



## Database Technologies prima e seconda rivoluzione

### Prima rivoluzione dei database

- Il primo DBMS appare sui mainframe computeri
- I primi modelli utilizzati erano:
  - Network model
  - modello gerarchico

### Seconda rivoluzione dei database

- Il modello relazionale appare in una prima implementazione detta System R (IBM)

- Vengono definiti i principi fondamentali dei database:

- Chiarissima distinzione tra rappresentazione logica e fisica dei dati
- Transazioni e proprietà ACID

### Terza rivoluzione dei database:

- I DBMS relazionali non erano adatti a gestire i volumi e le velocità dei dati che Google deve affrontare
- Nel 2003 Google introduce GFS (Google file system), un nuovo file system distribuito. Nel 2006 svela MapReduce, l'algoritmo di elaborazione parallela e distribuita. Nel 2006 invece rivelò i dettagli di BigTable, il suo database strutturato distribuito
- Unendo tutti questi concetti nacque nel 2007 il progetto Hadoop iniziò in Yahoo!

### Sfide di Amazon agli RDBMS

- siti web e di e-commerce hanno bisogno di gestire transazioni su larga scala.
- Oracle-RAC ha tentato di offrire una soluzione per la scalabilità illimitata, ma non ha avuto successo
- La soluzione adottata è lo "sharding", ovvero la suddivisione orizzontale dei dati su più database in base ad un attributo chiave (ad esempio, customer-id). I dati degli utenti Facebook e Twitter sono stati suddivisi su un numero molto elevato di database MySQL. La maggior parte dei dati di un singolo cliente risiedeva su un unico database
- Lo sharding presenta però degli svantaggi importanti: molte operazioni relazionali e le proprietà ACID vengono perse dunque infatti impossibile fare join e mantenere integrità transazionale fra shard
- Nel 2008 Amazon ha proposto una nuova soluzione: Dynamo il primo database chiave-valore

### Sfide del cloud computing agli RDBMS

- Tra il 2006 e il 2008 Amazon ha introdotto Elastic Compute Cloud (EC2) che metteva a disposizione immagini di macchine virtuali ospitate sull'hardware di Amazon. La piattaforma è nota come Amazon Web Services, prima implementazione del modello Infrastructure as a Service.
- Gli RDBMS non erano adatti alla scalabilità elastica di tale piattaforma. Amazon ha quindi proposto SimpleDB e DynamoDB come soluzioni per la gestione dei dati.

### Sfide dei document database agli RDBMS

- L'adozione degli strumenti di mapping object-relational hanno risolto solo parzialmente il disallineamento tra modelli orientati agli oggetti e modelli relazionali.
- Si usa il linguaggio di programmazione AJAX. Si basa su Javascript e comunica con un backend trasferendo messaggi in formato XML.
- Sono stati sviluppati nuovi sistemi di database in grado di memorizzare direttamente documenti JSON come MongoDB e CouchDB.

## Evoluzione degli RDBMS verso i sistemi NoSQL

- Nel 2007 si sostiene che le ipotesi hardware alla base dell'architettura relazionale non erano più applicabili.
- Si ritiene che contesti applicativi diversi richiedono architetture fisiche differenti.

2 approcci proposti:

- H-Store: un database distribuito puramente in memoria
- C-store: un database columnare

Si svilupperono numerosi nuovi sistemi di database. Possono essere raggruppati in una nuova classe chiamata: sistemi di gestione di database distribuiti non relazionali.

Prima rivoluzione: conseguenza dell'aumento dei computer elettronici:

- Solo design fisico del DB
- Dati bloccati nel sistema
- Accesso ai dati riservato solo ai programmati

Seconda rivoluzione: introduzione di una base solida formale e matematica per i DB

- Progettazione logica indipendente da quella fisica
- Mecanismo di query flessibile, linguaggi dichiarativi SQL
- Oltre 20 anni di dominio commerciale

Terza rivoluzione: non si basa su una singola architettura. Un'unica architettura per i database non è più sufficiente per affrontare le sfide del mondo digitale odierno.

Social network con centinaia di milioni di utenti e IoT, con miliardi di macchine che producono flussi di dati mettendo a dura prova i sistemi RDBMS

Le nuove applicazioni richiedono la gestione di dati con struttura e dimensioni imprevedibili.  
I professionisti dei DB dovranno scegliere con cura la tecnologia più appropriata per le loro applicazioni.  
Non sempre la tecnologia relazionale sarà la scelta migliore.

## Big Data e Hadoop

Big data è un termine ampio e con molteplici definizioni. Identificativi sono 2 eventi chiave che cambiano il ruolo dei dati nell'informatica e nella società:

- Maggior quantità di dati: la tecnologia ci fornisce la capacità di archiviare ed elaborare tutti i dati: generati da macchine, multimediali, provenienti da social network e transazioni
- Maggior impatto: le soluzioni disponibili per il machine learning, l'analisi predittiva e altre analisi consentono di estrarre più valore dai dati a disposizione

L'aumento del volume dei dati è sempre stata una questione critica nella storia dei DBMS. Ma ora si è decantati a una vera e propria "rivoluzione industriale" nel mondo dei dati.

Prima tutti i dati venivano prodotti internamente, creati dai sistemi operativi interni. Ora invece provengono da qualsiasi luogo: clienti, social network, sensori.

Ciò che ha alimentato la generazione di nuovi dati:

- Cloud
- Mobile
- Social Media

Tecnologie che hanno contribuito alla generazione di nuovi dati:

- Cloud computing
- Social network
- Smartphones e IoT



## Approccio di Google alla gestione dei big data

Il primo approccio di Google è stato l'utilizzo di un insieme di server standard, ma ha presto abbandonato questa soluzione.

Google ha creato una nuova architettura hardware e software in grado di scalare in modo esponenziale e illimitato.

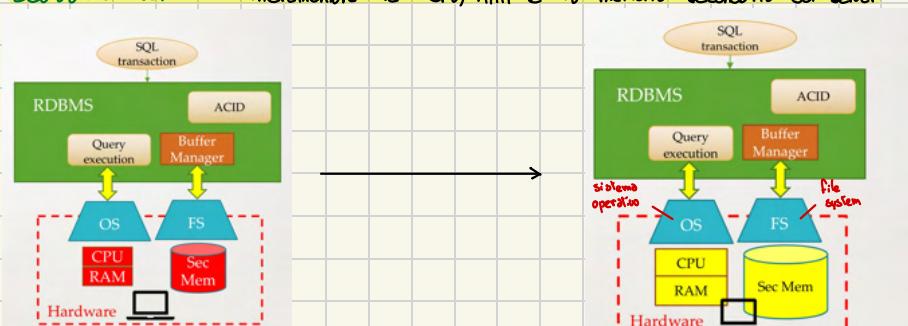
Per quanto riguarda l'architettura HDFS la soluzione di Google si basa sull'unità di elaborazione fondamentale chiamata **Google Modular Data Center**.

Per l'architettura SQL gli elementi chiave della riduzione di Google sono: **Google File System** (un nuovo file system distribuito),

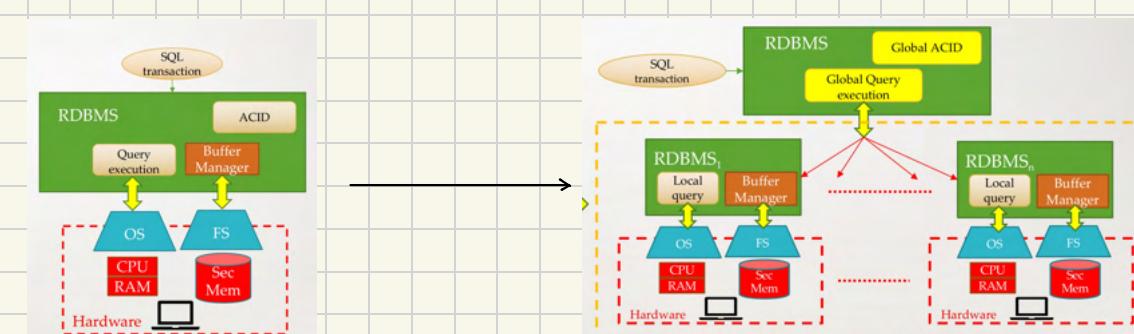
**MapReduce framework** (un nuovo approccio per l'elaborazione distribuita) e **BigTable** (un database non relazionale).

## Come poi scalare un RDBMS?

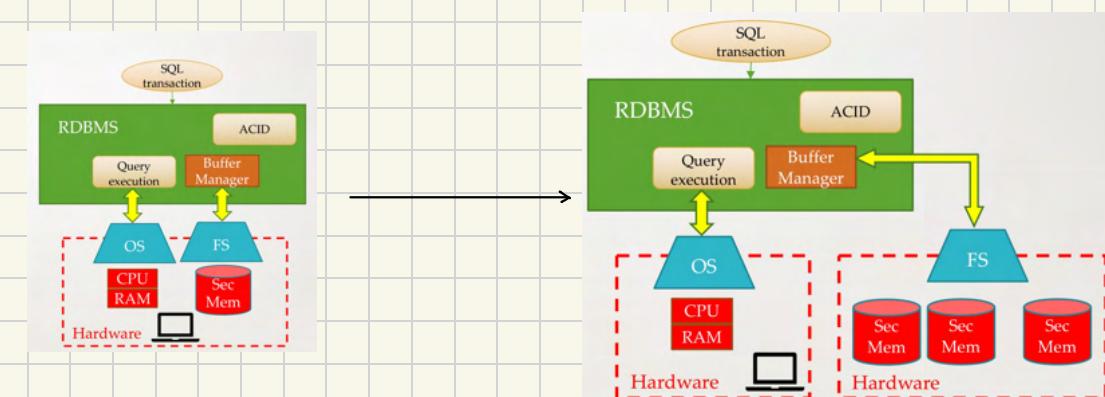
**Scalabilità verticale:** incrementare le CPU, RAM e la memoria secondaria del server.



**Scalabilità orizzontale:** aumentare il numero di macchine (single server → multiple servers)



**Scalabilità verticale:** aumentare le risorse della memoria secondaria elaborando esternamente la gestione della memoria secondaria

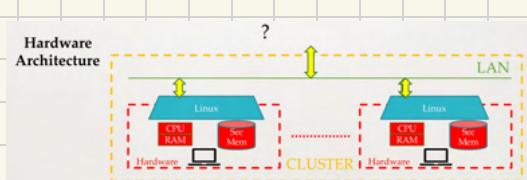


Nessuna di queste implementazioni è la soluzione corretta o migliore.

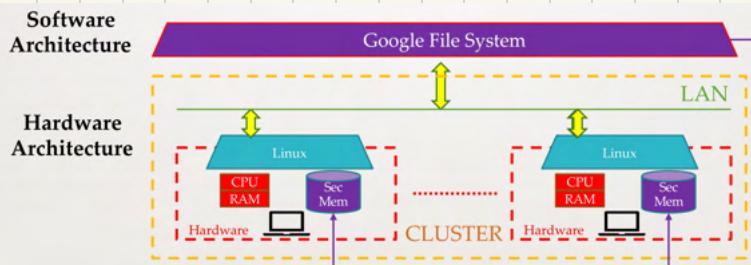
Google ha ridisegnato la gestione dei dati rifiutando completamente l'architettura hardware e software dei tradizionali sistemi DBMS per creare una nuova generazione.

**Google Data Modular Data Center:** è un container che ospita migliaia di server Intel che eseguono Linux.

**Data Center:** un insieme di Google Modular Data Center chiamato anche cluster.



Google ha introdotto un file system distribuito per cluster che consente di accedere a tutte le unità della memoria secondaria del cluster come se fossero un unico file system enorme, distribuito e ridondante.



• La ridondanza consiste nella replica dei dati, in genere si ne fanno 3 copie, il numero di copie può variare ma in genere non è necessario.

E' più facile, con le repliche, che si verifichi l'inconsistenza dei dati.

La caratteristica principale del filesystem distribuito di Google è la ridondanza dei dati: vengono memorizzate copie multiple dei dati su server multipli del cluster.

Questo permette di ottenere:

- (i). affidabilità (avendo la conservazione dei dati in caso di guasti)
- (ii). applicabilità di un modello di elaborazione scalabile

### MapReduce

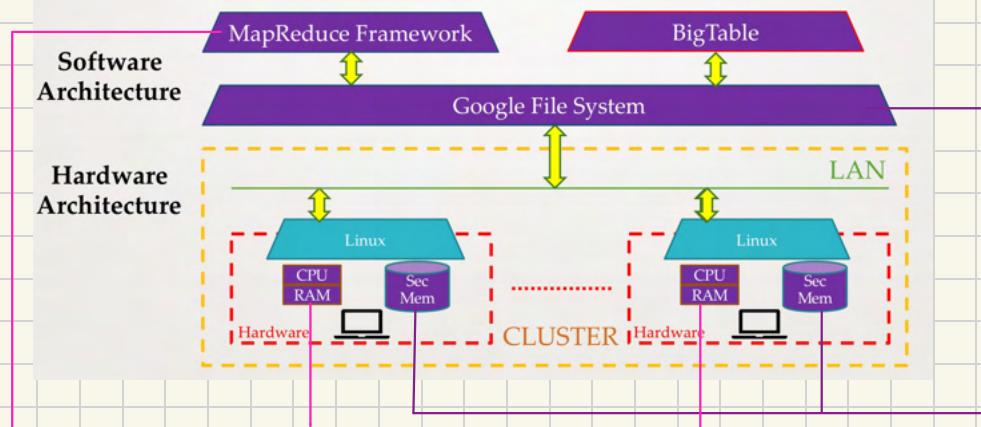
Google introduce un nuovo framework per l'elaborazione distribuita di algoritmi parallelizzati.

Cioè permette:

- (i). l'esecuzione parallela su un vasto numero di server non affidabili
- (ii). l'applicazione di algoritmi complessi a dati set enormi.

### BigTable

Google introduce un database system non relazionale che utilizza Google file system per lo storage.



### Approccio di Google

Google ha sviluppato il progetto essenziale per ciascuno di questi componenti: in una serie di paper. Da questi documenti è nato Hadoop, un progetto open source di Apache. L'architettura innovativa di Google è stata adattata da Hadoop. Hadoop è stato reso disponibile per il download nel 2007. Nel 2012 il cluster Hadoop di Facebook superava i 100 petabyte di memoria secondaria, affermandosi come sostegno di data warehousing superando completamente Oracle.

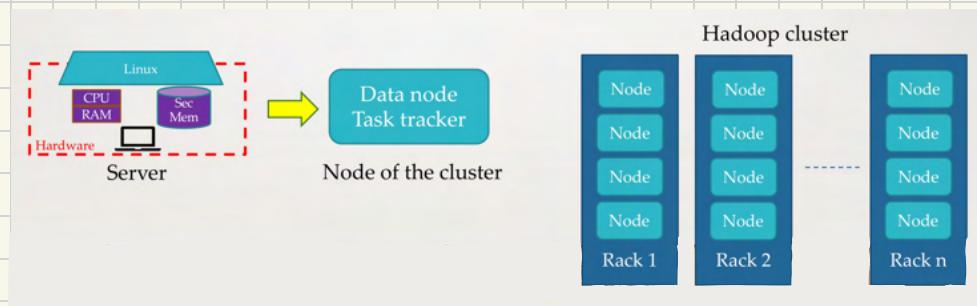
### Vantaggi di Hadoop per le applicazioni Big Data

- Modello di storage scalabile ed economico
- Elevata capacità di I/O scalabile
- Affidabilità
- Modello di elaborazione Scalabile
- Schema on read

## Architettura di Hadoop

- I componenti dell'architettura di Hadoop sono:
  - (i) Il file system distribuito di Hadoop (**HDFS**)
  - (ii) Il MapReduce framework
  - (iii) Il primo sistema database noSQL formale (**HBase**)

Ogni nodo di un Hadoop Cluster è un server che fa parte da data node che da task tracker.



- **JobTracker:** coordina la pianificazione dei job eseguiti su cluster
- **NameNode:** è una sorta di directory che fornisce la mappatura dei blocchi sui Data node ai file
- **Data node:** contiene una replica dei dati da elaborare
- **TaskTracker:** esegue una determinata attività sui DataNode

Nell'architettura di Hadoop 2.0 è stato introdotto un nuovo modulo per la pianificazione e l'allocatione delle risorse, chiamato **YARN**.

Questo modulo consente a Hadoop di eseguire task basati su modelli di elaborazione più complessi, andando oltre il framework **MapReduce**.

## MapReduce

Il paradigma **MapReduce**: fornisce un modello di programmazione che astrae il problema delle letture e scritture su disco, trasformandolo in un calcolo su insiemi di chiavi e valori.

Modello dati di MapReduce: si basa sui seguenti costrutti:

• **Record:** è l'elemento fondamentale del modello per la rappresentazione dei dati. Ogni record è una coppia:  $(k, v)$ , dove  $k$  è un'istanza della chiave e  $v$  è un valore con una struttura sconosciuta al modello. Si noti che la chiave può essere univoca o meno, quindi non è paragonabile alla chiave primaria di una tabella relazionale.

• **File di record:** è il costrutto che consente di definire raccolte di istanze di dati, rappresentate da record.  
Ogni file è una lista di record:  $[(k_1, v_1), (k_2, v_2), \dots]$

• Il modello dati di MapReduce appartiene alla famiglia dei modelli **KEY-VALUE**.

La caratteristica importante di MapReduce è che si divide il calcolo in due parti:

- Una fase di mapping, in cui i dati vengono suddivisi in chunk che possono essere elaborati da un insieme di task MAP
- Una fase di reduce, in cui uno o più task REDUCE combinano l'output dei mapper nel risultato finale
- Entrambe vengono eseguite in parallelo, ma molto frequentemente ci sono molte task map che elaborano una porzione del dataset e una task reduce che successivamente combina i risultati prodotti dalle task map
- MapReduce è un processore di query batch

## Esempio pratico

```
0057  
332130 # USAF weather station identifier  
99999 # WBAN weather station identifier  
19500101 # observation date  
0300 # observation time  
4  
+51317 # latitude (degrees x 1000) Vindicante  
+028783 # longitude (degrees x 1000)  
FM-12  
+0171 # elevation (meters)  
99999  
V020  
320 # wind direction (degrees)  
1 # quality code  
N  
0072  
1  
00450 # sky ceiling height (meters)  
1 # quality code  
C  
N  
010000 # visibility distance (meters)  
1 # quality code  
N  
9  
-0128 # air temperature (degrees Celsius x 10)  
1 # quality code  
-0139 # dew point temperature (degrees Celsius x 10)  
1 # quality code  
10268 # atmospheric pressure (hectopascals x 10)  
1 # quality code
```

Programma che raccoglie dati meteorologici. I sensori raccolgono dati ogni ora in molte località del mondo e generano un grande volume di dati. Questo è un esempio di file di record.

queerry: Qual è la temperatura più alta registrata per ogni anno nel dataset?

Bisogna esprimere la query come un job MapReduce.

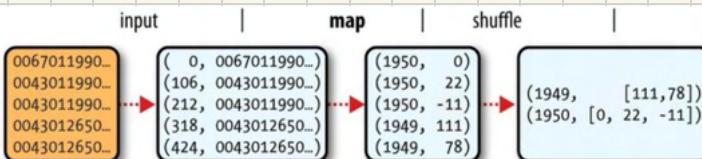
Ogni file ha coppie chiave-valore come input e output, i cui tipi possono essere scelti dal programmatore. In alcuni casi, la chiave può essere inutile come nel nostro esempio.

Il programmatore deve specificare due funzioni: la funzione di map e la funzione di reduce.

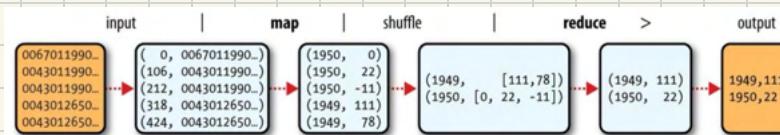
Nell'esempio pratico, ogni file di input è organizzato in coppie chiave-valore. Utilizza come valore una riga e come chiave l'offset della riga nel file.

La nostra funzione di map è semplice:

- Estraie l'anno e la temperatura dell'aria, quindi è solo una fase di preparazione dei dati per la funzione di reduce. La funzione di map può anche filtrare temperature mancanti, non valide o errate.
- Produce un elenco di coppie chiave-valore come mostrato di seguito: si noti che il risultato del mapper ha un contenuto diverso rispetto ai file di input, è sempre un file contenente record con la struttura <anno, temperatura>, ma ora la chiave è l'anno e il valore è la temperatura.
- L'output del task map è processato dal MapReduce framework per essere mandato al reduce task.
- Il processo chiamato shuffle ordina e raggruppa le coppie chiave-valore in base alla chiave.



- Alla fine ogni anno compare con una lista di tutte le sue letture della temperatura dell'aria.
- Cambia dunque ancora il contenuto del risultato: la chiave è l'anno come prima, ma il valore è una lista di temperature.
- Ora per ogni anno la reduce function deve iterare attraverso le liste e prendere il massimo letto.



## Esecuzione di un job MapReduce su Hadoop

- Un job MapReduce è un unità di lavoro che l'utente vuole eseguire. È composto da:

- Dati in input
- Programma MapReduce
- Informazioni di configurazione

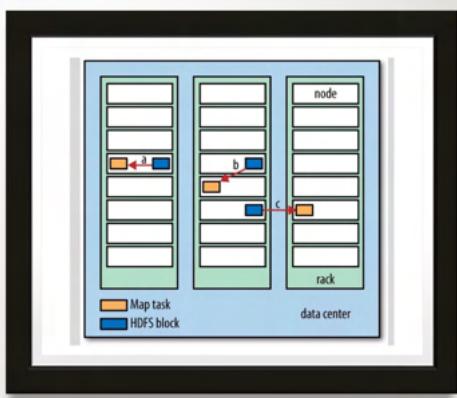
- Hadoop esegue il job suddividendolo in task: task di map e task di reduce.
- I task vengono schedulati da YARN ed eseguiti sui nodi del cluster.
- Hadoop divide il file di input in pezzi di dimensione fissa chiamati SPLIT.
- Hadoop crea una task di map per ogni split, che esegue la funzione di map definita dall'utente per ogni coppia chiave-valore nello split.
- La dimensione dello split determina il numero di task di map, in genere corrisponde al blocco HDFS, attorno 64 o 128 MB.

