Appunti di Data Management

Leonardo Alchieri

2019/2020

Indice

1 Informazioni Generali

Ci sarà una sfida sponsorizzata dal Sole 24Ore per produrre un'analisi dati con dei premi; verranno fornite maggiori informazioni dopo.

Esame L'esame è formato da uno scritto, 40% del voto, in cui ci avranno delle domande a risposta aperta in cui si chiederà di ragionare sulle cose fatte; e da un progetto che darà il 60% del voto. Dovremo in questi casi trovare dei dataset di nostro piacimento. Dovremo poi fare delle analisi descrittive (non predittive).

Per maggiore libertà, si possono separare progetto e scritto, e valgono per un anno accademico.

Progetto Il progetto deve essere fatto in gruppo, e deve essere consegnato come relazione tecnico-scientifica. Vi sarà poi una presentazione orale, in cui racconteremo che cosa abbiamo fatto.

Come soglia, dobbiamo usare almeno 2GB di dati raccolti, ma dobbiamo pensare come se dovessimo gestire grandi quantità di dati.

Dovremo scegliere 2 delle 3 `V` proposte, ovvero *velocità*, *varietà* e *volume*. [1]

2 Introduzione

Ciclo di vita dei dati La parte essenziale del corso è quella di analizzare il ciclo di vita dei dati, che consiste nella raccolta dati, memorizzazione, trasformazione e integrazione e, successivamente, la costruzione di un modello predittivo. Il corso ovviamente è strettamente legato a quello di Machine Learning.

È estremamente importante come vengono raccolti i dati, anche per evitare che si commettano degli errori sistematici. Ci saranno grandi quantità di dati, che devono essere raccolti e analizzati in tempo reale. Si dovranno catturare i dati, processarli, arricchirli e analizzarli.

[1] Vedi slide per che cosa significa.

3 Data Lifecycle

3.1 Gestire con cura

Piccolo Aneddoto Nel 1974 un professore del Minnesota prese una classe di economia, divise la classe in 2 gruppi, diede al primo informazioni sintetiche e al secondo complete, e scoprì che la parte che aveva i dati grezzi era più convinta delle proprie decisioni, ma erano sbagliato; viceversa, quella con i dati sintetici erano meno sicuri della propria decisione, ma erano giuste.

Quando ci mettiamo ad ascoltare dei campioni, dobbiamo considerare soprattutto da dove li prendiamo. Il fatto di avere tanti dati, non è sempre detto che essi non abbiano un *bias*.

I dati non dicono bugie, ma torturando i dati si possono dire tutte le bugie del mondo. Quando si mettono insieme i dati, se non si è certi, si può produrre solamente spazzatura.

Cosa fare Riuscire a porsi la domanda giusta è fondamentale per poter giungere a delle conclusioni sensate.

Come organizzare i dati È importante capire dove trovare i dati, e come iniziare a trattarli. In particolare, il 70% di un lavoro di analisi dati coinvolge la raccolta e la preparazione di essi.

3.2 Trovare i dati

Dove si possono trovare i dati che servono?

Interni Per prima cosa, vi sono dei dati interni: un'azienda che *vive* ha di per sé dei dati **interni**, che possiede, e.g. informazioni sui dipendenti, dati sulle macchine con sensori, sito web personale, etc. Le aziende sono sanno usare tutte le informazioni che posseggono, e spesso si appoggiano a degli istituti terzi.

Esterni Vi sono poi tutte le fonti di dati **esterne**, e.g. social networks, internet in generale, fonti di open data e altro. Nel mondo del web però non è sempre legale o tollerato poter scaricare i dati (si veda la GDPR).

Fare attenzioni Si deve considerare anche la qualità e il costo per i dati che si cercano: alcune aziende vanno a chiedere spesso molti soldi.

Prendere i dati Ci sono diversi modi per prendere i dati che ci si trova davanti: si possono scaricare, fare una *query SQL*, usare delle API oppure fare *web scraping*. Ci sono dei diversi prodotti *open source* per poter eseguire web scraping, sebbene ci siano diversi meccanismi e strategie per limitare questo fenomeno.

