Shared Nothing in quanto i DBMS di ogni singola macchina sono autonomi e collaborano tra di loro.

Vediamo ora un esempio di passaggio da architettura centralizzata a distribuita. Un'azienda che si occupa di progettazione e vendita ha tre sedi: una a Milano, una a Roma e una a Padova. La sede centrale di Milano è l'unica che dispone di un DB, quindi tutte le informazioni dei contratti, progetti, anagrafica ecc. di tutte le sedi risiedono nel DB centrale di Milano. In questa situazione iniziale il nodo di Milano dispone delle capacità di Data Processing e di Application Processing, mentre i nodi di Roma e Padova possiedono solo la capacità di Application Processing.

Per esigenze di efficienza e/o affidabilità può essere utile spostare i dati secondo la logica di spostare i dati vicino a dove servono. Quindi si può frammentare la base di dati in frammenti scelti sulla base degli accessi dai vari nodi (la sede di Roma accede ai dati delle vendite di Roma molto più spesso di quanto non lo faccia la sede di Padova). Facendo ciò si da vita ad un database distribuito dove ad esempio nel DB di Roma saranno presenti i Progettisti di Roma, i Dipartimenti di Roma e i Contratti di Roma, mentre nel DB di Milano ci saranno i Progettisti di Milano + Roma, i Dipartimenti di Milano e i Contratti di Milano. Questo consente alle sedi di Roma e Padova di acquisire la capacità di Data Processing sui dati appartenenti ai loro DB. In alcuni casi come ad esempio con i Progettisti può essere necessaria effettuare una operazione di replica dove duplico la stessa informazione su più DB.

La distribuzione:

Nella realizzazione di un sistema il primo problema che si pone è la distribuzione, ovvero decidere in che modo distribuire applicazioni, dati, ecc.. In un sistema distribuito a tal riguardo si verifica sempre almeno una delle due seguenti condizioni:

- Le varie applicazioni collaborano tra di loro e risiedono su più nodi elaborativi (elaborazione distribuita)
- L'archivio informativo è distribuito su più nodi (base di dati distributiva)

Nella distribuzione ad esempio nel caso di fusione o acquisizione tra banche è necessario tenere conto anche dei sistemi informativi già esistenti per entrambi le banche e dunque valutare come unire le logiche applicative e unire i sistemi di archiviazione. Molto spesso le banche decidono di mantenere uno tra i due modelli e di cestinare l'altro. La parte più difficile in questi processi è quella del merging dei dati in quanto gli stessi dati sono rappresentati in maniera differente dai sistemi informativi di banche diverse e questo potrebbe causare enormi problemi e anche l'eventuale perdita di dati.

Criteri di classificazione dei database distribuiti:

Un DBMS Distribuito Eterogeneo Autonomo è in generale una federazione di DBMS che collaborano nel fornire servizi di accesso ai dati con livelli di trasparenza definiti. Con Eterogeneo si intende che è presente una diversa rappresentazione dei dati che descrivono il DBMS. Autonomo identifica chi decide se creare o meno una tabella ecc., quindi in questo caso ogni nodo ha autonomia nel gestire i dati. Trasparenza vuol dire che chi scrive l'interrogazione in un sistema distribuito in base ai livelli di trasparenza può non essere consapevole che il DB sia distribuito eterogeneo autonomo. Con trasparenza si intende la proprietà generale di nascondere le diversità tra basi di dati nei nodi del sistema, negli aspetti di distribuzione, eterogeneità e autonomia. Ad esempio se un utente scrive un'interrogazione

per ottenere dei dati da un sistema distribuito, in base ai livelli di trasparenza, può non essere a conoscenza che le tabelle da cui prende i dati risiedono in nodi diversi, oppure, al contrario, può sapere che i dati richiesti risiedono in parte in una tabella appartenente ad un nodo e in parte all'interno di un'altra tabella di un altro nodo con magari un modello di rappresentazione di dati differente.

L'esigenza di integrare a posteriori sistemi di tipo federato emerge in molti casi come nell'esempio precedente della fusione tra banche. In questi casi è possibile dividere i livelli di federazione su tre caratteristiche ortogonali (indipendenti tra loro): Autonomia, Distribuzione, Eterogeneità.

L'autonomia fa riferimento a quanto è indipendente un nodo rispetto ad un altro e ci sono diverse forme di autonomia:

- Di progetto: livello massimo di indipendenza dove ogni nodo adotta un proprio modello dei dati e sistema di gestione delle transazioni
- Di condivisione: i nodi, pur condividendo uno schema di dati comune, decidono quale porzione di dati condividere con altri nodi
- Di esecuzione: le transazioni possono essere eseguite su nodi diversi i quali decidono in che modo eseguirle

Vi sono inoltre diversi tipi di autonomia:

- DBMS strettamente integrati: sistemi con i dati logicamente centralizzati e privi di autonomia, quindi con un unico DBA (admin) che organizza tutto e i DBA locali eseguono le direttive del main DBA
- Sistemi Semi-autonomi: una parte dei dati dello schema è condiviso e ogni DBA è autonomo ma deve partecipare a transazioni globali
- Sistemi Totalmente autonomi (Peer to Peer): sistemi in cui ogni DBMS è autonomo e collabora con altri DBMS in maniera da pari a pari

La distribuzione fa riferimento alla distribuzione dei dati e si può distinguere in tre livelli:

- Distribuzione client/server: i dati sono concentrati su un nodo (server), mentre l'ambiente applicativo e la presentazione sono gestite dai client
- Distribuzione peer-to-peer: non ci sono distinzioni tra client e server in quanto tutti i nodi del sistema hanno identiche funzionalità DBMS
- Nessuna distribuzione: nessun dato viene distribuito

L'eterogeneità può riguardare diversi aspetti tra cui:

- Modello dei dati: posso rappresentare le informazioni in formati diversi come ad esempio XML oppure relazionale oppure object oriented
- Linguaggio di query: a parità di modello relazionale esistono diversi linguaggi o dialetti di SQL che possono essere utilizzati
- Gestione delle transazioni: esistono diversi protocolli per la gestione delle transazioni forniti dai vendor di DBMS
- Schema concettuale e schema logico: è possibile che i diversi nodi abbiamo uno schema concettuale e logico differente tra di loro

Tipologie di sistemi di database DEA:

Esistono diverse tipologie di database che si basano sul loro livello di distribuzione, autonomia ed eterogeneità. Il DBMS Distribuito Omogeneo (DDBMS) è un database improntato alla distribuzione ma che manca di autonomia (è un unico elemento) ed eterogeneità; può essere identificato come una rete di nodi dello stesso vendor. Il DBMS Eterogeneo Logicamente Integrato è un sistema non autonomo, non distribuiti, ma con delle eterogeneità. Il DBMS Distribuito Eterogeneo sono dei database che nonostante siano distribuiti e abbiano delle eterogeneità di schema mancano di autonomia. I DBMS Federati Distribuiti hanno un'alta autonomia e distribuzione in quanto è presente la federazione degli elementi, ma manca di eterogeneità nei concetti degli schemi.

I Sistemi Distribuiti Federati Eterogenei sono sistemi che dispongono sia di distribuzione che di eterogeneità e di autonomia (classico esempio di quando due compagnie vogliono condividere le informazioni tra di loro). Infine ci sono i multidatabase che possono essere più o meno distribuiti ed eterogenei, ma che sono totalmente autonomi.

DBMS Distribuito Omogeneo (DDBMS):

In questo tipo di DBMS abbiamo due architetture di riferimento: l'architettura dati e l'architettura funzionale (ovvero l'insieme delle tecnologie a supporto dell'architettura dati). A differenza del DBMS Centralizzato che aveva una architettura di dati che collegava lo schema logico direttamente allo schema fisico, nel DDBMS seppur mantenendo lo stesso schema (in quanto non si ha l'eterogeneità) tra lo schema logico e lo schema fisico vengono inseriti dei nuovi elementi. Non si avrà più un unico schema logico ma uno schema logico globale che è associato a diversi schemi logici locali che non sono altro che delle sue, infatti si ha una organizzazione LAV (Local As View). In questa organizzazione se si vuole interrogare il sistema, si interroga lo schema logico globale che attraverso una tecnologia instrada l'interrogazione verso lo schema logico locale in grado di risolvere tale richiesta. Il LAV viene progettato prima degli schemi a livello locale in quanto fornisce una idea generale di schema logico.

Dato che è necessario pensare prima allo schema globale e successivamente a quello locale, cambia anche il processo di progettazione in quanto non può più essere utilizzato il classico l'approccio top-down, in quanto deve essere aggiunta una fase tra la progettazione concettuale e la progettazione logica che si occupi della distribuzione dei dati, ovvero la progettazione della distribuzione. Una volta terminata la progettazione della distribuzione si potrà dunque procedere con la progettazione logica locale, che traduce dallo schema logico globale soltanto alcuni concetti e la progettazione fisica locale.

Nei DDBMS si presenta il problema della portabilità ovvero la capacità di eseguire le stesse interrogazioni su ambienti diversi (macchine con versioni di SQL diverse tra loro). Per garantire l'interoperabilità nei DBMS DEA sono stati introdotti alcune tecnologie tra cui il middleware ODBC Database Connectivity che consente la connettività tra database con dati diversi. Esistono anche dei protocolli come l'X-Open Distributed Transaction Processing (DTP) che consente di eseguire delle transazioni con logica diversa.

FRAMMENTAZIONE E REPLICAZIONE

Nei DDBMS tre tipi di architetture nelle quali è possibile distribuire i dati:

- Le architetture Shared Everything dove non c'è alcuna distribuzione di dati in quanto il DB e il Disco di memoria sono nella stessa macchina
- Le architetture Shared Disk dove diverse componenti software agiscono sulla stessa SAN (Storage Area Network) che prevede la presenza di più dischi in assetto raid
- Le architetture Shared Nothing in cui vi sono più DB e dischi che vivono indipendentemente dagli altri. Questa soluzione garantisce una scalabilità detta orizzontale che consente di espandere indefinitamente il numero di DB e dischi associati.

Proprietà generali di un DDBMS:

La località è un principio fondamentale che si occupa di spostare i dati vicino a dove servono. La partizione dei dati dunque corrisponde spesso ad una partizione naturale come ad esempio per le organizzazioni con sedi diverse i dati degli operatori di una sede risiederanno vicino alla sede stessa, nonostante siano comunque globalmente raggiungibili.

La flessibilità è legata alla dinamicità nella distribuzione dei dati, ad esempio si può scegliere in base alla necessità se trasportare un'intera tabella verso un particolare nodo applicativo oppure spostare un sottoinsieme della tabella oppure ancora di replicare i dati e distribuirli.

La resistenza ai guasti è di particolare importanza in questo tipo di architettura che prevede la presenza di maggiori componenti all'interno della rete comportando così un maggiore rischio di guasti. In tal caso la replica dei dati all'interno di diverse componenti garantisce la ridondanza dei dati e quindi una maggiore resistenza ai guasti.

Le prestazioni nei DDBMS sono notevolmente maggiori soprattutto in presenza di una grande quantità di dati che viene distribuita su nodi diversi. Ogni nodo dovrà gestire un DB di dimensioni ridotte e potrà gestire ed ottimizzare in maniera indipendente e più semplicemente le applicazioni locali.

Funzionalità specifiche dei DDBMS rispetto ai DBMS centralizzati:

In una architettura shared nothing si hanno un certo numero di server collegati tra di loro da una rete. Ogni server ha una buona capacità di gestire le applicazioni in modo indipendente. Le richieste che raggiungono il server dalla rete possono essere delle query di richiesta di dati da parte di un utente, oppure dei pezzi di transazione distribuita che coinvolge più server. Nel caso delle query di interrogazione arriva una richiesta dalle applicazioni e i risultati provengono da un solo server, mentre nel caso in cui si tratti di transazioni l'accesso in scrittura chiede anche dei dati di controllo per il coordinamento. Da queste considerazioni possono essere ricavate alcune funzionalità specifiche dei DDBMS:

- Trasmissione: i nodi si scambiano frammenti di DB (dati) oppure dati di controllo rispettivamente nel caso di queries e transazioni
- Frammentazione/replicazione/trasparenza: per l'allocazione dei dati ottimizzata e per raggiungere i dati in maniera efficiente attraverso dei livelli di trasparenza
- Query processor: attraverso l'interrogazione di uno schema logico globale definisce una strategia di accesso agli schemi logici locali per la risoluzione delle query
- Controllo di concorrenza: fondamentale per le operazioni di scrittura
- Strategie di recovery di singoli nodi e gestione di guasti globali

Frammentazione, replicazione e trasparenza:

La frammentazione è la possibilità di allocare porzioni del DB su nodi diversi; ad esempio avendo la tabella R la spezzo in due tabelle di dimensioni inferiori R1 e R2 e le assegno a due DB differenti. Questo processo può essere effettuato su in due modalità diverse:

- Frammentazione orizzontale: divido la tabella in righe che vengono allocate a nodi differenti. In questo caso vengono generate delle tabelle più piccole che mantengono lo stesso schema. Le nuove tabelle vengono generate attraverso un operazione di selezione su un attributo (es. Tabella1 gli impiegati con numero identificativo <=3, Tabella 2 gli impiegati con numero identificativo >3)
- Frammentazione verticale: divido la tabella in colonne. In questo caso è necessario garantire un principio fondamentale affinché la frammentazione verticale avvenga con successo ossia condividere la colonna contenente la chiave primaria della tabella in ogni frammentazione della stessa affinché si possa ricomporre la tabella di partenza tramite una operazione di JOIN. Le tabelle di questo tipo vengono generate attraverso un operazione di proiezione (es. selezioni le colonne della tabella senza condizioni where)

Le frammentazioni devono rispettare delle regole di correttezza che sono:

- Completezza: ogni record della relazione originale deve essere ritrovato in almeno uno dei frammenti
- Ricostruibilità: è possibile ricostruire la relazione R di partenza senza perdite di informazioni quando si uniscono i frammenti
- Disgiunzione/ Replicazione: idealmente ogni record è rappresentato una sola volta in uno solo dei frammenti, in alternativa se è necessario avere più copie dello stesso record in diversi frammenti si ha la replicazione

Ogni frammento è allocato in generale su un nodo diverso, quindi serve uno schema di allocazione (mapping) che dichiara rispetto allo schema logico locale dove sono stati inseriti i frammenti e in particolare su quale nodo è stato

inserito un particolare frammento. Per fare ciò viene introdotto un catalogo che è una tabella relazionale che indica il luogo in cui risiede ogni frammento.

Un possibile problema che si potrebbe presentare quando si compiono azioni all'interno di frammentazioni è il seguente: supponiamo di avere due Frammenti del tipo E1(EmpNum, NameEmp), E2(EmpNum, Tax, Salary) dove EmpNum è chiave primaria del frammento di origine, se volessi effettuare una operazione di Update sulla variabile EmpNum mettendo la condizione WHERE Tax=20 allora la query andrà ad agire solo sulla tabella E2 modificandone il valore nella colonna EmpNum facendo così perdere il collegamento tra le chiavi primarie delle tabelle E1 ed E2 rendendo così impossibile l'operazione di ricostruzione. Per evitare questo è necessario imporre la propagazione delle modifiche delle chiavi primarie anche negli altri frammenti.

La replicazione è la possibilità di replicare un frammento di una tabella o la tabella intera su due nodi differenti, in questo modo viene reso possibile l'utilizzo degli stessi dati da applicazioni diverse. Se gli accessi sono solo in lettura non vi sono problemi ma se ci sono casi di scrittura potrebbero presentarsi dei problemi, infatti in tal caso è necessario predisporre una gestione delle transazioni e updates di copie multiple degli stessi dati che devono essere tutte aggiornate.

La trasparenza da la possibilità alle applicazioni di accedere ai dati senza che esse sappiano dove questi siano allocati tramite l'utilizzo di query. Viene dunque separata la logica applicativa dalla logica dei dati, ma per fare ciò è necessario introdurre uno strato software in grado di gestire la traduzione dallo schema unico globale ai sottoschemi locali.

Esistono due tipi di trasparenza nei DBMS:

- Trasparenza logica: dove vi è indipendenza dell'applicazione dalle modifiche dello schema logico
- Trasparenza fisica: dove vi è indipendenza dell'applicazione dalle modifiche dello schema fisico

I livelli di trasparenza nei DBMS sono:

- Trasparenza di Frammentazione: l'applicazione è scritta in SQL standard e ignora l'esistenza dei frammenti
- Trasparenza di Replicazione (Allocazione): l'applicazione è a conoscenza dell'esistenza dei frammenti ma non ne conosce la allocazione sui nodi
- Trasparenza di Linguaggio: l'applicazione deve specificare sia i frammenti che il loro nodo e in alcuni casi è possibile che alcuni nodi forniscano interfacce scritte non in standard SQL

TECNOLOGIE PER LA DISTRIBUZIONE DATI

I DDBMS hanno diversi modi per processare le loro operazioni più importanti. Le query sono le operazioni più importanti che si possono effettuare dentro un DB, queste possono essere di sola lettura (SELECT * FROM) oppure di scrittura e in base a questa distinzione vengono adottate strategie differenti.

Query processing nei DDBMS in fase di lettura:

Nel caso delle interrogazioni di sola lettura non vi sono problemi di concorrenza nel caso in cui diversi utenti tentino di accedere alla medesima risorsa. Il processo di lettura si articola nei seguenti passi:

- L'utente fa un'interrogazione nello schema globale attraverso una query
- Il DBMS riconosce che rispetto alla richiesta dell'utente vi sono dei dati frammentati
- Il DBMS decompone la query secondo una localizzazione specifica dei frammenti ricercati
- Il DBMS esegue una ottimizzazione globale sull'esecuzione della query
- Il gestore delle interrogazioni manda ai singoli nodi il frammento di query che deve svolgere
- Come ultimo passo si esegue una ottimizzazione locale per eseguire la query appena ricevuta

Questi passi possono essere riassunti in 4 fasi:

1- Query decomposition: fase in cui il DBMS opera sullo schema logico globale non tenendo conto della distribuzione e cerca di costruire un albero di interrogazione centralizzata attraverso l'algebra relazionale. L'albero così generato tuttavia non è ottimizzato rispetto ai costi di comunicazione.

2- Data localization: si considera la distribuzione dei frammenti e ci si rende conto dell'esistenza di tabelle frammentate su più nodi attraverso l'utilizzo dello schema dei Fragment. Ottimizza le operazioni rispetto alla frammentazione con delle tecniche di riduzione restituendo in output una query che opera in modo efficiente sui frammenti (es. se ho una frammentazione orizzontale dei dati su delle persone e in un frammento ho le persone con ID compreso tra 1 e 500, se sto cercando l'ID 56 interrogherò soltanto quel frammento e non altri contenenti dati che non necessito).

3- Global query optimization: in questa fase avviene l'ottimizzazione globale utilizzando le statistiche sui frammenti. L'ottimizzazione avviene aggiungendo agli operatori di algebra relazionale del query tree anche gli operatori di comunicazione (Send/Receive tra nodi). L'obiettivo preposto dall'ottimizzazione è quello di ottenere il migliore ordinamento possibile delle operazioni definite dalla fragment query di partenza. Le decisioni più importanti vengono effettuate sulle operazioni di JOIN perché a differenza delle operazioni di selezione e di proiezione riducono il volume dei dati trattati mentre l'operatore JOIN tendenzialmente aumenta il numero di dati a disposizione. In particolare sulle operazioni di JOIN si decide l'ordine dei JOIN n-ari e la scelta tra JOIN e SEMIJOIN. Oltre ad ottimizzare l'ordinamento delle operazioni questa fase si occupa anche della gestione degli eventi non predicibili (es. ritardo nella rete) implementando una fase di re-ottimizzazione in run time.

4- Local optimization: ottimizzazione locale dove ogni nodo riceve una fragment query e la ottimizza in modo indipendente utilizzando tecniche analoghe a quelle dei sistemi centralizzati.

Esempio di distribuzione dei dati:

Supponiamo di avere due tabelle Employee(eno,ename,title) e AssiGN(eno projectno, resp, dur) e supponiamo di avere 400 Employee e 1000 AssiGN. Supponiamo di volere ottenere il nome dei dipendenti che sono manager di progetti; normalmente utilizzando una query di selezione ed effettuando un JOIN tra le due tabelle che hanno la stessa chiave "eno" e mettendo come condizione WHERE resp= "manager" otterremmo ciò che cerchiamo. Supponiamo di avere 5 nodi a disposizione e di volere distribuire i dati; a questo punto inseriremo in due nodi i dati di Employee e in altri due nodi i dati di AssiGN, infine in un ultimo nodo inseriremo il risultato finale. I dati vengono suddivisi dentro i nodi secondo dei criteri precisi utili alla interrogazione, ossia in ASG1 e EMP1 tutti i dati degli impiegati con eno<=3 e in ASG2, EMP2 i dati degli impiegati con eno>3.