- Hadoop dovrebbe eseguire le task di map sul nodo in cui risiedono i dati di input in HDFS, in modo da minimizzare l'utilizzo della banda del cluster (i nodi sono collegati tramite la LAN del cluster). Questa operazione è chiamata ottimizzazione della località dei dati.
- A volte la località non può essere raggiunta, quindi le task di map devono lavorare su uno split archiviato su un altro nodo.

HADOOP MAP TASKS EXECUTION ON A CLUSTER  
a: locality, map task and split on the same node

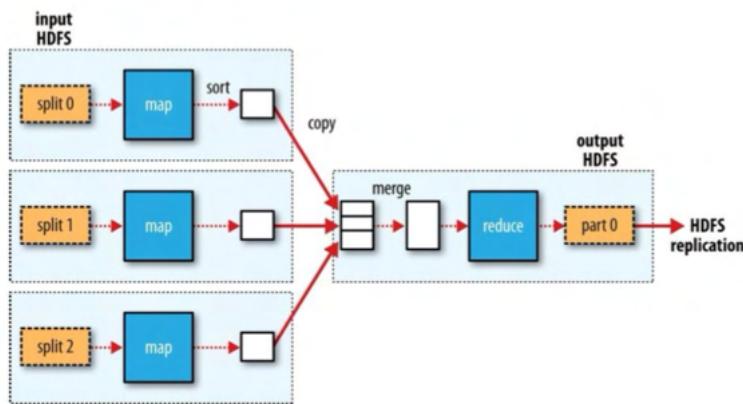
b: map task and split on different nodes of the same rack

c: map task and split on different nodes of different racks

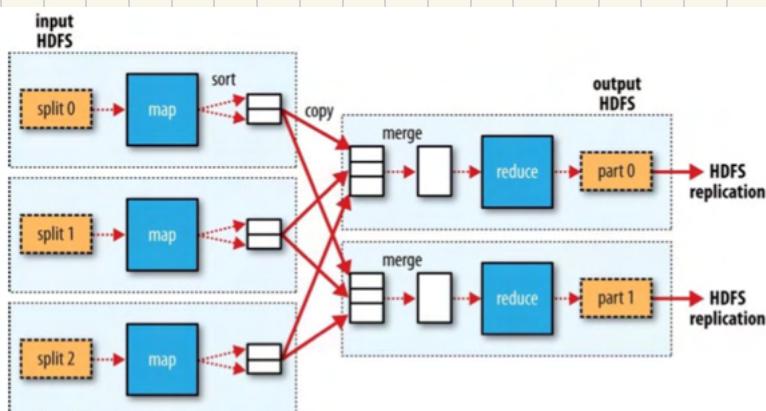


- Le task di map stampneranno il loro output sul disco locale.
- Il loro output viene processato dalle reduce task per produrre l'output finale, una volta che il task reduce ha completato il suo lavoro l'output del map può essere gettato.
- Se molte reduce tasks sono schedolate allora le task di map devono partizionare il loro output. Con le regole generate che il record per una data chiave deve trovarsi tutto in una singola partizione.

### Single reduce task



### Reduce Task Multiple



## Hadoop Interface

La piattaforma Hadoop non avrebbe potuto raggiungere il suo pieno potenziale se solo le persone in grado di scrivere programmi MapReduce avessero potuto accedere al sistema.

- Gli utenti non programmati avevano bisogno di uno strumento di query flessibili, potente e accessibile per estrarre dati da Hadoop
- Due soluzioni a questo problema sono state sviluppate indipendentemente da Facebook e Yahoo!: Hive e Pig

## HIVE

Generalmente conosciuto come SQL per Hadoop

È la prima soluzione che adotta un linguaggio di query simile a SQL per un sistema completamente diverso dagli RDBMS

Per farlo funzionare:

- È necessario mappare le tabelle del mondo SQL nei file key-value di Hadoop
- È necessario implementare un modulo software che traduce una query SQL in un job di Hadoop

### HIVE architecture

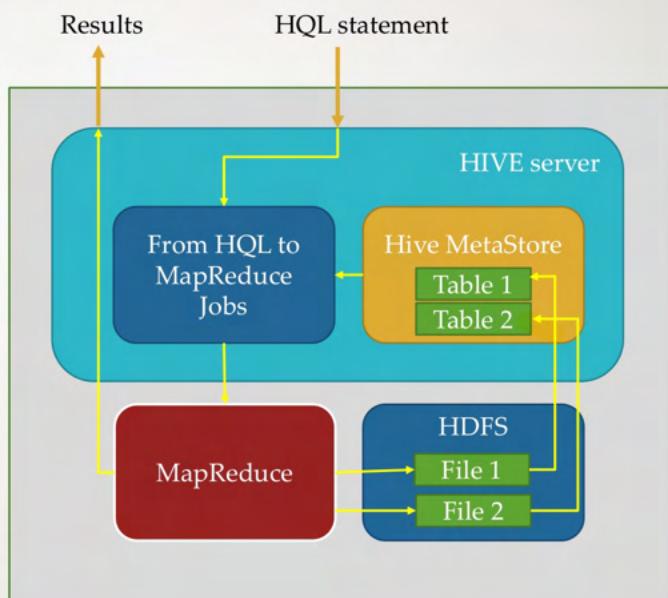
#### Mapping di query SQL to Hadoop

Hive MetaStore gestisce i metadati relativi alla struttura dei file HDFS. "Schematizza" questi file fornendo nomi di colonna e tipi di dati.

Poiché Hive fa sembrare Hadoop un database tradizionale ciò ha contribuito a creare aspettative poco realistiche. Infatti a differenza delle query SQL che vengono eseguite in tempo reale, Hadoop è orientato all'elaborazione batch e non può eseguire query in modalità real-time.

Soluzioni:

- IMPALA, creato da Cloudera
- Miglioramento di Hive con una maggiore integrazione con YARN e TEZ



HQL è un linguaggio basato su SQL molto simile a SQL-92

- La select è quasi identica
- Supporta lo join nello clause FROM
- Supporta la sottoquery nello clause FROM (Ad esempio visto)
- Supporta limitato alle sottoquery nello clause WHERE:
  - Solo IN/NOT IN con sottoquery non corrente
  - Solo EXIST/NOT EXIST con sottoquery corrente
    - Le sottoquery devono essere uno o più predicati corretti
    - I riferimenti alla query principale sono supportati solo nella clausa WHERE delle sottoquery

## Differenze tra HQL e SQL-92:

- HQL offre un insieme di funzioni per generare tabelle, queste funzioni consentono di espandere dati strutturati in righe e colonne
- `Explode()` restituisce una riga per ogni elemento in un campo array o in una mappa di coppie chiave - valore `< nome: valore >`
- `Json_tuple()` esplode un documento JSON incorporato, convertendolo in righe e colonne leggibili
- Nuove clausole fornite da HQL:
  - Clausola `SORT BY`: specifica che l'output venga ordinato SOLO all'interno di ogni reducer. Questo non produce un ordinamento globale dei dati, ma solo all'interno di ogni operazione di riduzione
  - Clausola `DISTRIBUTE BY`: indica che la distribuzione dell'output ai reducer non si basa sull'hashing dei valori chiave, ma in base ai nomi di colonna specificati nella clausola. Questo consente un controllo più granulare sulla distribuzione dei dati.
  - `DISTRIBUTE BY + SORT BY`: combinate, queste clausole possono produrre un ordinamento totale dell'output. La notazione abbreviata equivalente è `CLUSTER BY`

## PIG

- Pig è la risposta di Yahoo! all'esigenza di interfacce di accesso semplificate al sistema di Hadoop.
- Similmente a HIVE, il codice PigLatin viene compilato in codice MapReduce. Pig Latin è un linguaggio procedurale di alto livello per i flussi di dati.
- Essendo uno scripting language, Pig Latin può esprimere tutte le query di HQL, ma può anche andare oltre, consentendo la creazione di pipeline di operazioni molto più complesse.
- È importante notare che HQL mantiene l'approccio DICHIARATIVO di SQL, mentre Pig Latin non lo fa

I dati possono essere caricati da file con strutture simili a tabelle, ovvero collezioni di record con struttura complessa

```
Students = LOAD '../Stud.rec' AS (Code: chararray,  
                                  Name: chararray, BirthYear: int,  
                                  Town: chararray, Degree: chararray);
```

Il contenuto delle tabelle può essere visto chiamando il comando DUMP

```
DUMP Students;  
(09, John Smith, 1970, London, Computer Science)  
(83, Mary Green, 1964, Washington, Physics)
```

## Pig Latin: data model

Per definire tuple con una struttura complessa si possono utilizzare i costrutti di tipo: tuple (semantica record), bag (collezione con duplicati) e map (insieme di coppie chiave - valore)

```
Students = LOAD '../Stud.rec' AS  
          (Code: chararray, Name: chararray,  
           BirthYear: int, Degree: chararray,  
           Phone: tuple(prefix: chararray,  
                         number: chararray));
```

Dot notation utilizzata per navigare dentro la struttura delle tuple

Il formato per i dati complessi è il seguente:

• **Tuple**: racchiuso da `(`), oggetti separati da `,"`

• Tuple non vuota `(oggetto1, oggetto2, oggetto3)`

• Tuple vuota `()`

• **Bag**: racchiuso da `{}` , tuple separate da `,"`

• Bag non vuoto `{(tuple1), (tuple2), (tuple3)}`

• Bag vuoto `{}`

• **Mappa**: racchiusa da `[]` , oggetti: separati da `,"` chiave e valori separati da `#`

• Mappa non vuota `[{key1:#valore1}, {key2:#valore2}]`

• Mappa vuota `[]`

## Query semplici

- Projection: find the name, phone prefix and number of all students

```
StudPROJ = FOREACH Students GENERATE  
    Name, Phone.prefix, Phone.number;  
DUMP StudPROJ;
```

- Selection: find the students having a phone prefix '045'

```
StudSEL = FILTER Students BY Phone.prefix='045';  
DUMP StudSEL;
```

- Selection&Projection: find the name and degree of the students having a name that contains the letter 'a'

```
StudSEL = FILTER Students BY name MATCHES '.*a.*';  
StudSELPProj = FOREACH StudSEL GENERATE Name, Degree;  
DUMP StudSELPProj;
```

Join: supponiamo di avere una nuova tabella

```
Exams = LOAD '../Exam.rec' AS  
    (SCode: chararray, Grade: int,  
     Course: chararray);
```

- Find the code, degree, course name and grades of all the students:

```
StudWithExams = JOIN Students BY Code,  
    Exams BY SCode;  
StudPROJ = FOREACH StudWithExams GENERATE Code,  
    Degree, Course, Grade;  
DUMP StudPROJ;
```

## Query con unione

- Find the union between the name of the degree and the name of the courses

```
DegreeNames = FOREACH Students GENERATE Degree;  
CourseNames = FOREACH Exams GENERATE Course;  
Names = DegreeNames UNION CourseNames;  
DUMP Names;
```

## Query con ordinamento

- Order the students by birth year

```
StudOrd = ORDER Students BY BirthYear;  
DUMP StudOrd;  
(83, Mary Green, 1964, Washington, Physics)  
(09, John Smith, 1970, London, Computer Science)
```

- Order and rank the students by birth year

```
StudRank = RANK Students BY BirthYear;  
DUMP StudRank;  
(1, 83, Mary Green, 1964, Washington, Physics)  
(2, 09, John Smith, 1970, London, Computer Science)
```

↑ rank

## Query con differenza

- Find the difference between the name of the degree and the name of the courses

```
DegreeNames = FOREACH Students GENERATE Degree;  
CourseNames = FOREACH Exams GENERATE Course;  
JoinNames = JOIN DegreeNames LEFT BY Degree  
    CourseNames BY Course;  
DegreeNoCourse = FILTER JoinNames BY Course IS NULL;  
DegreeDiff = FOREACH DegreeNoCourse GENERATE Degree;  
DUMP DegreeDiff;
```

## Query con raggruppamento

- Group the students by birth year

```
StudGroup = GROUP Students BY BirthYear;  
DUMP StudGroup;  
Schema of the result
```

group ↓ Students

```
(1964, [(83, Mary Green, 1964, Washington, Physics),  
         (21, Paul Esting, 1964, New York, Physics)]),  
     (1970, [(09, John Smith, 1970, London, Computer Science)])
```

↑ bag of tuples

- Count the number of students for each degree:

```
StudGroup = GROUP Students BY Degree;  
StudCount = FOREACH StudGroup  
    GENERATE group, COUNT(Students);  
DUMP StudCount;
```

## Esempio con Pig Latin e esempio del meteo

```
-- max_temp.pig: Finds the maximum temperature by year  
records = LOAD 'input/ncdc/micro-tab/sample.txt'  
AS (year:chararray, temperature:int, quality:int);  
filtered_records = FILTER records BY temperature != 9999 AND  
quality IN (8, 1, 4, 5, 9);  
grouped_records = GROUP filtered_records BY year;  
max_temp = FOREACH grouped_records GENERATE group,  
MAX(filtered_records.temperature);  
DUMP max_temp;
```

## Un altro esempio

**Query:**  
count the number  
of customers for  
each country of  
Asia having more  
than 500  
customers,  
reporting  
the name of the  
country and the  
number of  
customers.

**Pig Latin**  
**countries** = load 'PIG\_COUNTRIES' as (country\_id, country\_name, country\_subregion, region);  
**customers** = load 'PIG\_CUSTOMERS' as (cust\_id, first\_name, last\_name, postcode, city, country\_id);  
  
**asianCnts** = filter **countries** by region matches 'Asia';  
**joined** = join **customers** by country\_id, **asianCnts** by country\_id;  
**grouped** = group **joined** by country\_name;  
**agg** = foreach **grouped** generate group, COUNT(**joined.customers**::cust\_id) as custCount;  
**moreThan500** = filter **agg** by custCount > 500;  
**ordered** = order **moreThan500** by custCount desc  
dump **ordered**;

### HQL

```
SELECT country_name, COUNT(cust_id) as custCount  
FROM countries AS co  
JOIN customers AS cu  
ON (co.country_id = cu.country_id)  
WHERE co.region='Asia'  
GROUP BY country_name  
HAVING COUNT(cust_id) > 500  
CLUSTER BY custCount DESC
```

## HIVE e Prg

lavorano su un modello di dati che è molto vicino all'approccio chiave-valore di HDFS

- Sia HIVE che Prg aggiungono una certa struttura alla parte dei valori dei record memorizzati in un file HDFS (simile alla struttura di una tabella relazionale), ma non sono altre
- Al contrario, HBASE fornisce un modello dei dati completamente nuovo su HDFS, rappresentando il primo tentativo di creare un nuovo livello logico per un sistema di database NoSQL

HBase è in grado di creare e gestire su HDFS strutture dati di dimensioni veramente massicce, ben oltre le capacità dei tradizionali RDBMS come MySQL e Oracle. Queste strutture dati sono chiamate **tabelle**. E' importante sottolineare che il modello dati di HBase è completamente diverso dal modello relazionale, anche se entrambi utilizzano il concetto di **TABELLA**.

Inoltre, poiché ogni elemento di dati in HDFS viene replicato 3 volte, HBase eredita da HDFS la ridondanza per proteggere le tabelle dalla perdita di dati. Infine, è bene notare che HBase impone una struttura sui file HDFS.

I dati sono organizzati in tabelle con righe e colonne

Le tabelle sono insiemi di righe e sono individuate e ordinate da un singolo **rowkey**

Una tabella include una o più **column family**, il cui nome è specificato dalla definizione di tabella

Le famiglie di colonne sono composte di **colonne**. I nomi delle colonne sono dinamici. Nuove colonne possono essere create al volo durante l'inserimento o l'eliminazione di dati. I valori nelle colonne hanno un **time stamp**. Perciò ogni famiglia di colonne in una riga definisce una mappa multidimensionale in cui il valore è identificato dal nome della colonna e dal time stamp

I dati di una specifica famiglia di colonne sono memorizzati assieme in memoria secondaria

## Esempio HB

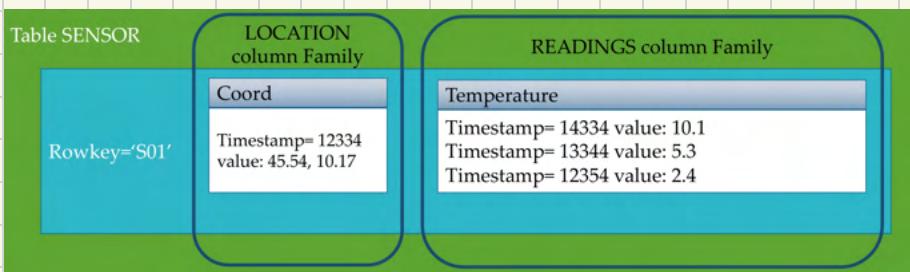
Se dovessimo immagazzinare dati su un enorme insieme di clienti. Questi dati includono:

- (i) Diverse proprietà relative ai loro dati personali
- (ii) L'elenco dei loro amici insieme al loro indirizzo email

Per le chiavi di riga utilizziamo il login utente

Table USERS		PERSONAL DATA column Family		FRIENDS column Family	
Rowkey=Guy	Name	Jo	John	John	Mary
	Guy Harrison	jo@gmail.com	john@hotmail.com	john@hotmail.com	mr@hotmail.com
Rowkey=Jo	Name	John	Mary		
	Joanna Fill			john@hotmail.com	mr@hotmail.com

Ogni cella in una famiglia di colonne può memorizzare versioni multiple dello stesso valore, indicate dal timestamp.



## HBASE

### Configurazione di gestione delle versioni:

- I valori sono memorizzati in ordine decrescente di timestamp
- Per ogni famiglia di colonne un parametro di configurazione indica il massimo di versioni memorizzabili di ogni dato
- C'è anche un numero minimo di versioni
- C'è un Time To Live, dopo quel tempo il dato è cancellato automaticamente dal server

### HBASE interface

- HBase non ha un linguaggio di interrogazione simile a SQL
- L'accesso alle tabelle HBase avviene tramite shell e un API
- La shell è adatta per esperimenti semplici e condivisi dei dati
- La maggior parte del lavoro viene svolto all'interno dei programmi JAVA

- Available operations in shell:
  - Create:** to create a table with a set of column Families
    - create 'ourfriends', {NAME => 'info'}, {NAME => 'friends'}
  - Put:** to populate a cell within a row of a table
    - put 'ourfriends', 'guy', 'info:email', 'guy@gmail.com'
    - put 'ourfriends', 'guy', 'friends:Jo', 'jo@gmail.com'
  - Get:** to pull values from a specific row a table specifying its rowkey
    - get 'ourfriends', 'guy'

```
Result of the get operation: get 'ourfriends', 'guy'  
COLUMN           CELL  
friends:Jo      timestamp=1445, value=jo@gmail.com  
info:email       timestamp=1542, value=guy@gmail.com
```

HQL può esprimere qualsiasi query di SQL-2, quindi copre tutte le classi identificate in TRC

$$HQL \equiv \{ C_1, \dots, C_6 \}$$

Pig Latin può esprimere tutte le query dell'algebra relazionale, pertanto copre tutte le classi identificate in TRC

$$Pig \equiv \{ C_1, \dots, C_6 \}$$

HBase non può esprimere molte query di SQL-2, quindi copre solo parzialmente una classe identificata in TRC capacità di lettura delle tabelle

$$HBase \equiv \{ C_1 \}$$

## Data Design

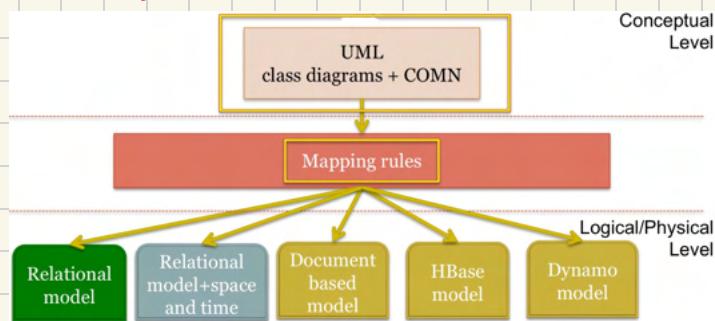
È necessario trovare una metodologia per la progettazione di raccolte dati. Le sue caratteristiche sono:

- Indipendenza dalla tecnologia specifica che verrà utilizzata per la rappresentazione dei dati: a livello specifico
- Si basa su:
  - uno strumento di specifica orientato agli oggetti (UML)
  - un nuovo strumento di modellazione (COMN) che consente la descrizione formale del sistema informativo per il quale è progettata la soluzione
- Può produrre schemi fisici in qualsiasi tecnologia di database T a condizione che venga definito un insieme di regole di mappatura per T.

La metodologia è formata:

- Permette la definizione delle specifiche dei dati (modelli di dati)
- Ogni modello di dati descrive la raccolta di dati come un insieme di oggetti a struttura complessa
- In un modello di dati, il programmatore può specificare le associazioni semantiche tra gli oggetti e definire le gerarchie di oggetti
- La specifica del modello di dati si basa su linguaggio formale, è univoca e può essere tradotta automaticamente nella specifica corrispondente a livello fisico (tecnologia specifica)
- Ogni specifica richiede insieme specifico di regole di mappatura. Ci possono essere ulteriori scelte di progettazione → secondo della specifica tecnologia. In particolare per i sistemi noSQL

### Visone generale



### Modellazione concettuale dei database nell'era noSQL

Nell'era DBMS noSQL è comune avere sistemi "schema-less" o "schema free", comunque comunque modellare il nostro database?