3.3 Comprensione del dato

Questa è la parte più importante, e spesso i dati che si posseggono presentano degli errori, e.g. formattazione dei dati, mancanza di unità di misura, etc.

4 Modellizzazione dei dati

4.1 Che cosa è

Che cosa si intende per *modellizzazione dei dati*? Si tratta del procedimento di prendere i dati e renderli utilizzabili per il fine che si ha.

A seconda del taglio che si va a fare dei dati, si possono ottenere delle informazioni differenti.

4.2 Workload

A seconda di come si utilizza un database, è noto come *workload*, ovvero il confronto tra quanti scrivono e quanti leggono. Spesso si parla di milioni di operazioni al secondo.

Per esempio, ci sono diverse modi per gestire dati se ci sono molte persone che scrivono e molte che leggono, oppure poche che scrivono e molte che leggono.

L'ideale sarebbe che un modello fosse *machine readable*, ovvero che sia direttamente utilizzabile dalla macchine. Purtroppo questo non è sempre possibile, e bisogna usare diverse tecniche per riuscire a rappresentare in una tabella relazionale le informazioni che si hanno a disposizione.

Spesso si può spezzare l'informazione da una tabella in più tabelle: è il grande limite dei modelli relazionali.

I modelli relazionali dovrebbe avere alcune caratteristiche importanti:

- *Machine readable*
- Abbia potere espressivo
- Semplice
- Flessibile
- Standardizzato

4.3 Modello relazionale

La grande intuizione del modello relazionale è quella che si possono spostare le colonne dove si vuole, e si collegano le informazioni tramite la semantica. Questo fa sì che si possano anche collegare diverse tabelle tra di loro, anche se può capitare che non ci sia stessa semantica.

Per riuscire a superare questo problema, si possono collegare colonne diverse attraverso gli stessi codici.

Esistono diversi linguaggi per il modello relazionale, il più importante **SQL**.

Le informazioni possono essere espresse in maniera differente, a seconda di diversi principi. Per esempio, con il *principio minimale* può capitare che si preferisca usare due tabelle piuttosto che una. Viceversa, spesso è più comodo lavorare solamente con una sola tabella, sebbene questa posta presentare nessuna informazione in alcune caselle.

Nelle caselle in cui non si ha alcun tipo di informazione, nella logica relazionale ci sono diversi casi: *vero*, *falso*, *vuoto*.

Nel modello relazionale, tutto quello che si sa è al suo interno: è un *modello chiuso*.

Per collegare due tabelle, spesso si crea una terza che li unisce, ovvero si esegue un'operazione di *join*.

La maggior parte dei modelli tabellari è organizzato tramite dei programmi, e.g. Excell, mySQL, Oracle.

Principi Nato nel 1970, è basato su *assunzione di mondo chiuso* e *principio di minimizzazione*, su cui si basa *SQL*. Questo ha anche dei motivi storici: si cercava di utilizzare meno memoria possibile. A oggi, però, con lo spazio di memorizzazione, serve meno il principio di minimizzazione: viene solamente usato per questioni di qualità, ed è possibile sprecare più spazio rispetto a un tempo.

Un'altra questione è anche la **velocità**: le persone vogliono tutte le cose subito. Il problema del modello relazionale è che è **rigido**, oltre che un problema di *performance*. Spesso il costo computazionale di fare alcune operazioni, e.g. join, non è indifferente.

Scalare Da quando si hanno grossissime quantità di dati, ci sono dei limiti fisici e tecnologici: oltre un certo volume, non si riesce a gestirlo. Per riuscire in parte e risolvere il problema, si fa *scalabirità orizzontale*, ovvero si riempiono degli armadi di computer modulari che si uniscono tra di loro.

NoSQL Con un software ottimizzato, si possono gestire in parallelo i processi che devono essere eseguiti, distribuendoli tra tutte le unità disponibili. Accanto a *SQL* è stata introdotta la semantica **NoSQL**, ovvero per gestire i database in Linux in maniera diversa.