Una prima esecuzione della query in questo nuovo ambiente distribuito potrebbe essere la seguente:

- Nel nodo dove c'è ASG1 faccio l'interrogazione resp= "manager" ottenendo ASG1'
- Il risultato ottenuto lo sposto nel nodo contenete EMP1 (unica tabella che può effettuare il JOIN con ASG1)
- Da EMP1 prendo il nome dei dipendenti che sono dei manager e sposto il risultato EMP1' nel nodo contenente il risultato
- Faccio la stessa procedura per ASG2 e EMP2, infine nel nodo risultato farò l'unione dei dati ricevuti dai nodi

È possibile implementare un altro tipo di soluzione allo stesso problema, ovvero trasferire i dati risultato di ASG1 e ASG2 nel nodo risultato e trasferire sempre nello stesso nodo anche i risultati dei nodi EMP1 e EMP2 ed eseguire all'interno del nodo risultato il calcolo del risultato. Questo potrebbe causare dei rallentamenti in quanto i tutti i dati devono raggiungere il nodo risultato affinché possa essere calcolata la soluzione e quindi nel caso di un nodo particolarmente lento ci sarebbero dei ritardi nella soluzione.

Confrontiamo ora le due strategie per capire quando è meglio usare una o l'altra. Supponiamo che:

- Il costo di accesso ad un record è 1
- Il costo di trasferimento di un record è 10
- Vi sono 10 manager sia su ASG1 che su ASG2

Utilizzando la prima strategia si ottengono i seguenti costi:

1- Il calcolo di ASG1' e ASG2' costa 10+10 = 20 per accesso diretto ai dati (ci sono 10 manager quindi 1X10)
2- Il trasferimento di ASG1' e ASG2' nei nodi contenenti EMP1 e EMP2 costa 20X10 = 200
3- Il calcolo di EMP1' e EMP2' che prevede un JOIN tra tabelle costa (10+10) X2 = 40
4- Il trasferimento di EMP1 e EMP2 nel nodo risultato costa 20X10 = 200

Per un totale di 460.

Adottando la seconda strategia si ottengono i seguenti costi:

1- Il trasferimento di EMP1 e EMP2 sul nodo risultato costa 400X10 = 4000 (ci sono 400 Employee in totale)
2- Il trasferimento di ASG1 e ASG2 sul nodo risultato costa 1000X10 = 10000 (ci sono 1000 AssiGN in totale)
3- Il calcolo di ASG' costa 1000
4- Il JOIN tra ASG' e EMP costa 20X400 = 8000

Per un totale di 23000.

Si nota facilmente come i costi di comunicazione siano i più dispendiosi. Nelle grandi reti geografiche vale la regola che dice che i costi di comunicazione siano molto maggiori dei costi di elaborazione dei dati, mentre nelle reti locali questi costi sono pressoché uguali. Se si desidera minimizzare il tempo di risposta sarà dunque necessario aumentare il parallelismo che comporterà però un aumento del costo totale in quanto verranno effettuate un numero maggiore di trasmissioni. Se si desidera minimizzare il costo totale non si tiene conto del parallelismo ma si incrementa il response time. Queste due azioni di ottimizzazione vengono eseguite nella fase di global query optimization. In conclusione il primo esempio ottimizza il costo totale mentre il secondo esempio parallelizza di più ma, in questo caso, non riesce a minimizzare il tempo di risposta.

JOIN e SEMIJOIN:

Il JOIN nelle reti distribuite causa notevoli problemi in quanto non è possibile trasportare da un nodo all'altro gli indici delle tabelle perdendo quindi parte dei dati. Per risolvere questo problema è possibile eseguire un'operazione diversa. Supponiamo di dovere effettuare un'operazione di JOIN tra due tabelle R e S situate in due nodi differenti. In alcune circostanze è possibile utilizzare il SEMIJOIN come soluzione più efficiente.

L'operazione di SEMIJOIN su un attributo A è il JOIN tra la tabella R ed S proiettato su R*, dove R* è l'insieme degli attributi di R. Il SEMIJOIN non gode della proprietà commutativa. Il SEMIJOIN come prima cosa porta dalla tabella S alla tabella R non tutta la tabella, ma bensì solamente la serie di attributi su cui è richiesto di effettuare il JOIN. Gli attributi una volta raggiunta la tabella R vengono effettuati i confronti tra i dati di R ed S che saranno incredibilmente più piccoli in quanto solo una parte della tabella S è stata trasferita. Una volta identificate le righe di R che sono in JOIN con le righe di S vengono trasferiti in S soltanto le righe di R che soddisfano la richiesta di JOIN con S.

In generale l'uso del SEMIJOIN è conveniente se il costo del suo calcolo e del trasferimento del risultato è inferiore al costo del trasferimento dell'intera relazione e del costo del JOIN intero (se una tabella ha 50 attributi è più veloce spostarne solo alcuni che sono strettamente necessari alla operazione di JOIN).

Controllo di concorrenza nei DDBMS:

Le transazioni nei DDBMS che prevedono operazioni di lettura e scrittura sono più complicate da gestire rispetto alle operazioni di sola lettura appena viste. Nelle transazioni ogni operazione è diretta ad un unico server, tuttavia le

transazioni possono modificare più di un DB e queste devono essere coordinate tra sistemi diversi. Sotto il punto di vista delle proprietà ACID nel caso delle transazioni nei DDBMS è necessario rivedere le proprietà di atomicità e di isolamento.

Serializzabilità e controllo delle repliche:

Ogni transazione può essere scomposta in sotto-transazioni che possono essere associate a diversi nodi di elaborazione, perciò lo schedule globale dipende dalle schedule locali su ogni nodo. Quando si effettuano transazioni su più nodi bisogna prestare attenzione alla serializzabilità, in quanto se a livello locale si ha una schedule seriale ciò non è garantito anche a livello globale. Lo schedule globale è serializzabile se gli ordini di serializzazione sono gli stessi per tutti nodi coinvolti. Nel caso in cui il DB contenga delle repliche questa regola non vale; se ad esempio abbiamo due nodi replicati e vogliamo effettuare due operazioni di scrittura, nel caso in cui venga selezionato la stessa replica per le due operazioni si otterrà un conflitto che non consente ad entrambe le transazioni di essere completate, nel caso in cui si decide di scrivere un dato su una replica e l'altro su un'altra si avranno due schedule seriali ma verrà a mancare la mutua consistenza dei DB locali in quanto a fine delle transazioni si avranno due valori diversi per due copie che dovrebbero essere uguali.

Protocollo ROWA:

Per far fronte a questo problema è necessario introdurre un protocollo di controllo delle repliche chiamato ROWA (Read Once Write All). Il funzionamento di questo protocollo è il seguente:

- Dato un oggetto X (tabella, riga, ecc) con una serie di repliche X1,X2…Xn
- X viene identificato come oggetto logico mentre X1,X2….Xn sono definiti oggetti fisici
- Le transazioni vedono solamente il livello logico (quindi X)
- Il protocollo consente la lettura dei dati da uno qualunque dei nodi replicati, ma impone la scrittura su tutte le copie dello stesso nodo

Quindi le transazioni non terminano finché il nuovo valore non è stato scritto all'interno di ogni nodo replicato. Questo incide notevolmente sulle performance ma garantisce consistenza.

Protocollo 2PL:

Il protocollo 2PL in ambito distribuito mette a disposizione due strategie, una strategia centralizzata che si basa sui siti e una strategia sulle copie primarie che si basa sulle copie. Il protocollo in generale prevede che una transazione richieda tutti i lock e successivamente possa iniziare a rilasciarli quando la transazione si è conclusa.

In una strategia centralized 2PL ogni nodo dispone di un Lock Manager e viene eletto un LM coordinatore che gestisce i lock per l'intero DDB. In generale il Transaction Manager del nodo dove inizia la transazione è considerato TM coordinatore e gli altri nodi su cui viene eseguita la stessa transazione sono considerati Data Processor. La strategia centralizzata 2PL avviene dunque con i seguenti step:

1- Il TM coordinatore chiede al LM coordinatore i lock
2- Il LM concede i lock utilizzando un 2PL
3- Il TM comunica ai DP di potere eseguire le transazioni in quanto sono stati garantiti i lock
4- I DP comunicano al TM di avere terminato le operazioni che a sua volta lo comunica al LM e vengono rilasciati i lock

Questo meccanismo comporta un problema di collo di bottiglia in quanto vi è un unico LM coordinatore che deve gestire tutte le richieste di lock. Per evitare questo problema viene introdotto il meccanismo della copia primaria. Per ogni risorsa viene individuata una copia primaria prima della assegnazione dei lock. I diversi nodi possiedono dei LM che gestiscono le proprie risorse, quindi il LM non è più centralizzato ma distribuito. Ogni risorsa richiesta dalla transazione viene richiesta dal TM al LM responsabile della copia primaria che concede il lock alla risorsa. Questa

operazione risolve il problema del collo di bottiglia ma richiede di determinare a priori tutti i LM assegnati a ciascuna risorsa.

Deadlock e protocolli di risoluzione:

Un altro problema che si pone è il problema del Deadlock causato da una attesa circolare tra due o più nodi. Viene dunque introdotto un algoritmo asincrono e distribuito. Assumiamo che tutte le sotto-transazioni siano attivata in modo sincrono, ossia quando una transazione richiede di eseguire una operazione su un nodo esegue una RPC (Remote Procedure Call bloccante) e dunque se due transazioni tentano di accedere ad una risorsa in questo modo si possono ottenere due tipi di attesa:

1- Attesa da RPC: attesa del termine di una sotto-transazione su un differente nodo
2- Attesa da rilascio di risorsa: attesa classica dove una transazione aspetta che venga rilasciato il lock di una risorsa preso precedentemente da un'altra transazione

È possibile caratterizzare le condizioni di attesa su ciascun nodo tramite l'utilizzo di condizioni di precedenza. Non avendo una visione globale ogni nodo dovrà essere in grado di riconoscere quali sono le transazioni eseguite internamente in attesa per una chiamata esterna o di una risorsa. Per fare ciò ogni nodo integra la sua sequenza di attesa con le sequenze di attesa degli altri nodi a lui legati con delle condizioni di EXT (chiamata ad un nodo remoto), successivamente analizza le condizioni di attesa sul nodo e rileva i deadlock locali, infine comunica le sequenze di attesa agli altri nodi (ed eventualmente la presenza di un deadlock). Per evitare che gli stessi deadlock vengano scoperti più volte il nodo verifica che localmente non vi siano dei deadlock prima di passare le informazioni al nodo immediatamente successivo.

Come garantire l'atomicità di una transazione (Recovery Management):

Un sistema distribuito oltre ad essere soggetto a guasti locali può soffrire di perdita di messaggi su rete e partizionamento della rete. Un esempio di partizionamento della rete può essere identificato con un insieme di nodi che si vedono tra di loro ma non vedono il resto della rete, questo implica degli errori nella gestione delle transazioni che occupano più nodi quando uno dei nodi impiegato nella transazione non risulta più visibile agli altri nodi.

Protocollo 2PC:

Il protocollo 2PC (Two Phase Commit) consente ad una transazione di effettuare una decisione di abort/commit su ciascuno dei nodi che partecipano ad una transazione. Il protocollo utilizza un coordinatore chiamato Transaction Manager (TM) e i sever che sono chiamati Resource Manager (RM) e la decisione di effettuare commit/abort tra due o più server partecipanti è coordinato appunto dal TM.

In assenza di guasti il protocollo 2PC prevede le seguenti fasi:

1- Il TM chiede a tutti i nodi (Prepare) come intendono terminare la transazione (con commit /abort). Ogni nodo decide autonomamente in quale modo terminare la transazione e lo comunica unilateralmente e la sua decisione è irrevocabile (ready to commit/not ready to commit)
2- Una volta ricevute tutte risposte dei nodi, il TM verifica e, se tutti comunicano un commit, si conclude la transazione comunicandolo ai nodi. Se anche uno solo dei RM comunica un abort allora viene comunicato un abort globale.

Nei file di log di ogni singolo nodo vi sono due tipi di record: i record di transazione che contengono informazioni sulle operazioni effettuate e i record di sistema che contengono l'evento di Checkpoint e l'evento di Dump che indicano rispettivamente l'elenco delle transazioni attive/completate e la fotocopia del database in un certo stato.

In assenza di guasti il protocollo viene applicato come segue:

- Il coordinatore scrive sul proprio log *begin commit* e quindi invia un messaggio a tutti i partecipanti di *prepare* e si mette in attesa della ricezione di tutte le risposte da parte di tutti i partecipanti della transazione
- I partecipanti ricevono la richiesta e valutano se sono pronti a fare commit, se sono pronti scrivono *ready to commit* nel file di log, altrimenti scrivono *abort*
- Il coordinatore una volta ricevute le risposte se ha ricevuto anche solo un messaggio di abort, scrive *global abort* sul file di log e poi comunica a tutti i nodi la decisione di abort globale e termina la transazione, oppure scrive *global commit* nel file di log e lo comunica a tutti i nodi con un commit globale
- I partecipanti ricevono la risposta e se ricevono abort scrivono sul file di log *abort*, mentre se ricevono commit scrivono sul file di log *commit* e infine comunicano al TM di avere ricevuto il messaggio

Nella fase di prepare e nella fase di global decision vengono introdotti due timeout che stabiliscono un intervallo di tempo entro il quale è necessario che vengano ricevute le risposte dei nodi dal TM coordinatore nel primo caso (il timeout viene fisato quando il TM scrive *prepare* nel suo log) e nel secondo che venga ricevuta la risposta dei nodi di avere ricevuto il messaggio di commit/abort globale del TM coordinatore (il timeout viene fissato quando il TM comunica il commit o abort globale). Se scatta il primo timeout significa che almeno un nodo non ha risposto per tempo quindi potrebbe essere guasto e quindi scatta un abort globale. Se scatta il secondo timeout significa che almeno un nodo non ha confermato la decisone globale di commit/abort, quindi viene settato un nuovo timeout e inviato un nuovo messaggio ai nodi da cui il TM non ha ricevuto risposta di decisone globale.

Dopo che gli RM hanno comunicato lo stato di ready al TM, questi rimangono in attesa di una risposta da parte del TM, questo intervallo di tempo viene chiamato finestra di incertezza. In questo intervallo di tempo le risorse non possono essere rilasciate e nel caso di un malfunzionamento del TM o RM esse non saranno mai rilasciate.

I casi in cui il RM si guasta sono i seguenti:

- Se uno o più RM si guastano prima della operazione di *prepare*, non saranno restituiti i loro messaggi di *ready* e quindi scatterà il primo timeout che porterà ad un abort globale
- Se uno o più RM si guastano dopo avere comunicato al TM la loro disponibilità al commit/abort, quando il nodo viene riattivato si mette in attesa di un invio da parte del TM della decisione globale oppure tramite una richiesta di remote recovery chiede al coordinatore quale è stata la decisione globale

I casi in cui il TM si guasta sono i seguenti:

- Se il TM si guasta prima della fase iniziale viene attivato un timeout che porta ad un abort globale
- Se il TM si guasta dopo avere comunicato il messaggio di *prepare* ai nodi e questi hanno comunicato il loro stato di *ready* vengono utilizzati degli algoritmi bizantini che consentono agli RM di decidere tra di loro che decisione prendere (questa operazione può essere più o meno agevole in base alla visibilità che nodi hanno degli altri)
- Se il TM si guasta dopo avere effettuato la decisione globale, quando viene ripristinato il suo funzionamento informa nuovamente i nodi della decisione globale

È possibile effettuare delle ottimizzazioni nel protocollo 2PC come ad esempio ridurre il numero di messaggi spediti nel caso in cui un RM che partecipa alla transazione non effettua operazioni di scrittura ma solo di lettura (Read Only) che al messaggio di *prepare* risponde con un messaggio di *read-only* in modo tale che il TM ignori i nodi che hanno risposto con questo messaggio nella seconda fase di comunicazione della decisione globale. Un'altra possibile ottimizzazione avviene quando il TM comunica un abort globale, in questo caso non è necessario attendere la risposta di ricezione del messaggio da parte degli RM.

REPLICHE

La replica è quel processo di creazione più istanze dello stesso database allineate fra di loro, questo consente la condivisione di dati ma deve tenere conto dei cambiamenti progettuali del database ovvero se viene modificato un elemento all'interno di un nodo replicato la modifica deve essere applicata anche a tutte le repliche dello stesso. Elemento fondamentale per le repliche è la sincronizzazione ossia il processo che consente di avere le copie del database allineate. La replica può essere utilizzata sia in maniera sincrona che asincrona. Nel primo caso si cerca di aggiornare tutte le repliche in contemporanea tramite diversi metodi (es. protocollo ROWA Read Once Write All) e se uno o più nodi di replica non risultano raggiungibili impossibilitandone l'aggiornamento la transazione non potrà essere completata. Nel secondo caso il meccanismo è diverso infatti prima si completa la transazione e si aggiorna il database e poi in un secondo momento si vanno ad aggiornare le repliche. La replica sincrona obbliga ad aggiornare due o più storage contemporaneamente e nel caso in cui non fosse possibile si effettua una operazione di rollback della transazione; questa tecnica viene utilizzata principalmente per le cosiddetta disaster recover ossia dei down in reti che non possono permetterselo come le banche e dunque necessitano di avere una replica in un luogo sicuro. Al contrario la replica asincrona effettua una copia dello storage principale e con calma sposta i dati in un nodo di replica, questo implica costi più bassi e maggiore flessibilità. Questa tecnica viene utilizzata maggiormente per gli accessi offline, per l'efficienza del calcolo ecc.. tuttavia il problema principale di questa tecnica è la possibilità di perdere dei dati in quanto le repliche richiedono più tempo.

Contesti di applicazione della replica:

Vi sono diversi contesti nei quali può essere applicata la replica e in ognuno di questi viene applicata in maniera differente seguono alcuni contesti:

1- Condivisione dei dati tra utenti disconnessi tra di loro: ad esempio nel caso in cui vi sono dei venditori che utilizzano dei laptop senza internet ed effettuano più vendite sullo stesso prodotto con quantità limitata e questi dati vengono inseriti nel database non in contemporanea ma solo quando il dispositivo torna in azienda creando così la possibilità di vendere più di quanto si ha. Per far fronte a questi problemi viene utilizzata la tecnica di merge replication.
2- Consolidazione dei dati: le compagnie possono avere sedi diverse con i propri dati e le modifiche vengono effettuate solo nei nodi locali, ma ad un certo punto è necessario replicare questi dati ed inserirli nel DB centrale per aggiornarlo.
3- Distribuzione dei dati: utilizzato principalmente per le applicazioni di E-Commerce che necessitano di una continua sincronizzazione bidirezionale in real time tra le applicazioni per garantire ai clienti una certa qualità di servizio (es. Amazon)
4- Performance
   4.1- Incrementare l'accessibilità dei dati: se nella soluzione che vogliamo costruire non ci sono update immediati si può utilizzare la replica del nodo garantendo così migliore accesso ai dati e migliori tempi di risposta
   4.2- Bilanciamento del carico: nel caso in cui un singolo DB è utilizzato da troppi utenti e la sua CPU supera l'80% di carico è conveniente effettuare una replica del DB per alleggerirne il carico
   4.3- Incrementare la disponibilità: per evitare la perdita di dati in caso di guasto del DB centrale si può creare una copia remota dello stesso a cui indirizzare i nodi nei casi di disaster recovery
5- Separazione dei data entry e data reporting: se il DB riceve una grande quantità di dati e gli stessi dati devono essere utilizzati per il reporting allora può essere utile separare i due elementi, in quanto le transazioni (data entry) bloccando tabelle intere non consentirebbero le operazioni di lettura del reporting, tramite la replicazione è possibile disaccoppiare una operazione di data entry e di data recovery sullo stesso dato
6- Coesistenza di applicazioni: per esigenze di aggiornamento potrebbero verificarsi complesse trasformazioni dei dati per renderli compatibili a nuove applicazioni. In questo caso si effettua una replica per mantenere il servizio attivo mentre si effettua la trasformazione dei dati e la migrazione degli stessi per le nuove applicazioni

Vi sono casistiche in cui la replicazione del DB non dovrebbe essere usata quali i casi in cui vi sono frequenti update di record su più copie del DB e quando la consistenza dei dati in real-time è fondamentale (es. l'acquisto di biglietti aerei). Nel caso di frequenti aggiornamenti sulle copie è possibile che si verifichino conflitti tra i record (es. Amazon vende 5 paia di scarpe e ne vengono comprate 6), questi conflitti devono essere risolti manualmente e quindi richiedono un notevole quantitativo di tempo.

Tipologie di replica:

Tra le varie tipologie di replica disponibile alcune tra le più comuni sono:

- Replica 1: many dove la sorgente distribuisce in maniera sincrona/asincrona le varie copie passive (solo lettura)
- Replica Peer-to-Peer dove ogni nodo può effettuare operazioni di scrittura/lettura e si aggiornano tra di loro (gestibile con approccio ROWA)
- Replica 1: many per Data Consolidation in cui delle sorgenti locali trasferiscono i dati che vengono consolidati a livello centrale in unico DB
- Replica bidirezionale tipicamente usata per identificare soluzioni ai conflitti dove vi sono due nodi in rapporto Peer-to-Peer
- Replica Multi-tier Staging in cui vi sono dei meccanismi intermedi tra la sorgente e il target che forniscono una funzione di deposito

Come realizzare una replica:

Vi sono diversi modi di effettuare una replica alcuni di questi sono:

- Effettuare un backup fisico staccando il disco dal server, creandone una copia e poi riattaccando entrambe
- Effettuare una copia di backup e spostarla fisicamente e fare il restore in un altro luogo
- Effettuare una replica incrementale dove si effettua un full backup e successivamente si sposta solamente il log transazionale che consente alla replica di aggiornare in automatico le transazioni nel log evitando di trasportare enormi quantità di dati come le tabelle intere

SECONDO INCONTRO

Risultati del test di comprensione:

DOMANDA: QUALE DELLE PROPRIETA' ACID GARANTISCE IL PROTOCOLLO 2PL?