- Spesso emergono pattern nella struttura dati:
  - gran parte dei dati memorizzati condivide uno schema significativo.
- Se si salta la fase di modellazione dei dati, la rappresentazione dei dati deve essere riorganizzata frequentemente.
  - costi elevati in termini di tempo, spazio di archiviazione e restituzione del codice
- Un modello di dati offre l'opportunità di sperimentare con i dati e analizzare l'impatto di scelte differenti prima di preoccuparsi della rappresentazione fisica all'interno di una tecnologia specifica
  - senza costi di riorganizzazione dei dati già memorizzati
- Implementare in un DBMS schema-less senza un modello significa che una volta completa l'implementazione non sopravvive alcuna documentazione formale dei dati.
  - la conoscenza dei dati rimane internamente ai programmati e nel codice

Alcuni DBMS NoSQL supportano gli schemi e quindi spesso altamente desiderabile imponere la conformità a uno schema per:

- Garantire validità dei dati memorizzati;
- Assicurare alle applicazioni che i dati abbiano un modello comune da sfruttare.

Per qualsiasi progetto significativo è necessario sviluppare e mantenere un modello di dati completo come parte di un progetto di sviluppo model-driven. Se si salta il processo di modellazione si rischia di:

- Limitare le capacità del sistema di adattarsi a requisiti mutevoli;
- Incorrere in problemi difficilmente risolvibili;
- Compromettere completamente il progetto.

La metodologia proposta si basa su:

- COMN: per la specifica del contesto applicativo del sistema informativo che vogliamo modellare
- UML - Diagrammi di classe: per la specifica del modello concettuale dei dati che descrive la raccolta di oggetti astratti che devono essere rappresentati nel DB
- UML - Diagrammi di classe con stereotipi: per la specifica del modello di dati logico/fisico che descrive la raccolta di oggetti frusci memorizzati nella memoria (secondaria) che rappresentano gli oggetti astratti del modello concettuale di dati.

### UML per il design concettuale

La progettazione concettuale di qualsiasi database può essere realizzata adottando la piattaforma UML e in particolare mediante la definizione di diagrammi di classe.

I diagrammi di classe UML programmano una rappresentazione grafica di classi che può essere utilizzata per la rappresentazione a livello concettuale dello stato e del comportamento di una collezione di tipi che descrivono le caratteristiche di una raccolta di dati.

Questo approccio può essere particolarmente efficace in quanto è indipendente dalla tecnologia e il diagramma delle classi ottenuto dal processo di progettazione può essere tradotto in diversi modelli logici/fisici, quando viene scelta la tecnologia target per l'implementazione del DB.

Supponiamo che il nostro obiettivo sia rappresentare in UML le seguenti caratteristiche:

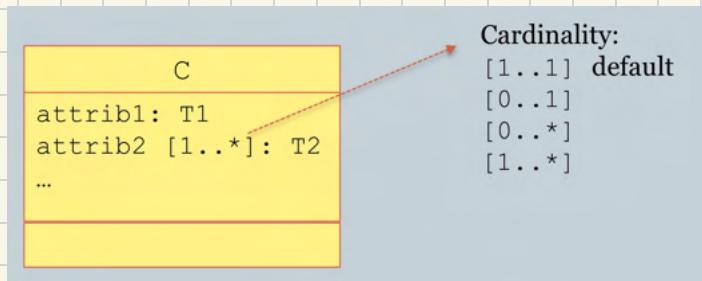
- I tipi che descrivono gli attributi degli oggetti;
- Le associazioni tra tipi che collegano oggetti tra loro tramite ruoli;
- Le gerarchie di tipi per oggetti;
- I tipi utilizzati per rappresentare enumerazioni statiche o dinamiche

## Classe UML

La classe UML viene utilizzata per definire una popolazione di istanze che rappresentano una collezione di oggetti/concetti specifici nei requisiti dell'applicazione.

### Sintassi:

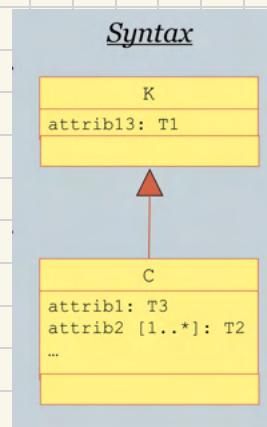
Una classe con due attributi: `attrib1` e `attrib2`, il primo tipo base `T1` e il secondo che contiene un insieme di valori tipo `T2`.



### Ereditarietà delle classi UML

Una classe `C` di UML può essere sottoclasse di un'altra classe UML `K`. Questa dichiarazione ha due conseguenze:

- intensionale, tutte le proprietà di `K` sono ereditate da `C`
- estensionale, tutte le istanze di `C` sono anche istanze di `K`

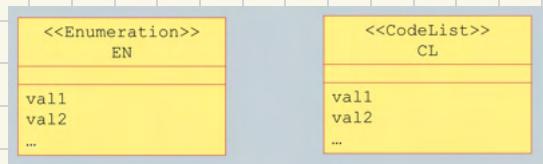


### Enumerazione

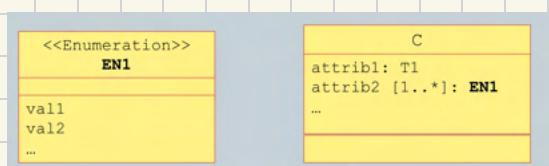
La dichiarazione dei tipi di enumerazione si ottiene utilizzando un classificatore con specifici stereotipi:

- `Enumeration` per le enumerazioni statiche
- `Code List` per le enumerazioni dinamiche

### Sintassi:



### Esempio



## Rappresentazione delle relazioni tra istanze:

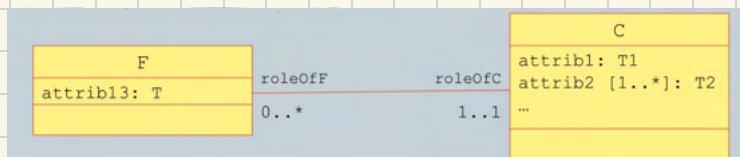
Per rappresentare le relazioni binarie tra istanze appartenenti all'estensione di classi, si può usare il costrutto **associazione** di UML.

Tale costrutto è in grado di specificare sia ruoli delle classi che partecipano all'associazione, sia vincoli di cardinalità che devono essere soddisfatti dalle istanze delle classi coinvolte.

N.B. il nome del ruolo e i vincoli di cardinalità corrispondenti sono specificati nella rappresentazione grafica del modello, vicino alle classi a cui si riferiscono.

Questo produce una rappresentazione grafica del modello in cui i vincoli di cardinalità sono specificati in una posizione opposta rispetto a uno schema ER.

Sintassi per le associazioni:



In the ER model:



## Classi astratte

Una classe può essere specificata come **classe astratta**.

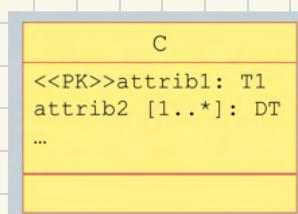
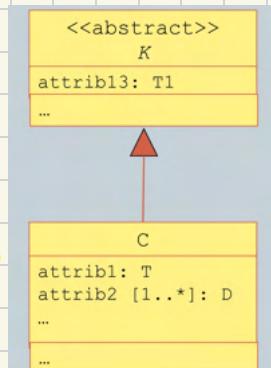
In questo caso non può avere istanze (oggetti concreti).

Queste classi sono generalmente utilizzate nelle gerarchie per **fattorizzare**

la definizione di proprietà comuni tra classi figlie. Una classe astratta

è identificata dallo stereotipo **<<abstract>>** In molti strumenti di

progettazione basati su UML, una classe astratta è rappresentata graficamente scrivendo il suo nome in corsivo.



## Identificatori UML

Ulteriori dettagli sulla specifica di un modello di dati in UML riguardo all'identificazione degli oggetti: lo stereotipo **<<PK>>** può essere utilizzato con l'obbligo di etichettare gli attributi/ruoli che verranno utilizzati per identificare gli oggetti della classe nel contesto dell'applicazione.

Ulteriori dettagli sull'indirizzo di classi come tipi di attributi:

Una classe C definita nel diagramma di classe può essere utilizzata come tipo per l'attributo A di un'altra classe K solo se vengono le seguenti condizioni:

- L'insieme di oggetti della classe C non verrà mai considerato come una collezione indipendente di istanze di informazioni, ma vi si accede sempre a partire dalla classe K ospita l'attributo A.

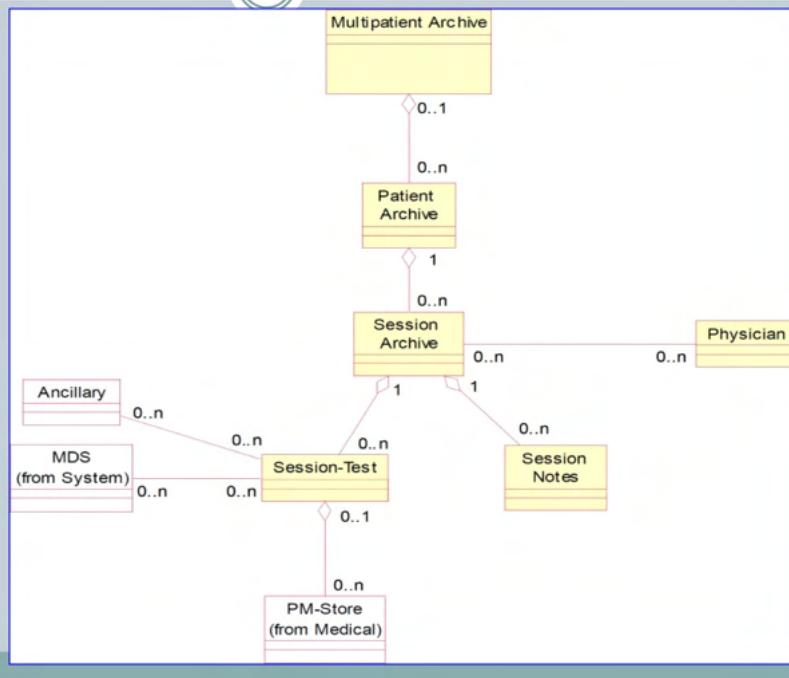
- Non esiste alcuna associazione che coinvolga gli oggetti della classe C. L'unica associazione è quella con la classe K ed è implicitamente rappresentata dall'incapsulamento degli oggetti C nell'attributo A di K.

## Esempio



Schema UML  
from  
ISO/IEEE  
International  
Standard for  
Health  
Informatics:

Patient archive.



Software : argoUML

Mapping UML verso HBase

Primo passo: Definizione delle raccolte di record da memorizzare in HBase

- Considerando l'insieme di classi definite nel diagramma di classe, vanno scelte le classi da rappresentare come raccolta di righe in una tabella HBase
- Cioè andrebbe scelto considerando gli access path e le query più frequenti nel nostro contesto
- E' il passo più importante

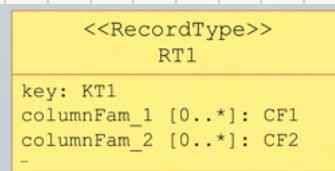
- Ogni percorso di accesso AP è definito assegnando una classe principale CAP da cui inizia il percorso
- Per ogni percorso di accesso, dovrebbe essere definita una raccolta di record. E' necessario un record per memorizzare ciascuna istanza della classe CAP, con una famiglia di colonne per tutti i suoi attributi
- Successivamente è necessario includere le classi collegate a CAP con le associazioni

Secondo passo: Definizione del tipo di record

- Definisci i tipi di record che verranno memorizzati nella tabella HBase. Per fare ciò bisogna introdurre lo stereotipo <> recordType <<

E' necessario definire ora:

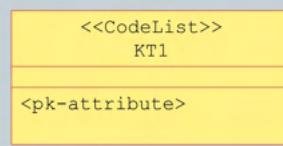
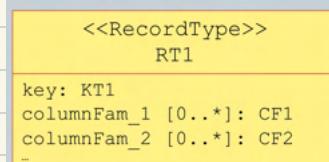
- un tipo per la chiave
- un tipo per ogni column family



Record Type: definizione del tipo di chiave:

Per la definizione di un tipo record è necessario definire una chiave e un insieme di column family

Per la rappresentazione della chiave il record type ha sempre un attributo Key, il cui tipo è definito tramite una lista di codici. Gli attributi con lo stereotipo <> pk <<definiti nelle classi a livello concettuale possono aiutare a definire la struttura della chiave



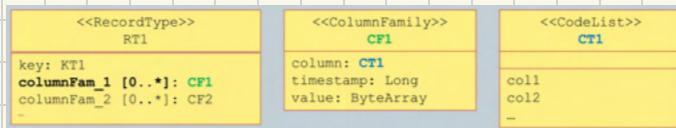
The key can also be a composite key like:  
<whole-id><part-id>

## Record Type - Definizione dei tipi column family

Per la definizione di un record dobbiamo definire almeno una column family.

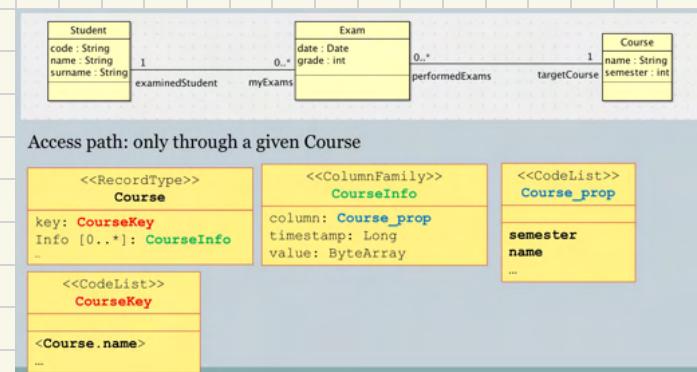
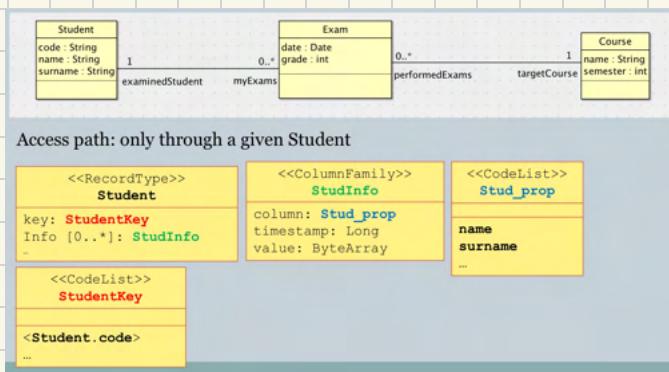
Per la rappresentazione di una famiglia di colonne Column Fam viene aggiunto al record type un attributo Column Fam; il tipo di ColumnFam è definito tramite una classe separata con stereotipo <<columnFamily>>

Inoltre, viene utilizzata una lista di codici per elencare i valori possibili per le colonne



Il tipo base **ByteArray** viene utilizzato per rappresentare qualsiasi valore possibile la cui struttura è nota solo all'applicazione.

### Esempio



### Mapping per le associazioni

Per ogni classe C collegata a CAP, con un'associazione di tipo uno a molti (inclusi i casi in cui l'associazione è una composizione o un'aggregazione), ci sono due possibili scelte:

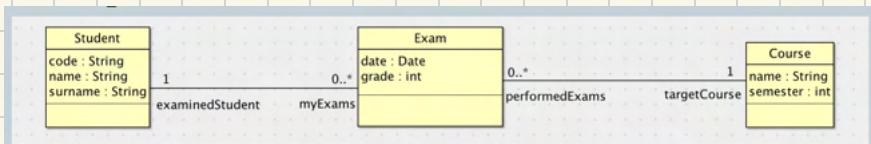
#### 1 Mappatura esterna

- Definisci un tipo record per C
- Definisci una chiave composta per C, dove ogni chiave è composta dalla chiave di CAP e una chiave per l'istanza di C che è collegata alla stessa istanza CAP

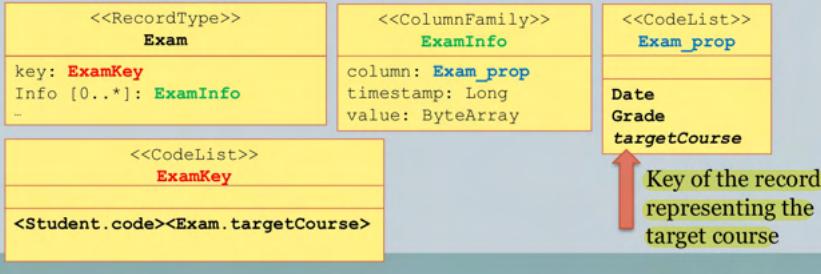
#### 2 Mappatura interna

- Definisci una famiglia di colonne in CAP per memorizzare tutti gli attributi di C encapsulati in CAP

## Esempio



Access path: only through a given Student – EXTERNAL mapping of Exam



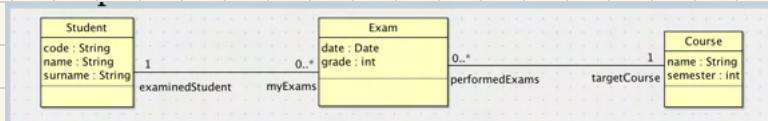
Mapping di una classe C collegata uno a molti con C<sub>AP</sub>

Access path: only through a given Student – EXTERNAL mapping of Exam  
EXAMPLE of record instances

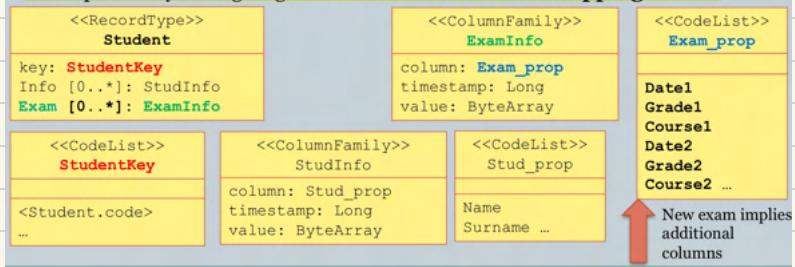
This is not a Table! Record types will be mapped to table later.

Key	Column Family 1
VR993	Info:Name='Mario', Info:Surname='Rossi'
VR993-Databases	Info:Date='1/2/20', Info:Grade=23
VR993-Algebra	Info:Date='13/3/20', Info:Grade=24
Algebra	Info:Semester='I Sem'
Databases	Info:Semester='II Sem'

## Esempio



Access path: only through a given Student – INTERNAL mapping of Exam



Mapping di una classe C collegata uno a molti con C<sub>AP</sub>

Access path: only through a given Student – INTERNAL mapping of Exam  
EXAMPLE of record instances

This is not a Table! Record types will be mapped to table later.

Key	Column Family 1	Column Family 2
VR993	Info:Name='Mario', Info:Surname='Rossi'	Exam:Date1=='1/2/20', Exam:Grade1=23, Exam:Course1='Databases', Exam:Date2=='13/3/20', Exam:Grade2=24, Exam:Course2='Algebra'
Algebra	Info:Semester='I Sem'	
Databases	Info:Semester='II Sem'	

## Mapping per le associazioni

Per ogni associazione A molti a molti che collega  $C_{1AP}$  con un'altra main class  $C_{2AP}$ , ci sono due possibili scelte

- Internal mapping in  $C_{1AP}$  (o in  $C_{2AP}$ ) definiamo una column family A in  $C_{1AP}$  (o in  $C_{2AP}$ ) per memorizzare tutte le chiavi dei record di  $C_{2AP}$  collegati (o di  $C_{1AP}$ )
- Internal mapping sia in  $C_{1AP}$  che in  $C_{2AP}$  (mapping rindondante): definiamo una column family A in  $C_{1AP}$  e un'altra column family A in  $C_{2AP}$  per memorizzare tutte le chiavi dei record collegati di  $C_{2AP}$  ( $C_{1AP}$ )

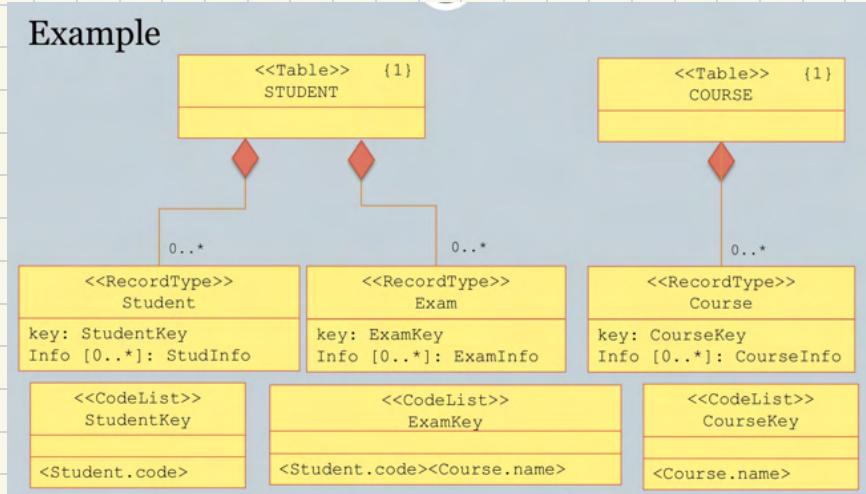
L'external mapping non è un'opzione in questo caso, poiché le istanze di  $C_{2AP}$  sono già rappresentate in un record type separato

## Mapping dei record type alle tabelle

- Mapping normalizzato: una tabella per ciascun record type
- Mapping di accesso path based: Ogni  $C_{AP}$  in una tabella assieme ai record che dipendono dalla sua chiave (chiave composta)
- One table mapping: Tutti i record in una tabella, ciò richiede l'aggiunta di un'etichetta record type

## Mapping di un accesso path based

### Example

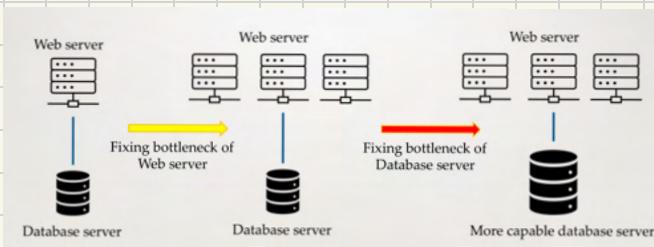


## Sharding and Amazon

- Nel 1995 viene rilasciato MySQL e nei successivi 10 anni Internet è passato da essere una curiosità ad essere il più importante mezzo di comunicazione per la nostra Società.
- Nonostante ciò i sistemi di database usati erano sempre gli stessi.
- Tuttavia la capacità dei database relazionali di soddisfare le esigenze era arrivata al limite.