Diverse aziende a partire dal 2000 hanno introdotto diversi modelli, diversi da quelli relazionali, con nomi differenti, per gestire grosse quantità di dati; molti di questi modelli sono stati forniti in maniera gratuita e open access a tutti. Dal 2009 il termine *NoSQL*. Questo modello è nato in ambito di ricerca industriale, spesso da aziende che producono pubblicità (e quindi devono gestire grandi dati).

Con **NoSQL** si intendono dei modelli dati basati su alcuni principi fondamentali:

- *Senza schema*, ovvero si può modificare lo schema senza che si abbia il passato impattato.
- Teorema CAP.
- *Principio BASE*.

  I modelli relazionali sono basati su un principio noto come *ACID*, che garantisce isolamento e persistenza dell'informazione. Questo fa sì che le informazioni siano robuste, e non ci siano sovrapposizioni. In situazioni come la gestione delle banche è essenziale che questo non abbia dei problemi.

  In alcuni casi, però, questo non è per forza essenziale, e.g. i social network: noto come *eventually consistency*.

  Spesso è più importante che il servizio sia sempre disponibile, sebbene non ci sia l'ultimo dato disponibile.

In molti modelli NoSQL si assume l'assunzione di **mondo aperto**.

Un teorema importante, noto come **teorema CAP**, che dice che un sistema non può garantire sempre *consistenza*, *disponibilità* e *tolleranza di partizionamento*, ma solamente due di queste. Il modello relazionale salva consistenza e disponibilità, ma non si ha tolleranza. Si possono quindi costruire sistemi differenti in NoSQL.

A seconda di quello che si deve fare, si possono andare a scegliere diversi modelli. Piuttosto che avere un unico modello, si ha oggi una grandissima libertà, a seconda del problema che ci si trova davanti.

4.4 Modelli NoSQL

Vedremo alcuni modelli importanti basati sui principi NoSQL:

- *Key-Value Stores*
- *Column Friendly Stores*
- *Document Databases*
- *Graph Databases*
- *RDF databeses*

Key-Value È un database che gestisce solamente una chiave e un valore: si può cercare solamente per chiave, a cui è associato un valore. Sono dei sistemi subito disponibili e altamente scalabili.

Come esempio, un sito di eshopping deve associare a un singolo utente un singolo carrello, e serve subito e in maniera efficiente.

Un modello simile è stato inventato da un Italiano, noto come *Revis*, utile per avere immediatamente dei valori in memoria.

Column Introdotti nel 2006, da BigTable di Google, tramite una singola chiave sono associati diversi valori, riferiti a delle entry di una colonna. Il sistema è altamente scalabile.

Grafi Il modello a grafi è uno dei modelli più vecchi, basato sulla *teoria dei grafi*. Questo modello viene usato per collegare dei nodi, a cui sono associati delle chiavi: spesso si va a cercare il cammino minimo.

Document Il più importante è JSON, o*Java Script Object Notation*, un formato che viene usato per gestire chiavi a cui sono associati più dati, organizzati in maniera gerarchica.

Paragone tra modelli *Vedi slide del prof per i primi 10min della lezione.* Tutti i modelli forniscono lo stesso tipo di informazione, anche se sono trattati in maniera differente.

5 Modello documentale, JSON

Modello ad albero gerarchico, può possedere anche un documento, a cui sono associati dei sotto-documenti, e così via; si ottiene quindi uno schema di un albero.

Quale scegliere Il modello documentale è caratterizzato da **embedding**. Si può comunque scegliere se organizzare per *referecing*, tipico del tradizionale modello relazionale, oppure embedding, tipico del documentale. La scelta tra le due configurazioni dipende da quanto alcuni elementi devono essere usati: se si deve continuamente cercare due cose insieme, la tecnica *embedding* risulta più comoda computazionalmente.

Nessuna scelta è sbagliata, ma in base al tipo di analisi che si deve fare si ottengono dei risultati più o meno migliori. Il modello documentale, avendo l'**assunzione di mondo aperto**, può anche presentare casi in cui si ha mancanza di informazione. Aggiungo solamente gli elementi dove servono, e non esiste uno schema generale.