Il protocollo 2PL si occupa del controllo di concorrenza e previene problemi come gli aggiornamenti fantasma, la perdita di update, le letture sporche ecc... NON SI OCCUPA DELL'ATOMICITA'. Il protocollo garantisce di evitare che delle transazioni utilizzino delle risorse che sono già utilizzate da altre transazioni, non si occupa di garantire che la transazione si concluda con un commit/abort e che il DB ricordi la modifica effettuata per sempre. NON SI OCCUPA DELLA CONSISTENZA, in quanto sul DB sia che il protocollo 2PL eviti la perdita di un update sia che non vi sia un protocollo a protezione di tale errore, la variabile salvata sarà comunque consistente.

DOMANDA: QUANTE VOLTE PUO' ESSERE LOGICAMENTE ESEGUITO IL COMANDO COMMIT ALL'INTERNO DI UNA TRANSAZIONE?

La risposta non è nessuna di quelle proposte, in quanto il comando commit può essere eseguito 0 o 1 volta, in quanto viene eseguita 0 volte nel caso in cui si effettua un rollback e viene eseguita 1 volta nel caso in cui si esegue il commit a termine della transazione.

DOMANDA: IL CASO ALITALIA COSA PUO' AVERE CAUSATO L'EMISSIONE DI DUE CARTE DI IMBARCO PER LO STESSO POSTO?

L'overbooking spesso viene utilizzato dalle compagnie aeree e consiste nel vendere più biglietti dei posti a sedere in quanto è probabile che qualche passeggero per qualche problema non riesca a presentarsi in aeroporto. Questo non è possibile in quanto l'overbooking si effettua sulla vendita dei biglietti non sul check-in.

Lo schedule potrebbe non essere stato isolato ovvero l'ordine con cui sono state eseguite le transazioni non ha garantito il controllo di isolamento previsto dal protocollo 2PL. Questa risposta non è corretta in quanto le operazioni di check-in sono state effettuate in intervalli di tempo differenti.

Per trovare la risposta corretta è necessario ragionare sulla architettura dati. Dato che un check-in è stato effettuato online e uno in aeroporto l'architettura dati potrebbe prevedere la presenza di due DB, uno che si occupa delle prenotazioni web e uno che si occupa delle prenotazioni in loco in aeroporto. La presenza di repliche del DB garantiscono anche un ulteriore livello di controllo e di sicurezza dei dati dagli attacchi. In questo caso il problema potrebbe essere identificato con un problema di aggiornamento tra repliche e quindi su un DB il posto era assegnato ad un passeggero mentre sull'altro era assegnato ad un altro passeggero.

TERZO INCONTRO

Domande e risposte test:

DOMANDA: COSA SI INTENDE PER ETEROGENEITÀ NEI DDBMS

L'eterogeneità si riscontra nei modelli, negli schemi, nei vendors e sui prodotti software.

DOMANDA: QUANTI SCHEMI LOGICI SI DEVONO PROGETTARE IN UN DDBMS

Uno schema logico globale e schemi logici locali per ogni nodo.

DOMANDA: CHE COS'È X-OPEN DTP

È una architettura (non è un middleware ne un software) implementata da vari vendors che consente la comunicazione tra applicazioni diverse

DOMANDA: QUALI SONO I PASSI AGGIUNTIVI CHE SONO PREVISTI NELL'OTTIMIZZAZIONE GLOBALE DI UNA QUERY DISTRIBUITA RISPETTO ALL'OTTIMIZZAZIONE DI UNA QUERY IN UN SISTEMA CENTRALIZZATO?

Query decomposition, data localization, global optimization, local optimization. Local optimization non deve esserci perché è una ottimizzazione che viene eseguita anche nei sistemi centralizzati

DOMANDA: QUALE TRA LE SEGUENTI INFORMAZIONI È CORRETTA RISPETTO ALLA PRIMA FASE DEL 2PC

Se la transazione è read only il TM non fa nulla

DOMANDA: SE DURANTE LO SVOLGIMENTO DEL PROTOCOLLO 2PC SI PARTIZIONA LA RETE AVENDO DUE SOTTORETI UNA CON TM E RM1 È LA SECONDA SOTTORETE CON RM2 , RM3 , COSA SUCCEDERE?

C'è ambiguità perché non viene specificato in che fase ci si trova. Se avviene durante la prima fase avverrà un abort globale in quanto non riceverà nessuna risposta dai RM partizionati e dunque andrà in timeout. Se avviene nella seconda fase, la global decision è già stata presa e quindi il TM continuerà a comunicare la decisione globale agli RM scollegati finché non otterrà una risposta di avvenuta ricezione del messaggio dagli stessi.

DOMANDA APERTA: RISPETTO ALLA SEPARAZIONE FRA PRESENTAZIONE, APPLICAZIONE E GESTIONE DATI, COME POSSONO ESSERE SUDDIVISI TALI COMPONENTI FRA CLIENT E SERVER NEL CASO DI MOG (MULTPLAYER ONLINE GAME)?

Se i giochi sono complessi anche il DB deve essere in locale. Affermazione sbagliata perché non tiene conto dei dati degli altri giocatori a meno che non si tratti di una architettura peer-to-peer ma anche in quel caso ci sarebbero dei problemi di latenza.

Deve esserci un client graficamente dotato che esegua le richieste, mentre la gestione dei dati viene gestita dal server centrale. Non può esserci un solo server per gestire milioni di connessioni, servono più server divisi per aree o piattaforma, in quanto gestire singole partite è meno pesante rispetto al gestire milioni di utenti in simultanea. Se ad esempio sparo ad un giocatore in un gioco online il messaggio di fuoco e di danno deve essere inviato in broadcast a tutti gli altri giocatori che vedono l'azione e passare tramite un server per queste operazioni non è ottimale, sarebbe meglio utilizzare una connessione di tipo peer-to-peer. Nelle partite dunque un database come quelli di questo corso probabilmente non serve per quanto riguarda lo svolgimento della partita stessa, ma serve per memorizzare il risultato della partita. Fortnite ha un database distribuito diviso per aree geografiche per avere una latenza minore, probabilmente il gioco non ha alcun sistema di persistenza e ha tutto nella memoria locale e trasmette messaggi agli altri giocatori.

Object persistence:

Uno dei temi fondamentali in un sistema DDBMS è capire se i dati memorizzati in un'applicazione devono essere persistenti o meno. Se non è necessaria la persistenza dei dati un DB non è utile, ma se è necessaria la persistenza, dato che l'applicativo prima o poi verrà terminato, di conseguenza viene implicata anche la presenza di un DB. Dato che gli applicativi lavorano su oggetti e i DB su tabelle risulta difficile la comunicazione e lo scambio di informazioni tra i due, allora per creare un oggetto persistente e inviarlo in maniera corretta ad un DB è necessario l'Object-Relational Mapping. Questa tecnologia consente di salvare gli oggetti sotto forma di righe all'interno delle tabelle e viceversa di ottenere i dati dalle tabelle e di trasformarli in oggetti. Queste operazioni sui dati vengono eseguite dai Data Mapper/ Data Access Object che eseguono le operazioni CRUD (Create, Retrieve, Update, Delete).

NoSQL

BLOCKCHAIN

DEFINIZIONE DI BLOCKCHAIN E CONCETTI BASE:

La blockchain è un registro pubblico, condiviso e decentralizzato che memorizza le proprietà dei beni digitali. Con registro si intende un luogo in cui vengono memorizzate le informazioni relative alla proprietà di informazioni digitali (es. io possiedo tot di una determinata criptovaluta ecc…) e le transazioni. Questo registro è decentralizzato ossia non esistono degli amministratori, tutti sono allo stesso livello (Peer-to-Peer) e nessuno può fare di più rispetto ad altri.

Questo registro è organizzato a blocchi: il blocco di partenza chiamato genesis block da il via alla blockchain, i blocchi collegati tra di loro costituiscono il corpo della blockchain. I blocchi sono legati tra di loro tramite delle funzioni di hash crittografico, quindi il collegamento tra un blocco e il successivo è dato dal fatto che il valore di hash di tutto il primo blocco (impronta hash) è contenuto all'interno del blocco successivo. Questo rende estremamente difficile alterare il contenuto di un blocco dato che ogni blocco contiene diverse transazioni e l'impronta hash è generata da queste, quindi modificare una transazione all'interno di un blocco ne modificherebbe l'impronta hash e dunque sarebbe facile individuare delle manipolazioni dei dati.

Gli utenti che fanno parte di una blockchain vengono definiti nodi in assetto P2P e il loro compito consiste nel osservare le transazioni proposte, verificarne la validità (es. osservano che non avvenga il double spending cioè cedere a più utenti lo stesso prodotto) e successivamente eseguire un protocollo di consenso tra più nodi per accettare le transazioni ed inserirle in nuovo blocco posto in cima alla blockchain. Una volta raggiunto il consenso la blockchain viene memorizzata su ciascuno dei nodi della rete P2P.

È possibile che due blocchi siano collegati ad un nodo di origine, in questo caso la blockchain si dirama e si inizia ad espandere i due rami generati, ma la diramazione con la catena più lunga avrà la prevalenza su quella più corta e dunque si utilizzerà sempre il ramo più lungo lasciando morire quello più corto (orphan node). Le transazioni all'interno di un orphan node vengono dimenticate e devono essere riconsiderate come transazioni da inserire nei blocchi della blockchain.

BITCOIN:

Il Bitcoin è una criptovaluta nata nel 2008 che aveva come obiettivo di sostituire digitalmente il denaro. Unendo diversi protocolli per la crittografia si tentò di creare una valuta che non necessita di un intermediario come la banca per effettuare le transazioni. Le proprietà memorizzate all'interno della blockchain di Bitcoin sono ad esempio il numero di Bitcoin (BTC) posseduti. Le transazioni di Bitcoin avvengono coi seguenti step:

- Utente A vuole inviare 1 Bitcoin ad utente B
- Usando il proprio client specifica il quantitativo di bitcoin da inviare e l'indirizzo di B (l'indirizzo si ottiene utilizzando una chiave pubblica)
- L'utente A dimostra di essere il proprietario del Bitcoin inviato tramite una firma digitale con la propria chiave segreta
- L'utente A invia la richiesta di transazione alla rete P2P che dopo averla approvata la inserirà all'interno di un blocco della blockchain (i blocchi vengono aggiunti alla blockchain con cadenza di 10 minuti)

All'interno di una blockchain possedere un bitcoin è indicato dalla possibilità di avviare una transazione per dare tale bitcoin ad un altro utente.

I MINERS:

I miner sono i nodi della rete P2P e la loro funzione è quella di analizzare e validare le varie transazioni proposte dai client, sceglierne 1000 circa e poi formare un nuovo blocco da aggiungere alla blockchain. L'operazione di mining diventa dispendiosa a livello di calcolo in quanto è necessaria una proof-of-work ossia una prova che richiede uno sforzo di calcolo che gli accrediti il diritto di aggiungere il blocco all'interno della blockchain. La proof-of-work consiste nel prendere il dato di cui è necessario calcolarne l'impronta hash, aggiungergli una quantità casuale (nonce) e calcola l'hash del dato concatenato alla quantità casuale e verifica se il risultato possiede un certo numero di bit significativi uguale a 0.

Le funzioni di hash sono funzioni crittografiche che prendono in input una sequenza di bit di qualsiasi dimensione e ne produce una sequenza univoca di poche centinaia di bit. Seguono alcune proprietà fondamentali delle funzioni di hash:

1- È possibile, seppur estremamente raro, che il valore di hash calcolato da un file sia uguale ad un altro valore di hash calcolato da un file diverso, in questo caso si ha una collisione di hash.
2- Data un'impronta è estremamente difficile trovare un input in grado di produrre la stessa impronta.
3- Se cambio anche solo un bit della sequenza di bit in input, il risultato sarà completamente differente

Avere un'impronta con dei bit significativi di valore 0 corrisponde ad una riduzione del numero dei possibili output validi. Se ottiene un valore che possiede il numero richiesto di bit significativi uguali a 0 allora ha avuto successo, altrimenti utilizza un nuovo nonce e ricalcola finché non ottiene il valore ricercato. Una volta ottenuto il valore desiderato invia il blocco nella rete P2P dove verrà validato dagli altri nodi della rete e, dopo averlo accettato, lo inseriranno in cima alla blockchain. Il miner che tra tutti riesce ad inserire il proprio blocco in cima alla blockchain viene premiato con dei Bitcoin in quanto ad ogni nuovo blocco inserito nella blockchain vengono generati nuovi bitcoin.

I miners delle 1000 transazioni che devono scegliere prima di procedere con l'operazione di mining scelgono quelle transazioni che possiedono il valore più alto possibile di "mancia" ossia un compenso in bitcoin proposto da chi genera la transazione affinché i miners si occupino di far si che la transazione venga inserita all'interno della blockchain.

ETHEREUM:

Ethereum è un altro tipo di blockchain in cui si pone l'attenzione non sulle transazioni, ma sulle computazioni. Ogni utente che si iscrive ad Ethereum ottiene un account con una certa quantità di criptovaluta Ether (ETH). Le transazioni avvengono in maniera similare a quelle dei bitcoin con il coinvolgimento dei miners e della proof-of-work, anche se per motivi di efficienza energetica sono passati ad una proof-of-stake. La differenza sostanziale di Ethereum è la possibilità di scrivere degli smart contract tramite la programmazione con un linguaggio che si chiama Solidity. Questo linguaggio è molto simile a JavaScript e consente di generare dei contratti che vengono compilati in un bytecode che verrà memorizzato nella blockchain. Le transazioni dunque non servono solo a trasferire della valuta, ma anche a compiere delle funzioni stipulate all'interno dei contratti. I miner eseguono le operazioni del contratto sulla loro macchina e guadagnano in base all'esecuzione di ogni singola operazione del bytecode. La scrittura dei contratti deve essere fatta con estrema precauzione e precisione in quanto un contratto che presenta delle vulnerabilità o incorrettezze a livello di codice può comportare seri problemi sulla gestione delle transazioni o manipolazioni delle stesse da malintenzionati.

QUANDO E QUALI BLOCKCHAIN UTILIZZARE:

Esistono molte varianti di blockchain a seconda delle esigenze e talvolta le blockchain non sono necessarie, vediamo come in base alle esigenze si può necessitare o meno di una blockchain e di quale tipo di blockchain si necessita:

- Se NON si necessita di memorizzare uno stato allora non si necessita di alcuna blockchain

- Se si necessita di memorizzare uno stato ma NON ci sono scrittori multipli (quindi mi fido di chi scrive) allora non si necessita di alcuna blockchain
- Se si necessita di memorizzare uno stato, di avere degli scrittori multipli e si necessita di una Trusted Third Party (TTP) come ad esempio le banche nelle transazioni comuni, allora non si necessita di alcuna blockchain
- Se si necessita di memorizzare uno stato, di avere degli scrittori multipli e non si vuole utilizzare una TTP e i writers NON sono tutti noti (non si conoscono tutti fra di loro), allora si necessita di una Permitionless blockchain ossia quella utilizzata da Bitcoin ed Ethereum dove chiunque può scaricarsi il software e diventare miner
- Se si necessita di memorizzare uno stato, di avere degli scrittori multipli e non si vuole utilizzare una TTP, i writers si conoscono fra loro e si fidano l'uno dell'altro allora non c'è bisogno di una blockchain
- Se si necessita di memorizzare uno stato, di avere degli scrittori multipli e non si vuole utilizzare una TTP, i writers si conoscono fra loro e NON si fidano l'uno dell'altro allora se la verificabilità pubblica è richiesta si dovrà utilizzare una Pubblic Permissioned Blockchain, altrimenti si utilizzerà una Private Permissioned Blockchain

Nelle blockchain permissioned i nodi della rete P2P si conoscono tra di loro ma non si fidano tra loro e quindi vanno a formare un consorzio in cui i partecipanti sono potenzialmente in conflitto fra loro. Dato che nessun nodo si fida di nessun altro nodo, tutto quello che fa un nodo deve essere scritto all'interno della blockchain e se le affermazioni scritte siano vere o meno viene stabilito da un servizio di audit esterno che se individua una dichiarazione disonesta fa uscire dal consorzio il nodo interessato. In questo tipo di blockchain non esiste mining ne proof-of-work, quindi lo scopo principale è quello di avere un punto in cui sono implementate delle politiche di trust dove ogni nodo non può dichiarare il falso.

QUARTO INCONTRO

ESEMPI DI BLOCKCHAIN:

La blockchain può essere utilizzata per la tracciabilità e in particolare per evitare i falsi veri (es. la borsa di gucci tarocca ma che è uguale all'originale). Un caso particolare di tracciabilità è quella del parmigiano reggiano che è un marchio DOP ma che è facilmente contraffabile. Per potere tracciare le forme di parmigiano prodotte secondo la regolamentazione si potrebbe utilizzare un numero identificativo (hash code) dentro la forma che però non sia cancellabile a differenza delle impressioni a fuoco classicamente utilizzate. Un altro tipo di problema proposto sempre riguardante il parmigiano è la tracciabilità dei documenti che stabiliscono come è stato fatto il prodotto secondo le regolamentazioni nazionali; entreranno in gioco vari utenti che effettueranno uno o più passaggi nella creazione del parmigiano e che dovranno documentare dettagliatamente ciò che hanno fatto. Per verificare se questo caso è risolvibile attraverso l'utilizzo di una blockchain occorre osservare l'elenco puntato *di quando e quali blockchain utilizzare*. Si può notare come in questo caso vi siano le seguenti caratteristiche: è necessario memorizzare uno stato, vi sono scrittori multipli, vi è una TTP. La TTP in questo caso è identificabile con il consorzio DOP che può gestire la tracciabilità e affidabilità delle documentazioni. Quindi osservando le caratteristiche del caso si può dire che NON è necessario utilizzare una blockchain in quanto esiste una TTP. Supponendo di volere eliminare la TTP del consorzio DOP è necessario implementare una blockchain. I dati che andranno a comporre i blocchi saranno le datificazioni di ogni fase che percorre il parmigiano partendo dalla materia prima del provider arrivando fino al consumatore.

IL CASO GITLAB:

Gitlab è un sito web che gestisce pubblicamente l'intero ciclo DevOps di progetti. Dei malintenzionati sono entrati nel DB di GitLab e stavano creando degli snippet (richieste di scrittura di grandi quantità di dati) nel DB primario creando così dei lock nel sistema che ne impedivano il funzionamento. Per risolvere hanno fatto un ripristino prima dell'attacco utilizzando il DB secondario, ma questa operazione non va a buon fine in quanto il sistema risulta sovraccarico. Per risolvere un operatore decide di rimuovere i dati scritti durante l'attacco, ma per errore ha eliminato i dati nel DB primario anziché quelli del DB secondario. A questo punto si tentò di effettuare un backup (i quali vengono fatti ogni 24 ore) ma ci si accorse che i backup recenti erano vuoti per problemi di incompatibilità delle versione di Postgress

utilizzate. La morale della storia è controllare che le repliche e i backup effettuati funzionino in caso di problemi prima che i problemi si presentino.

INTRODUZIONE A NOSQL

ASPETTI POSITIVI DEL MODELLO RELAZIONALE:

- Modello ben definito e stretto
- Closed word assumption: assume che tutti i dati che potrebbero interessare alle applicazioni risiedono all'interno del DB
- Diversi anni di migliorie alla sicurezza, ottimizzazione e standardizzazione
- Proprietà ACID
- Molto diffuso ed utilizzato e difficilmente si smetterà di utilizzare (perché il porting sarebbe troppo rischioso)

LIMITI DEL MODELLO RELAZIONALE:

- Modello stretto, in quanto i principi di minimizzazione e di closed word assumption sono limitanti per le esigenze moderne
- Per ogni attributo all'interno di una riga vi può essere un solo valore
- I linguaggi di programmazione moderni ragionano ad oggetti, mentre il modello relazionale ragiona con tabelle e righe
- Hanno difficoltà coi modelli ciclici
- Difficile modificare le tabelle
- Non scalabile causato dal dover garantire le proprietà ACID e il 2PC che incrementa esponenzialmente il costo di organizzazione limitando dunque la scalabilità

ATTENZIONE IL MODELLO NOSQL NON SERVE SOLO PER LE APPLICAZIONI CHE LAVORANO COI BIG DATA

NOSQL CONCETTI CHIAVE:

NoSQL (Not only SQL) è un insieme di modelli che hanno tre caratteristiche fondamentali:

1- Tutti i modelli sono Schema Free
2- Tutti i DBMS sfruttano il CAP theorem
3- Viene utilizzato il modello BASE (Basic available, Soft state, eventually consistency)

Schema free è una caratteristica che, a differenza dei modelli relazionali che prima definiscono i modelli e successivamente i dati, stabilisce che non vi è un modello fisso ma un modello che si basa sui dati che vengono inseriti nel DB. Questo comporta che tutti i modelli NoSQL assumono la open word assumption e quindi non è necessario inserire i valori NULL nel caso in cui un dato non ci sia.