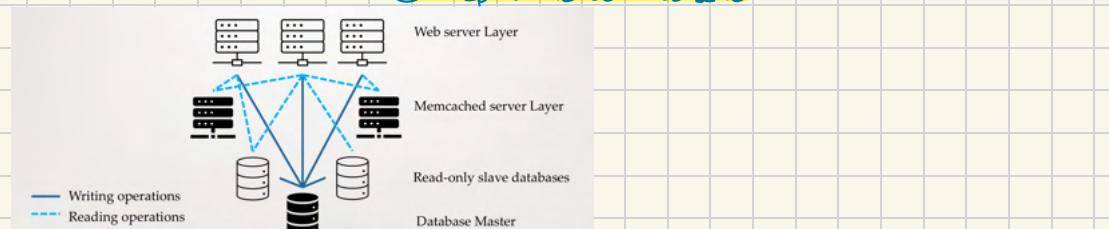
### Web Applications

- All'inizio il web era una collezione di file HTML statici senza contenuti dinamici e gestione delle transazioni. (il Web 1.0)
- Dopo gli e-commerce portano alla generazione di pagine web dinamiche, recuperando i dati da un database server.
- Common Gateway Interface (CGI) permette alle richieste HTTP di invocare uno script che genera codice HTML (il Web 2.0).
- A questo punto la problematica principale era la crescente richiesta di risorse dei sistemi che supportano le applicazioni del mondo Web 2.0 per fare fronte al carico di dati generato da centinaia di migliaia di utenti.



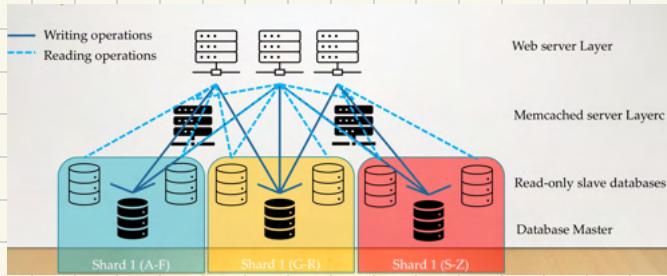
I server database di grandi dimensioni hanno due problemi principali:

- Affidabilità: un singolo fallimento può generare il crash di tutta l'architettura.
  - Lo scale up richiede sempre di cambiare i server e quando nel Web 2.0 le compagnie giunsero allo scala globale, neanche il database centralizzato più grande poteva soddisfare le esigenze.
- Le compagnie volevano una soluzione intelligente che potesse crescere a seconda delle richieste.
- Questo approccio terminò dopo il crash del dot.com.
- Dopo esso Linux, Apache web Server e MySQL divennero le tecnologie basiche per una nuova Web App opensource nel Web 2.0.
- MySQL richiedeva meno risorse di Oracle, e nonostante fosse meno scalabile di esso divenne la tecnologia chiave per un architetture del Web 2.0.
- Il nuovo approccio è basato su:  
① Replicazione dei server MySQL per le operazioni di lettura.  
② Cache di oggetti distribuita implementata con la soluzione memcached.  
③ Replicazione dei web server.



Il bottleneck delle Web App 2.0 che utilizzano una soluzione MySQL sono le scritture nel database

La soluzione più semplice da applicare consiste nello: SHARDING



Le tabelle grandi erano partizionate su più server fisici. Ogni partizione veniva denominata shard

Questo partizionamento si basava su un valore chiave ad esempio la chiave primaria, come può essere lo user id

Ogni SHARD era gestita da un server master del database

Data una specifica istanza di informazione ad esempio un utente tutte le tuple che rappresentano dati di tale istanza di informazione verranno memorizzati nello stesso SHARD

Facebook nel 2011 usava 1200 shard di MySQL e 9000 server Memcached. Ciò supporta 1.6 miliardi di letture al secondo e 3.5 milioni di cambiamenti di righe al secondo

### Sfide dello sharding

Lo sharding comporta significativa complessità computazionale e compromessi

Lo Sharding è concettualmente semplice ma difficile nella pratica

Indirizzare la richiesta di accesso ai dati allo shard corretto è complesso

Alcune richieste che necessitano di accedere a più shard richiedono una codifica complessa

### Svantaggi dello sharding

Complessità delle applicazioni: è compito delle applicazioni indirizzare la richiesta SQL allo shard corretto. Inoltre è richiesto un livello di routing dinamico

SQL limitato: le operazioni di join fra shard e query aggregate group by non possono essere implementate, inoltre solo i programmatore possono accedere all'intero DB

Perda di integrità delle transazioni: le transazioni ACID su più shard non sono implementate in MySQL. Il protocollo Two Phase commit che teoricamente è in grado di interagire con transazioni distribuite non è implementato senza problemi di risoluzione dei conflitti e bottleneck

Complessità operazionale: il load balancing tra i vari shard diventa problematico, in particolare quando nuove shard vengono aggiunte continuamente

## Soluzione di Oracle ai database sharded:

- Oracle's Real Application Cluster è il più significativo esempio di soluzione che può garantire Scalabilità conformità alle proprietà ACID e cluster relazionale
- In Oracle RAC ogni database node lavora con dati situati su dispositivi di archiviazione condivisi
- Nuovi nodi possono essere aggiunti senza rimbalscimento dei dati e viene implementata una memoria cache tra i nodi

## Motivi del fallimento di Oracle RAC come alternativa a MySQL

- Troppo costoso
- Non completamente scalabile, in particolare sui siti web più grandi
- Divenne evidente che nessun database relazionale (con le proprietà ACID) potessero soddisfare le necessità dei siti web più grandi

## Teatrmo CAP

Ci dice che un sistema distribuito non può garantire simultaneamente:

- **Consistenza**: ogni utente ha la stessa vista dei dati in qualsiasi istante
- **Disponibilità**: in caso di fallimento di un certo nodo il database rimane operativo
- **Partition tolerance**: in caso di fallimento della connessione (partizioni in due o più segmenti) il database rimane operativo

Negli anni 2000 non c'erano dei veri e propri data center multipli, allora la partition tolerance non era un problema pratico. Ma quando i più grandi siti web divennero globali e si desiderava continua disponibilità, allora la partition tolerance cominciò ad essere un problema.

Quando avviene un partizionamento abbiamo due possibili scelte:

- (i) mostrare a ogni user una differente vista dei dati
- (ii) chiudere una partizione e disconnettere gli utenti ad essa collegati. Questo è ciò che fa Oracle RAC

## Consistenza alla fine (Eventual Consistency)

- Nel mondo dei social network e degli e-commerce la soluzione desiderabile è mantenere la disponibilità e sacrificare la consistenza tra gli utenti.
- Mantenere la consistenza è un problema più di performance che di disponibilità.
- Per assicurare la consistenza c'è da propagare i cambiamenti del database a nodi multipli in maniera sincrona e immediata.
- Ciò comporta un inevitabile aumento della latenza quando i nodi sono distanti geograficamente.
- Per le banche questo ritardo è inevitabile, però per molti altri siti web la consistenza sincrona non è necessaria e un ritardo nell'aggiornamento di tutte le copie dei dati può essere accettabile (**consistenza alla fine**)

## Definizione formale del teorema CAP

Consideriamo un sistema distribuito molto semplice DS.

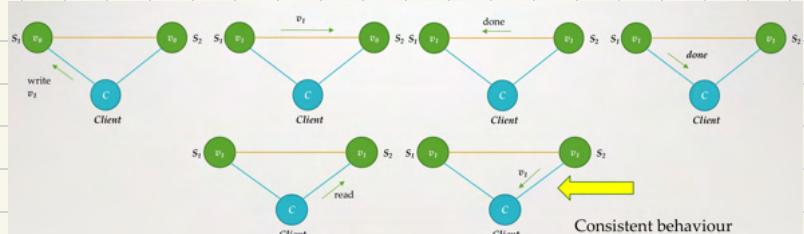
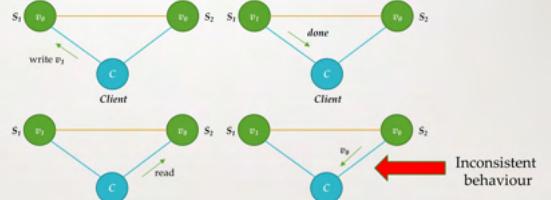
DS è composto da due server  $S_1$  e  $S_2$ . Entrambi contengono la stessa variabile  $v$ , il cui valore iniziale è  $v_0$ .  $S_1$  e  $S_2$  possono comunicare tra loro e possono anche comunicare con client esterni.



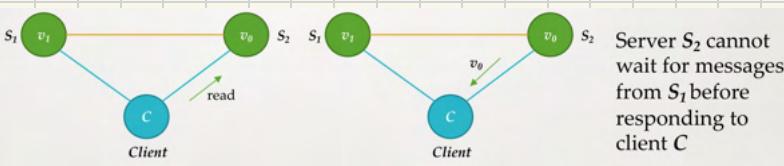
Il client può richiedere di leggere o scrivere la variabile  $v$  a qualsiasi server



**Consistenza:** qualsiasi operazione di lettura su una variabile  $v$  che inizia dopo una scrittura che è stata completata (come  $v=v_0$ ) deve ritornare  $v_0$  o il risultato di una scrittura successiva.

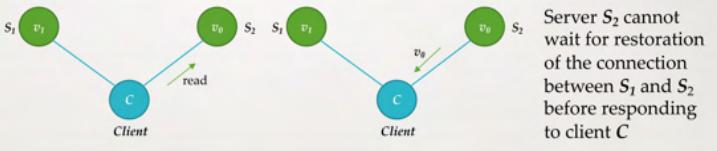


**Disponibilità:** qualsiasi richiesta da un non-failing node nel sistema deve risultare in un immediata risposta. Perciò, il server non può ignorare le richieste dei clienti



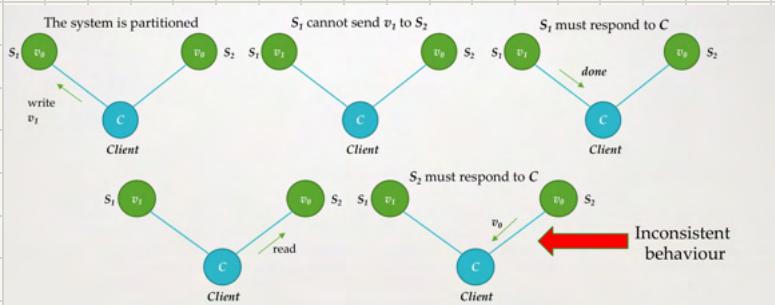
**Partial tolleranza:** la rete può perdere arbitrariamente messaggi mandati da un nodo all'altro

Un sistema partizionato sembrerà simile a questo

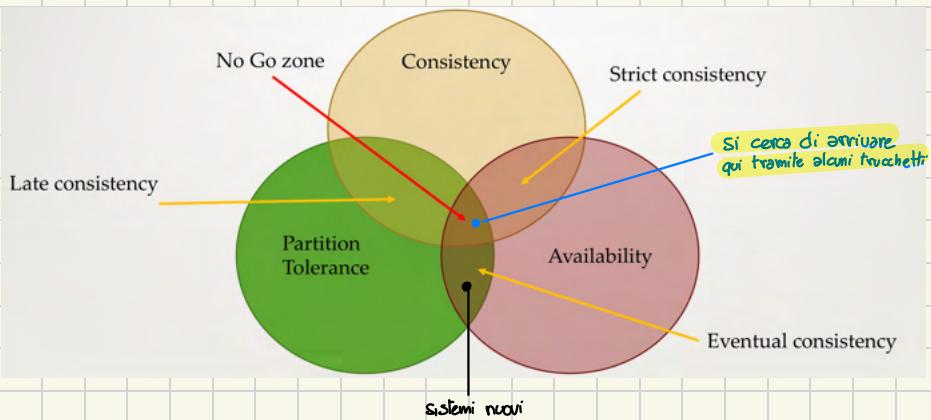


Proof formale del teorema CAP

Assumiamo per assurdo che esista un sistema consistente, disponibile e partition tollerant!



Facendo queste assunzioni notiamo che comunque esiste un'esecuzione che porta all'inconsistenza. Perciò non può esistere un sistema con tali proprietà presenti contemporaneamente.



Dynamo è un sistema non relazionale alternativo sviluppato da Amazon, costruito per soddisfare questi requisiti:

- Disponibilità continua: non sono ammesse interruzioni di servizio
- Network partition tolerance: queste non devono costituire perdite di disponibilità
- Risoluzione dei conflitti senza perdite: nessun ordine e nessun oggetto aggiunto al Carrello deve risultare perso
- Efficienza ed economicità
- Scalabilità incrementale

- I sistemi basati su Dynamo devono perdere consistenza solo se necessario per garantire la disponibilità
- Supportano solo accessi basati sulla primary key senza data model. L'approccio chiave-valore si applica qui, ma con l'assunzione che ogni valore memorizza un oggetto binario non strutturato con dimensione minore di 1 MB

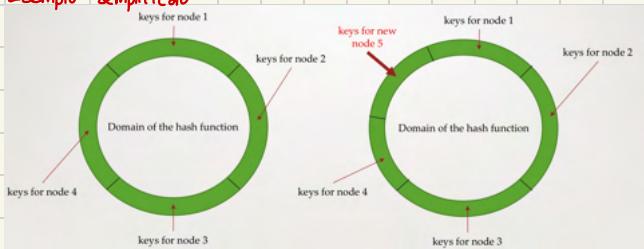
Nuove tecniche applicate in Dynamo sono:

- Hashing consistente
- Consistenza sintonizzabile
- Data versioning

### Hashing Consistente

- Questa tecnica permette al sistema di assegnare una chiave a un nodo usando la funzione di hashing.
- L'hashing lavora bene per distribuire i dati uniformemente su un numero fisso di nodi
- Ma quando il numero di nodi cambia (ad esempio: aggiunta di un nodo) allora tutto l'assegnamento delle chiavi deve essere ricalcolato
- L'Hashing consistente prova a ridurre la quantità di lavoro necessaria per ridistribuire le chiavi sui nodi in caso di variazione del numero di nodi (alcuni approcci usano i nodi virtuali)

### Esempio semplificato

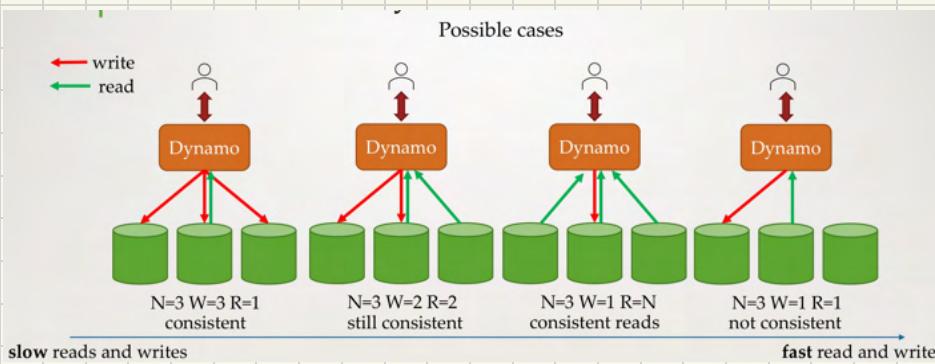


## Tunable consistency

Dynamo permette all'applicazione di scegliere il livello di consistenza da applicare per maneggiare i suoi dati.

La notazione NWR è utilizzata per descrivere tale livello.

- $N$  è il numero di copie di ciascun oggetto che il database dovrà mantenere (in genere 3 copie).
- $W$  è il numero minimo di copie del dato che devono essere state scritte prima che la write possa essere completata.
- $R$  è il numero di copie a cui l'applicazione accederà durante la lettura per cercare il valore attuale.



## Data versioning

- È possibile che ci siano più versioni di uno stesso oggetto nel sistema.
- Alcune volte il conflitto può essere risolto dal sistema stesso, ma a volte è necessario che lo risolvano l'applicazione o lo user.

## Sistemi ispirati da Dynamo

Sono in genere i sistemi key-value. Che includono:

Dynamo DB: sistema database Cloud-based di Amazon

Riak e LinkedIn Voldemort: praticamente la copia di Dynamo

Cassandra: implementa modelli Dynamo con hashing consistente e Tunable consistency, combinato con varie versioni del data model di Big Table

I sistemi come Dynamo non introducono nessuna struttura nei dati che memorizzano. Ciò li rende impenetrabili a tutti, ad eccezione dei programmatore.

## Store chiave - valore

- Non è una definizione formale su come i dati dovranno essere rappresentati in un database chiave-valore.
- Esso accetta qualsiasi valore binario, con un determinato limite di dimensione per la chiave e qualsiasi dato binario per il valore.
- Il metodo di accesso principale delle istanze  $(k_i, v_i)$  è attraverso la chiave.
- potrebbe essere fornito un secondo indice aggiuntivo. Si basano su termini non chiave contenuti nel valore dei record chiave-valore.

## Sistemi Riak

In Riak, un archivio chiave-valore basato sulle specifiche di Dynamo i seguenti **datatype** (tipi che verranno usati per la parte di valore dei record) sono disponibili:

- **Replicazione convergente dei datatype**: Questi tipi permettono la risoluzione automatica di operazioni in conflitto dovute all'aggiornamento dei dati.

• **Document type**: come XML, JSON e testo

• **datatype custom**

## Data model di Dynamo DB

I costrutti principali sono:

- Tabelle

- Oggetti
- Attributi

- Chiavi primarie

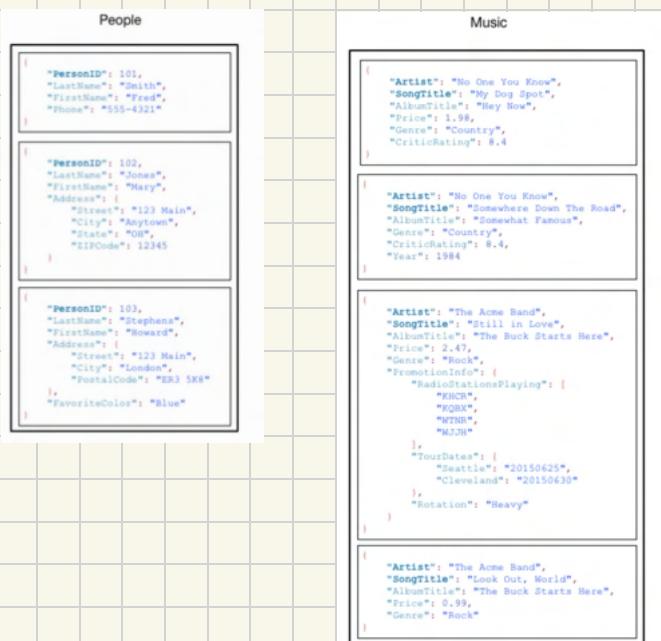
- Indice secondario

**Tabella**: questo costrutto permette di definire una struttura per memorizzare una collezione di istanze di informazione. Ogni istanza è descritta da un oggetto. Pertanto una tabella è un insieme di elementi.

**Elemento**: Questo costrutto permette di rappresentare un'istanza di informazione elencando un gruppo di coppie attributo-valore che ne descrivono le proprietà. Ogni oggetto è identificabile unicamente tra tutti gli altri oggetti. Non c'è limite sul numero di oggetti che puoi memorizzare in una tabella (servizio cloud).

**Attributo**: Questo costrutto permette di definire una proprietà di un oggetto. Il suo valore può essere **scalare**: numero o stringhe, ma può essere anche una struttura complessa. DynamoDB supporta fino a 32 livelli di nesting.

**Primary key**: Con la creazione di una tabella in aggiunta al nome di tabella, bisogna specificare la chiave primaria della tabella. La chiave primaria identifica unicamente ogni oggetto nella tabella, perciò due oggetti non possono avere la stessa chiave. La chiave è usata anche per accedere ai dati in maniera efficiente.



## Chiave primaria

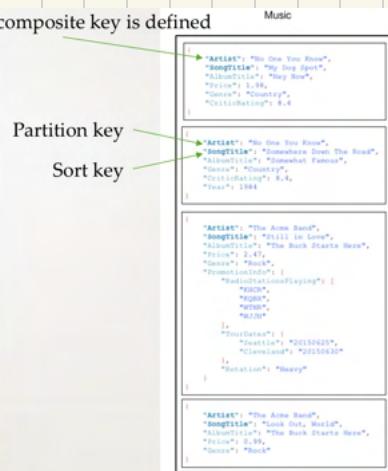
Ci sono due possibili strutture per la chiave primaria:

- Una **partition key** è composta da un attributo. DynamoDB utilizza il valore della chiave di partizione per determinare la partizione fisica di archiviazione in cui l'elemento sarà memorizzato. Nella tabella `Persons` abbiamo un attributo `PersonID`. Questo permette di accedere ai dati assegnando un `PersonID`.
- Una **chiave composta**: è composta da una partition key e una sort key: la partition key è usata come nel caso precedente per scegliere la partizione fisica in cui archiviare gli elementi. Tutti gli oggetti con la stessa partition key sono memorizzati assieme ordinati per il valore di chiave di ordinamento. Questo permette di accedere ai dati assegnando un determinato valore alla chiave di partizione o a tutti i valori della chiave di ordinamento.