La prima difficoltà è che non sappiamo a priori come è fatto lo schema: senza la conoscenza di dati non si può sapere tutti i valori. In questo caso, ci si deve basare sulla documentazione o su alcuni programmi che identificano quali siano tutte le categorie.

Efficienza L'efficienza del modello non è lineare, ma risulta esponenziale: l'algoritmo, che è molto semplice, va ogni volta a controllare già la presenza dei valori. Con numeri troppo grandi, ci mette troppo tempo: per memorizzare in JSON cambio come descrivo il modello dati.

Il modo più semplice è quello di duplicare i dati, risparmiando tempo in copiatura: in questo modo si ha un andamento lineare. In questa maniera, al posto che avere *embedding*, ho un dato per ogni coppia.

Specie quando si ha un grande volume di dati, a volte si deve lavorare in maniera contro-intuitiva: in alcuni contesti, quando il volume di dati è troppo grosso, è molto più facile replicare, al fine di ottimizzare i tempi di calcolo.

MongoDB A differenza del modello relazionale, nei sistemi documentali esistono vari prodotti: uno dei più utilizzati è **MongoDB**. I dati sono memorizzati in formato *JSON binario*(Bson). Vedi informazioni per confronto per i nomi.

Esistono altri modelli, con funzionamento leggermente differente.

6 Modello a grafi

Introduzione Il grafo è un sistema di reti che collegano diversi punti, detti nodi. Il modo in cui i nodi sono connessi viene modellato tramite i grafi. Si può usare un nodo per rappresentare un nodo, e collegarlo ad altri tramite una qualche relazione. La teoria dei grafi è stata sviluppata a fine '700, per cui vi è molta letteratura in questo ambito.

Le applicazioni sono svariate, e.g. social network, informazioni geografiche, transazioni finanziare e altro.

Quando si rappresentano grafi con solo due nodi, *persone* e *oggetti*, si parla di **grafi bi-partiti**.

Anch'esso, come il modello documentale, è **schema-less**, ovvero si descrivono le caratteristiche e schema insieme.

Grafi in altri modelli Si possono memorizzare nativamente i dati in maniera *grafo*, oppure rappresentarli con altri modelli e poi applicare dei grafi. Per esempio, si può rappresentare un grafo anche con modello relazionale e/o documentale, e.g. RangoDB. Quando si hanno però una grossa quantità di nodi,[2] questo sistema diventa inefficiente, e i database nativi a grafo risultano molto più veloci.

Modello ad albero In alcuni casi, il modello ad albero riesce a sostituire un grafo; quando però si è in presenza di cicli, non è possibile. Vale però fino a un certo punto: si possono duplicare e/o mettere riferimenti.

Le tecnologie basate sui nodi hanno comunque un problema: più i dati sono divisi in nodi, maggiore è la velocità. Il problema principale è che *i grafi non scalano*: se infatti il grafo è connesso, non si riesce a distribuire i dati. Il vantaggio di eseguire richieste in parallelo si può fare solamente in modelli documentali, mentre è difficili fare interrogazioni in parallelo in un grafo.

Modellizzazione dei grafi Un grafo, oltre a **nodi** e **archi**, si possono dare dalle **proprietà** ai diversi nodi. Gli archi legano i nodi tramite delle proprietà. Il modello a grafo è quindi molto vicino al modello relazionale.

Esistono ovviamente delle scelte modellistiche, a seconda dell'obiettivo e del tipo di dato che si ha a disposizione.

Da un punto di vista della visualizzazione, è molto semplice creare un grafo: sono delle semplici mappe, e rappresenta in maniera efficace le informazioni.

Linguaggi a grafi Il modello a grafo, oltre alla difficoltà di avere dei linguaggi di interrogazione diversi, presenta approcci diversi per scrivere "a macchina" in grafo.

Cypher Lavora a *pattern-matching*, ovvero cerca in base a dei pattern tipici tra tutti i sottografi. È possibile indicare dei nodi, archi e caratteristiche possedute da entrambi. Quando si fa una ricerca, si può ottenere o un grafo o una tabella di valori, che rovina la logica *modello-risultato ricerca*.

[2] Noto come `JOIN bobbing`.