Il CAP theorem dice che in un sistema distribuito di nodi non è possibile garantire contemporaneamente i seguenti aspetti:

- Consistenza: tutti i nodi uguali hanno gli stessi dati nello stesso istante
- Disponibilità (Availability): ogni richiesta deve ricevere una risposta che sia di successo o di fallimento
- Partizionamento: il sistema deve essere operativo nonostante la possibilità di perdita di messaggi o di fallimenti da parte del sistema

Date queste caratteristiche il teorema dice che è possibile soddisfare solo due di queste tre in un sistema distribuito. Gli RDBMS (es. Postgres) generalmente soddisfano le condizioni CA (consistenza e disponibilità), i sistemi NoSQL possono essere CP (manca la disponibilità quindi i DBMS non possono lavorare 24h su 24) come ad esempio i database di MongoDB o AP (manca la consistenza dei dati alle volte) come ad esempio i database Cassandra.

Il principio BASE stabilisce che:

- Deve essere garantita la Basic Availability, ossia che una risposta viene comunque data prima o poi anche se la consistenza non del tutto garantita.
- Viene abbandonata la consistenza che stabilisce il modello ACID in cambio di un cosiddetto Soft State
- La consistenza c'è anche se non immediata, quindi prima o poi i dati saranno allineati ma non subito

MODELLI NOSQL:

Seguono alcuni modelli di NoSQL:

- Key-Value Stores: sono tabelle di hash dove la chiave punta ad un particolare valore (es. Dynamo di Amazon)
- Column Family Stores: tabelle le cui ogni chiave può assumere dei valori diversi e fare sì che punti a colonne multiple (es. Cassandra, BigTable)
- Document Databases: sono dei modelli che si basano su dei documenti all'interno dei quali ci possono essere coppie di chiavi/valore o oggetti. Questi documenti sono indirizzati nel DB tramite una chiave unica ed è possibile effettuare una ricerca nei documenti.
- RDF Databases

DOCUMENT BASED SYSTEM

L'idea dei Document Based System è quella di avere una evoluzione dei modelli chiave/valore; esiste una chiave, un object identifier, di solito i dati vengono memorizzati in formato JSON e i documenti possono essere ricercati a qualunque livello. Il modello documentale è rappresentabile con un albero con un nodo radice e tanti nodi foglia e l'accesso ad un dato parte dal nodo radice e arriva fino alla foglia interessata, questo comporta una fondamentale differenza rispetto al modello relazionale dove non esistono elementi più importanti di altri. Il modello ad albero dunque ha le seguenti caratteristiche:

- È un modello multidimensionale
- Ogni campo può contenere un valore oppure nessuno oppure contenere altri documenti
- Le interrogazioni si possono fare su qualunque livello dei documenti
- Lo schema è flessibile nel senso che posso avere un documento che ha soltanto un id, una chiave e un valore e poi posso avere un documento della stessa collezione che contiene due o più attributi o addirittura dei sotto-attributi
- È possibile aggiungere direttamente in linea dei dati
- Avendo embedding tra le relazioni che sussistono tra gli elementi sono richiesti meno incidici e quindi le performance sono migliorate

REFERENCING E EMBEDDING:

Per modellare il modello documentale si utilizzano due operazioni: il referencing e l'embedding. Il referencing funziona come il modello relazionale dove vengono creati più documenti i cui dati sono collegati da dei riferimenti. L'embedding invece pone all'interno di una collezione di documenti dove ogni documento contiene tutti i suoi dati, questo comporta la possibilità di avere repliche dei dati. La flessibilità dello schema documentale si denota ad esempio nel caso in cui fosse necessario aggiungere degli attributi aggiuntivi a dei dati; se nel modello relazionale è necessario modificare lo schema dei dati aggiungendo i nuovi attributi e valorizzare a NULL tutti gli altri dati che non possiedono tale attributo, nello schema documentale si aggiungono i nuovi attributi senza andare a modificare altri documenti ma solo il diretto interessato. Questo però potrebbe portare ad un problema in quanto quando si vuole effettuare un'interrogazione su uno schema non è possibile sapere quando un documento può possedere o meno un dato attributo senza andare a scorrere tutti i documenti.

QUERY:

I linguaggi di interrogazione dei Document Based System non si limitano ad utilizzare solo SQL ma ce ne sono diversi.

MongoDB:

MongoDB è un DBMS non relazionale dove i dati sono memorizzati in un modello Binary JSON (BSON)che consente un risparmio in quanto i dati vengono rappresentati in formato binario. Il modello consente di non eseguire i JOIN perché utilizza l'embedding dei dati anche se consente comunque il referencing seppur in maniera più complessa rispetto al modello relazionale. L'insieme delle tabelle che costituiscono il DB si chiamano Collezioni dove ogni documento è equivalente ad una riga della tabella e una colonna è equivalente ad una chiave il cui valore può essere a sua volta un documento, un valore oppure un array di valori.

La replicazione in MongoDB avviene tramite architetture master slave dove tutte le scritture avvengono sul nodo master e le scritture sulle repliche secondarie avvengono in un secondo momento. La replica avviene in modo parziale ed incrementale, ovvero un nodo prende tutti i dati dal nodo master (PRIMARY) e successivamente dal file di log del nodo primario esegue tutte le ultime operazioni. Ogni nodo può assumere i seguenti stati:

- STARTUP: un nodo che non è un membro effettivo di nessun replica set
- PRIMARY: l'unico nodo nel replica set che può accettare operazioni di lettura
- SECONDARY: nodo in grado di effettuare la sola replicazione dei dati (eventualmente può essere promosso a PRIMARY)
- RECOVERING: nodo che sta effettuando una qualche operazione di recupero dei dati causato da un ROLLBACK o un DOWN (può essere promosso a PRIMARY)
- STARTUP2: nodo appena entrato nel replica set (può essere promosso a PRIMARY)
- ARBITER: nodo non atto alla replicazione dei dati ma che prende parte alle elezioni per promuovere i nodi a PRIMARY (anche lui può essere promosso a PRIMARY)
- DOWN/OFFLINE: un nodo irraggiungibile
- ROLLBACK: un nodo che sta effettuando un rollback per cui sarà inutilizzabile per le letture (può essere promosso a PRIMARY)

DURANTE LE OPERAZIONI DI ELEZIONE LE SCRITTURE SI INTERROMPONO, ECCO PERCHÉ NON È GARANTITA LA PERSISTENZA IN MONGODB.

Ogni replicaSet manda un heartbeat agli altri nodi con un timeout, se un nodo non risponde entro il timeout questo viene considerato morto. Se il nodo morto è il nodo primario allora viene indetta una elezione. Ogni nodo ha un identificativo intero che rappresenta la sua priorità durante l'elezione; più alto è il valore di questo intero più è probabile che venga eletto PRIMARY, di conseguenza quando un nodo primario muore il nodo con priorità più alta immediatamente dopo al vecchio nodo primario richiama per primo delle nuove elezioni. Un replcia set può avere al massimo 50 membri di cui solo 7 votanti, tuttavia è possibile che vi siano dei nodi non votanti i quali avranno una priorità uguale a 0 e accettano solo operazioni di lettura.

Le operazioni di scrittura avvengono sul nodo primario e possono essere definite delle politiche di scrittura chiamate writeConcern che definisce le modalità di replica. I parametri che possono essere passati a writeConcern sono:

- w: il numero di nodi in cui la scrittura deve essere confermata prima che la scrittura possa essere confermata a livello globale
- j: stabilisce se c'è l'intenzione di scrivere sui log di Wired Tiger (write ahead logging) PRIMA che l'operazione sia stata eseguita. In sostanza prima si scrive sul file di log e poi sul disco per garantire la persistenza del dato.
- wtimeout: imposta il tempo limite da aspettare prima di concludere l'operazione

In base ai valori di W si hanno differenti risultati:

- W= 0: indica nessuna certezza dell'inserimento in quanto nessun nodo deve confermare
- W= 1: constata la scrittura sul nodo primario
- W= n: constata la scrittura su n nodi (a maggioranza)
- W= majority: almeno la metà più uno dei nodi deve avere scritto prima di avere conferma dell'operazione

Nel caso in cui si deve caricare una grande quantità di dati nota, per motivi di efficienza si caricano i dati sul nodo primario risparmiando il tempo di conferma di avvenuta ricezione nel caso in cui si selezionassero più nodi.

La frammentazione in MongoDB viene garantita con un meccanismo di sharding ovvero con la scalabilità orizzontale dei propri elementi. Lo sharding viene eseguito in tre modi differenti: hash-based sharding (sharding sulla chiave) , range-based sharding e tag-aware sharding (sharding su alcuni attributi che assumono dei valori particolari). Il meccanismo fornito è dinamico (pay as you go) ovvero c'è un bilanciamento automatico da parte di MongoDb che aggiunge uno shard nel caso sia superato il volume richiesto per ogni shard. Una chiave di shard suddivide la chiave in vari chunk e il numero massimo di questi chunk che è possibile definire su una sharded key definisce il numero massimo di shard in un sistema.

I ruoli fondamentali che ci sono nel processo della frammentazione sono tre:

1- Router: utilizza MongoS è un software di lancio delle query che va a leggere il Config Server
2- Config Server: un server che ospita i file di configurazione che conosce come sono costituite le shard, ovvero conosce quali i sono i nodi frammentati e replicati
3- Shard: una o più istanze di MongoDB che viene partizionata

Dunque il router quando riceve una interrogazione chiede al config server dove sono situati i frammenti ricercati e successivamente se i dati si trovano su una sola shard esegue una target query, se invece l'interrogazione va eseguita su tutte le shard esgue una broadcast query.

Nella architettura di MongoDB esiste una differenza tra i Primary shard contenente l'intero database e shard secondari che contengono solamente i frammenti e repliche di alcune collezioni.

Le letture utilizzano le politiche di readConcern che possono essere:

- Local: valore di default per la lettura dei dati che viene effettuata senza verificare che il dato sia stato scritto sul resto dei nodi, ovviamente se la richiesta è del nodo primario dato che i dati stanno sullo stesso nodo l'operazione non è distribuita
- Available: valore di default per leggere i dati dai nodi secondari dopo averne verificato la consistenza
- Majority: valore utilizzato per leggere i dati quando la metà più uno delle repliche ha ricevuto l'ack in scrittura sul dato

Le query su MongoDB sono diverse dalle operazioni SQL di fatto assomigliano molto ad un linguaggio di programmazione ad oggetti, segue un esempio:

```
db.user.find({age:{$gt:18}},{name:1,address:1})
```

Dove:

- ({age:{$gt:18}} corrisponde alla clausola WHERE age>18
- user corrisponde a FROM user
- {name:1,address:1} corrisponde a SELECT name, address

Il risultato di una query MongoDB è un documento JSON o una collezione di documenti JSON.

Le operazioni di insert in MongoDB avvengono come segue:

```
db.user.insertOne({
        name:”Andrea”,
        age:26
        Teach[<<datawarehouse>>, <<architecture>>]
})
```

Le operazioni di update in MongoDB avvengono come segue:

```
db.user.updateAll({
        name:{$eq: <<Andrea>>},
        {$set: {Role: <<PA>>}},
        {$upsert:true}
}
```

Qui vado ad aggiornare tutti i documenti che hanno come nome Andrea e vi assegno il ruolo PA e applico una clausola upsert che inserisce il dato se questo non è presente nella tabella e nel caso in cui fosse presente lo va a modificare.

Per effettuare l’import massivo di dati è messa a disposizione la utility mongoimport alla quale bisogna specificare il nome del DB, il nome della collezione, il tipo di dati (CSV o TSV), se si devono ignorare gli spazi vuoti e il path del file.

Le transazioni sono gestite da MongoDB dalla sua versione 4.0 utilizzando le tecnologie di Wired Tiger. Introduce il concetto di log nel nodo primario che comporta l’utilizzo dei lock. I lock possono essere dei seguenti tipi:

- S: un lock di tipo condiviso (sharded) utilizzato per le letture
- X: un lock di tipo esclusivo utilizzato per le scritture
- IS: esprime l’intenzione di bloccare in modo condiviso uno dei nodi che discende dal nodo corrente (intent sharded)
- IX: esprime l’intenzione di bloccare in modo esclusivo uno dei nodi che discende dal nodo corrente (intent exclusive)

QUINTO INCONTRO

DOMANDE E RISPOSTE:

DOMANDA: COME AZIENDE COME AMAZON FACEBOOK GOOGLE SVILUPPANO PER CONTO LORO SISTEMI FONDAMENTALI PER LE LORO ATTIVITÀ E METTONO IL SORGENTE DISPONILI A TUTTI, COMPRENDENDO POTENZIALI NEMICI. COME SE LA MCLAREN PUBBLICASSE I PROGETTI DELLA SUA AUTOMOBILE DI F1 E DESSE L'AUTORIZZAZIONE A TUTTI, ANCHE ALLA FERRARI, DI POTER USARE QUEI PROGETTI.

Queste aziende traggono profitto principalmente da pubblicità e cloud computing, nessuna vende tecnologia. Queste aziende gestiscono una quantità enorme di dati e per fare ciò si sono sviluppati da se delle tecnologie in grado di gestirli, a differenza delle banche che si affidano a terzi per la gestione dei dati. Questa conoscenza creata in casa è stata poi resa pubblica per abituare gli utenti alle proprie tecnologie, ma soprattutto per aumentare la mole di dati che gira all’interno delle loro tecnologie e perché l’unica cosa che conta tenere segreta in realtà sono i dati stessi non la tecnologia che li gestisce.

DOMANDA: COME SI PUÒ MODELLARE IN MANIERA EFFICENTE UN SISTEMA PER LA GESTIONE DELLE REVIEW DEI PRODOTTI?

Si potrebbe pensare di avere un documento sul prodotto e annidare al suo interno tutte le review degli utenti. In questo caso si presenta il problema delle fake review e per ovviare a ciò sarebbe opportuno inserire delle informazioni riguardanti l'utente che ha scritto la recensione. L'idea migliore sarebbe quella di denormalizzare i dati e replicare alcune informazioni sugli utenti che scrivono le recensioni. Un altro problema si presenta nel caso in cui vi sono migliaia di review per un dato prodotto; non è possibile annidare tutte i documenti delle review nel documento di un prodotto, l'opzione migliore sarebbe quella di fare vedere un numero ragionevole di review e, nel caso in cui si volesse visualizzare tutte le review del prodotto, fare una reference ad un elenco più ampio.

GRAPH DB

Il modello a grafo è uno tra i modelli più famosi per la costruzione di DB. Il grafo è una collezione di nodi e archi dove gli archi rappresentano le relazioni fra loro. In un grafo vi sono caratteristiche simili ai modelli relazionali, ovvero i concetti più importanti (le entità in ER) sono rappresentati nei grafi come nodi. La caratteristica fondamentale che contraddistingue il modello a grafo da quello relazionale è come le entità sono collegate fra loro, infatti nei grafi i nodi sono collegati da archi che costruiscono delle relazioni semantiche che associano gli elementi fra di loro. Alcuni esempi di applicazioni sono i social network, le raccomandazioni di prodotti, la fraud detection nelle transazioni, ecc…

Un esempio che marca la fondamentale differenza tra RDBMS e Graph DB lo si può riscontrare nella creazione dei social network. Se ci si pone il problema di trovare gli amici degli amici in un modello relazionale è necessario effettuare diverse operazioni di JOIN causando così il JOIN BOMBING, ovvero il fatto di eseguire un numero elevato di JOIN che richiedono uno sforzo non indifferente. Per quanto riguarda i sistemi document-based invece i dati sono connessi tra di loro mediante un identificatore che punta ad un altro documento (referencing) oppure annidiati (embedding). Nei grafi le relazioni che intercorrono tra due nodi vengono rappresentate tramite le etichette poste sugli archi che li collegano, quindi navigando all'interno del grafo è possibile individuare gli amici degli amici.

DATA MODELING:

La modellazione dei dati avviene come su una "lavagna" ovvero assomiglia molto alle mappe concettuali che identificano il ragionamento umano. Un primo modello di rappresentazione dei dati mediante grafo è il Property Graph che prevede che sia i nodi che le relazioni possano avere delle proprietà (espresse con una chiave/valore) e inoltre accetta che i nodi e le proprietà abbiano un tipo.

Per gestire i grafi o si utilizza un graph database ovvero un dbms che gestisce in maniera persistente un grafo ed esegue delle transazioni, oppure un graph compute engines ossia delle tecnologie per l'analisi offline di grafi.

I graph DB devono supportare le operazione CRUD (Create, Read, Update, Delete) e le interrogazioni vengono risolte tramite l'attraversamento di un grafo che comporta un alta performance (non fanno i JOIN) a differenza dei DB relazionali. Per lo storage dei dati si possono utilizzare dei sistemi nativi o non nativi:

- Native Graph Storage: sistemi ottimizzati per la progettazione e gestione di grafi
- Non Native Graph Storage: sistemi che trasformano i dati in un grafo relazionale object oriented

I graph nativi memorizzano i dati in maniera intelligente, ovvero si cerca di posizionare fisicamente vicini i nodi collegati fra loro da archi, ottenendo così l'index free adjacency che comporta una grande efficienza in fase di lettura ma bassa in scrittura.

Uno tra i vari linguaggi di interrogazione per i graph DB è NEO4J CYPHER. Cypher è un linguaggio di query pattern-matching di tipo dichiarativo che prende spunto dai famosi linguaggi di query. Consente di effettuare operazioni di aggregazione, ordine, limit (trovare le top query) e di update del grafo. Cypher rappresenta un grafo in formato

testuale nel seguente modo: (c)-[:KNOWS]->(b) dove (c) rappresenta il nodo C, [:KNOWS] rappresenta la relazione ed infine -> rappresenta la direzione. L'interrogazione classica di Cypher viene rappresentata come segue:

MATCH (c:user)

WHERE (c)-[:KNOWS]->(b)-[:KNOWS]->(a), (c)-[:KNOWS]->(a), c.user="Michael"

RETURN a,b

Quindi trova tutti gli utenti che conoscono Michael e almeno un altro amico e questo altro amico conosce anche Michael. Il RETURN che indica a e b nonostante questi siano nodi il risultato ottenuto sarà una tabella.

Altre caratteristiche di NEO4J sono:

- Garantisce le proprietà ACID
- Garantisce la scalabilità in senso di alta disponibilità (vengono definiti dei cluster di nodi NEO4J)
- Latenza bassa, ovvero un tempo di risposta basso in quanto le interrogazioni sono dei semplici attraversamenti di nodi del grafo

DIFFERENZE TRA MODELLO RELAZIONALE E MODELLO A GRAFO:

Per evidenziare le differenze consideriamo l'esempio di una server farm dove un utente può accedere ad una applicazione, che gira su una macchina virtuale ospitata dentro un server, per effettuare operazioni su un DB. Se si volesse identificare eventuali componenti malfunzionanti della server farm utilizzando il modello relazionale sarebbe necessario effettuare un numero considerevole di JOIN, mente utilizzando il modello a grafo basta vedere quali asset (server, applicazioni o macchine virtuali), con un grado massimo di profondità nel grafo rispetto al nodo di partenza, possiedono lo status down (ad ogni nodo viene assegnato uno status di up o down). Questa operazione avviene mediante un semplice attraversamento del grafo.

ARANGODB:

Può succedere che in alcune applicazioni reali vengano utilizzati diversi modelli per diverse componenti (es. MongoDB per il catalogo, RDBMS per i dati finanziari e Neo4J per le raccomandazioni). Nelle soluzioni che usano più linguaggi si presenta un problema di persistenza dei dati e molti altri problemi. Per far fronte a tutto ciò ci sono soluzioni come ArangoDB che offre un concetto di database poliglotta; in particolare supporta i modelli key-value, document based, graph based e ricerca in full text. Il linguaggio di interrogazione di ArangoDB si chiama AQL e garantisce la possibilità di interrogare i dati sia con il modello documentale che con il modello a grafo oppure un key-value/text search. I dati sono memorizzati in database e collezioni di documenti ed esistono dei documenti particolari chiamati edges che svolgono la funzione degli archi nei modelli a grafo per interconnettere i nodi fra loro (_from, _to). ArangoDB offre diverse possibilità di scalabilità tra cui:

- Single instance
- Master/Slave
- Active failover: questo meccanismo risolve il problema delle proposte master/slave dove se il master fallisce si blocca tutto. Infatti l'active failover esiste una componente chiamata Agency che nel caso di malfunzionamento di un master si occupa della elezione del nuovo master
- Cluster: introduce degli Agent in maniera analoga all'active failover e in più introduce dei coordinatori che comunicano con i client e si occupano di eseguire le interrogazioni sui DBMS
- Multiple datacenters

KEY VALUE

INTRODUZIONE AL MODELLO KEY-VALUE:

I key-value sono la formulazione più semplice dei modelli di architetture dati dove ad ogni chiave è assegnato un valore. I valori sono considerati come dei blob, ovvero non sono accessibili. Le operazioni che si effettuano sono: inserire un valore data la chiave, trovare un valore data la chiave e cancellare una chiave. Alcune soluzioni come ad esempio Redis consentono di associare anche un tipo al valore (string, int, list…). Avendo un meccanismo a chiave è possibile utilizzare le strutture di hash di conseguenza è possibile ottenere una scalabilità orizzontale con delle DHT. Le operazioni elementari che possono essere effettuate con le key-value sono: inserimento di una coppia, modifica di una coppia, cancellazione di una coppia, trovare il valore associato ad una coppia. In questo modello le tabelle vengono chiamate bucket, le righe key-value e gli identificatori key.