Here a partition key is defined



Here a composite key is defined



## Indice secondario

Un indice secondario permette di fare una query dei dati nella tabella utilizzando una chiave differente rispetto alla query effettuata con la chiave primaria.

DynamoDB supporta due tipi di indici:

- Indice secondario globale: un indice con una partition key e una sort key che possono essere diverse da quelle della tabella.
- Indice secondario locale: un indice che ha la stessa partition key della tabella ma una sort key diversa.

Ogni tabella in DynamoDB ha 20 indici secondari globali e 5 indici secondari locali per tabella.



# Regole di mapping UML da Dynamo DB

Primo passo: Definizione della collezione di elementi da memorizzare in DynamoDB

- Considerando l'insieme delle classi che sono definite nel diagramma delle classi. Vanno scelte le classi che vogliamo rappresentare come collezione di oggetti in una tabella di DynamoDB.
- Cioè andrebbe fatto considerando il percorso d'accesso e le query più frequenti nella nostra app.
- Questo è il passo più importante.

Definizione di collezione di elementi:

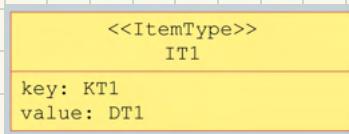
- Ogni percorso di accesso AP è definito assegnando una chiave principale Cap dove il percorso parte.
- Per ogni access path deve essere definito un insieme di elementi.
- Abbiamo bisogno di un tipo di elemento per memorizzare le istanze delle classi Cap con un insieme di attributi per rappresentare tutte le proprietà.
- Successivamente bisognerà includere le classi collegate a Cap attraverso le associazioni.

Secondo passo - Definiamo l'Item type

Definiamo l'item type che poi verrà memorizzato nelle tabelle di DynamoDB. Per fare ciò è stato introdotto il tipo <<ItemType>>

Ora è necessario definire

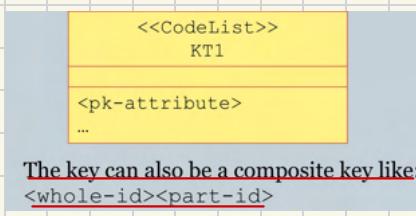
- Un tipo per le chiavi
- Un tipo per i valori



Definizione del Key Type:

Per la definizione di un tipo di elemento è necessario definire una chiave e un tipo per il valore.

Per la rappresentazione della chiave l'ItemType ha un attributo chiave, il cui tipo è definito attraverso un elenco di codici. L'attributo con lo stereotipo <<pk>> definito nella classe a livello concettuale può aiutare nel definire la struttura della chiave.

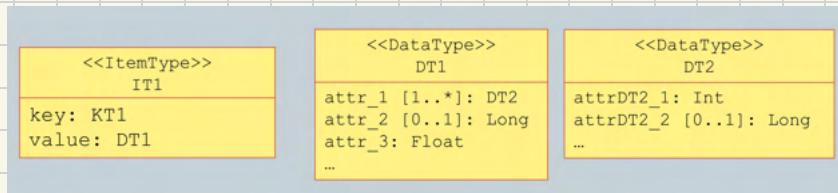


Definizione del datatype:

Per la definizione di un Item Type, è necessario definire un DataType per il valore.

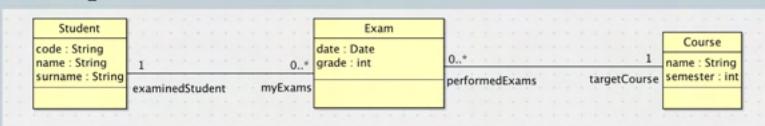
Il datatype è definito da una classe separata con lo stereotipo <<DataType>>. Ha uno o più attributi. Ogni valore può avere:

- Valore Scalare: appartiene a un tipo base come: String, Int, Float, .....
- Valore complesso: appartiene ad altri datatype specificabili nel datamodel.



## Mapping della classe principale CAP

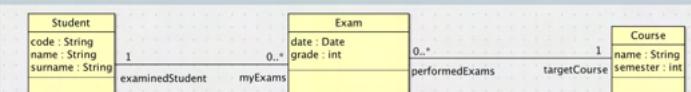
### Example



Access path: only through a given Student



### Example



Access path: only through a given Course



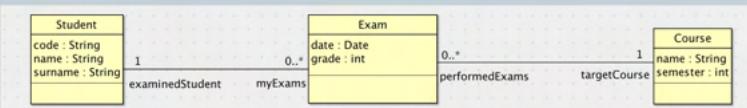
## Definizione della collezione di oggetti

Per ogni classe  $C$  collegata a CAP con un associazione uno a molti ci sono due possibili scelte:

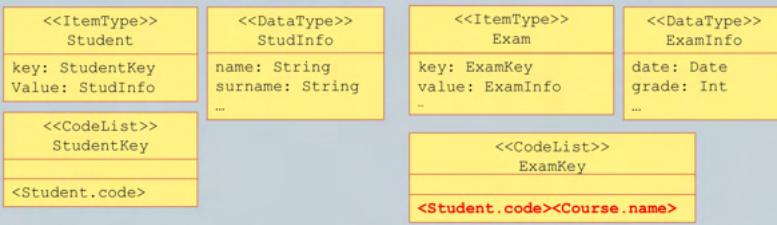
- External mapping:**
  - definiamo un item type per  $C$
  - definiamo una chiave composta per  $C$ , dove ogni classe è una chiave composta dalla chiave per CAP (partition key) e una chiave per le istanze di  $C$  che sono collegate alle stesse istanze di CAP (sort key)
- Internal mapping:**
  - Definisce un attributo aggiuntivo multistore nel datatype che rappresenta le istanze di CAP per memorizzare tutte le istanze di  $C$ ; un nuovo datatype verrà definito per memorizzare tutti gli attributi di  $C$  incapsulati in CAP

## Mapping di una classe collegata uno a molti: Cap

### Example



Access path: only through a given Student – EXTERNAL mapping of Exam



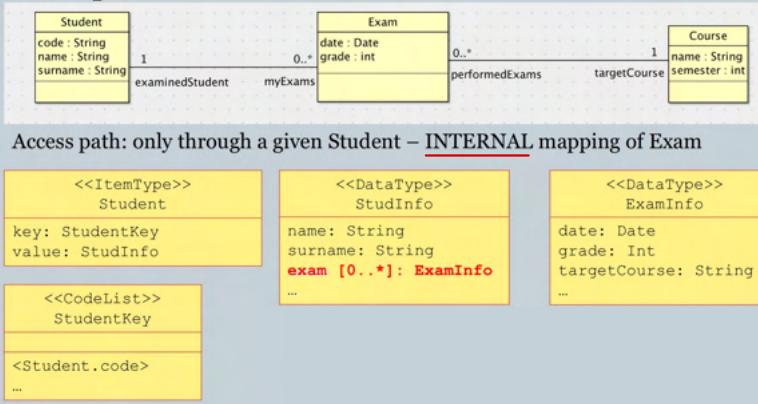
Access path: only through a given Student – EXTERNAL mapping of Exam  
EXAMPLE of record instances

This is not a Table! Item types will be mapped to tables later.

PartitionKey	SortKey	Attribute 1	Attribute 2
VR993	Info	name: 'Mario',	surname: 'Rossi'
VR993	Databases	date: '1/2/20',	grade: 23
VR993	Algebra	date: '13/3/20',	grade: 24
Algebra	Info	semester: 1	
Databases	Info	semester: 2	

### Internal mapping

### Example



Access path: only through a given Student – INTERNAL mapping of Exam  
EXAMPLE of record instances

This is not a Table! Item types will be mapped to table later.

Partition Key	Attribute 1	Attribute 2	Attribute 3
VR993	Name: 'Mario'	Surname: 'Rossi'	Exam: [{Date: '1/2/20', Grade: 23, targetCourse: 'Databases'}, {Date: '13/3/20', Grade: 24, targetCourse: 'Algebra'}]
Algebra	Semester: 1		
Databases	Semester: 2		

### Associazioni molti a molti

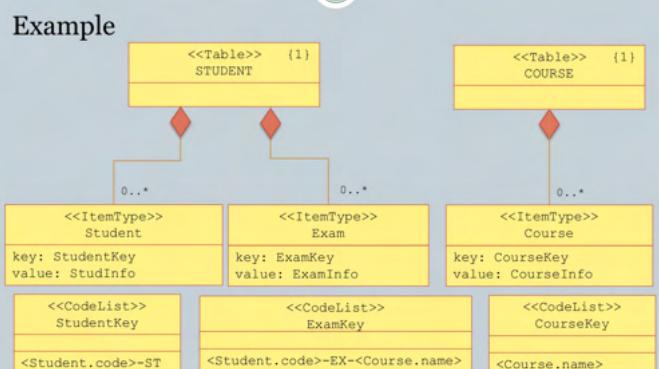
Per ogni associazione molti a molti che connette Cap con un'altra main classe Cap, ci sono due possibili scelte:

- Internal mapping in Cap: definiamo un attributo multivalue nel datatype che rappresenta le istanze di Cap per memorizzare tutte le chiavi degli item collegati di Cap
- Internal mapping in Cap: definiamo un attributo multistore nel datatype che rappresenta le istanze di Cap per memorizzare tutte le chiavi degli item collegati a Cap

### Mapping dell' item type alle tabelle

- Mapping normalizzato: una tabella per ciascun item type
- Mapping basato sull'access-path: Ogni Cap in una tabella assieme agli item dipendenti dalla loro chiave (chiave composita)
- One table mapping: tutti gli oggetti nella stessa tabella: richiede di aggiungere un'etichetta item type

### Esempio di mapping access path:



## Indice secondario

• È possibile creare un indice secondario per consentire l'accesso agli elementi utilizzando una chiave di accesso diversa dalla chiave primaria.

• Introduciamo uno stereotipo per l'indice secondario << Secondary index >> per personalizzare il classificatore.

• Ogni indice secondario è definito su una collezione di oggetti e consente di rappresentare una voce che contiene la mappatura tra:

- Un valore della chiave di accesso
- Un valore della chiave di un elemento

Ogni indice secondario contiene esattamente un attributo chiamato **access Key**, per memorizzare la chiave di accesso e un attributo chiamato **Key** per memorizzare la corrispondente **item Key**.

## Document database

Definizione: un document database è un database non relazionale che memorizza dati come documenti strutturati generalmente in formato XML o JSON.

- La definizione non dice nulla sul fatto che document DB implementano le transazioni ACID o sul modello clustering che può implementare.
- Tuttavia in genere tali sistemi forniscono un modesto supporto transazionale.

Introdotti perché:

- Possono gestire il conflitto tra i database relazionali e la programmazione orientata agli oggetti.
- Si adattano al paradigma dominante in contesto web based: il modello di programmazione AJAX.
- Aggiungono uno schema autocrittivo ai dati e forniscono accesso tramite query al database, cosa che era assente nei sistemi key-value puri.
- MongoDB divenne la scelta principale di molti web developer.

## Il primo Document DB: XML

• Appare come risultato di uno sviluppo migliore per la rappresentazione di dati semistruzzurati.

• Nel 2008 rappresentava lo standard

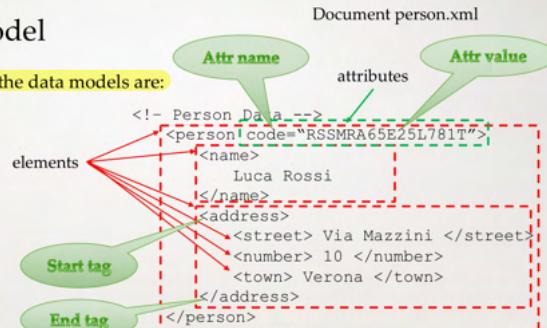
• XML schema: è un linguaggio formale per progettare la struttura di collezioni di documenti XML.

• XPath e XQuery sono linguaggi formali che permettono allo USER di navigare il primo e fare query il secondo su documenti XML.

• XSLT è un linguaggio per trasformare documenti XML in altri documenti XML o in istanze di dati di altri formati.

### XML data model

- Basic constructs of the data models are:
  - Elements
  - Attributes



The document person.xml contains a root element:

<person>

The element <person> has a structure: it contains an element <name> and an element <address>.

The element <name> contains only text.

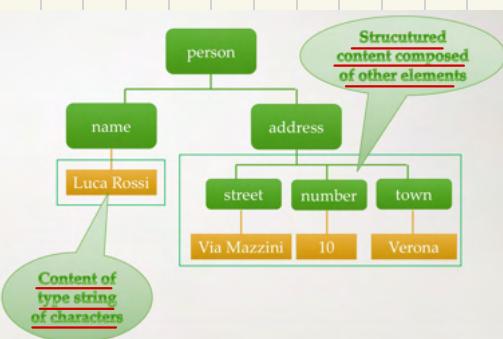
The element <address> has its own structure, containing the elements: <street>, <number> e <town>.

The element <person> has an attribute code.

## Rappresentazione grafica di un documento XML

Ogni documento XML può essere visto come un albero di elementi.

rappresentazione di person.xml



### Esempio 2

```
<messages>
  <note id="501">
    <to>Tove</to>
    <from>Jani</from>
    <body>Don't forget me this weekend!</body>
  </note>
  <note id="502">
    <to>Jani</to>
    <from>Tove</from>
    <heading>Re: Reminder</heading>
    <body>I will not</body>
  </note>
  <note id="505">
    <to>Jani</to><from>Tove</from>
    <body>I will not</body>
    <cc>John</cc>
    <cc>Rose</cc>
  </note>
</messages>
```

### Punti chiave del data model XML

- I dati sono modellati come una collezione di documenti
- Ogni documento ha una struttura complesso e variabile
- Ogni istanza di informazione è rappresentata da un elemento
- Il contenuto dell'elemento può essere strutturato o rappresentato come testo puro
- Può essere applicata la Data encapsulation producendo rindondanza ma aziendo così accessi efficienti

## JSON vs XML document DB

- XML è stato criticato perché sposta spazio e perché è un processo troppo costoso
- Nuova proposta negli anni 2000 il JSON che permetteva vantaggi migliori
- È molto simile al linguaggio XML ma meno verboso.
  - I database XML sono usati come sistemi di gestione di contenuti: organizzano e mantengono collezioni di testi file in formato XML come documenti accademici o aziendali
  - I database JSON sono utilizzati per la loro velocità e flessibilità. Il formato JSON è leggero e facilmente utilizzabile dalle applicazioni web, rendendolo perfetto per memorizzare e manipolare dati dinamici in ambienti web interattivi, come le applicazioni che usano AJAX
- Un Document database basato su JSON è un sistema per memorizzare e gestire in un ambiente distribuito un insieme di documenti JSON

Il datamodel è caratterizzato da:

- Un'unità base di storage chiamato document e specificata in JSON corrisponde a una riga di tabella relazionale. Un documento ha una struttura complessa:
  - Contiene una o più coppie chiave - valore (delle attributi o proprietà)
  - Contiene uno o più array di valori o array di documenti
  - Permette di fare nesting di documenti con annessa incapsulazione e una struttura gerarchica complessa

- Una collezione o data bucket, ovvero un insieme di documenti aventi scopo comune, corrisponde a una tabella nel modello relazionale. I documenti di un stesso raccolta non devono necessariamente essere dello stesso tipo.
- L'istanza di un document database può essere descritta come un insieme di collezioni di documenti.

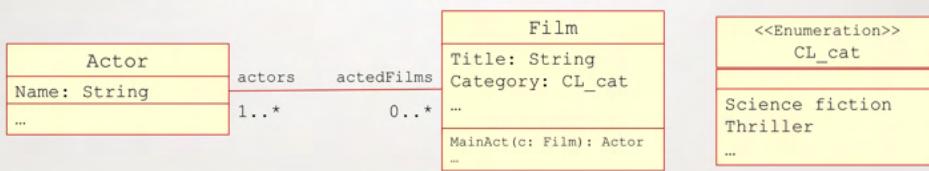
Il data model dei documenti database è il più complesso dei sistemi NoSQL, possiamo introdurre alcune considerazioni generali sulla modellazione dei dati, come nel modello relazionale:

- Teoricamente i documenti database potrebbero implementare uno schema in terza forma normata come i sistemi relazionali
- Più il numero di collezioni è inferiore e più il nesting dei documenti viene applicato

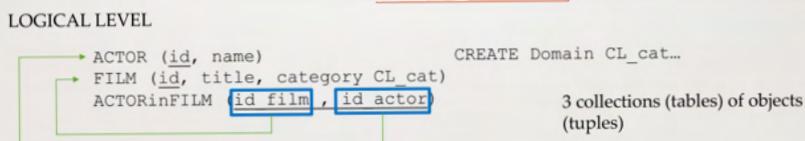
## Example of data modelling

We want to model a set of data for representing movies and actors.

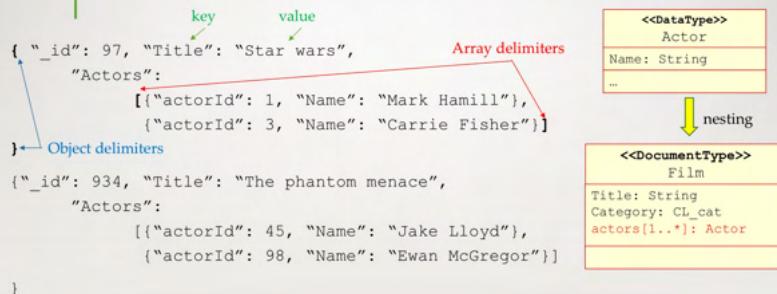
At conceptual level we use UML for the class description



Hopping attraverso Data model relazionale



## Mapping to the physical representation in JSON



## JSON database - CouchDB

- Sviluppato nel 2005 scritto in C++ e memorizza documenti JSON
- Incluse poi documenti JSON nel 2007 come formato principale
- Incluse successivamente altri paradigmi come una implementazione JS di MapReduce, la consistenza all'fine e multiple versioning model e uno sharding model hash based per permettere di scalare attraverso i nodi
- Altri sistemi interessanti nello stesso periodo era Membase che nel 2015 introduce un livello simile a SQL per l'accesso ai documenti N1QL

## N1QL

- È una versione di SQL per il document database implementata nei server di CouchDB
- Permette all'utente di specificare in modo dichiarativo query complesse su un document database
- Ha clausole aggiuntive rispetto a SQL per navigare attraverso alle parti nested di un documento

Considering the following bucket of documents (films) we show some queries in SQL++:

```
{ "_id": 97, "Title": "Star wars", "Category": "Science Fiction",
  Length: 125, "Actors":
    [{"actorId": 1, "Name": "Mark Hamill"}, {"actorId": 3, "Name": "Carrie Fisher"}] }

{ "_id": 934, "Title": "Spiderman", "Category": "Science Fiction",
  Length: 133, "Actors":
    [{"actorId": 45, "Name": "Tom Holland"}, {"actorId": 98, "Name": "Zendaya"}] }
```

- ① Find the title of the film with id = 97.

```
SELECT 'Title' FROM films WHERE _id = 97;
```

The query execution will produce the following documents listed in the property "results":

```
"results": [ {"Title": "Star wars" } ],
  "status": "success"
```

- ② Find the name of the first actor of the films of category "Science fiction".

```
SELECT Actors[0].'Name' FROM films
WHERE 'Category' = 'Science fiction';
```

```
"results": [ {"Name": "Mark Hamill"}, {"Name": "Tom Holland" } ],
  "status": "success"
```

- ③ Find the title and the category of the films having among their actors at least one actor of name "Tom Holland".

```
SELECT 'Title', 'Category' FROM films
WHERE ANY Actor IN films.Actors SATISFIES
  (Actor.'Name' = "Tom Holland") END;
```

variable name  
"results": [ {"Title": "Spiderman", "Category": "Science Fiction" } ],
 "status": "success"

- ④ Find the title of the films that last more than 130 minutes together with the name of their actors/actresses.

```
SELECT x.'Title', y.'Name' FROM films x UNNEST x.Actors y
WHERE x.'Length' > 130;
```

```
"results": [ {"Title": "Spiderman", "Name": "Tom Holland" },
  {"Title": "Spiderman", "Name": "Zendaya" } ],
  "status": "success"
```

- Suppose to have another document bucket, called `award`, containing the films that have been selected for a film award. It contains documents having an attribute `filmID`.
- Find the title of the films that have been selected for the film award.

```
SELECT x.'Title' FROM award
    JOIN films x ON KEYS award.filmID;
"results": [ {"Title": "Star wars" } ],
"status": "success"
```

Suppose that `award` contains only the film with `_id: 97`.