Gremlin Si tratta di un linguaggio per *attraversare grafi*, ovvero si ha come risultato un nodo che si ottiene dopo aver percorso un particolare cammino. La logica per l'interrogazione è molto diversa rispetto a *Cypher*. Si deve in questo caso specificare **come arrivare** al vertice.

7 Altri modelli

Modello key value I modelli sono legati da delle chiavi; non si può fare molto, e ovviamente tutto è legato alla chiave. Implementando la funzione di *hashing* permette di passare dallo spazio delle chiavi a uno spazio delle hashi, e permette di rendere il modello molto scalabile.

Wide column Nasce storicamente da `Google's Big Table`. Una riga è una mappa multidimensionale caratterizzata da una chiave, una colonna e un *timestamp*.

HBase è organizzato con diverse famiglie di colonne, legate a diverse parole chiave. Tutte le chiavi devono avere le stesse famiglie di colonne, ma possono avere diverse colonne. Scala molto bene, e su di esso si appoggiano altri sistemi.

8 Integrazione e arricchimento dei dati

Integrazione Dati due dataset, si può andare a capire come sono fatti e integrarli. Per l'integrazione dei dati, si hanno due insiemi di dati, non per forza disgiunti. Da un punto di vista pratico, in **SQL** si tratta di eseguire in *full outer join*; ovviamente quello che si esegue dipende dal tipo di modello che si usa.

Arricchimento Con arricchimento, si aggiungono a un insieme di dati delle informazioni provenienti da un altro insieme. In **SQL** questo corrisponde a eseguire un *left outer join*. In qualche maniera, il concetto di arricchimento prevede un meccanismo di matching tra i due dataset differente dall'uguaglianza, ovvero spesso in maniera più rilassata.

Risolvere eterogeneità Nella pratica è più comodo decidere a priori quale modello usare e poi eseguire l'integrazione. Si possono ovviamente avere casi diversi:

- **Sinonimia**, ovvero dove attributi sono sinonimi e il contenuto degli elementi è lo stesso.
- **Omonimia**, ovvero termini usati per descrivere stessi concetti ma presentano informazioni differenti. Integrare tabelle con semantiche diverse, si può incorrere in problemi.
- **Iperonimie**, ovvero quando si hanno dei tipi di relazioni all'interno degli schemi che si stanno integrando.

È molto importante avere a disposizione la documentazione dei dati.

Azioni di integrazione Quando si hanno due set di dati, si dovrà andare a trasformare lo schema, fare matching tra i due schemi e andare a integrare questi, in maniera sensata. Questo ovviamente vale sia in modello relazionale sia in altri tipi di modelli. I sistemi di integrazione dati prendono delle sorgenti e li mettono insieme, cercando di ottenere uno schema integrato che abbia tutte le informazioni del primo e del secondo schema insieme.

La complessità dello *schema matching* può essere molto alta, e i problemi possono essere affrontati in maniera molto diversa.

Numerosità di schemi Per poter integrare o arricchire una grande quantità di schemi, ci sono diverse strategie, che prevedono due scelte fondamentali:

- Integrazione binaria a coppie. Si ottiene una soluzione **a scala**, e ci si trova sempre nelle condizioni di dover lavorare con due schemi.
- Integrazione a gruppi di N, spesso se hanno a che fare con le stesse cose.

Relativismo semantico Si può spesso rappresentare lo stesso concetto in maniera diversa: dipende da come si deve utilizzare.

Classificazione dei conflitti Quando si esegue *matchibg*, si possono avere dei conflitti. Esistono alcuni conflitti di tipo **descrittivo**, in particolare dei nomi che descrivono in maniera diversa le stesse cose; oppure di tipo **booooh**.

Risoluzione conflitti Ci possono essere problemi di frammentazione durante l'integrazione: alcuni tabelle possono essere divise in due tabelle differenti, e si cerca di unirle insieme. Questo tipo di gestione di conflitti da valore all'integrazione dei dati.

Mapping Rules Una volta definiti gli schemi che si hanno e la metodologia che si deve applicare per l'integrazione, si possono definire delle regole per eseguire l'operazione, e.g. come legare nomi diversi tra due schemi. Su progetti su grande scala, è buona norma codificare quello che si esegue in una documentazione a supporto.