ESEMPIO DI MODELLO KEY-VALUE E UTILITÀ:

Redis è il risultato della ricerca di Salvatore Sanfilippo che voleva trovare una soluzione di real time web analytics in quanto insoddisfatto della lentezza di MySQL. Le chiavi in redis sono stringe in ASCII, i valori primitivi sono delle stringhe e ci sono anche dei Containers di stringhe che possono essere delle hash list, delle liste, delle liste ordinate. Lo sharding avviene effettuando un partizionamento delle chiavi tramite funzioni hash, successivamente le shard vengono distribuite in maniera incrementale, ovvero quando lo spazio disponibile in una macchina si esaurisce viene creato uno shard. Le shard possono essere distribuite anche in modo sparso dove viene settato un numero massimo di nodi utilizzati per la distribuzione delle shards. La High Availability in Redis è applicata in due modi: il primo dove esistono dei nodi su un cluster Redis e altri nodi su un secondo cluster e si scrive solo su uno dei due cluster mentre sul secondo si effettua una sync dei dati asincrona, oppure la soluzione chiamata Active-Active Setup in cui è possibile leggere e scrivere in entrambi i cluster.

WIDE COLUMN STORE

Ultimo modello di NoSQL analizzato nel corso. Si può considerare una evoluzione del modello key-value, infatti accanto alla chiave non vi è un blob ma una riga contenente degli attributi monovalore. Questo modello è stato introdotto da Google con bigTable.

BIGTABLE E HBASE:

bigTable è una mappa multidimensionale ordinata, persistente e sparsa. La mappa viene indicizzata attraverso la chiave della riga, la colonna della riga e il timestamp. La versione distribuita di bigTable viene chiamata HBASE dove le tabelle sono composte da colonne organizzate in column family (ogni column family può avere un numero variabile di colonne). Rispetto al modello relazionale, il modello colonnare non pone particolare attenzione ad i JOIN in quanto per evitare di fare i JOIN si utilizza la denormalizzazione degli elementi. HBASE fornisce delle colonne dinamiche, ovvero il nome delle colonne è codificato nelle celle e quindi celle differenti possono avere colonne differenti. Esiste un version number unico all'interno di ogni chiave. HBASE si base su una architettura master/slave composto da un master, un client e diversi region server. La region è un sottoinsieme delle righe che possono essere partizionate orizzontalmente e il region server gestisce un insieme di righe di una o più tabelle (effettuando operazioni di scrittura) e la sincronizzazione avviene tramite file di log. Esiste un coordinatore tra master e region server denominato ZooKeeper nelle operazioni di lettura e scrittura.

CASSANDRA:

Cassandra è un modello utilizzato da Facebook che si basa su un modello key-value ma che implementa anche i concetti e i modelli delle wide column. In Cassandra esistono dei Key Space all'interno dei quali è possibile definire delle Column family. Ogni column family ha un insieme di rowid e diversamente da HBASE le column family sono equivalenti ad una tabella e le coppie key-value sono come delle righe. Il valore di una riga è una serie di coppie key-

value pairs ossia delle columns, quindi ogni riga contiene sempre almeno una colonna. Il linguaggio di interrogazione è chiamato CQL che è molto simile a SQL. In Cassandra cambia molto l'architettura che è una architettura distribuita P2P. Cassandra utilizza un meccanismo di transparency elasticity che consente di aggiungere o rimuovere nodi in maniera efficace. All'interno di Cassandra vi è una topologia ad anello in cui i valori delle chiavi sono distribuiti ai nodi con delle regole che garantiscono un accesso veloce ai dati. I dati solitamente vengono replicati 3 volte, ovvero ogni volta che c'è un'operazione di scrittura su un nodo questa avviene anche sul nodo successivo nell'anello e anche sul successivo del successivo. I sistemi per individuare dei malfunzionamenti utilizzano dei protocolli di Gossip che prevedono che ogni nodo chieda ai due successivi se stia funzionando tutto regolarmente. Le operazioni di scrittura sono precedute da una scrittura su file di log. Una componente interessante di Cassandra è la possibilità di selezionare delle politiche che impongono delle letture/scritture consistenti; determina quanti nodi devono rispondere per considerare completata correttamente la fase di lettura/scrittura (uno, tutti, metà più uno). Le operazioni di cancellazione non esistono, in realtà i dati vengono resi non disponibili perché è più semplice cambiare un flag piuttosto che cancellare. Questo a lungo andare causa dei problemi di spazio e per far fronte a ciò si utilizzano delle operazioni di merge sovrascrivendo i valori non più disponibili.

SESTO INCONTRO

DOMANDA: COME POSSO MODELLARE IL FATTO CHE UN UNA PERSONA HA PIÙ NUMERI DI TELEFONO (PER LAVORO, CASA, CELLULARE) IN UN GRAFO?

(telefono)-[proprietario: Tipo="Cellulare"]->(studente) <-[proprietario: Tipo="Casa"]-(telefono)

Questa soluzione è ottima nel caso in cui l'applicazione effettua ricerche del tipo: trova tutti i numeri di telefono associati ad una persona.

(telefono)-[Cellulare]->(studente) <-[TelefonoCasa]-(telefono)

Questa soluzione è ottima nel caso in cui l'applicazione effettua ricerche del tipo: trovami tutti i numeri di telefono cellulare associati a più persone.

Quindi la risposta esatta è dipende.

DOMANDA: SI PROPONGA UNA SOLUZIONE MODELLISTICA PER INDIVIUARE IL GRADO DI INTERDISCIPLINARIETÀ DI UN DIPARTIMENTO MEDIANTE LO STUDIO DELLE LORO PUBBLICAZIONI. (È POSSIBILE USARE UN QUALUNQUE DEI MODELLI DATI CONOSCIUTI)

Se si riesce ad associare ad ogni articolo pubblicato la disciplina si riesce ad ottenere il grado di interdisciplinarietà del dipartimento. Si potrebbe anche associare la disciplina allo stesso autore dell'articolo, per fare ciò si potrebbe creare una gerarchia tra le varie discipline e definire successivamente delle parole chiave da assegnare alle discipline. In ambito informatico ci sono degli insiemi di tassonomie che contengono ad esempio tutte le informazioni riguardanti il campo dell'informatica ma questo non risolve il problema di associazione di una disciplina ad una persona/pubblicazione. Si può utilizzare un modello a grafo con: dipartimento1 -> articolo1, dipartimento2-> articolo1, quindi due dipartimenti sono associati ad uno stesso articolo quindi la pubblicazione è interdisciplinare. Il problema è che non è così ovvio che in un dipartimento vi siano persone che si occupano della stessa disciplina (es. ci sono informatici anche nel dipartimento di medicina). Quindi la soluzione migliore sarebbe quella di aggiungere autori al grafo come segue: dipartimento1 -> autori -> articolo1.

DOMANDA: COME POTREBBE AMAZON USARE IL SUO PRODOTTO DI KEY VALUE DYNAMO (ORA DYNAMODB)?

I key-value sono in memory, ovvero NON sono persistenti (se va via la corrente vanno via anche i dati). Quindi Amazon non può utilizzare Dynamo per molte delle operazioni che fornisce con la sua piattaforma. Lo scenario in cui un sistema in memory può essere utile per Amazon è la gestione del carrello, in particolare associare ad ogni persona il

suo carrello. È molto semplice applicare ciò in quanto basta associare un user id ad un carrello, tutte le altre operazioni di pagamento ecc saranno gestite da altre componenti.

ARCHITETTURE DI INTEGRAZIONE

ARCHITETTURE PER DATA INTEGRATION

La data integration è uno step fondamentale nella fase di sviluppo di un progetto che consente di accedere in maniera uniforme e univoca ad un insieme di sorgenti di dati che sono eterogenei e autonomi tramite una vista unificata dei dati. Nello sviluppo di un progetto, una volta raccolti tutti i dati in vari formati ma genericamente sotto forma di tabella, è necessario integrare tutte le tabelle in un unico data set. La prima fase della data integration prevede la scelta dell'architettura da utilizzare per integrare i propri dati. Una prima soluzione possibile per l'integrazione dei dati è il cosiddetto consolidamento dei dati che prevede la raccolta dei dati necessari, l'integrazione degli stessi e infine vado a salvare il risultato in un file o in un DB. Se si volesse rappresentare i vari schemi mantenendo inalterati i dati, senza memorizzare il risultato finale in nessun punto, ma fornire all'utente la possibilità di effettuare on-the-fly delle query che vanno ad interrogare questi dati, bisogna utilizzare un differente approccio. La data integration fornisce diversi vantaggi tra cui la capacità di gestire l'eterogeneità di schemi superiori e l'eterogeneità a livello di istanza rispetto ai sistemi federati.

Vi sono in generale due approcci alla data integration:

- Integrated Rad-Only View: integra i dati lasciandoli nel loro formato originale e cerca di metterli insieme per poterli leggere
- Integrated Read-Write View: integra i dati e fornisce sia la possibilità di leggere che di scrivere/aggiornare i dati

Generalmente l'integrazione dei dati avviene come segue:

1- Dispongo di diversi database (relazionali, documentali, ecc..) ognuno con i loro schemi locali
2- Definisco uno schema globale
3- Tramite mapping associo i concetti degli schemi locali con lo schema globale
4- Quando l'utente effettua una query il sistema a runtime passa dallo schema globale allo schema locale tramite funzioni di mapping

Dunque esistono tre elementi fondamentali nella architettura delle data integration: glli schemi locali, lo schema globale e il mapping che li collega.

WRAPPER E MEDIATOR:

Esistono due componenti fondamentali che consentono la comunicazione tra utente-schema globale-schema locale e sono i Wrapper e i Mediator. I wrapper hanno il compito di agganciare lo schema locale con lo schema globale consentendo la rappresentazione della sorgente come una sorgente descritta con uno schema compatibile con lo schema globale, inoltre ricevono in ingresso le interrogazioni espresse nel linguaggio del modello dello schema globale, le traducono in modo tale che lo schema locale possa risolvere tale interrogazione e infine devono restituirle con lo stesso modello dello schema globale. Il mediatore riceve in ingresso la query e la deve frammentare e riscriverla in modo tale che sia eseguibile nei nodi locali, successivamente deve unire i dati, integrarli, risolvere i problemi di eterogeneità e restituire una risposta all'utente. Certamente un compito fondamentale mediatore è quello di effettuare la riconciliazione delle istanze, ovvero unire le risposte da parte dei vari database locali evitando la replicazione e inconsistenza dei dati. Il mediatore svolge le sue attività in due fasi: la publishing phase e la querying phase. Nella prima fase il mediatore definisce lo schema globale e i mapping partendo dagli schemi locali creando uno schema unificato modellando e definendone i linguaggi, nella seconda fase vengono eseguite le query in base al modello scelto per rappresentare lo schema unificato e tramite i mapping vengono consegnate agli schemi locali e restituite allo schema globale.

SCHEMA GLOBALE E MAPPING:

Vi sono diversi modi per approcciarsi allo schema globale, ovvero come descrivere ogni elemento dello schema globale per fare riferimento agli elementi degli schemi locali e viceversa. Il mapping descrive come ogni elemento degli schemi locali sia associato allo schema globale e viceversa. Lo schema globale definisce delle tabelle virtuali e vi sono diversi approcci per definire tali tabelle e di conseguenza i relativi mapping e sono:

- GAV (Global As View): assume che lo schema globale sia una vista degli schemi locali, quindi una unione di tutti gli elementi degli schemi locali. Ogni singola tabella dello schema globale è una vista degli schemi locali. Il verso del mapping in questo caso va dallo schema locale allo schema globale. Se si dovesse aggiungere una nuova sorgente dati lo schema deve essere ricomposto nuovamente. Le query quindi interrogheranno la vista come se fosse una tabella facendo una operazione di unfolding.
- LAV(Local As View): costruisce uno schema globale indipendentemente dalla sorgente dati e successivamente va a vedere negli schemi locali quali informazioni che necessita prendere. Potrebbe capitare che un dato dello schema locale non sia rappresentato nello schema globale ma per questo approccio la cosa non è rilevante. Il verso del mapping in questo caso va dallo schema globale allo schema locale. Nel caso venga aggiunta una nuova sorgente dati lo schema globale resta invariato. La gestione delle query è più complicata del GAV perché richiede più trasformazioni dei dati per renderli compatibili con gli schemi locali/globale.
- GLAV (Global and Local As View): alcuni concetti sono costruiti con approccio GAV altri con approccio LAV.

DATA INTEGRATION

Come detto precedentemente la data integation sostanzialmente si occupa di creare un unico schema contenete tutti gli elementi di altri schemi unificati e laddove è possibile effettua una operazione di matching. L'operazione di enrichment è leggermente diversa, infatti questa operazione ha come obiettivo arricchire le informazioni di un dato insieme. Dal punto di vista di SQL questa operazione assomiglia ad un left outer join dove si prende tutti gli elementi di una tabella e laddove possibile si aggiunge tutte le informazioni della tabella che serve ad arricchire.

ETEROGENEITÀ:

Negli schemi utilizzati per la data integration vi sono tre tipi di eterogeneità:

1- Eterogeneità dei nomi: legato a come sono etichettati le entità, gli attributi e in generale i concetti di primo livello
2- Eterogeneità dei tipi: legato a come fisicamente viene modellato un pezzo della realtà
3- Eterogeneità di modello: come già visto prevede la scelta di un modello e traduce gli altri modelli affinché siano compatibili con quello scelto

Nelle eterogeneità dei nomi abbiamo i problemi della omonimia, sinonimia e iperonimia. Si ottiene un fenomeno di sinonimia quando si rappresenta lo stesso concetto in maniera diversa che sia una tabella o un attributo (es. corso e esame). L'omonimia identifica dei termini utilizzati per descrivere dei concetti che hanno lo stesso nome ma semantiche diverse (es. città di residenza e città di nascita). Le iperonimie si presentano quando si utilizzano concetti di un livello superiore ad altri (es. persona rispetto a uomo/donna).

Le eterogeneità di tipo portano a differenti problematiche da affrontare:

- Ci sono domini diversi per gli stessi attributi in schemi diversi (es. dominio delle città italiane e dominino delle città francesi)
- Un attributo in uno schema è un valore derivato in un altro schema (es. in uno schema è indipendente, nell'altro dipende da un attributo)
- In uno schema un concetto è rappresentato come un'entità di secondo livello (es. un arco) in un altro schema invece come un'entità di primo livello (es. un nodo, un documento)
- Lo stesso problema di iperonimia della eterogeneità dei nomi

- Differenze di cardinalità
- Differenze di granularità
- Conflitti di chiave

FASI DELLA DATA INTEGRATION:

Le fasi che costituiscono il processo di data integration sono:

1- Trasformazione degli schemi: fase propedeutica per far comunicare gli schemi tra loro che riceve in input tutti gli schemi e produce degli schemi omogenei
2- Investigazione delle corrispondenze: individuo quali concetti di uno schema sono presenti anche in un altro schema
3- Definizione dello schema integrato e regole di generazione dei mapping: genera lo schema integrato e le regole di mapping ricevendo in input gli schemi omogenei e le corrispondenze

L'integrazione degli schemi può essere eseguita secondo diverse strategie che possono essere raggruppate in integrazioni binarie e integrazioni n-arie. Le integrazioni binarie possono essere:

- Composed (a scala): integro uno schema con un altro, il risultato lo integro ad un altro schema e così via fino ad ottenere un unico schema
- Balanced: integro gli schemi vicini tra loro fino ad ottenere un unico schema

Le integrazioni n-arie possono essere:

- One step: integro tutti gli schemi in un unico in una volta sola
- Iterative: integro n schemi alla volta in base alla loro similarità ottenendo degli schemi intermedi che a loro volta saranno integrati fino ad ottenere un unico schema

Identificare le corrispondenze tramite operazioni di matching e quindi identificare i casi di omonimia, sinonimia e iperonimia consente di generare una lista dei possibili conflitti che sono:

- Conflitti di classificazione: riguardano l'iperonimia, ovvero il caso in cui vi sia una generalizzazione (es. fiore con rosa)
- Conflitti di tipo descrittivo: riguardano i nomi delle entità o gli attributi che le descrivono (es. employee con attributo name e surname in uno schema e nell'altro con attributo salary e department).
- Conflitti strutturali: riguardano il tipo degli elementi in schemi differenti (es. in uno schema libro è un'entità, in un altro schema libro è un attributo di autore). Questo problema generalmente si risolve scegliendo di utilizzare la struttura meno vincolata tra le due (es. scegliere libro come entità)
- Conflitti di frammentazione: in uno schema un'entità è rappresentata da una sola tabella, in un altro schema rappresentata da più tabelle. Una semplice soluzione e l'aggregazione delle tabelle

REGOLE DI MAPPING:

Le regole di mapping sono definite tra lo schema integrato e gli schemi sorgente e, tramite l'identificazione dei conflitti all'interno degli schemi sorgente, identificano delle soluzioni ad hoc per gestire i dati in arrivo.

Nel caso in cui in uno schema un dato ha un valore e in un altro schema lo stesso dato ha un valore differente è necessario capire quale mantenere di questi. In questo caso entrano in gioco delle funzioni di risoluzione dei conflitti (funzione MIN/MAX/RANDOM/ECC..) che cercano di restituire il valore più probabile.

Per le istanze ci sono due processi per risolvere i problemi di duplicazione:

- Deduplication: processo in cui si individuano i concetti replicati all'interno di una sola tabella, spesso vengono utilizzate funzioni che calcolano la distanza tra valori per risolvere questi tipi di problemi

- Record linkage: processo per analizzare più tabelle e verificare se vi sono dei concetti replicati al loro interno. Questa operazione riceve in input gli schemi e produce in output o dei match nel caso in cui vi siano dei concetti identici o sufficientemente simili, produce in output anche dei non match di dati e infine produce una lista di possibili match ma non è sicuro (in questo caso si risolve a mano)

Il problema fondamentale del Record linkage è che se si va ad integrare delle tabelle con un alto numero di righe i confronti che si devono eseguire per verificare i possibili match per ogni dato delle tabelle richiedono una quantità molto alta di operazioni di calcolo. Per risolvere questo problema si utilizzano i blocking method, dei metodi che riducono lo spazio di ricerca.

Le fasi che costituiscono il processo di record linkage sono:

1- Preprocessing: attività di normalizzazione dei formati dei dati.
2- Blocking: riduce lo spazio di ricerca
3- Compare: scelta della funzione di distanza da utilizzare per individuare i match
4- Compare: costruzione di un campione di coppie di dati che si sa essere identici e verificare che la funzione di distanza stabilita applicata a questo campione ottenga dei risultati soddisfacenti
5- Decide: verificare che la funzione di distanza stabilita rispetto agli schemi in input
6- Decide: identificare il valore MIN e MAX per identificare un match/non match ed eventualmente un possibile match

RIASSUNTONE DELLE FASI DELLA DATA INTEGRATION:

1- Schema matching
2- Istance level matching
3- Definizione di funzioni di fusione di dati e risoluzione conflitti

SETTIMO INCONTRO

SALTATO MI SEMBRAVA INUTILE RIPRENDEVA LA DATA INTEGRATION.

OTTAVO INCONTRO

DOMANDE E RISPOSTE SU INTEGRAZIONE DATI:

DOMANDA: QUALI SONO LE ETEROGENEITÀ DI SCHEMA POSSIBI LI?

I tipi di eterogeneità possibili negli schemi sono: eterogeneità di schema , di nome e di relazione

DOMANDA: LE ARCHITETTURE DI INTEGRAZIONE DATI DI TIPO WRAPPER MEDIATOR COME POSSONO ESSERE DEFINITE?

Sono delle architetture di integrazione virtuale (in quanto lo schema globale in realtà non esiste fisicamente)

DOMANDA: PROVATE A INTEGRARE GLI SCHEMI DI QUESTI DATASET SUI FILM:

https://www.kaggle.com/shivamb/netflix-shows

https://www.kaggle.com/jrobischon/wikipedia-movie-plots

https://www.kaggle.com/stefanoleone992/imdb-extensive-dataset

OPPURE QUESTI SULLE PROPRIETÀ IMMOBILIARI:

https://www.kaggle.com/dansbecker/melbourne-housing-snapshot

https://www.kaggle.com/htagholdings/aus-real-estate-sales-march-2019-to-april-2020

Per Netflix, Wikipedia e Imdb si nota che ci sono tre tabelle csv che non possono essere tabelle relazionali. Questo perché le tabelle contengono delle liste. I modelli non relazionali che potrebbero rappresentare configurazioni di questo tipo sono i modelli documentali, dato che le liste possono essere trattate solo dal modello documentale (JSON). Una volta definito il modello da utilizzare è possibile passare ad elaborare la integrazione dei dati delle tre tabelle. In questo caso supponiamo che tutte le tabelle contengano le stesse informazioni riguardanti gli stessi film; l'approccio di integrazione più adatto allora sarebbe quello a scala (composed), utilizzando come prima tabella quella di IMDB in quanto teoricamente dovrebbe essere quella più completa delle tre e integrarla prima a Wikipedia e poi il risultato con Netflix.