• **SQL++ permette di scrivere tutte le query di SQL 2**

• Può anche esprimere condizioni su informazioni nested all'interno dei documenti

## Mongo DB

- Json oriented database document, open source
- Memorizza internamente i documenti in una variante codificata binaria di JSON detta BSON
- BSON supports un minore overhead di parsing
- MongoDB offre funzionalità di query basate su Javascript molto ricca ed espanso tramite la propria shell
- Database predefinito nella moderna community di sviluppo web
- Le collezioni di documenti includono un metodo `find()` che consente di costruire query complesse

### Document bucket used in the following examples

```
{ "_id": 97, "Title": "Star wars", "Category": "Science Fiction",
  Duration: 125, "Actors":
    [{"actorId": 1, "Name": "Mark", "Surname": "Hamill"}, 
     {"actorId": 3, "Name": "Carrie", "Surname": "Fisher"}] }
{ "_id": 934, "Title": "Spiderman", "Category": "Science Fiction",
  Duration: 133, "Actors":
    [{"actorId": 45, "Name": "Tom", "Surname": "Holland", "Age": 24}, 
     {"actorId": 98, "Name": "Zendaya", "Surname": "Coleman", "Age": 24} ] }
```

## Queries in MongoDB

### Simple queries:

- Find all films:  
`db.films.find( {} )`
- Find films of Category "Science fiction":  
`db.films.find( {Category: "Science fiction"} )`
- Find films of Category "Science fiction" and Length equal to 125 minutes:  
`db.films.find( {Category: "Science fiction", Duration: 125} )`
- Find films of Category "Science fiction" or "Horror":  
`db.films.find( {Category:{$in:["Science fiction", "Horror"]}} )`

### Queries on nested documents

- Find all films having at least one actor of name «Tom»:  
`db.films.find( {Actors.Name: "Tom"} )`
- Find all films having at least one actor under the age of 25:  
`db.films.find( {Actors.Age: { $lt : 25 } } )`

## Operatori Logici

Name	Description
<code>\$and</code>	Joins query clauses with a logical AND returns all documents that match the conditions of both clauses.
<code>\$not</code>	Inverts the effect of a query expression and returns documents that do <i>not</i> match the query expression.
<code>\$nor</code>	Joins query clauses with a logical NOR returns all documents that fail to match both clauses.
<code>\$or</code>	Joins query clauses with a logical OR returns all documents that match the conditions of either clause.

Name	Description
<code>\$eq</code>	Matches values that are equal to a specified value.
<code>\$gt</code>	Matches values that are greater than a specified value.
<code>\$gte</code>	Matches values that are greater than or equal to a specified value.
<code>\$in</code>	Matches any of the values specified in an array.
<code>\$lt</code>	Matches values that are less than a specified value.
<code>\$lte</code>	Matches values that are less than or equal to a specified value.
<code>\$ne</code>	Matches all values that are not equal to a specified value.
<code>\$nin</code>	Matches none of the values specified in an array.

## Queries with projections

- Find the title of the films of Category "Science fiction":

```
db.films.find( {Category: "Science fiction"}, {title: 1} )
```

- Find the title of the films of Category "Science fiction" but the property \_id:

```
db.films.find( {Category: "Science fiction"},  
               {title: 1, _id: 0} )
```

- Find the id of the films and the name of their actors/actresses:

```
db.films.find( { }, { _id: 1, Actors.Name: 1} )
```

## Join queries (initially not supported by MongoDB)

In order to execute a join query it is necessary to use the function aggregate with the \$lookup option:

```
db.films.aggregate([  
  { $lookup:  
    { from: "award",  
      localField: "_id",  
      foreignField: "filmID",  
      as: "awards"  
    }  
  }])
```

20

## Queries in MongoDB

### Join query result

```
{ "_id": 97, "Title": "Star wars", "Category": "Science Fiction",  
  "Duration": 125, "Actors":  
  [{"actorId": 1, "Name": "Mark", "Surname": "Hamill"},  
   {"actorId": 3, "Name": "Carrie", "Surname": "Fisher"}],  
  "award": [{"_id": 1, "filmID": 97, "name": "SF_Prize", "type": "Best soundtrack"},  
            {"_id": 2, "filmID": 97, "name": "SF_Prize", "type": "Best director"}]}  
  
{"_id": 934, "Title": "Spiderman", "Category": "Science Fiction",  
  "Duration": 133, "Actors":  
  [{"actorId": 45, "Name": "Tom", "Surname": "Holland", "Age": 24},  
   {"actorId": 98, "Name": "Zendaya", "Surname": "Coleman", "Age": 24}]}  
awards:  
  {_id = 1, filmID: 97,  
   name: "SF-Prize",  
   type: "Best soundtrack"}  
  {_id = 2, filmID: 97,  
   name: "SF-Prize",  
   type: "Best director"}
```

### "Group By" queries:

```
db.films.aggregate([  
  {$match: {"Duration": {$lt: 150} } },  
  {$project: {"Category": 1} },  
  {$group: {"_id": "$Category",  
            "count": { $sum: 1} },  
  {$sort: { "count": -1} },  
  {$limit: 5}  
])
```

Other group by accumulators: \$avg, \$min, \$max, \$first, \$last

### Equivalent SQL query:

```
SELECT Category, count(*) as count  
FROM films  
WHERE Duration < 150  
GROUP BY Category  
ORDER BY count DESC  
LIMIT 5
```

## Document Database

### N1QL (SQL++)

- Si pronuncia Nikol ed è una versione di SQL per i document database implementata nei server Couchbase 4.0
- Permette allo user di specificare in maniera dichiarativa query complesse su un database di documenti contenenti differenti buckets di istanze di documenti
- Ha clausole aggiuntive rispetto a SQL per navigare attraverso le parti nested di un documento

### SQL++ vs RC

Considering the following relational schema concerning some tests performed on patient in medical centers:

- MEDICAL\_CENTER(CenterCode, Name, Municipality, Region, Lat, Long);
- TEST(Date, Patient, Type, Time, Processed, Result, Center)
- PATIENT(PatCode, Age, Nationality, DateOfBirth, ResMunicipality, ResRegion)

Referential integrity constraints:

- TEST.Center → MEDICAL\_CENTER
- TEST.Patient → PATIENT

Supponiamo di rappresentare i JSON con due bucket di documenti: uno per i centri e uno per i pazienti.

Centers	Patients
{ "_id": 1, "Code": "C1", "Name": "Centro VR1", "Municipality": "Verona", "Region": "Veneto", "Lat": 45, "Long": 13.1 }	{ "_id": 13, "Code": 12392, "Age": 34, "Nationality": "ita", "DateOfBirth": "10-10-1987", "ResMunicipality": "VR", "ResReg": "Veneto", "Tests": [ { "_id": 103, "Date": "1-1-2021", "Time": "07:03", "Processed": true, "Result": "positive", "Center_id": 1, "Type": "molecular swab"} ] }
{ "_id": 2, "Code": "C2", "Name": "Centro VR2", "Municipality": "Verona", "Region": "Veneto", "Lat": 45, "Long": 13.2 }	{ "_id": 123, "Code": 12392, "Age": 61, "Nationality": "ita", "DateOfBirth": "12-12-1960", "ResMunicipality": "VI", "ResReg": "Veneto", "Tests": [ { "_id": 133, "Date": "1-3-2021", "Time": "11:13", "Processed": true, "Result": "negative", "Center_id": 3, "Type": "rapid test"}, { "_id": 412, "Date": "1-4-2021", "Time": "11:43", "Processed": true, "Result": "negative", "Center_id": 2, "Type": "molecular swab"} ] }
{ "_id": 3, "Code": "C3", "Name": "Centro VII", "Municipality": "Vicenza", "Region": "Veneto", "Lat": 44.9, "Long": 14 }	

### Query modello relazionale vs SQL++

1. Q1: find the medical center of Veneto that are located neither in 'Verona' nor in 'Venice', reporting in the result the code, the name and the coordinates (lat, long) of the center.

RC:  $\{c, n, lat, long \mid \exists m, r. (\text{MEDICAL\_CENTER}(c, n, m, r, lat, long) \wedge r = 'Veneto' \wedge \neg(m = 'Verona' \vee m = 'Venice'))\}$

SQL++:

```
SELECT Code, Name, Lat, Long
FROM Centers
WHERE Region = "Veneto" AND
      NOT (Municipality = "Verona" OR Municipality = "Venice");
```

SQL++:

```
SELECT Code, Name, Lat, Long
FROM Centers
WHERE Region = "Veneto" AND
      Municipality NOT IN ["Verona", "Venice"];
```

2. Q2: find the patients with age greater than 55 and nationality 'english' or 'irish', reporting in the result the code, the municipality of residence and the nationality of the patient:

RC: {**c, rm, n** |  $\exists a, db, r. (\text{PATIENT}(c, a, n, db, rm, r) \wedge a > 55 \wedge (n = \text{'english'} \vee n = \text{'irish'}))}$ }

SQL++:

```
SELECT Code, ResMunicipality, Nationality
FROM Patients
WHERE Age > 55 AND
    Nationality IN ["english", "irish"]
```

3. Q3: find all the tests of the patients coming from 'Sicily', reporting in the result the patient code, the result and the date of the tests.

RC: {**(c, r, d)** |  $\exists a, n, db, rm. (\text{PATIENT}(c, a, n, db, rm, \text{'Sicily'})) \wedge \exists ty, tm, p, ct. (\text{TEST}(d, c, ty, tm, p, r, ct))}$ }

SQL++:

```
SELECT pat.Code, t.Result, t.Date
FROM Patients AS pat
UNNEST pat.Tests AS t
WHERE pat.Region = "Sicily"
```

4. Q4: find the patients from 'Lombardy' that in the same day have done one test of type 'rapid test' and another test of type 'molecular swab', reporting in the result the code and the nationality of the patient together with the date in which these tests were done.

RC: {**c, n, d** |  $\exists a, db, rm. (\text{PATIENT}(c, a, n, db, rm, \text{'Lombardy'})) \wedge \exists tm, p, r, ct. (\text{TEST}(d, c, \text{'rapid test'}, tm, p, r, ct)) \wedge \exists tm', p', r', ct'. (\text{TEST}(d, c, \text{'molecular swab'}, tm', p', r', ct'))}$ }

SQL++:

```
SELECT pat.Code, pat.Nationality, t1.Date
FROM Patients AS pat
UNNEST pat.Tests AS t1 UNNEST pat.Tests AS t2
WHERE pat.Region = "Lombardy" AND t1.Date = t2.Date AND
    t1.Type = 'rapid test' AND t2.Type = 'molecular swab'
```

5. Q5: find for each patient, if they exist, the pairs of consecutive positive tests of type 'molecular swab' reporting the code of the patient and the date of both tests (use only the attribute "Date" for ordering the tests on the temporal axis).

RC: {**c, d<sub>1</sub>, d<sub>2</sub>** |  $\exists a, n, db, m, r. (\text{PATIENT}(c, a, n, db, m, r)) \wedge \exists tm, p, ct. (\text{TEST}(d_1, c, \text{'molecular swab'}, tm, p, \text{'true'}, ct)) \wedge \exists tm_2, p_2, ct_2. (\text{TEST}(d_2, c, \text{'molecular swab'}, tm_2, p_2, \text{'true'}, ct_2)) \wedge d_1 < d_2 \wedge \neg \exists tm_3, p_3, ct_3. (\text{TEST}(d_3, c, \text{'molecular swab'}, tm_3, p_3, \text{'true'}, ct_3)) \wedge d_1 < d_3 \wedge d_3 < d_2)}$ }

SQL++:

```
SELECT pat.Code, t1.Date, t2.Date
FROM Patients AS pat
UNNEST pat.Tests AS t1 UNNEST pat.Tests AS t2
WHERE t1.Date < t2.Date AND t1.Type = 'molecular swab' AND
    t2.Type = 'molecular swab' AND t1.Result = 'positive' AND
    t2.Result = 'positive' AND
    NOT ANY tx IN pat.Tests
        SATISFIES t1.Date < tx.Date AND tx.Date < t2.Date END
```

6.

Q6: find the centers, that has done no one test on '1-1-2021' reporting in the result the code of the center and the tests done on '21-5-2021'.

RC:  $\{n, c, p_1, ty_1, tm_1, r_1 \mid \exists m, r, lt, lg. (MEDICAL\_CENTER(c, n, m, r, lt, lg) \wedge \neg \exists p, ty, tm, pr, r. (TEST('2021 - Jan - 1', p, ty, tm, pr, r, c)) \wedge \exists p_1. (TEST('2021 - May - 21', p_1, ty_1, tm_1, pr_1, r_1, c)))\}$

SQL++:

```
SELECT cen.Code, te AS tests
FROM Patients AS pat UNNEST pat.Tests te
JOIN Center AS cen ON KEYS te.Center_id
WHERE NOT cen._id
WITHIN (SELECT t.Center_id
        FROM Patients UNNEST Patients.Tests t
        WHERE t.Date = '1-1-2021')
AND te.Date = '21-5-2021'
```

## Query modello relazionale vs MongoDB

1. Q1: find the medical center of Veneto that are located neither in 'Verona' nor in 'Venice', reporting in the result the code, the name and the coordinates (lat, long) of the center.

RC:  $\{c, n, lat, long \mid \exists m, r. (MEDICAL\_CENTER(c, n, m, r, lat, long) \wedge r = 'Veneto' \wedge \neg(m = 'Verona' \vee m = 'Venice'))\}$

MongoDB:

```
db.Centers.find(
  { Region: "Veneto",
    $not: { $or: [{Municipality: "VR"}, {Municipality: "VE"}]} }
),
{Code: 1, Name: 1, Lat: 1, Long: 1})
```

2. Q2: find the patients with age greater than 55 and nationality 'english' or 'irish', reporting in the result the code, the ResMunicipality and the nationality of the patient.

RC:  $\{c, rm, n \mid \exists a, db, r. (PATIENT(c, a, n, db, rm, r) \wedge a > 55 \wedge (n = 'english' \vee n = 'irish'))\}$

MongoDB:

```
db.Patients.find(
  { Age: {$gt: 55},
    Nationality: { $in: ["english", "irish"] }
  },
  {Code: 1, ResMunicipality: 1, Nationality: 1})
```

3. Q3: find all the tests of the patients coming from 'Sicily', reporting in the result the patient code, the result and the date of the tests.

RC:  $\{(c, r, d \mid \exists a, n, db, rm. (PATIENT(c, a, n, db, rm, 'Sicily')) \wedge \exists ty, tm, p, ct. (TEST(d, c, ty, tm, p, r, ct)))\}$

MongoDB:

```
db.Patients.find( {Region: "Sicily"},
  {Code: 1, "Tests.Result": 1, "Tests.Date": 1})
```

5. Q5: find for each patient, if they exist, the pairs of consecutive positive tests of type 'molecular swab' reporting the code of the patient and the date of both tests (use only the attribute "Date" for ordering the tests on the temporal axis).

RC:  $\{c, d_1, d_2 \mid \exists a, n, db, m, r. (PATIENT(c, a, n, db, m, r)) \wedge \exists tm, p, ct. (TEST(d_1, c, 'molecular swab', tm, p, 'true', ct) \wedge \exists tm_2, p_2, ct_2. (TEST(d_2, c, 'molecular swab', tm_2, p_2, 'true', ct_2) \wedge d_1 < d_2 \wedge \neg \exists tm_3, p_3, ct_3. (TEST(d_3, c, 'molecular swab', tm_3, p_3, 'true', ct_3) \wedge d_1 < d_3 \wedge d_3 < d_2)))\}$

MongoDB: it cannot be done by using the function:

```
db.Patients.find(...)
```

6.

Q6: find the centers, that has done no one test on '1-1-2021' reporting in the result the code of the center and the tests done on '21-5-2021'.

```
{n, c, p1 | ∃m, r, lt, lg.(MEDICAL_CENTER(c, n, m, r, lt, lg) ∧
    ¬∃p, ty, tm, pr, r.(TEST('2021 - Jan - 1', p, ty, tm, pr, r, c)) ∧
    ∃ty1, tm1, pr1, r1.(TEST('2021 - May - 21', p1, ty1, tm1, pr1, r1, c)))}
```

MongoDB: it cannot be done by using the function:

```
db.Patients.find(...)
```

it can be done by using the function aggregate:

Q6: find the centers, that has done no one test on '1-1-2021' reporting in the result the code of the center and the tests done on '21-5-2021'.

MongoDB: using the aggregate function with a pipeline we can obtain this:

```
db.Patients.aggregate([
  { $unwind: "$Tests" },
  { $out: "Tests" }
]);
db.Patients.aggregate([
  { $unwind: "$Tests" },
  { $match: { "Tests.Date": "21-05-2021" } },
  { $out: "TestsOn210521" }
]);
db.Centers.aggregate([
  { $lookup: { from: "Tests", localField: "_id", foreignField: "Tests.Center_id", as: "TestsAtCenter" } },
  { $lookup: { from: "TestsOn210521", localField: "_id", foreignField: "Tests.Center_id", as: "TestsOn210521" } },
  { $match: { $not: { $eq: { "TestsAtCenter.Date": "1-1-2021" } } } }
]);
↑ Similar to not exists one test on 1-1-2021
```

## Regole di mapping da UML a Document DB

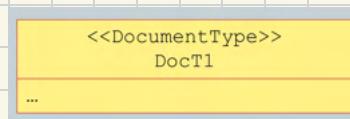
Primo passo: Definizione dei bucket da memorizzare in un Document DB

- Consideriamo l'insieme delle classi definite nel diagramma delle classi, scegliamo le classi che vogliamo rappresentare come collezioni di documenti in un bucket
- Cioè andrebbe fatto considerando l'access path e le query che avranno più frequentemente nell'applicazione
- E' il passo più importante

- Ogni Access path è definito assegnando una classe principale CAP che il path parte
- Per ogni access path deve essere definita una collezione di documenti
- E' necessario un document type per memorizzare le istanze della classe CAP con un set di attributi per rappresentare tutte le proprietà
- E' necessario includere le classi collegate a CAP con le associazioni, ma prima ci occupiamo del resto

## Secondo passo: Definizione del Document type

Definiamo il document type che verrà memorizzato in un Document DB bucket  
Per fare ciò è stato introdotto lo stereotipo <>Document type<>

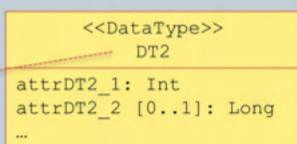
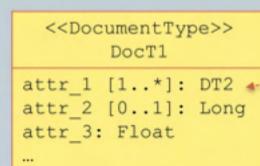


Ora è necessario specificare gli attributi che definiranno la struttura del documento

## Specificazione degli attributi

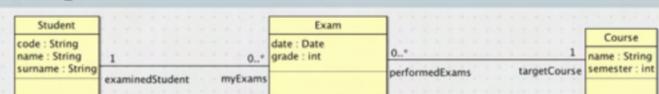
Per la definizione di un document type dobbiamo definire uno o più attributi. Ogni attributo può assumere

- valori semplici appartenenti a un tipo base
- valori complessi con una struttura: che sono specificati da datatype aggiuntivi che sono specificati nel datamodel

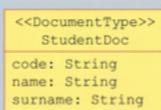


## Mapping della classe principale Cap

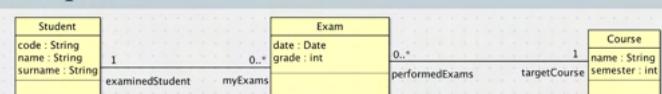
### Example



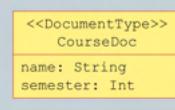
Access path: only through a given Student



### Example



Access path: only through a given Course



## Definizione di collezione di documenti:

Per ogni classe C che è collegata a Cap con una associazione uno a molti (incluso i casi in cui l'associazione è una composizione o una aggregazione) Ci sono 3 possibili scelte:

- Internal Mapping: definiamo un attributo multivalue aggiuntivo nel document type che rappresenti Cap per memorizzare tutte le istanze di C connesse a ogni documento di Cap. Un nuovo datatype verrà aggiunto per memorizzare tutti gli attributi di C incapabili in Cap

### External mapping con accesspath da Cap

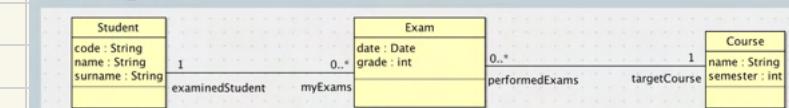
- Definiamo un documentType per C senza un link a Cap
- Definiamo un attributo in Cap per memorizzare i collegamenti al documento C connessi al documento corrente di Cap

### External mapping con accesso indipendente

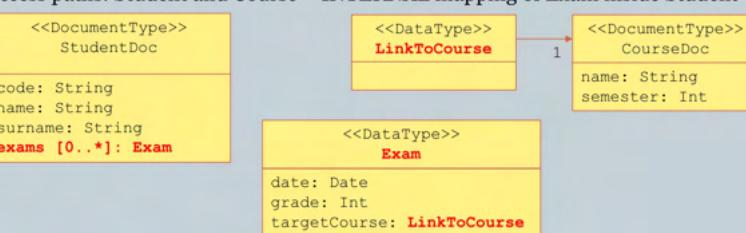
- Definiamo un documentType per C includendo anche un attributo per rappresentare il link a Cap
- Definiamo un attributo in Cap dove viene memorizzato il link al documento di C collegato al documento corrente di Cap

## Mapping di una classe collegata uno a molti con Cap

### Example



Access paths: Student and Course – INTERNAL mapping of Exam inside Student



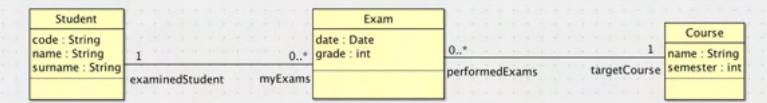
## Access path: Student and Course – INTERNAL mapping of Exam

EXAMPLE of document instances

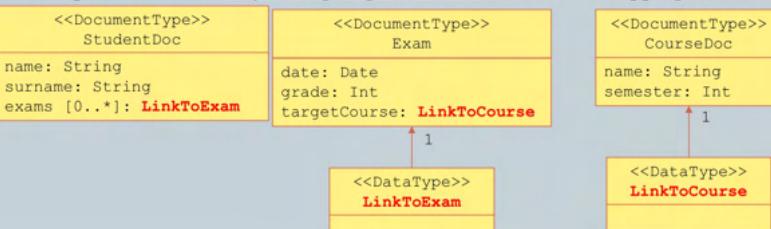
```
{
  "_id": 97, "Code": "VR0098", "name": "Mario", "surname": "Rossi",
  "exams": [
    [{"targetCourse": 23, "date": "10/1/20", "grade": 23},
     {"targetCourse": 7, "date": "3/7/21", "grade": 28}] ]
  {"_id": 934, "Code": "VR0011", "name": "Laura", "surname": "Verdi",
  "exams": [
    {"targetCourse": 13, "date": "13/3/20", "grade": 25} ] }

  {"_id": 23, "name": "Databases", "semester": 2 }
  {"_id": 7, "name": "Programming languages", "semester": 1 }
  {"_id": 13, "name": "Algebra", "semester": 1 }
}
```