Integrazione Istanze Se l'integrazione degli schemi è complicata, quella delle istanze lo è ancora di più. Questo problema appare sia nel caso di integrazione e di arricchimento: bisogna capire come le informazioni combacino.

Conflitto di chiave In alcuni casi non si riesce a capire se un dato descritto in un dataset viene descritto anche nell'altro dataset. Per risolvere a questo problema, esistono le **funzioni di risoluzione dei conflitti**, spesso del tipo *min* o *max*. Può essere una buona pratica di etichettare i casi di conflitto, per esempio per andare a vedere a mano questi casi (*human in the loop*).

Per le istanze, ci sono due soluzioni:

- **Deduplication**, ovvero se ci sono nomi del mondo reale che siano duplicati. In questo caso, non si ha uno *schema matching*.
- **Integrazione tra due tabelle**, come visto.

Deduplicazione In alcuni casi, delle tabelle presentano descrizioni dello stesso del mondo reale più volte. Per esempio, si può calcolare la distanza in caratteri tra differenti nomi, e si estraggono delle probabilità. Ovviamente rimangono tutte considerazioni probabilistiche, e non si ha certezza che le correzioni sia esatte.

Il problema è ancora aperto e dipende dalle situazioni che si incontrano: a oggi, le tecniche più avanzate usano algoritmi di Machine Learning per risolvere il problema.

Alcune tecniche, come il **blocking**, cercano di ridurre lo spazio di ricerca, dividendolo a blocchi.

9 Velocità

Mi manca una lezione.

Oramai ci sono una serie di attività per cui si può parlare si streaming data in tempo reale; è anche però difficile definire in maniera precisa tutto questo. Ci sono una serie di architetture volte alla gestione di dati che arrivano con grande velocità.

Per esempio, sistemi come quella a guida autonoma devono prendere decisioni immediate da dei dati che provengono in tempo reale. Ci sono quindi due aspetti: da un lato l'acquisizione in tempo reale, che devono prendere informazione immediata, e i sistemi di gestione e computazione, che devono lavorare il più velocemente possibile su questi dati.

Tempo reale A seconda del caso che si ha, il termine *tempo reale* può cambiare. Per esempio, un sistema di guida autonoma deve avere un tempo di reazione di pochi secondi mentre un sistema di rilevamento del riempimento dei cassonetti può anche avere un tempo di reazione di qualche ora.

Acquisizione È molto importante capire quando "pescare" i dati al momento giusto. Si deve quindi trovare il modo per catturarli e immagazzinarli, all'interno di una *repository*.

Tools Da un punto di vista informatico, per l'acquisizione dati serve un *middlewear*, ovvero una *pipeline* che riesca a gestire i dati in tempo reale. Un sistema di questi è **Apache Kafka**.

In questo, i personaggi fondamentali sono *producer*, ovvero colui che scrive sulla coda di Kafka, e un *consumer*. Questo meccanismo deve consentire al *producer* di scrivere il più velocemente possibile, e il consumatore deve avere possibilità di leggere con più calma. Vengono disaccoppiati i dati che si producono in tempo reale.

Questo sistema è noto come *publish-subscribe*.

Oltre a Kafka, si possono usare protocolli come $NPTT$, usato nelle interazioni macchina-macchina. Questo meccanismo è molto utile per i sensori, pensati per durare molto a lungo con un dispendio molto piccolo di energia.

Architettura lambda Quando si ha la necessità di elaborare, oltre che raccogliere, dati in tempi reali, si possono seguire alcuni *pattern procedurali*. L'architettura ricorda le funzioni **lambda** di `Python`. Dopo aver raccolto i dati in tempo reale, si usa uno *speed layer* che, appena riceve dati, usa delle librerie di programmazione pensate per lavorare in streaming, e.g. **Spark**.

Una volta elaborati i dati, si vanno a modificare alcuni parametri relativi ai dati. Nel qual caso però serva comunque mantenere l'informazione, in contemporanea si ha un *batch layer*, e.g. **Hadoop**, che memorizza i dati in maniera stabile.