CASO AOP:

Un consorzio ortofrutticolo (AOP) aveva bisogno che gli creassero un data warehouse coopetitivo (insieme di cooperazione + competizione). Il loro obiettivo era quello di:

- Ottenere una visione globale del mercato
- Effettuare analisi storiche circa l'intera produzione del consorzio (es. quanto vendono rispetto a quanto producono)
- Condurre analisi previsionali per anticipare le richieste della grande distribuzione (GDO)

Per lo sviluppo di questo progetto si sono subito presentati dei problemi:

- Proprietà dei dati: essendo aziende in coopetizione non volevano dare i loro dati (es. a quanto comprano la materia prima rispetto a quanto la vendono) per paura dei competitors, quindi questi dati dovevano restare nelle basi dati delle singole aziende ed essere visualizzati in maniera aggregata e non singola per evitare di divulgare i dati.
- Privacy: informazioni sensibili riguardanti i clienti o i prezzi ad esempio

Da questi problemi si deduce che è necessario un approccio di integrazione di dati virtuale, ovvero implementare un virtual operational data store che costruisce uno schema globale che riceve i dati e anziché materializzarli vengono trasformati e caricati direttamente sul data warehouse.

L'approccio all'integrazione è di tipo top-down in cui per prima cosa si delinea lo schema globale, successivamente con un approccio LAV si identificano i dati utili all'interno degli schemi locali per poi portarli nello schema globale. Per quanto riguarda le istanze è stata concordata un'anagrafica condivisa che descrivesse le istanze di un concetto allo stesso modo per tutti gli schemi locali ed infine è stato definito il mapping che consentiva lo scambio di dati tra schemi locali e schema globale.

Per risolvere il problema della privacy è stata posta la condizione che i dati venissero caricati sul data warehouse solo se almeno 3 partecipanti fornivano i dati, in modo tale che non fosse possibile decomporre il messaggio e ottenere i dati dei competitors.

Non tutti i concetti dello schema globale però potevano essere visti da tutti, perciò si è scelto di comune accordo tra i partecipanti di creare una classificazione dei concetti che definiva le informazioni come:

- Unclassified: informazioni liberamente condivise
- Secret: informazioni condivisibili in forma aggregata e con riduzione della qualità tramite l'inserimento di rumori o delay
- Confidential: informazioni condivisibili solo in forma aggregata
- Top-Secret: informazione non condivisibile

DATA QUALITY

I dati sono una rappresentazione della realtà e la data quality è in un certo senso il grado di accuratezza con cui questi dati sono rappresentati in base all'uso che se ne deve fare. Alcuni esempi di caratteristiche di qualità di dati identificati in coppie di dati simili sono:

- Qualità della fonte: se ad esempio ho una foto di Marte della NASA e una foto di Marte di un sito amatoriale
- Quantità di informazione nei dati: in alcuni casi una eccessiva quantità di informazioni può essere controproducente, in altri casi può essere utile
- Usefullness: la precisione con cui si rappresentano i dati come ad esempio una immagine sfocata e la stessa immagine non sfocata
- Faithfullness: l'aderenza alla realtà dei dati come ad esempio una immagine con nebbia che oscura un cartello e una immagine fotoritoccata senza nebbia dove si vede il cartello (in questo caso meglio quella con la nebbia)

Una delle tante definizioni di data quality dice che un dato è di buona qualità se è adatto all'uso che se ne vuole fare (Fitness for use). In generale la qualità è una caratteristica di un artefatto che soddisfa una necessità o una aspettativa esplicita o implicita. La qualità può essere rappresentata da varie dimensioni che sono delle caratteristiche che definiscono la qualità dell'informazione associata al dato. Le metriche sono un insieme di procedure che consentono di misurare un certo valore; quindi per la stessa misura possono esserci diverse metriche e per la stessa metrica ci sono diversi modi per misurare. Una qualità dei dati è composta da dimensione di qualità e le dimensioni e sotto-dimensioni sono misurate da delle metriche.

Vi sono diverse dimensioni in grado di esprimere la qualità dei dati tra cui l'accuratezza, la completezza e la consistenza. Vi sono moltissime altre dimensioni quindi è comodo raggrupparle in cluster per individuare delle classi di dimensioni di qualità dei dati, i cluster individuati sono:

- Accuratezza/correttezza: dimensione che identifica quanto il dato rappresenti accuratamente il vero valore della realtà
- Completezza/pertinenza: dimensione che identifica quanto il dato rappresenti gli aspetti della realtà che ci interessano
- Ridondanza/minimalità: dimensione che identifica che il dato rappresenta gli aspetti della realtà che ci interessano con il minore numero possibile di risorse
- Consistenza/coerenza: dimensione che identifica quanto il dato abbia delle caratteristiche coerenti ai suoi dati simili
- Comprensibilità: dimensione che identifica quanto il dato sia comprensibile dagli utenti
- Accessibilità: dimensione che identifica quanto il dato possa essere utilizzato con successo dagli utenti
- Volatilità: dimensione che identifica il valore temporale del dato

Quando si va a definire delle dimensioni di qualità, le misure di qualità utilizzate possono essere oggettive (es. individuare i valori nulli) o soggettive (es. i questionari che dipendono dalla percezione umana).

ACCURATEZZA:

L'accuratezza impatta i valori di tipo alfanumerici e si può applicare anche alle tuple e alle relazioni. L'accuratezza è definita come la vicinanza tra un valore V nel data set e il valore V' che rappresenta il vero valore del fenomeno. L'accuratezza si divide in due tipi:

- Accuratezza semantica: definisce un metro di giudizio per identificare se il valore di un dato rappresenta correttamente il valore reale. Questa operazione è molto difficile in quanto bisognerebbe verificare per ogni singolo dato la sua accuratezza.
- Accuratezza sintattica: identifica se il valore inserito è presente in un insieme di valori di riferimento che considero corretti. Ad esempio se si sta parlando di città italiane la reference table sarà una tabella contenete

tutte le città italiane che servirà per verificare se c'è corrispondenza oppure se c'è quantomeno una somiglianza. Le metriche utilizzate per misurare l'accuratezza sintattica, ovvero la distanza tra V e V', si possono basare o sulle stringhe alfanumeriche oppure sui singoli token ottenuti suddividendo la stringa. Le metriche utilizzano delle Edit Distance per calcolare, in base alla loro distanza, quanto due stringhe differiscono tra di loro.

COMPLETEZZA:

La completezza impatta tutti i dati di una base dati. La completezza misura il grado di copertura con la quale un fenomeno osservato è rappresentato nell'insieme di dati. La qualità dei dati si può rappresentare a livello di completezza di tupla, attributo o tabella in base al numero di attributi nulli all'interno delle stesse. Le metriche di qualità restituiscono 0 se vi sono solo valori nulli oppure 1 se il dato è completo. La completezza di oggetti è difficile da determinare in quanto è possibile che in un primo momento si abbiano tabelle con un certo numero di oggetti, ma che successivamente si venga a conoscenza che gli oggetti che dovrebbero essere presenti nella tabella sono maggiori.

VOLATILITÀ O PROPRIETÀ TEMPORALI:

Gli aspetti temporali sono particolarmente complicati e di diversa natura. Un esempio di aspetto temporale è la Currency che misura la rapidità con cui i dati sono aggiornati rispetto al mondo reale. La metrica utilizzata per misurare la currency calcola la differenza tra quando il dato è aggiornato nel mondo reale rispetto a quando il dato è aggiornato nel sistema informativo (sostanzialmente il ritardo con cui viene aggiornato). Un altro tipo di metrica utilizzabile per la currency è definire la stessa come la differenza tra il tempo di arrivo all'organizzazione del dato aggiornato e il tempo in cui tale aggiornamento è effettuato. Tra le varie metriche per misurare la currency da menzionare è anche la differenza rispetto al metadato "ultimo aggiornamento effettuato". Un'altra aspetto temporale è la tempestività che misura quanto i dati sono aggiornati rispetto al processo che li utilizza, questo vuol dire che, a differenza della currency, la tempestività dipende dal processo.

CONSISTENZA:

La consistenza si riferisce ai dati che possiedono dei vincoli di integrità definiti sullo schema (es. il CF deve essere consistente con i dati nome cognome ecc), ma si può anche riferire alle diverse rappresentazioni di uno stesso oggetto della realtà in basi di dati differenti. Per quanto riguarda i vincilo di integrità tra dati ce ne sono molti e alcuni di questi sono:

- Vincoli di integrità in logica: es. un impiegato non può guadagnare più del suo capo
- Data edit nelle indagini statistiche: non puoi avere 10 anni ed essere sposato
- Vincoli di integrità per le dipendenze funzionali
- Vincoli di integrità referenziale

CONCLUSIONE:

Un aspetto importante è che esiste un tradeoff tra le qualità, ovvero ad esempio la consistenza e completezza nel modello relazionale possono non essere conciliabili se si deve rispettare l'integrità referenziale. Nell'ambiente web c'è tradeoff tra tempestività e accuratezza (es. nel meteo dei tornado si privilegia la tempestività).

QUALITY IMPROVEMENT

L'obiettivo della fase del miglioramento della qualità è quello di "pulire" i dati in maniera tale da migliorarne la qualità. Nella fase del miglioramento ci sono due strategie utilizzabili: data-driven in cui si cerca di migliorare il dato in quanto tale e process-driven cerca di migliorare i dati migliorando il processo con cui si raccolgono i dati.

Le strategie process-driven riprogettano i processi di raccolta dei dati affinché i dati stessi ne traggano beneficio. Ci sono due principali tecniche adottate da questo tipo di strategia:

- Process control: inserisce degli elementi di controllo nelle fasi di creazione di nuovi dati, update dei data sets e quando nuovi data set sono utilizzati dal processo. In questo modo è possibile bloccare per tempo e reagire di conseguenza a possibili errori.
- Process redesign: riprogetta il processo di acquisizione dei dati per rimuovere la bassa qualità dei dati e introduce nuove attività che producono dati di qualità superiore

Le strategie data-driven offrono diverse tecniche per migliorare i dati e alcune di queste sono:

- Acquisizione di nuovi dati: migliora i dati acquisendo dati di alta qualità da una nuova fonte
- Record linkage: individuare un altro dataset che contiene la stessa descrizione dell'oggetto reale di bassa qualità presente nel data set attuale e pulire il dato di bassa qualità con le informazioni di alta qualità riguardanti lo stesso oggetto provenienti dall'altro data set
- Sorgenti di dati affidabili: selezionare le data sources sulla base della qualità dei loro dati

In conclusione le strategie process-driven sono più efficienti sul lungo periodo dato che eliminano il problema alla fonte, mentre le strategie data-driven sono più cost efficienti a breve termine.

MIGLIORAMENTO DELLE DIMENSIONI DI QUALITÀ:

- Accuratezza: confrontare i valori con una tabella di riferimento
- Completezza: usare la conoscenza dei dati
- Consistenza: identificare quali sono le relazioni di integrità o di vincolo

Alcune tecniche di miglioramento dei dati sono:

- Localizzazione e correzione degli errori
- Deduplicazione dei dati nello stesso data set. Riceve in input una tabella e raggruppa delle tuple in tre gruppi e le restituisce in una nuova tabella. I tre gruppi sono riconducibili alla somiglianza tra tuple, quindi sono match, non-match e possibile match tra tuple.
- Normalizzazione o denormalizzazione dei dati

RECORD LINKAGE E DATA FUSION

Il record linkage è uno dei task più importanti della data quality che serve a collegare dei dati presenti su schemi differenti che identificano caratteristiche differenti di una o più entità presenti in ogni schema, dove non è possibile collegare le tabelle attraverso procedure standard come il JOIN (es. tabella con informazioni di un cliente, tabella con twitter del cliente, tabella con anagrafica del cliente dove non posso fare un JOIN in quanto non dispongo di una chiave comune). Il record linkage in realtà in base alla situazione risolve diversi tipi di problemi che assumono nomi diversi ma che di fatto applicano un processo molto simile di fusione dati, questi processi sono:

- Record linkage: definito sopra
- Deduplication: nel caso in cui i dati di una sola tabella possiedano informazioni duplicate che descrivono lo stesso oggetto della realtà
- Canonicalization: raccolgo tutti i dati li integro per ottenere il record più completo possibile
- Referencing: viene utilizzato per verificare se in una nuova sorgente sono presenti file già presenti all'interno del DB e, nel caso in cui non lo siano, li aggiunge o eventualmente aggiunge le informazioni aggiuntive non presenti nel DB riguardati quell'oggetto della realtà.

Il record linkage riceve in input un insieme di tabelle e restituisce 3 tipi di output:

- Match di tuple
- Non match di tuple
- Possible match di tuple

CI sono diverse tecniche per il data linkage alcune di queste sono:

- Empirica: si osservano le tabelle e si cerca di capire se ci siano delle corrispondenze o similarità (es. Carlo e Crlo sono simili)
- Probabilistica: utilizza un campione di tuple delle quali si conosce se restituiscono un match o non match e confrontandolo con la tabella si stabiliscono delle soglie per determinare il match o non match
- Knowledge based: utilizza delle regole che devono rispettare le tuple prese in considerazione (es. ho 11 anni non posso essere sposato)
- Mista: utilizza sia la tecnica probabilistica che la knowledge based

Il record linkage può essere effettuato su dataset con differenti tipi di dati come ad esempio dei JSON, delle immagini o delle tabelle. Nel caso delle tabelle il processo di data linkage è il seguente:

1- Pre-processing dei dati dove si cerca di rendere il più possibile uguali gli elementi delle tabelle tramite tecniche di normalizzazione dei dati.
2- Identifico un sistema di blocking che riduce lo spazio di ricerca. Un esempio di approccio al sistema di blocking è il Sorted Neighbour. Questo approccio prende un attributo della tabella e si cerca di ordinare la tabella sugli insiemi degli attributi in modo tale che gli attributi simili stiano vicini, con lo stesso attributo si riordina anche l'altra tabella e poi si confrontano i primi n elementi delle due tabelle.
3- Identifico un sistema di comparison tra gruppi di dati simili e la decisione che si deve prendere per ottenere i tre insiemi di output. Vi sono diversi metodi di comparison e si possono raggruppare in: probabilistic, supervised, unsupervised.

METODI DI COMPARISON:

L'approccio probabilistico cerca di calcolare la probabilità di match tra le coppie di dati utilizzando la somma pesata dei vettori di distanza.

L'approccio supervisionato è stato introdotto recentemente in concomitanza dello sviluppo del machine learning. Alcuni di questi sono: support vector machines, ensembles of classifiers e conditional random fields.

Gli approcci non supervisionati sono: Expectation Maximization e Hierarchical Clustering.

DATA FUSION:

Una volta che si identificano due descrizioni di un oggetto della realtà, appartenenti a due data set differenti, che coincidono bisogna unirle. Vi sono diversi tipi di conflitti nel processo di data fusion e questi solitamente sono o ignorati o evitati o affrontati. Generalmente non si ignorano i conflitti perché questo vorrebbe dire restituire 0. Per evitare i conflitti si utilizzano approcci basati sui metadati o sulle istanze. Infine se si decide di risolvere i problemi conflittuali si deve andare a lavorare o sulla qualità delle sorgenti dati oppure mediando i risultati.

BIG DATA

I Big Data sono un grande volume, con grande velocità e varietà di informazioni. L'analisi di una grande quantità di dati su una singola macchina è costosa, lento e difficile, perciò per i big data si opta spesso per soluzioni distribuite. Le soluzioni distribuite tuttavia portano a problemi di sincronizzazione, deadlock, ecc… Le caratteristiche fondamentali dei big data sono:

- Elaborare grandi quantità di dati
- Scalabilità lineare all'aumentare dei nodi
- Spostare le elaborazioni verso i dati e non viceversa dato che i dati sono così grandi da richiedere troppe risorse di tempo ed elaborazione per trasferirli, quindi è meglio spostare il programma per effettuare l'elaborazione dei dati che pesa pochi mega

- Problemi di estendibilità e failure che devono essere gestiti

L'architettura classica dei big data analytics prevede:

1- Caricare i dati e salvarli in un'area di staging
2- Processare i dati
3- Gestire i dati dal punto di vista della sicurezza, pulizia, monitoraggio
4- Accedere ai dati per ricavarne dei valori

Avere una grande quantità di dati non assicura un lavoro migliore, di fatto è il modo in cui si analizzano i dati non la quantità degli stessi che determina la qualità di un lavoro.

HDFS:

La tecnologia più efficace per memorizzare dei dati di qualunque volume, a qualunque tasso di velocità di arrivo e di qualunque varietà è il file system (come il sistema del nostro PC che salva qualsiasi tipo di dato di qualsiasi dimensione). Per i big data dunque esiste il Hadoop Distributed File System (HDFS) che è un file system distribuito fortemente scalabile e consente la replica di dati tra nodi anche nel caso di fallimento di nodi e infine ha un approccio write once read many. In HDFS la procedura di analisi dei dati avviene nel seguente modo:

1- I file vengono spezzettati in blocchi
2- Ogni blocco viene distribuito su almeno 3 nodi diversi della rete
3- Tramite un'architettura master/slave vengono coordinati gli accessi ai dati
4- I dati vengono elaborati e il client quando fa richiesta si rivolge ai nodi dopo aver ricevuto la posizione del blocco contenente il dato ricercato da un NameNode

Una componente molto importante del HDFS è il NameNode, egli possiede tutti i metadati in memoria, ovvero ha l'elenco dei file, l'elenco dei blocchi, l'elenco dei DataNodes per ogni blocco e gli attributi del dato. Si occupa principalmente di coordinare le operazioni sui dati nei DataNodes, di mantenere in salute il sistema eliminando i nodi morti ad esempio e infine di direzionare i client quando fanno richieste di dati sui nodi interessati.

I DataNode ricevono i dati, ne verificano la correttezza e li salvano per poi consegnarli ai client su richiesta.

MAP REDUCE:

Il map reduce è un modello di programmazione di HDFS che prevede di eseguire dei programmi in parallelo su grandi cluster di macchine. Le attività da svolgere vengono spezzate e distribuite a dei worker che producono un risultato (MAP) e successivamente i diversi risultati vengono uniti in un risultato finale (REDUCE). Questo consente di analizzare dei dati in maniera scalare dato che tanto più sono i worker quanti più dati si possono analizzare in tempi ragionevoli.

HADOOP:

Hadoop possiede un sistema di storage distribuito (HDFS) e un sistema di calcolo distribuito in parallelo (MAP REDUCE). Hadoop venne rilasciato con alcuni software tra cui Pig e Hive. Il primo consentiva di scrivere delle operazioni di trasformazione di dati che venivano tradotti in automatico come dei job di map reduce, il secondo è un'interfaccia di tipo SQL-like per i dati memorizzati su HDFS. Hive per fare delle interrogazioni trasformava le query inserite dagli utenti in uno o più job di map reduce.

YARN:

Nella prima versione di hadoop il jobtracker che assegnava e coordinava le operazioni da svolgere dei vari worker eseguiva troppo lavoro, perciò si decise di disaccoppiare la gestione dei nodi dalla esecuzione dei job sviluppando lo YARN (Yet Another Resource Negotiatior). Vengono introdotti quindi un ResourceManager e un ApplicationManager, che rispettivamente si occupano di gestire del cluster controllato da Hadoop e di gestire le risorse per effettuare i vari job.

NONO INCONTRO

ARCHITETTURE DATI E MACHINE LEARNING:

La pipeline di un sistema di machine learning in produzione è la seguente:

1- Ho dei dati in input di training
2- I dati vengono preparati e validati
3- Viene eseguito il training set per poter addestrare il modello
4- Il modello così addestrato viene utilizzato nella fase applicativa di serving data
5- I dati in input vengono preparati e messi a disposizione per essere dei serving data da integrare con il modello generato e ottenere i risultati che ci si aspetta

Nel machine learning sono molto importanti i dati utilizzati per la fase di training e di serving (TRAINING DATA e SERVING DATA). La maggior parte del tempo è spesa quindi nella data preparation, in quanto i dati sono da modellare in base alle esigenze e tali dati devono essere il più corretti possibile per ottenere dei risultati soddisfacenti. Sono necessarie dunque delle operazioni di data understanding, data visualization e sanity checks (es. controllare i valori minimi e massimi che una variabile può assumere) prima di partire con la fase di training.

La data visualization fornisce degli strumenti di visualizzazione dei dati che consente di osservare graficamente gli elementi che compongono il data set e di comprenderne dunque meglio la natura e la correttezza.

I sanity checks servono per effettuare delle operazioni di pulizia e di controllo dei dati al fine di costruire un training set il meglio possibile. Un esempio può essere quello di controllare i valori massimi e minimi che una variabile può assumere nel data set.

Durante la fase di training non ci si deve fermare con la data understanding, di fatto ci sono tre punti in cui la comprensione dei dati è fondamentale:

1- Feature-based analysis: analizzare la sensitività di una feature rispetto ai dati, ovvero se cambia una feature in che misura cambia anche il risultato
2- Data lifecycle analysis: comprendere se le feature scelte possiedono delle dipendenze fra loro, in quanto potrebbe portare a degli errori
3- Open questions: vi sono varie questioni aperte tra cui la fairness dei modelli di machine learning, ovvero come i dati riescano ad identificare correttamente la realtà

DATA VALIDATION:

Una volta che il sistema va il produzione se il fornitore dati cambia il modo in cui un dato viene rappresentato questo crea dei problemi nel modello di apprendimento non riconosce i nuovi dati in input. Il processo che fa fronte a questa problematica è quello della data validation che consiste nel cambiamento della forma dei dati con cui si effettuano le domande (es. cambiare il modo di misurare l'età). Il modello non è in grado di rispondere a ciò per cui non è stato addestrato, perciò è necessario introdurre un controllo automatico che tramite dei playbook, ovvero delle linee guida per correggere eventuali errori ed effettuare operazioni di data cleaning.