## Example



## Access path for exams: only through a given Student – External mapping of Exam



## Access path: only through a given Student – External mapping of Exam

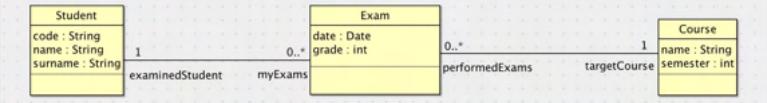
EXAMPLE of document instances

```
{
  "_id": 97, "Code": "VR0098", "name": "Mario", "surname": "Rossi",
  "exams": [ 2, 8 ] }
  {"_id": 934, "Code": "VR0011", "name": "Laura", "surname": "Verdi",
  "exams": [ 9 ] }

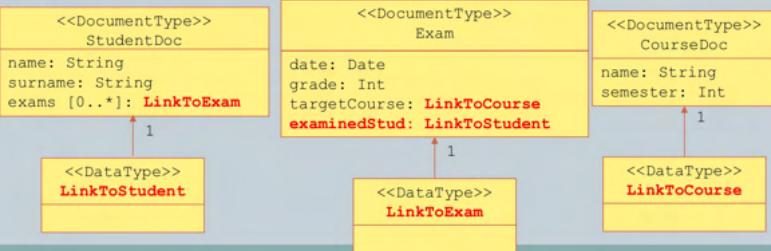
  { "_id": 2, "targetCourse": 23, "date": "10/1/20", "grade": 23 }
  { "_id": 8, "targetCourse": 7, "date": "3/7/21", "grade": 28 }
  { "_id": 9, "targetCourse": 13, "date": "13/3/20", "grade": 25 }

  {"_id": 23, "name": "Databases", "semester": 2 }
  {"_id": 7, "name": "Programming languages", "semester": 1 }
  {"_id": 13, "name": "Algebra", "semester": 1 }
}
```

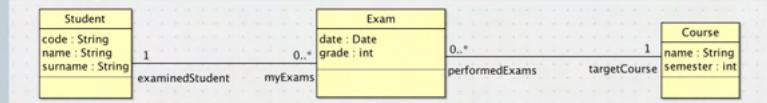
## Example



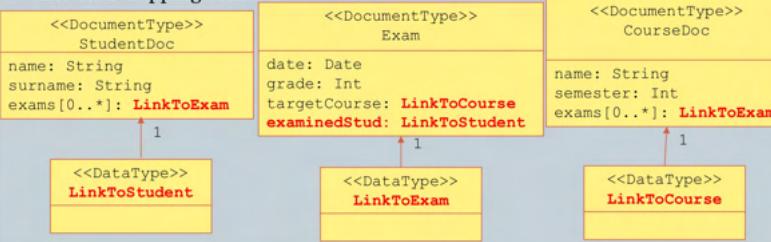
## Access path for exams: independent access (for Student) - External mapping of Exam



## Example



## Access path for exams: independent access (for both Student and Course) - External mapping of Exam



## Access path: Student, Course and Exam – External mapping of Exam

EXAMPLE of document instances

```
{
  "_id": 97, "Code": "VR0098", "name": "Mario", "surname": "Rossi",
  "exams": [ 2, 7 ] }
{
  "_id": 934, "Code": "VR0011", "name": "Laura", "surname": "Verdi",
  "exams": [ 9 ] }

  { "_id": 2, "targetCourse": 23, "date": "10/1/20", "grade": 23,
    "examinedStud": 97 }
  { "_id": 7, "targetCourse": 7, "date": "3/7/21", "grade": 28,
    "examinedStud": 97 }
  { "_id": 9, "targetCourse": 13, "date": "13/3/20", "grade": 25,
    "examinedStud": 934 }

{ "_id": 23, "name": "Databases", "semester": 2 }
{ "_id": 7, "name": "Programming languages", "semester": 1 }
{ "_id": 13, "name": "Algebra", "semester": 1 }
```

Associazioni molti a molti:

- Per ogni associazione molti a molti che collega Cap con un'altra main class Cap'. Abbiamo 4 possibili scelte

- Internal mapping in Cap: definiamo un attributo multivalue aggiuntivo nel document type che rappresenti Cap per memorizzare tutte le referenze dei documenti collegati di Cap'
- Internal mapping in Cap': definiamo un attributo multivalue aggiuntivo nel document type che rappresenti Cap' per memorizzare tutte le referenze dei documenti collegati di Cap
- Internal mapping in entrambe le classi Cap e Cap': definiamo un attributo multivalue aggiuntivo in entrambe le classi. La ridondanza deve essere gestita correttamente in questo caso
- External mapping: definiamo un document type per rappresentare le istanze delle associazioni. Approccio simile al relazionale.

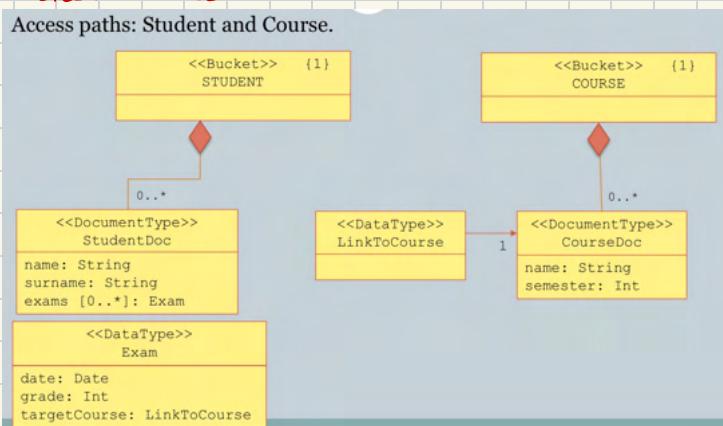
Mapping del document type ai bucket

Qui c'è una sola possibile scelta:

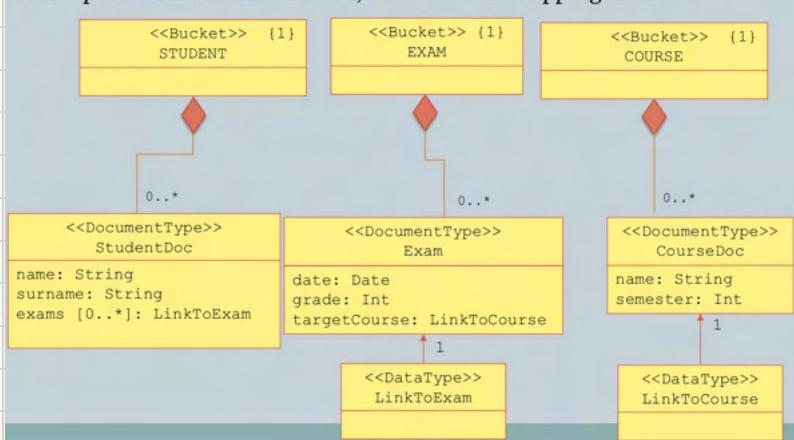
Un bucket per ogni document type che è stato definito nel data model

Collezione di documenti:

Access paths: Student and Course.



Access paths: Student and Course, with external mapping for exams.



## Database Colonnare

- Quando i primi file digitali sono stati creati ogni pezzo di informazione era stato rappresentato con un record, ovvero come una riga
- OLTP (Online Transaction Processing) era il carico di lavoro più importante per i database relazionali quando questi ultimi fecero la comparsa sulla scena
- L'OLTP si basa sull'elaborazione di un record alla volta
- Tuttavia con l'aumento di carichi di lavoro di data warehousing e analisi dei dati, l'organizzazione fisica orientata alle righe è diventata meno ideale, a favore di un accesso orientato alle colonne dei dati memorizzati
- Negli anni 90 un numero crescente di database relazionali iniziarono a supportare applicazioni di analitica e di decision support chiamati anche **data warehouse** che lavorano in parallelo con i sistemi OLTP
- L'acronimo OLAP (Online Analytic Processing) differenzia tale carico di lavoro rispetto a quello dei sistemi OLTP
- I sistemi OLAP richiedono schemi diversi dai sistemi OLTP
- Vennero definiti gli schemi a stella per alimentare le applicazioni OLAP

## Data Warehousing

- È un processo per l'estrazione e l'integrazione di dati storici da sistemi transazionali differenti e non omogenei (OLTP) per supportare il sistema decisionale dell'azienda

Database → conoscenza utile per l'azienda

### Motivazioni del data warehousing

- I dati memorizzati possono essere utili per supportare decisioni sulle scelte future dell'azienda

Sfide:

- Grandi quantità di dati da elaborare
- Molti database eterogenei da integrare
- I dati devono essere raggruppati e classificati:

nuove soluzioni per supportare OLAP sono necessarie. Gli RDBMS non sono sufficienti: il loro target è il carico di lavoro OLTP



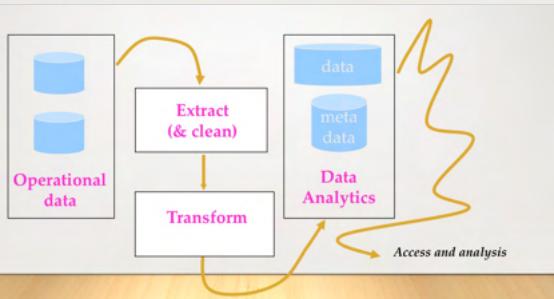
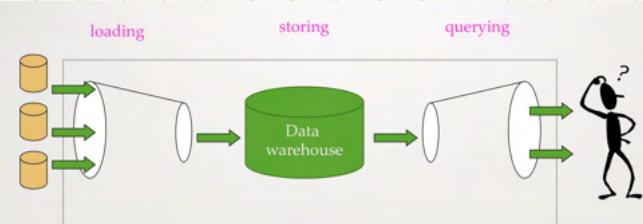
Il data warehouse è una collezione di informazioni che è

- **Orientata al soggetto**: considera il soggetto di interesse per l'azienda e non dipende dai processi organizzativi
- **Integrato**: i dati provengono da sorgenti differenti e con formati differenti. Perciò dovrò integrare i dati per memorizzarli in un formato consistente  
ad es: l'indirizzo può essere rappresentato sotto forma di stringa o da campi separati: stato, città, nazione
- **Completa al tempo**: i dati sono necessari per confrontare diversi scenari e prevedere nuove criticità, situazioni o per estrarre tendenze. ad es: dati che descrivono vecchi clienti possono aiutare a prevedere quanti clienti chiuderanno il loro account il prossimo anno
- **Persistente**: una volta caricati nel DW dei dati non è possibile aggiornarli è possibile solo cancellarli tutti e ricaricare nuove collezioni aggiornate



- **Orientata al support decisionale**

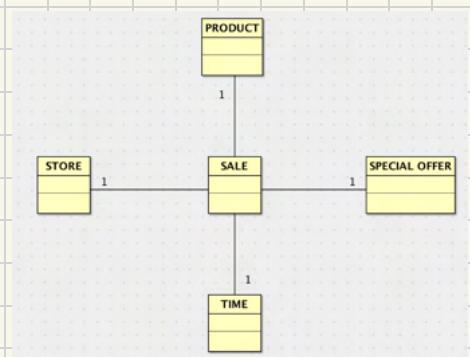
## Architettura del data warehouse



## Schema del data warehouse

Lo schema di un data warehouse è diviso in parti dette DATA MART. Ogni DATA MART indirizza un goal specifico dell'analisi.

Ogni DATAMART ha uno schema concettuale che spesso è rappresentato da una stella.



Al centro dello schema a stella c'è una collezione di dati che rappresentano i FACTS i cui attributi memorizzano la misura dei FACTS.

Alle estremità della stella la dimensione dell'analisi è rappresentata (i cui attributi rappresentano i diversi livelli di dimensione).

## Data analysis

Questi sistemi supportano le decisioni, poiché consentono all'utente di scrivere query (tramite interfaccia grafica in genere) che estraendo e generano report, tabelle e diagrammi.

Le informazioni estratte dal data mart riassumono i fatti del data warehouse applicando funzioni di aggregate.

## Scrivere query in interfaccia grafica

Esempio

dim1	dim2	fact (F.c)	fact (F.d)
current values	current values		
selection	selection		
group by	group by	OP1 <sub>Aggr</sub>	OP2 <sub>Aggr</sub>

## Equivalent SQL query

```

SELECT D1.a, D2.b, OP1Aggr(F.c), OP2Aggr(F.d)
FROM Facts AS F, Dim1 AS D1, Dim2 AS D2
WHERE <join predicates (F,D1)> AND
<join predicates (F,D2)> AND <selections>
GROUP BY D1.a, D2.b
ORDER BY D1.a, D2.b

```

## Evoluzione dei tool per l'analisi di un DWH



### Strumenti per MDA (multi dimensional analysis)

- La struttura dati per l'MDA può essere rappresentata come un cubo di Rubik di dati
- Lo user può modificare il cubo applicando diversi operatori che gli permettono di aggiungere o rimuovere una dimensione al cubo e di applicare funzioni di aggregazione ai dati in esso contenuti;
- In questo modo è possibile testare e analizzare vari scenari "what-if"

### Data Cube

#### Data Cube

Clauses: CUBE for an SQL query with GROUP BY.

Originale table T

Month	Store	Product	Qty
Jan	A	Wine	10
Feb	A	Wine	20
Mar	A	Water	30
Jan	B	Water	40
Feb	B	Wine	50
Mar	B	Wine	60
Jan	C	Wine	70
Feb	C	Water	80
Mar	C	Water	90

SELECT Month,  
Product, Sum(Qty)  
FROM T GROUP BY CUBE  
(Month, Product)

Result on next slide

#### Data Cube

Result of the previous query with the clause  
GROUP BY CUBE.

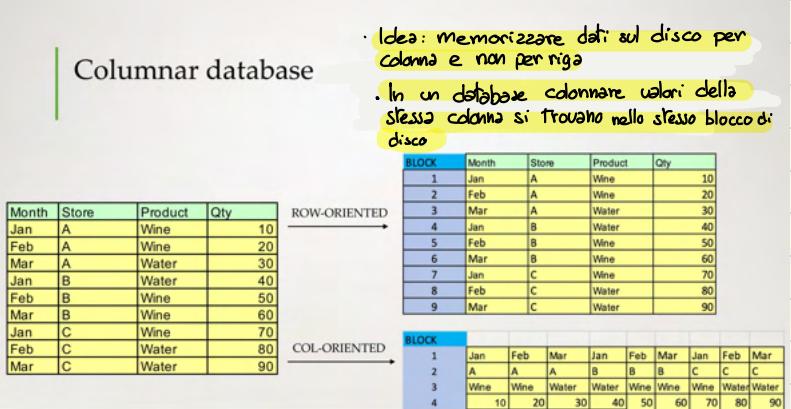
Month	Product	sum(Qty)
Jan	Wine	80
Feb	Wine	70
Mar	Wine	60
Jan	Water	40
Feb	Water	80
Mar	Water	120
Jan	ALL	120
Feb	ALL	150
Mar	ALL	180
ALL	Wine	210
ALL	Water	240
ALL	ALL	450

### Operatori sul cubo:

1. Slice and dice: selezione di un sottinsieme di celle nel cubo
2. Drill-down: da informazioni più aggregate a informazioni meno aggregate (aggiungi una dimensione
  - o un livello più specifico di una dimensione esistente)
3. Roll-up: da informazioni meno aggregate a informazioni più aggregate (elimina una dimensione
  - o si considera un livello più generale di una dimensione esistente)

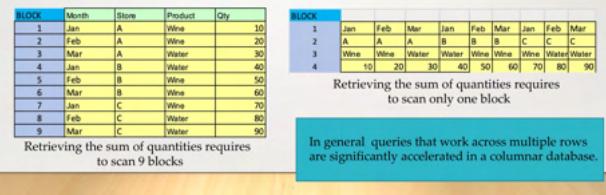
## Introduzione dei database colomnari

- Quasi tutti gli RDBMS adottarono lo schema di indicizzazione e di ottimizzazione SQL per accelerare le query su schemi a stella
- Nonostante queste ottimizzazioni, l'elaborazione dello schema a stella nei data warehouse è rimasta fortemente intensiva in termini di CPU e I/O
- Con la crescita del volume dei dati, è aumentato il malcontento verso le prestazioni tradizionali dei data warehouse
- Sono state proposte nuove soluzioni, tutte basate sull'approccio colomnare

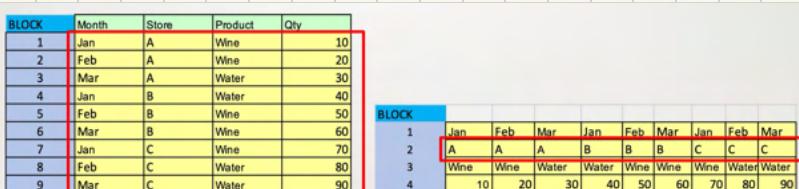


Vantaggi:

- Le query che mirano ad aggregare i valori di una specifica colonna sono ottimizzati, perché necessitano solo di accedere a pochi blocchi della memoria secondaria



- Gli algoritmi di compressione lavorano meglio sui database colomnari perché i dati con ripetizioni localizzate ottengono rapporti di compressione più elevati; inoltre l'overhead della CPU per la compressione si riduce notevolmente se può lavorare su blocchi di dati isolati



• Nei sistemi row-based la compressione è meno efficace

• Nei sistemi column-based la compressione può lavorare su ripetizioni locali

## Svantaggi

Ciò comporta grandi overhead su operazioni su singole righe, che sono le più frequenti in un'applicazione OLTP.  
 In un database columnare recuperare una singola riga richiede di assemblare le righe da ogni record colonnaire.  
 Le operazioni DML in particolare gli inserimenti, sono pesanti per i database colonnaari poiché non c'è modo di fare l'operazione senza modificare tutte le colonne per ciascuna riga.

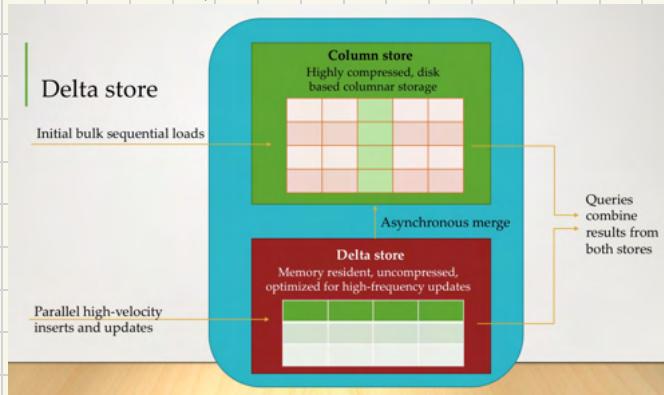
BLOCK	Month	Store	Product	Qty
1	Jan	A	Wine	10
2	Feb	A	Wine	20
3	Mar	A	Water	30
4	Jan	B	Water	40
5	Feb	B	Wine	50
6	Mar	B	Wine	60
7	Jan	C	Wine	70
8	Feb	C	Water	80
9	Mar	C	Water	90

BLOCK	1	2	3	4	5	6	7	8	9
1	Jan	Feb	Mar	Jan	Feb	Mar	Jan	Feb	Mar
2	A	A	A	B	B	B	C	C	C
3	Wine	Wine	Water	Water	Wine	Wine	Wine	Water	Water
4	10	20	30	40	50	60	70	80	90

- Nei sistemi column based molti blocchi devono essere acceduti per prendere una riga

## Database colonnari e data warehouse

- Molti data warehouse sono stati implementati usando un database colonnare ma le debolezze che riguardano le operazioni DML costringono a introdurre un classico processo ETL notturno (Extract, Transform, Load)
- Per superare questo problema e fornire informazioni "aggiornate" al sistema di supporto alle decisioni il sistema colonnaire ha introdotto una qualche forma di delta store ottimizzato per la scrittura.
- I delta store sono rappresentati in forma di riga. Il delta store risiede generalmente in memoria ed è in genere non compresso.
- Il contenuto del delta store riguarda la porzione del database che è stata recentemente modificata da operazioni DML.
- Il delta store è in grado di accettare modifiche ai dati ad alta frequenza.
- I dati nel delta store vengono periodicamente uniti al database principale.
  - L'unione viene eseguita periodicamente o quando si supera una determinata soglia
- Prima dell'unione, le query devono accedere sia al delta store che al column store per restituire risultati completi e accurati.



## Proiezioni

- Nel database colonnare ogni colonna di una tabella viene memorizzata in una sua area dedicata. Questo è sufficiente per supportare query su singole colonne, ma in pratica le query complesse richiedono la lettura di combinazioni di dati provenienti da più colonne.
- Per ovviare a ciò molti database colonnari archivano fisicamente le tabelle come una serie di proiezioni. Queste proiezioni contengono combinazioni di colonne frequentemente consultate insieme.
- Oltre alla proiezione base contenente l'intero set di colonne (super proiezione) vengono aggiunte proiezioni aggiuntive per supportare query specifiche.

### Projection (example)

Region	Cust	Prod	Sales
A	G	C	77
B	C	C	73
D	F	D	43
C	C	A	23
A	R	B	5

Super-projection

Region	A	B	C	A
Cust	G	C	F	C
Prod	C	C	D	A
Sales	77	73	43	23

Projection 1

Region	A	A	B	C	D
Prod	B	C	C	A	D
Sales	5	77	73	23	43

Projection 2

Cust	C	C	F	G	R
Sales	73	23	43	77	5

Projection in columnar databases are like indexes in a row-based store.

- Sistemi di questo tipo sono: C-store, Vertica e Apache Parquet

## Graph database

- Tutte le precedenti tecnologie per gestire i dati concordano su un punto: I database vengono utilizzati per memorizzare informazioni su " cose " (oggetti o concetti), che possono essere rappresentati come tuple di tabella, documenti in bucket o record in bigtable.
- Certe volte le relazioni tra " cose " sono di primo interesse.
- In questi ultimi casi usiamo il graph database.
- Alcune applicazioni che possono ottenere benefici la rappresentazione a grafi: social network, network topology...