Quando poi serve qualcosa, si possono unire i dati analizzati in tempo reale con quelli che vengono immagazzinati in maniera procedurale.

Questa soluzione architetturale è stata spesso criticata, in quanto la logica deve essere implementata due volte, spesso con librerie differenti.

Architettura Kappa Nominata dall'iniziale della persona che l'ha pensata, si basa sull'implementazione dei due layer insieme: sono messe una dopo l'altra le due attività, quella di elaborazione in *real time* e quella del *batch layer*. In questa maniera, il codice per trasformare i dati viene scritto una sola volta.

Questa architettura, come anche la *lambda*, sono *general purpose*, ovvero applicabili a tutti i tipi di problemi. Non sono però le uniche soluzioni possibili, orientate a obiettivi diversi.

Elastic Stack L'ecosistema ELK, formato da *kibana*, *elastisearch* e *logstash* sono alcuni sistemi utili per fare analisi dati in *real time*. L'obiettivo di questo ecosistema è quello di **monitorare** in tempo reale i dati, senza avere alcuna complessa analisi.

Elasticsearch è un motore di ricerca, che trova dati per parola chiave. La versione di questo sistema è scalabile, a cui è associata la velocità di esecuzione.

Logstash è un linguaggio di scripting per caricare i dati e metterli su *Hadhoop*.

Kibana serve invece per fare analitica in tempo reale e gestire i dati. Il sistema è molto semplice e interattivo.

Graphana graphite Nato per gestire infrastruttura di serve, **Graphana** è un visualizzatore che mostra dati in tempo reale da parte dei sensori di un computer. I dati vengono immagazzinati in dataframe temporali.

Graphana supporta un componente software noto come *Graphite*, il quale aiuta a leggere i dati.

10 Dataware houses

La **data integration** risulta una soluzione *lazy* per connettere i dati, fatta tramite il **warehouse**, molto più potente. Le integrazioni *lazy*, come mostrate prima, usano un *wrapper* per trasformare il modello e consentiva di interrogare direttamente il sistema.

La costruzione di una **warehouse** è molto lungo e il sistema è molto rigido, ovvero non si ha la possibilità di essere flessibili coi dati a disposizione. L'interrogazione risulta molto veloce, in quanto si ha solamente uno schema, e si possono anche eseguire richieste complesse. La sicurezza è anche aumentata e, ancora oggi, aziende medio-grandi utilizzano queste tecnologie.

Alcune aziende usano delle soluzioni Big Data che nascono al di fuori del mondo relazionale, tecnologia di aggregazione più utilizzata.

Hub-and-spoke Tipo di architettura dati in cui tutto viene mandato a un grosso centro dati, per poi essere fornito "a raggiera" a tutto ciò che serve.

Alcune aziende però hanno bisogno di velocità maggiori di trattamento dei dati. In particolare, si può usare un sistema in cui ci sono più centri di raccolta. Può ovviamente capitare che i dati non siano sempre coerenti tra di loro.

Il datawarehouse riesce ad arricchire le informazioni che si hanno a disposizione rispetto ai normali dati che si hanno.

OLAP Noti come sono fatti i dati, si deve capire come poterci girare. Una serie di Operatori OLAP consentono di navigare in essi, per poter lavorare.

DFM Da un lato ci sono gli attributi descrittivi, che danno informazioni aggiuntive.

Attributi cross-dimensionali: attributi il cui valore dipende da quello assunto da due valori diversi, e.g. IVA.

Gerarchie condivise.

From DFM to Star Schema È importante sapere come si passa dal modello DFM al modello relazionale. Ovvero si dovrebbe avere necessità di accedere ai dati in forma tabellare. L'operazione che si deve fare per eseguire questo passaggio si basa sulla **progettazione logica**: si ha un modo per passare da un modello concettuale di alto livello a uno logico.

Il concetto essenziale si basa sulla **ridondanza dei dati**: per poter andare velocemente, è necessario avere ridondanza e de-normalizzare le informazioni. In questa maniera, si ha una più grande efficienza nell'eseguire un `join`.

•