DATA PREPARATION:

La data preparation si occupa di migliorare i dati in input per ottenere performance migliori e prevede operazioni di feature engeneering (es. tramite normalizzazione affinché vengano generati dei dati più friendly per il processo di machine learning) o l'aggiunta di nuovi dati di training nel caso in cui ci si accorge che data set ha pochi dati e quindi darebbe un output poco attendibile.

FRAMMENTAZIONE NEI SISTEMI DISTRIBUITI E INTERROGAZIONI

Esercizio su basi di dati distribuite con lo scopo di partizionare le tabelle della base di dati centralizzata per costruire la base di dati distribuita rispettando determinate specifiche organizzative.

Consider the database:

PRODUCTION (SerialNumber, PartType, Model, Quantity, Machine)

PICKUP (SerialNumber, Lot, Client, SalesPerson, Amount)

CLIENT (Name, City, Address)

SALESPERSON (Name, City, Address)

Il primo obiettivo è frammentare i dati e allocarli in modo corretto. Assumendo che ci siano 4 centri di produzione posti a Dublino, Zurigo, San Jose e Taiwan e che ci siano 3 centri di vendita a Zurigo (Zurigo vende anche a quelli di Dublino), San Jose e Taiwan. Ogni centro di produzione è responsabile di produrre una componente tra le seguenti: CPU, tastiera, monitor, cavi. Assumiamo in fine che ogni zona geografica possiede il proprio DB per un totale di 4 DB.

FRAMMENTAZIONE ORIZZONTALE:

DOMANDA: progettare una frammentazione orizzontale delle tabelle PRODUCTION, PICKUP, CLIENT, SALESPERSON.

La relazione PRODUCTION sarà frammentata in 4 tabelle in quanto le componenti di un PC prodotte sono 4. Segue la frammentazione:

ATTENZIONE SIGMA è il simbolo σ e PROJECTION è un altro simbolo

PRODUCTION_1 = SIGMA_PartType= CPU(PRODUCTION)

PRODUCTION_2 = SIGMA_PartType= Keyboard(PRODUCTION)

PRODUCTION_3 = SIGMA_PartType= Screen(PRODUCTION)

PRODUCTION_4 = SIGMA_PartType= Cable(PRODUCTION)

Per la relazione PICKUP si avrà una frammentazione in 4 tabelle, una per ogni componente del PC, perciò è necessario effettuare un JOIN con la tabella PRODUCTION per poter utilizzare PartType come segue:

PICKUP_1 = PROJECTION_SerialNumber, Lot, Client, SalesPerson, Amount (SIGMA_PartType= CPU(PICKUP

SerialNumber = SerialNumber (PRODUCTION))

Questa è una frammentazione su PICKUP_1 dove si utilizza una proiezione di tutti gli attributi che si trovano in PICKUP e prendo tutte quelle tuple dove il tipo corrisponde a CPU. Si effettua la stessa procedura per le altre 3 frammentazioni.

Per frammentare SALESPERSON e CLIENT il discorso cambia. Nel caso di SALESPERSON si avrà la seguente frammentazione di 3 tabelle una per punto di vendita:

SALESPERSON_1 = SIGMA_City = “San Jose” (SALESPERSON)

E così via per i restanti due punti vendita.

Per la tabella CLIENT ci saranno nuovamente 3 tabelle una per ogni punto vendita, ma attenzione che nella tabella di Zurigo vanno inseriti anche i clienti di Dublino in quanto non c'è punto vendita in quel luogo e questi vengono serviti a Zurigo:

CLIENT _1 = SIGMA_City = "San Jose" (CLIENT)

CLIENT _2 = SIGMA_City = "Zurich" OR City = "Dublin" (CLIENT)

CLIENT _3 = SIGMA_City = "Taiwan" (CLIENT)

Una volta eseguita la frammentazione bisogna decidere la posizione fisica di ogni tabella. Quindi per ogni località avremo le seguenti tabelle:

San Jose -> company.sanJose.com

PRODUCTION_1

PICKUP_1

CLIENT_1

SALESPERSON_1

Allo stesso modo per le altre località con attenzione che Dublino non terrà traccia delle frammentazioni di CLIENT e SALES in quanto non si occupa della vendita.

TRASPARENZA DI FRAMMENTAZIONE E TRASPARENZA DI ALLOCAZIONE:

DOMANDA: determinare la quantità dei prodotti che possiedono il valore 77y6878

Per la trasparenza di frammentazione è necessario effettuare un'interrogazione come se il sistema non fosse frammentato:

Procedure Query1 (:Quan)

Select Quantity in :Quan

From PRODUCTION

Where SerialNumber = 77y6878

End Procedure;

Per la trasparenza di allocazione bisogna fare un'interrogazione che tiene conto dei frammenti:

Procedure Query2 (:Quan)

Select Quantity in :Quan

FROM PRODUCTION_1

Where SerialNumber = 77y6878

If :empty then

Select Quantity in :Quan

FROM PRODUCTION_2

Where SerialNumber = 77y6878

E COSÌ VIA PER LE ALTRE TABELLE.

Per la trasparenza dal linguaggio bisogna tenere conto sia delle frammentazioni che delle diverse allocazioni:

Procedure Query2 (:Quan)

Select Quantity in :Quan

FROM PRODUCTION_1@company.sanJose.com

Where SerialNumber = 77y6878

If :empty then

Select Quantity in :Quan

FROM PRODUCTION_2@company.Zurich.com

Where SerialNumber = 77y6878

E COSÌ VIA PER LE ALTRE TABELLE.

DOMANDA: determinare le macchine usate per la produzione di tastiere vendute al cliente Brown

Per la trasparenza di frammentazione si avrà:

Procedure Query3 (:Machine)

SELECT Machine in :Machine

FROM PRODUCTION JOIN PICKUP on PRODUCTION.SerialNumber=PICKUP.SerialNumber

WHERE PartTyper = Keyboard AND Client= “Brown”

End Procedure;

Per la trasparenza di allocazione si avrà:

Procedure Query3 (:Machine)

SELECT Machine in :Machine

FROM PRODUCTION_2 JOIN PICKUP_2 on PRODUCTION_2.SerialNumber=PICKUP_2.SerialNumber

WHERE Client= “Brown”

End Procedure;

In questo caso non è necessario specificare WHERE PartType= Keyboard in quanto in questa frammentazione ci sono solo tastiere.

DOMANDA: Modificare l’indirizzo del cliente Brown da una via di Dublino ad una via di Taiwan

Per la trasparenza di frammentazione si avrà:

Procedure Update1

Update Client

Set Address= “nuovavia”, City = Taiwan

Where name= “Brown”

End Procedure

Per la trasparenza di allocazione si avrà:

Procedure Update2

Delete CLIENT_2

where name =”Brown”

Insert into CLIENT_3 (Name, Address, City) values (Brown, nomevia, Taiwan)

End Procedure

In questo caso per fare l’update bisogna cancellare l’utente dalla tabella di Dublino e inserire una nuova riga nella frammentazione di Taiwan con i dati dell’utente.

MASSIMIZZARE IL PARALLELISMO INTER-QUERY:

DOMANDA: estrai la somma delle quantità di produzione raggruppate per tipo e modello delle componenti.

La query di base è la seguente:

Select sum(Quantity), Model, PartType

From Production

Group by Model, PartType

Per massimizzare il parallelismo utilizziamo le frammentazioni e quindi facciamo 4 query una per ogni singola tabella, in modo tale da distribuire il carico di calcolo:

Select sum(Quantity), Model, PartType

From PRODUCTION_1

Group by Model, PartType

E così via per le altre tabelle.

DOMANDA: estrai la media delle componenti vendute da salesperson raggruppate per tipo modello delle componenti.

Select avg(Amount), Model, PartType

FROM PICKUP JOIN PRODUCTION on serialnumber = serialnumber

Group by model, part type

MONGODB

MongoDB è un document based DB, nella esercitazione spiega come installare ed utilizzare MongoDB dal prompt dei comandi. Una volta scaricato un DB bisogna caricarlo in MongoDB con il comando:

mongoimport –d test –c movies –drop –file movies-mongo.json (quest'ultimo è il nome del file)

In questo esempio si userà Pyton come linguaggio.

Per prima cosa si importano le librerie di Pyton che ci permettono di implementare MongoDB e successivamente creare una connessione con un client con un nome ed una porta associata:

```
import pymongo

from pymongo import MongoClient

client = MongoClient('localhost', 27017)
```

Una volta creata la connessione sarà possibile accedere al DB del mongoServer. Sarà necessario specificare a quale DB ci si vuole connettere come segue:

```
db= client ['test']

movies = db ['movies']
```

OPERAZIONI CRUD (CREATE, READ, UPDATE, DELETE)

CREATE:

Le operazioni di Create sono:

1- Inserire un singolo documento nella collezione: db.collection.insertOne()
2- Inserire multipli documenti nella collezione: db.collection.insertMany()
3- Inserire un singolo documento o multipli documenti nella collezione: db.collection.insert()

Ovviamente bisogna passare il nome del documento da inserire

ESEMPIO:

```
newmovie = { 'title' : "star trek", 'year' : 1982}

x = movies.insertOne( newmovie)
```

READ:

Le operazioni di Read vengono eseguite utilizzando il metodo find() e tutte le interrogazioni vengono eseguite sulla singola collezione (in questo caso movies) che è composta da una serie di documenti e sta a noi specificare quali vogliamo interrogare:

```
cursor = movies.find({}).limit(1)

for document in cursor:

pprint(document)
```

Qui abbiamo preso un solo documento tramite la funzione find() e abbiamo salvato il risultato in un cursor. Successivamente abbiamo iterato l'operazione per ottenere tutti i documenti contenuti all'interno del cursore.

DOMANDA: estrarre tutti i documenti che hanno come anno 2000 (questo dato è al livello più esterno non sta ne in un array ne in un sottodocumento)

Cursor = movies.find({'year' : 2000})

For document in cursor:

pprint(document)

Se bisogna fare una query che prende informazioni da un sotto-documento all'interno di un documento bisogna fare:

cursor = movies.find({"awards.wins":1}) CHIAMI IL NOME DEL DOC IN CUI RISIEDE QUELLO CHE CERCHI PRIMA DEL . QUINDI OTTIENI COME RISULTATO TUTTI I FILM CHE HANNO VINTO 1 PREMIO, NEL CASO IN CUI WINS FOSSE UN DOC PRENDEVI TUTTO IL DOC

for document in cursor:

pprint(document)

Se vogliamo prendere i documenti che hanno un determinato valore in un Array:

cursor = movies.find({"languages":["English", "French"]}) SPECIFICI QUALI VALORI DEL ARRAY VUOI CHE SIANO PRESENTI QUINDI PRENDI TUTTI I FILM IN LINGUA INGLESE E FRANCESE

for document in cursor:

pprint(document)

Se non vogliamo ottenere un elemento di un documento bisogna settare il suo valore a 0 come segue:

Cursor = movies.find({'year' : 2000}, {'_id' : 0}) QUESTO RESTITUISCE TUTTO TRANNE ID, SE VUOI FAR VEDERE SOLO DETERMINATE COSE METTI COSA VUOI VEDERE CON 1

For document in cursor:

pprint(document)

Per ordinare in maniera crescente o decrescente si usa:

Cursor = movies.find().sort('year' , 1)

For document in cursor:

pprint(document)

Gli operatori di comparazione sono: $eq (un valore uguale a), $gt (maggiore di), $gte (maggiore o uguale),$in (verifica se c'è un valore all'interno di un array), ecc.. Vediamo un esempio:

Cursor = movies.find({'runtime' : {'$gt': 90, '$lt': 120}} , {'title' : 1}) NELLA PRIMA GRAFFA METTO I CRITERI DI SELEZIONE E NELLA SECONDA I DATI CHE VOGLIO VEDERE QUINDI IL TITOLO

For document in cursor:

pprint(document)

Gli operatori logici sono: $and,$not, $nor, $or.

ESEMPIO:

```
cursor = movies.find({'$or':
        [{'tomatoes.viewer.numReviews': {'$gt':95}},{"imdb.votes": {'$gt': 88}}]} QUI FINISCE LA CONDIZIONE DEL OR []
       ,{'_id':0, 'title':1,'tomatoes.viewer.numReviews': 1,'imdb.votes':1}).limit(2)
for document in cursor:
   pprint(document)
```

Esistono anche degli operatori di elementi che sono: $exists, $type.

UPDATE:

Le operazioni di Update sono:

1- Aggiornare un singolo documento: db.collection.updateOne()
2- Aggiornare tutti i documenti che possiedono una caratteristica specifica: db.collection.updateMany()
3- Sostituire un singolo documento che possiede una caratteristica specifica nonostante altri documenti possono avere la stessa condizione: db.collection.repalceOne()
4- Aggiornare o sostituire un singolo documento che soddisfa una determinata condizione: db.collection.update()

Alcuni operatori per la modifica disponibili nelle operazioni di update sono: $rename, $set, $max (aggiorna solo il campo che possiede un valore maggiore di quello specificato) $unset

Segue un esempio di Update:

```
movies.updateOne({‘title’: “Regeneration”},          QUI IDENTIFICO IL DOCUMENTO CHE VOGLIO AGGIORNARE
{‘set’ : {“awards” : {“wins”:8, “nominations” : 14}}})  QUI AGGIORNO IL CAMPO AWARDS CHE È UN DOCUMENTO CON I VALORI CHE SEGUONO
```

DELETE:

Per eliminare documenti dalla collezione è possibile utilizzare i seguenti metodi:

1- Eliminare un singolo documento che soddisfa una determinata condizione: db.collection.deleteOne()
2- Eliminare tutti i documenti che soddisfano una determinata condizione: db.collection.deleteMany()
3- Eliminare un singolo documento o tutti i documenti che soddisfano una determinata condizione: db.collection.remove()

Segue un esempio di Delete:

```
movies.deleteOne({‘title’: “Regeneration})
```

AGGREGAZIONE:

Con l’aggregazione è possibile processare dei record e restituire dei risultati composti (es. raggruppamento su più documenti restituendo un singolo risultato). Per effettuare una **aggregazione su MongoDB si usano i seguenti comandi:**

cursor =movies.aggregate([

{“$match: {“cast”: “Will Smith”}}, NON SI USA PIÙ IL FIND MA AGGREGATE E LE CONDIZIONI VANNO ESPRESSE DOPO IL MATCH

{“$project” : {“_id”:0, “title” : 1}}, PROJECT SERVE PER IDENTIIFCARE COSA VUOI AVERE INDIETRO

{“$limit” : 5}

])

for document in cursor:

pprint(document)

Qui abbiamo preso tutti i documenti dei film in cui Will Smith è stato un attore.

REFERENCING E EMBEDDING:

Utilizziamo un esempio di un dataset che possiede dati sulla compravendita di oltre 5000 oggetti in tutto il mondo negli ultimi 30 anni.

Nel caso in cui si volesse optare per una soluzione di embedding, la scelta migliore sarebbe quella di rappresentare con un documento uno scambio di merce. La struttura di ogni documento del DB sarà la seguente composizione di più documenti innestati in un unico documento:

- Regione o area di appartenenza
- Anno
- ID (generato automaticamente da MongoDB)
- Prodotto: nome, codice, categoria QUINDI UN SOTTODOCUMENTO
- Informazioni sul commercio del prodotto: dimensioni, peso, costo, quantità ALTRO SOTTODOCUMENTO

Nel caso in cui si volesse utilizzare il referencing vi saranno due collezioni di documenti, una per il commercio e una per i prodotti. Segue la composizione delle collezioni:

- Commercio: ID, anno, regione di appartenenza, informazioni scambio [dimensioni, peso, costo] ARRAY
- Prodotto: ID, nome, codice, categoria

In questo caso in qualche modo le due collezioni sono collegate fra loro tramite una chiave comune.

DOMANDA: trova lo scambio più costoso per ogni anno e per ogni paese

SE CI SONO DUE CONDIZIONI DI RAGGRUPPAMENTO CHE COINVOLGONO PIÙ DOCUMENTI (DI OGNI ANNO E PER OGNI PAESE) BISOGNA USARE L’AGGREGAZIONE:

Cursor= trades.aggregate([

{‘$group’ :

{“_id” : { “country” : “$country”, “year” : “$year”}, QUESTO RAGGRUPPA PER ANNO E PAESE

“max_value” : {“$max” : “$tradedetails.tradeUSD”} QUESTO PRENDE LO SCAMBIO PIU COSTOSO

} ,

```
{'$project' : {
        "_id" :0 , "country" : "_id.country", "year" : "_id.year", "mostexpensive" :
"$max_value"}}
])
For document in cursor:
pprint(document)
```

Il simbolo $ si usa per gli operatori o per quando si va a specificare il percorso di un campo. Nella prima parte della query $group serve per raggruppare le informazioni mentre nella seconda seleziono ponendo le condizioni della richiesta.

DOMANDA: identifica se il canada ha venduto più pecore o capre

```
Cursor= trades.aggregate([
        { '$match': { "country": "canada" , "commodity.name" : { '$in' : [ "SHEEP", "GOAT"]}
        }
        },
        '$group': {
        "_id": "$comodity.name",
        "quantity" : { '$sum': "$trade_details.quantity"}
        }
        { '$project' : { "_id" : 0,  "name" : "$_id" , "quantity" : "$quantity"}
])
For document in cursor:
pprint(document)
```

NEO4J

Neo4J è un graph database sul quale vedremo degli esempi. La differenza tra database relazionale e graph database fondamentale è che il primo utilizza le tabelle per rappresentare i dati, mentre il secondo utilizza dei grafi. Nel modello relazionale ogni oggetto o record è identificato da una riga, mentre nel modello a grafo è rappresentato da un nodo.

Il property graph model organizza i dati tramite i nodi, le relazioni e le proprietà sia dei nodi che delle relazioni. I nodi qui sono delle entità del grafo e possono contenere un numero qualsiasi di proprietà (ovvero attributi chiave/valore). Per potere distinguere i nodi fra loro è possibile dargli una etichetta che non solo permette di distinguerli ma consente

anche di inserire i metadati del nodo. Le relazioni collegano due nodi fra loro e hanno una direzione, un tipo e un nodo di inizio e un nodo di fine.

CYPHER:

Il linguaggio di interrogazione utilizzato dai grafi di Neo4J è il Cypher. Il linguaggio utilizza una sintassi di tipo ASCII-ART con le seguenti caratteristiche:

- I nodi sono rappresentati con delle parentesi tonde: (:Movie)
- Le relazioni sono definite con delle parentesi quadre: (:Movie)<-[:RATED]-(:User)
- Le aggregazioni vengono rappresentate in maniera implicita tramite l'utilizzo di funzioni come COUNT
- Esistono dei filtri in grado di limitare lo spazio di ricerca all'interno del grafo. In questo caso gli operatori logici e di confronto tra stringhe sono di comune utilizzo

Alcune tra le keyword utilizzate per interrogare il grafo sono:

- MATCH: ricerca nel grafo un nodo esistente, una relazione, una etichetta, una proprietà o un pattern in maniera molto simile alla SELECT di SQL. MATCH() ricerca qualsiasi tipo di nodo, MATCH (:Label) seleziona un sottoinsieme di nodi.
- RETURN: specifica quale valore o risultato deve essere restituito dalla query. È possibile richiedere di restituire nodi, relazioni, prorprietà o patterns. La funzione di RETURN non è richiesta per le funzioni di scrittura ma è necessaria per le funzioni di lettura. MATCH (node) RETURN node restituisce i nodi che soddisfano la condizione di MATCH. MATCH (node:Label) RETURN node restituisce la variabile node.

Per interrogare le relazioni esistono diversi modi:

- MATCH (node1) --> (node2): prende tutte le relazioni che esistono tra i due nodi
- MATCH (node1) –[:LABEL]->(node2): prende solo il sottoinsieme di relazioni che hanno l'etichetta LABEL
- MATCH (node1) –[r]->(node2) RETURN r: restituisce il risultato delle relazioni tra i nodi
- MATCH (node1) –[r:LABEL]->(node2) RETURN r: restituisci le relazioni di tipo LABEL tra i due nodi

Per rappresentare le proprietà in Cypher si utilizzano le parentesi graffe{} seguono due esempi:

- Proprietà di nodo: (a:Actor {name: 'Jack Black'})
- Proprietà di relazione: -[rel:rated{rating: 4}]->

Neo4J dispone di diverse clausole raggruppabili in: clausole di lettura, clausole di scrittura, clausole generali. Le clausole di lettura sono:

- MATCH: clausola utilizzata per la ricerca di dati con specifiche caratteristiche
- OPTIONAL MATCH: stessa funzione del MATCH solo che può utilizzare i valori NULL nel caso di informazioni mancanti
- WHERE: clausola che consente di estrarre dei sottoinsiemi che soddisfano una determinata condizione
- START: clausola utilizzata per indicare il punto di partenza degli indici
- LOAD CSV: clausola utilizzata per importare dei file CSV nel grafo

Le clausole di scrittura sono:

- CREATE: clausola utilizzata per la creazione di nodi, relazioni e proprietà
- MERGE: clausola che verifica se un determinato pattern esiste nel grafo e, se non è presente, crea il pattern
- SET: clausola utilizzata per aggiornare le etichette o proprietà dei nodi e relazioni
- DELETE: clausola utilizzata per cancellare nodi, relazioni e proprietà
- REMOVE: clausola utilizzata per rimuovere proprietà ed elementi dai nodi e dalle relazioni

Le clausole generali sono:

- RETURN: clausola utilizzata per definire cosa restituire come risultato da una query
- ORDER BY: clausola utilizzata per ordinare l'output di una query
- LIMIT: clausola utilizzata per limitare il numero di righe nell'output risultato
- SKIP: clausola utilizzata per definire da quale riga iniziare ad includere le righe nell'output
- WITH: clausola utilizzata per concatenare le parti della query assieme

Esempi di Create:

CREATE (node_name1, node_name2) utilizzata per creare più nodi

CREATE (node_name:label) utilizzata per creare un nodo con una etichetta

CREATE (node_name:label {key1: value, key2:value}) utilizzata per creare un nodo con delle proprietà

CREATE (node_name1)-[:RelationshipType]->(node_name2) utilizzata per creare delle relazioni tra nodi

Esempi di Set:

MATCH (node:label {properties})

SET node.property = value

RETURN node

Utilizzata per settare una nuova proprietà in un nodo

MATCH (n {properties})

SET n :label

RETURN n

Utilizzato per settare una etichetta su un nodo esistente

Esempi di Delete:

MATCH (n) DETACH DELETE n utilizzata per eliminare tutti i nodi

MATCH (node: label {properties}) DETACH DELETE node utilizzata per eliminare un nodo specifico

Esempi di Remove:

MATCH (node:label {properties})

REMOVE node.property

RETURN node

Utilizzato per rimuovere una proprietà da un nodo

ESEMPIO PRATICO:

Abbiamo un applicativo a grafo che contiene gli attori e i direttori dei film e un dataset con i vari film.

DOMANDA CREATE: inserire le informazioni dei film all'interno del grafo

Prima si creano i nodi sia dei film che degli attori, poi si creano le relazioni popolando il dataset. Ad esempio per ogni film si farà: CREATE (TheMatrix : Movie {title:'The Matrix', released: 1999….}

DOMANDA FIND: trovare l'attore Thom Hanks

MATCH (n:Person) WHERE n.name = "Thom Hanks" RETURN n

DOMANDA QUERY: trovare la lista di tutti i film di Thom Hanks

MATCH (tom:Person {name : "Thom Hanks"}) – [:ACTED_IN]-> (m:Movie)

Return tom, m

NOTARE COME tom E m IN REALTÀ SONO DEGLI OGGETTI CHE MI COSTRUISCO PER POI RESTITUIRE COME RISULTATO

DOMANDA SOLVE: trova tutti i film e attori fino a 4 hops di distanza da Kevin Bacon

MATCH (p:Person { name: "Kevin Bacon"}) – [*1..4]-> (m)

RETURN DISTINCT p,m

Qui praticamente dico che p è la persona di nome Kevin Bacon, la relazione dice che ha distanza da 1 a 4 da m

SECONDO ESEMPIO PRATICO:

Si desidera importare da file CSV un dataset in Neo4J, per farlo si utilizza la seguente sintassi:

USING PERIODIC COMMIT LOAD CSV WITH HEADERS FOM "pathfile" AS row CREATE (:Product {ProductName: row.ProductName, …})

Una volta creati i nodi che compongono il nostro grafo sarà necessario creare le relazioni tra i nodi come segue:

USING PERIODIC COMMIT LOAD CSV WITH HEADERS FOM "pathfile" AS row MATCH (product:Product {ProductName: row.ProductName, …}) MATCH (order:Order {orderID: row.orderID}) MERGE (order)- [:SOLD]-> (product)

DOMANDA: elenca le categorie dei prodotti forniti da ogni fornitore

MATCH (s:Supplier) → (:Product) →(c:Category)

RETURN s.companyName AS Company, collect(DISTINCT c.categoryName) AS Categories

Qui abbiamo collegato supplier e product da una relazione non specificata e product con category con un'altra relazione non specificata. Restituiamo il nome dell'azienda e con collect raccogliamo le categorie in una lista.

DOMANDA: trova un singolo prodotto tramite il suo nome

MATCH (p:Product) WHERE p.ProductName = "NOMEPRODOTTO" return p.ProductName

BIG DATA INTEGRATION – SCHEMA ALIGNMENT

Big Data Integration sono una combinazione degli approcci di data integration applicati ai big data. Quindi ricapitolando la data integration si occupa di fornire un accesso facile e veloce a diverse sorgenti dati, mentre big data riguarda le famose V: Volume, Velocity, Variety, Veracity e Value. La big data integration è necessaria per costruire i grafi di conoscenza di larga scala, effettuare analisi di dati scientifici, ecc..

La small data integration corrisponde a risolvere un puzzle con diversi pezzi e i processi coinvolti sono:

- Schema Allignment: funzioni di mapping dove se ad esempio due sorgenti di dati con schemi diversi vengono mappate per poter comunicare
- Record Linkage: individuare istanze che rappresentano lo stesso oggetto nella realtà nonostante siano rappresentate in maniera differente
- Data Fusion: ha come obiettivo quello di riconciliare i contenuti che sono in mismatch

Per la big data integration i puzzle sono molto più grandi e molto più complicati da risolvere. Vi sono diverse problematiche nuove nella big data integration rispetto alla small data integration e per risolverle vediamo alcuni casi d'uso.

Nel caso in cui avessimo un grande insieme di dati appartenenti ad un unico dominio specifico, una prima domanda potrebbe essere di definire come questi possano essere propagati nel web (es. come i dati sui ristoranti sono propagati nel web tramite i diversi providers) e se è possibile individuare incongruenze tra i dati. Il modo migliore è integrare le sorgenti di dati basandosi su attributi chiave (es. ISBN per i libri) e così si può notare come in base al numero di sorgenti integrate aumenta la percentuale di affidabilità del dato. La seconda domanda posta è come sono collegate fra loro le varie sorgenti dati in un dato dominio. In questo caso si può creare un grafo bipartito dove le entità e le sorgenti sono identificate come nodi e gli archi che le collegano identificano se ad esempio una pagina web contiene una determinata entità. Tramite lo studio del grafo è possibile identificare con che percentuale le entità sono collegate fra loro.

Un altro caso di studio analizza la qualità del web dove vengono studiati due domini uno dello Stock e uno degli Aerei. Per verificare se i dati sono consistenti è necessario analizzare la distribuzione dei data items nei domini e dei loro possibili valori, se abbiamo tanti valori diversi all'aumentare dei data set integrati avremo un problema di inconsistenza. Alcuni risultati di questa analisi potrebbero essere una forte percentuale di dati che possiedono una ambiguità semantica o di istanza.

Ritornando alla big data integration possiamo dire che è molto complicata in quanto a livello delle V abbiamo:

- Volume: abbiamo milioni di siti web con data set di un dominio specifico e di conseguenza milioni di tabelle e di sorgenti dati. Questo comporta una enorme difficoltà nell'eseguire lo schema allignment e rende impossibile in termini di costo l'utilizzo di data warehouse.
- Velocità: i milioni di siti web si aggiornano costantemente e nei relativi database vengono effettuate milioni di query. Questo comporta problemi nella comprensione dell'evoluzione della semantica, l'altissimo costo per l'utilizzo di data warehouse per avere uno storico dei dati e infine di catturare i dati che sono in continuo aggiornamento in maniera precisa ed affidabile.
- Varietà: alcune rappresentazioni sono testuali, altre tabellari, altre in formato video ecc.. Che rendono molto difficile comprendere se i dati sono rappresentati in maniera corretta o meno

SCHEMA ALLIGNMENT:

Lo schema allignment è composto dai seguenti passi:

- Mediated schema: creazione di uno schema che permette di modificare e allineare le informazioni degli attributi delle srogenti in un unico schema mediatore che funge da vista unica per tutte le sorgenti
- Attribute matching: identifica delle corrispondenze tra gli attributi dello schema mediatore e lo schema delle sorgenti dati
- Schema mapping: schema che sta tra lo schema mediatore e la sorgente dati che riformula le query che devono essere indirizzate allo schema mediatore. I tre tipici approcci sono (GAV, LAV, GLAV)

Alcune delle tecniche dello schema alligment che si propongono di risolvere i problemi legati al volume e alla variety dei Big Data sono:

- Integrare i dati del web con delle interfacce di query

- Operazioni di crawling e indicizzazione del web
- Sistemi di dataspace (approccio pay-as-you-go)
- Ricerche su parole chiave

(MANCA UN ESEMPIO DI RICERCHE DI PAROLE CHIAVE)

BIG DATA INTEGRATION – RECORD LINKAGE VARIETY

Il record linkage si occupa principalmente di individuare dei record con delle istanze uguali per poi fare un merge. Gli step da percorrere per effettuare il record linkage sono:

- Blocking: creare dei blocchi che sono dei sottoinsiemi di record simili per ridurre lo spazio di ricerca dei match
- Pairwise Matching: confronta tutte le coppie di record in un blocco e ne computa la similarità. I risultati possibili sono di match, non match oppure indecisione
- Clustering: ha come obiettivo il raggiungimento della consistenza globale e lo fa raggruppando set di record che matchano in delle entità

Le tecniche utilizzate per effettuare il record linkage sono:

- Map reduce: utilizza il modello di map reduce per parallelizzare la computazione di blocking di una grande mole di dati
- Record linkage incrementale: consente di aggiornare i link esistenti quando ci sono degli aggiornamenti dei dati costanti e veloci
- Matching tra dati strutturati e non strutturati: a causa della grande varietà nei big data cercare di matchare tipi di dati rappresentati in maniera differente è fondamentale. Ad esempio le inserzioni di prodotti identici da diversi venditori hanno descrizioni non strutturate e strutturate diverse tra loro che identificano lo stesso prodotto. Queste sono matchate tramite dei tag e dei parse che combinano i tag in maniera tale che ogni attributo abbia dei valori differenti.
- Link temporale dei record: consente di verificare la veridicità dei record

BIG DATA INTEGRATION – TRANSFORMATION & SEMANTIC ENRICHMENT

Le integrazioni di arricchimento della base di dati aiuta a migliorare la completezza dei dati. Le trasformazioni di dati consente di avere un unico tipo di rappresentazione di dati che vanno a comporre la base di dati. Queste due operazioni precedono la fase di record linkage e, seppur non necessarie, nelle soluzioni più moderne sono comunemente utilizzate. Uno dei principali metodi utilizzati per l'arricchimento dei dati è quello del pair matching basato sulle distanze tra chiavi che se soddisfatto consente di arricchire le informazioni associate ad una particolare entità facendo il merge delle due entità che hanno ottenuto un match. Un altro metodo per la trasformazione è quello che utilizza la distanza semantica tra i concetti per migliorare la completezza dei dati; ad esempio è possibile trasformare delle tabelle in modelli a grafo dove le intestazioni delle tabelle vengono identificate con le etichette di relazione dei rami e le celle della tabelle vengono trasformati in dei nodi. Una volta che i dati sono rappresentati allo stesso modo è possibile confrontarli tramite tecniche che utilizzano il calcolo della distanza semantica e, nel caso di match, arrichire il data set con i nuovi dati.

BIG DATA INTEGRATION – TEMPORAL SCOOPING OF FACTS

Quando si utilizzano delle misure come le altezze, eseguire il matching con tecniche di distanza semantica non è utile in quanto non ricondurrebbe a dei risultati corretti. Per questo tipo di dati si utilizza il cosiddetto Temporal Scooping che assegna ad ogni valore un intervallo temporale di validità. Una idea è quella di utilizzare le informazioni

strutturate e non strutturate provenienti dal web per poter assegnare degli intervalli temporali a delle variabili che descrivono una caratteristica identica di una entità (es. Pato che gioca al Milan e Pato che gioca al Corinthias). Le pagine web vengono interrogate utilizzando come parole chiave le informazioni contenute all'interno degli oggetti di cui dobbiamo assegnare l'intervallo temporale. Dopo avere ottenuto diversi valori temporali possibili dalla ricerca web si mappano questi valori in intervalli di tempo possibili e successivamente si prendono i valori che hanno una maggiore ricorrenza per generare l'intervallo di tempo.

BIG DATA INTEGRATION – TABLE INTERPRETATION

Nel web si trovano spesso delle tabelle non interpretabili o interrogabili. L'approccio per risolvere questo tipo di problema propone l'utilizzo della semantic table interpretation che ha come obiettivo quello di estrapolare il contenuto delle tabelle del web e arricchirle con delle annotazioni semantiche (es. un set di dati che identifica il nome dei monti avrà come annotazione montagne) utilizzando delle tecniche di entity linking, disambiguation ecc… Successivamente viene definito un modello descrittivo che permette ai computer di capire i contenuti presenti nelle tabelle non interpretabili e infine trasformare il contenuto di queste tabelle in un linguaggio naturale per gli utenti.

La semantic table interpretation è l'estrazione e arricchimento semantico dei contenuti delle tabelle. Dato un catalogo, ovvero un insieme di gerarchie di tipi con relazioni ed entità e data una tabella T, questa tabella è annotata quando ogni colonna della tabella è associata con uno o più tipi, ogni coppia di colonne sono annotate con una relazione binaria nel catalogo e infine ogni cella della tabella è annotata con degli identificativi (ID).

Vi sono diversi approcci per la semantic table interpretation, in questo caso vedremo le fasi dell'approccio più completo detto enhanced approach che è non supervisionato, completo ed automatico. Le fasi di questo approccio sono:

- Data preparation: prepara i dati
- Column analysis: task che si occupano della classificazione semantica in modo tale da assegnare i tipi alle colonne
- Concept and Datatype Annotation: si occupa di mappare delle colonne
- Predicate annotation: si occupa di individuare le relazioni
- Entity linking: mappa tutte le celle e entità in un Knowledge graph

La data preparation si occupa principalmente di sistemare i dati come togliere le minuscole, rimuovere gli HTML, normalizzare le unità di misura ecc..

La column analysis può assegnare uno di due possibili tipi alle colonne e sono i Named Entities (NE-Column) e Litteral Column(L-Column), successivamente si occupa di indentificare la colonna soggetto S-column. Per identificare le L-Column vengono applicate 16 regular expression per identificare le colonne Literal, tutto ciò che non è literal di conseguenza è un Named Entity Column. Per identificare la colonna soggetto vengono utilizzate delle feature che identificano una Named Entity come soggetto in base a quanto ognuna delle NE identifica in modo univoco le entità della tabella.

La fase di Concept and Datatype Annotation si occupa di fare una mappatura dei concetti. Vengono estratte un insieme di entità candidate per fare entity linking e applicando delle euristiche si individua un concetto più rilevante per ogni colonna e lo si assegna alla stessa.

La fase di Predicate annotation si occupa di definire i predicati e utilizza il concetto assegnato alla S-Column come Soggetto della relazione e tutti gli altri concetti delle altre colonne come gli Oggetti della reazione.

La fase finale di entity linking utilizza le annotazioni scoperte negli step precedenti per creare delle interrogazioni in modo tale da distinguere il contenuto delle celle e quindi consentire il mapping tra le celle e le entità in un Knowledge Graph.

DATA INTEGRATION – DATA FUSION RESOLVING DATA CONFLICTS

Vi sono molti conflitti di dati quando si effettua la data fusion, un semplice esempio può essere l'utilizzo di diversi strumenti per rappresentare le stesse informazioni. Gli errori possono essere intra-sorgenti o extra-sorgenti quindi interni alle stesse sorgenti dati o esterne, ovvero sorgenti che rappresentano lo stesso oggetto della realtà in maniera differente. Per risolvere i data conflicts si utilizza la data fusion delle tabelle e in particolare di operazioni di merge di intere sorgenti dati. Per correggere gli errori presenti nelle tabelle si potrebbe utilizzare una tabella di riferimento contenente valori certamente corretti per rimodellare i dati non corretti, oppure si potrebbe usare la standardizzazione e normalizzazione dei dati.

La data fusion è il processo di integrazione tra due sorgenti dati. Le sue fasi sono:

- Schema mapping: mapping sugli attributi simili tra le sorgenti
- Data transformation: normalizzazione delle istanze delle sorgenti per inserirle in un nuovo schema integrato
- Duplicate detection: identifica due record che rappresentano il medesimo oggetto della realtà tramite l'utilizzo di misure di distanza che riconoscono la similarità tra le informazioni
- Data fusion: fondere i dati duplicati in un unico dato

Alcuni tra i problemi più comuni nella data fusion sono: come affrontare i valori NULL, come affrontare le contraddizioni tra valori di dati ecc… L'incertezza creata dai valori NULL può essere risolta in diversi modi categorizzati in: ignorare il conflitto, evitare il conflitto, risolvere il conflitto. Per risolvere il conflitto si vanno ad analizzare coppie di tuple e queste possono essere:

- Tuple identiche: risolte utilizzando operazioni di unione
- Sottonsiemi di tuple: risolte con operazioni di unione minima
- Tuple complementari: risolte con operazioni di merge
- Tuple conflittuali: risolte con approcci relazionali come match, group, fuse ecc

La data fusion si occupa di risolvere le inconsistenze tra sorgenti dati diverse e si articola in tre passaggi fondamentali:

- Voting: individuare i valori ricorrenti nelle varie sorgenti dati per selezionare il valore più adeguato
- Source quality: individuare la sorgente con la qualità più alta e dare maggiore peso ai valori presenti in quella sorgente rispetto a quelli di una sorgente di bassa qualità
- Copy detection: individuare se le sorgente copiano tra di loro e ridurre il peso dei loro valori

Il metodo di risoluzione dei conflitti di base è il Naive Voting. Questa tecnica è molto funzionale per le sorgenti indipendenti che hanno una accuratezza simile tra loro. Per calcolare l'accuratezza di una sorgente è necessario calcolare una media di tutti i valori di una data sorgente; più precisamente si ha in input un data item D che assume i valori v0,v1…vn e, basandosi sulla regola di Bayes, si calcola la probabilità che i valori del data set siano veri.

SELEZIONE DELLE SORGENTI:

Alle volte non è sempre la scelta migliore integrare un numero maggiore di sorgenti, in quanto l'integrazione di sorgenti che portano dati ridondanti non apportano molto guadagno alla accuratezza e completezza dei dati. Bisogna anche tenere conto della qualità delle sorgenti integrate in quanto se i dati sono di scarsa qualità potrebbero compromettere l'intero database. Il metodo utilizzato per scegliere quali e quante sorgenti dati selezionare per l'integrazione è generalmente un criterio di massimizzare la qualità rispetto al proprio budget.

METODO DI OTTIMIZZAZIONE GRASP:

Una volta scelte le sorgenti si applica un algoritmo di ottimizzazione GRASP. Questo algoritmo consente di effettuare una ricerca ottimizzata in fase di integrazione. GRASP sceglie casualmente la fonte da integrare tra i top k candidati dei quali è stato calcolato il gain marginale, successivamente vi è un miglioramento delle soluzioni applicando una ricerca di tipo hill-climbing.

SANSA: SCALABLE RDF PROCESSING

Esistono molte informazioni strutturate in formato RDF e gestire questi tipi di grandi quantità di dati è complicato da gestire su singole macchine. Per gestire la mole di dati delle big data application l'ecosistema più utilizzato è Hadoop che consente di distribuire il data set su diversi nodi in un cluster e vengono processati in parallelo. SANSA sta per Scalable Semantic Analytics Stack e la sua funzione è quella di combinare il processare i dati eterogenei del semantic web (RDF) con gli algoritmi e le tecniche di machine learning. SANSA consente anche di interrogare una vasta gamma di dati eterogenei che non sono in formato RDF. L'idea è quella di combinare i Big Data con il Semantic Web in quanto i vantaggi di uno sono gli svantaggi dell'altro come ad esempio nella fase di data integration nel caso dei big data è necessaria una operazione manuale di pre-processing molto dispendiosa, mentre nel caso del Semantic Web questa operazione è parzialmente automatizzata e standardizzata.

SANSA STACK:

I layer che compongono SANSA sono:

- Distributed in-memory Processing
- Knowledge Distribution & Representation: detta anche Read Write Layer che permette di avere degli API per effettuare la lettura e scrittura di dati in diversi formati. Una delle tante feature messe a disposizione da questo layer è il OWL Support che consente di processare degli assiomi di OWL.
- Querying: questo layer si occupa delle interrogazioni rivolte ai grafi RDF utilizzando il linguaggio SPARQL. Le interrogazioni vengono eseguite su delle viste che rappresentano delle porzioni del data set. Nel caso in cui vi siano dei dati eterogenei in diversi formati diversi dal RDF l'approccio utilizzato è quello del Data Lake, in questo caso le interrogazioni di SPARQL sono spezzate in sotto query che vengono direzionate tramite mapping RML alle sorgenti interessate.
- Inference: questo layer si occupa di poter inferire delle asserzioni aggiuntive costruendo dinamicamente un grafo delle regole di dipendenza
- Machine Learning: applica degli algoritmi di machine learning distribuiti che lavorano su RDF utilizzandone la struttura e semantica. Alcuni algoritmi utilizzati sono quelli di graph clustering o di association rule mining.