### Definizione di grafo:

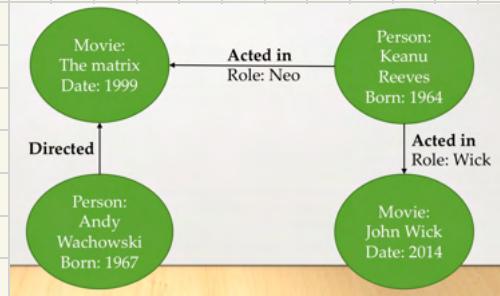
Il grafo è una struttura algebrica del tipo:

- Insieme di vertici:  $V$
- Insieme di archi:  $E \subseteq V \times V$
- In alcuni casi le label possono essere aggiunti sia a vertici che ad archi

Quando il grafo è usato come database, le label rappresentano proprietà e sono specificate con approccio chiave: valore, come ad esempio: dato un nodo che rappresenta una persona la seguente label può essere aggiunta ad essa per rappresentare il nome della persona

name: "John"

Esempio di grafo che rappresenta film e attori:



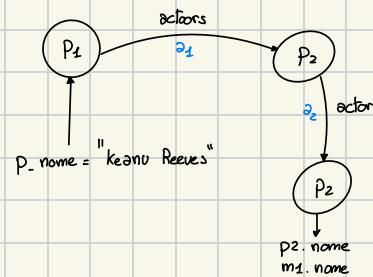
## Rappresentazione del grafo in RDBMS

```
• PEOPLE(PersonID, Name, BirthDate)  
• MOVIES(MovieID, Name, ReleaseDate)  
• DIRECTORS(MovieID, PersonID) ← relationships  
• ACTORS(MovieID, PersonID)  
  
Referential integrity constraints:  
DIRECTORS.MovieID → MOVIES, DIRECTORS.PersonID → PEOPLE  
ACTORS.MovieID → MOVIES, ACTORS.PersonID → PEOPLE
```

## Querying graph data in RDBMS

- La sintassi **SQL** non è adeguata per performare l'attraversamento di grafi. Specialmente quando la profondità non è nota. Per aggiungere un livello bisogna aggiungere una nuova sequenza di join
- Le performance delle query degradano velocemente quando più livelli di attrversamento vengono aggiunti:

```
SELECT p2.Name, m1.Name  
FROM PEOPLE p1 JOIN ACTORS a1 ON (p1.PersonID = a1.PersonID)  
JOIN MOVIES m1 ON (a1.MovieID = m1.MovieID)  
JOIN ACTORS a2 ON (a2.MovieID = m1.MovieID)  
JOIN PEOPLE p2 ON (p2.PersonID = a2.PersonID)  
  
WHERE p1.Name = 'Keanu Reeves'
```



## Resource Description Framework

RDF è uno standard web per rappresentare e processare i grafi che contengono dati. Le informazioni sono espresse in triple:

entità: attributo: valore

Per esempio:

The Matrix: is: Movie  
Keanu: is: Person  
Keanu: starred in: The Matrix

SPARQL è un linguaggio di interrogazione SQL-like per gli store RDF di triple.

## Neo4j

- È il sistema database a grafi più largamente usato
- Adotta il modello del grafo delle proprietà: due gli attributi possono essere associati ai nodi e agli archi del grafo
- Supporta grafi con milioni di nodi e transazioni ACID
- Ha un linguaggio di chiedere di interrogazione, detto Cypher

## Esempio di un grafo database

```
CREATE (TheMatrix: Movie {title:'The Matrix', released: 1999})  
CREATE (JohnWick: Movie {title:'John Wick', released: 2014})  
CREATE (Keanu: Person {name:'Keanu Reeves', born: 1964})  
CREATE (AndyW: Person {name:'Andy Wachowski', born: 1967})  
  
CREATE (Keanu)-[:ACTED_IN {roles:['Neo']}] -> (TheMatrix),  
      (Keanu)-[:ACTED_IN {roles:['John Wick']}] -> (JohnWick),  
      (AndyW)-[:DIRECTED] -> (TheMatrix)
```

## Esempi di query

```
Find the nodes that have a property name equal to "Keanu Reeves":  
MATCH (k: Person {name: "Keanu Reeves"}) RETURN k;  
Find the name of the director of the movie "Matrix":  
MATCH (m: Movie {title: "The Matrix"})-[:DIRECTED]->(d) RETURN d.name  
Find all actors who have ever acted in the same movie as Keanu Reeves.  
MATCH (k: Person {name: "Keanu Reeves"})-[:ACTED_IN]->(m)-[:ACTED_IN]->(y)  
RETURN y.name
```

## Processing dei grafi

- Processare query sui grafi può essere fatto in qualsiasi sistema come in RDBMS, ma con basse prestazioni con l'aumentare della profondità del grafo
- Il processing real time efficiente richiede un modo di muoversi attraverso il grafo senza indici
- I graph database nativi usano adiacenza senza indici per navigare nel grafo. Ogni nodo conosce la posizione fisica dei propri nodi di adiacenza. Perciò gli indici non servono
- I graph database nativi superano enormemente le altre implementazioni di attraversamento di grafo
- L'architettura di questo sistema presenta problemi quando viene applicata a sistemi distribuiti

## Conclusioni

- I graph database nativi sono molto efficienti nel graph processing, ma molto raramente invece sono ottimi anche per tutti i restanti carichi di lavoro delle applicazioni tipiche
- Questi sistemi non vogliono rimpiazzare gli RDBMS, ma per supportare altre tecnologie, specialmente quando è necessario una analisi delle relazioni
- Sono stati proposti i graph computer engines, sono in grado di elaborare grafi su dati conservati in altri formati e archiviati in altri sistemi (spesso Hadoop)

## Pattern Distribuiti: Internals

### Architetture distribuite: motivazioni

Gli amministratori delle web applications possono compiere due scelte quando le richieste dell'applicazione superano la capacità del database

- scaling up: aumentare la potenza dei singoli server
- scaling out: aumentare il numero di server

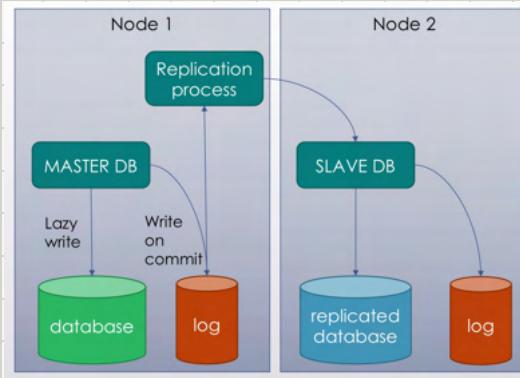
Per gestire le richieste delle web application una soluzione scaling out era sempre necessaria e una nuova generazione di sistemi distribuiti non relazionali nacque

### Replicazione: Primo passo nelle architetture distribuite

- La replicazione nei database venne adottata inizialmente per avere una maggiore disponibilità
- Veniva spesso sfruttato il log delle transazioni che molti sistemi relazionali usano per supportare le transazioni ACID
- Appena viene effettuato il commit di una transazione il record delle transazioni viene immediatamente scritto nel log così viene preservato per eventuali failure
- Il processo di replicazione può monitorare il log e applicare le modifiche a un database di backup, creando così una replica

## Processo di replica log based

La replicazione viene utilizzata per distribuire il carico della lettura in modo scalare orizzontale, sebbene le transazioni del database debbano comunque essere applicate al database master.



## Replicazione

- La replicazione può supportare solo semplici operazioni di lettura
- Spesso query complesse richieste dal carico di lavoro del data warehousing non può essere facilmente parallelizzate in una architettura con replication
- Sono necessarie nuove soluzioni

## Tassonomia dell'architettura dei database distribuiti

- Shared everything: Scenario classico con un nodo (il server) che condivide tutto: memoria, CPU e risorse del disco tra tutti i processi.
- Shared - disk: i processi possono esistere su nodi differenti nel cluster e hanno accesso alla memoria e alla CPU del loro nodo. Tutti i processi hanno accesso al disco dei dispositivi, che sono condivisi tra tutti i nodi del cluster.
- Shared - nothing: i processi hanno accesso solo alla memoria, alla CPU e al disco locali e ai loro sottosistemi del database (è applicato sharding). Perciò le query possono facilmente essere parallizzate su più nodi.

## Approccio Shared nothing

- È la base dei primi sistemi cluster database, poiché consente di parallelizzare l'esecuzione delle query
- Introduce però nuove problematiche per le transazioni ACID, a causa della necessità di coordinare le transazioni che modificano i dati su più nodi
- Il partizionamento dei dati: Tra i nodi diventa un aspetto critico, altrimenti il carico di lavoro del cluster si squilibrerà rapidamente

## Approccio Shared Disk

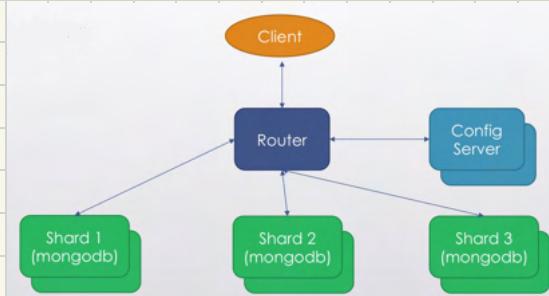
- Permette una Scalabilità maggiore e più elastica
- Non sono necessarie operazioni di ribilanciamento
- È più affidabile, poiché la failure di un nodo non comporta la perdita di dati
- La sfida in questo caso è coordinare la distribuzione dei dati tra i nodi
- Il sistema che gestisce i dischi condivisi può diventare un bottleneck quando la Scalabilità globale è richiesta

## Architettura dei database distribuiti della successiva generazione di sistemi (approccio shared nothing)

- Architettura sharding tradizionale: i dati sono suddivisi tra i nodi basandosi su una "shard-key"
- Hadoop / HBase model: un processo globale "master Omnisciente" determina dove i dati dovrebbero trovarsi
- Hashing model consistente di DynamoDB: una funzione matematica di hashing decide dove allocare i dati nei nodi del cluster

## Architettura distribuita di MongoDB

### MongoDB sharding e replication



### MongoDB sharding

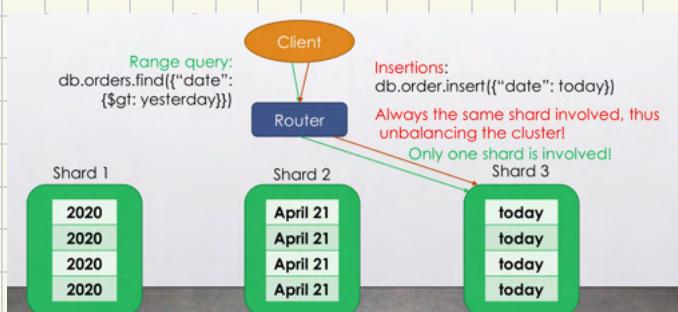
- Ogni shard è implementato da un distinto database MongoDB
- Un altro server MongoDB detto config server contiene i metadati che possono essere utilizzati per determinare come i dati sono distribuiti tra le shard
- Un processo router permette il routing di una richiesta alla shard corretta
- Per fare lo shard di una collezione di dati scegliamo una shard-key, che è composto da uno o più attributi indicizzati che verranno usati per distribuire i documenti fra le shard

### Meccanismo di Sharding di MongoDB

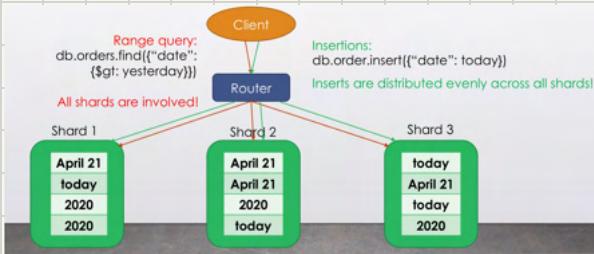
- La distribuzione dei documents tra le shard può essere sia:

- Range based: ad ogni shard è allocato uno specifico range di shard-key value
- Hash based: i documents sono distribuiti sulla base di una funzione hash applicata alla shard-key
- Ci sono pro e contro in ciascuno dei due meccanismi

### Sharding range-based



## Sharding hash-based



- Durante la vita del sistema il carico di lavoro può cambiare e la policy di sharding iniziale può non essere più valida, cioè sbilancia il cluster
- Per questo motivo MongoDB periodicamente valuta il bilanciamento delle shard ed effettua operazioni di ribilanciamento, se necessario.
- L'unità di ribilanciamento è detta shard chunk (tipicamente di dimensione 64 MB). Ciascun chunk contiene valori continui di shard key
- Se una shard è diventata sbilanciata, il balancer può muovere chunk da uno shard all'altro

## Mongo DB replication

- In MongoDB i dati dei nodi possono essere replicati attraverso i replica set
- Un replicaset è costruito da un nodo primario insieme a due o più nodi secondari. Il nodo primario accetta tutte le richieste di scrittura, che vengono prorogate in modo asincrono ai nodi secondari
- I nodi primari non sono fissi, ma vengono determinati da un'elezione che coinvolge tutti i nodi del replicaset. Per diventare primario, un nodo deve essere in grado di contattare più della metà dei membri del replicaset
- Il nodo primario memorizza le informazioni sulle modifiche ai documenti in una collezione all'interno del suo database locale, chiamata oplog. Queste modifiche vengono applicate continuamente ai nodi secondari del replicaset
- I membri di un replicaset comunicano frequentemente tramite messaggi heartbeat
- Se il nodo primario scopre di non essere più in grado di ricevere messaggi heartbeat da più della metà dei secondari, rinuncerà allo stato di primario e verrà indetto una nuova elezione
- Per impostazione predefinita, le operazioni di scrittura si completano quando il nodo primario ha ricevuto la modifica
- Il client può lanciare una chiamata bloccante, che attenderà fino a che la scrittura non verrà ricevuta da tutti i nodi secondari o da un numero specificato di essi
- Di default, tutte le richieste di lettura vengono inviate al nodo primario
- Il client può richiedere che una richiesta di lettura venga indirizzata a un nodo secondario, se il primario non è disponibile

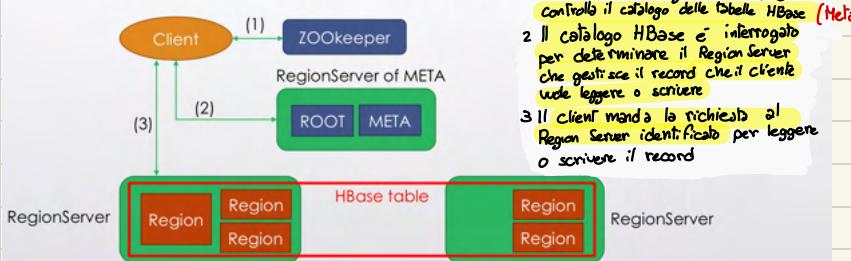
## Architettura distribuita HBase

- HBase è basato sul file system HDFS che applica già la replica
- Pertanto la replica non è un problema per il sistema HBase
- L'altro presupposto importante è che i record di valori chiave in un file HDFS non possono essere aggiornati o eliminati come un singolo record
- Cioè significa che invece di aggiornare i dati memorizzati, HBase ne genera una nuova versione dei record aggiornati e li archivia in un nuovo file HDFS
- Periodicamente questi file vengono uniti per ottenere una rappresentazione coerente della tabella HBase

### HBase Table: rappresentazione fisica

- Ogni tabella HBase è rappresentata da un numero variabile di file HDFS, detti Hfiles
- Le tabelle di dimensioni non banali vengono suddivise in più partizioni orizzontali, chiamate Regions. Ciascuna regione memorizza uno serie di record con un intervallo contiguo e ordinato di valori chiave
- Gli accessi in lettura e scrittura a una Region sono controllati da un Region Server, il quale è un processo co-locato con il DataNode di Hadoop

HBase distributed architecture



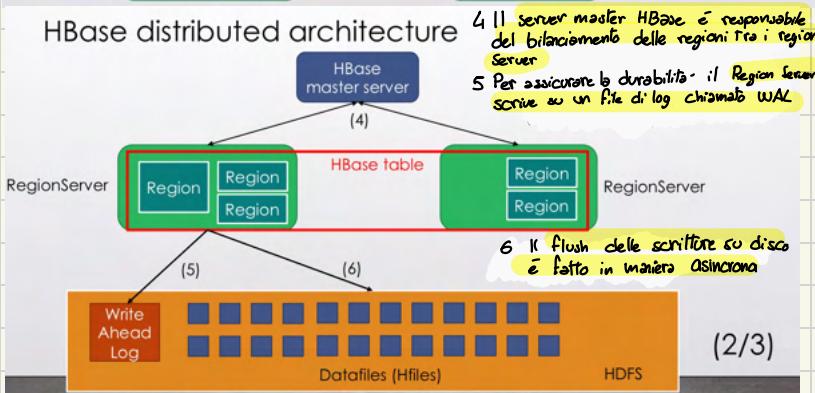
- Il client chiede allo Zookeeper la posizione del Region Server che controlla il catalogo delle tabelle HBase (Meta)
- Il catalogo HBase è interrogato per determinare il Region Server che gestisce il record che il cliente vuole leggere o scrivere
- Il client manda la richiesta al Region Server identificato per leggere o scrivere il record

A sé stante

Non è così

Succede tutto  
in maniera

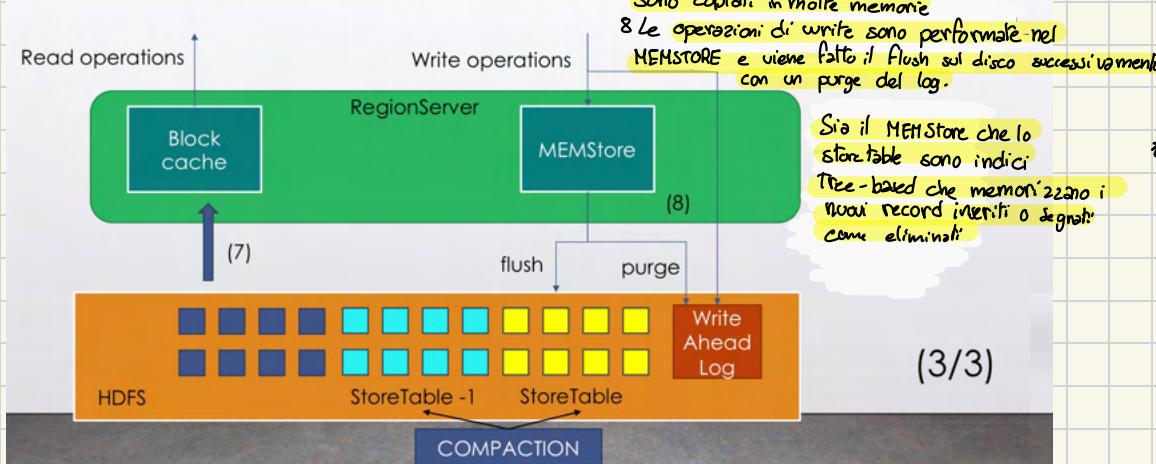
Diversi mezzi  
tutta collegato



- Il server master HBase è responsabile del bilanciamento delle regioni tra i region server
- Per assicurare la durabilità - il Region server scrive su un file di log chiamato WAL

A sé stante

HBase distributed architecture



- Usando il blocco cache molti record sono copiati in molte memorie
- Le operazioni di write sono eseguite nel MEMSTORE e viene fatto il flush sul disco successivamente con un purge del log.

Sia il MEMStore che lo storetable sono indici  
Tree-based che memorizzano i nuovi record inseriti o segnati come eliminati

A sé stante

(3/3)

**Compaction:** periodicamente il sistema compatta i store multipli in singoli file  
Durante la fase di compattamento le righe che sono frammentate su store table multiple sono consolidate e le righe cancellate vengono rimosse

## Replica delle regioni

segreto  
controllo in server  
p (che è una replica)

- Nelle versioni più recenti di HBase un guasto di un Region Server richiedeva un failover su un nuovo Region Server. I dati non vanno perso, poiché sono archiviati su HDFS, tuttavia ciò comporterebbe qualche interruzione del servizio.
- Region replica permettono alle copie ridondanti delle region di essere memorizzate su Region Server multipli. Se un region server si rompe si usano le repliche per rispondere alle richieste.
- Le scritture sulle repliche sono asincrone rispetto alle scritture sui Region server primari; perciò i dati nelle repliche non saranno sempre aggiornati.

## Architettura distribuita di Dynamo (Cassandra)

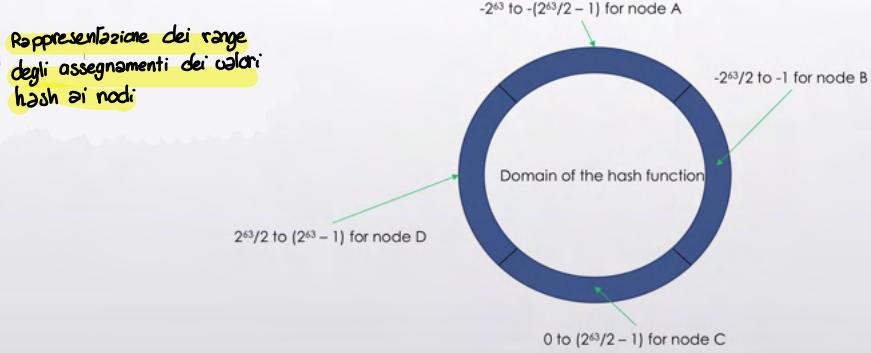
- Il nodo master ha specifiche funzioni di supervisione che includono il coordinamento degli altri nodi e lo storage dello stato corrente del database distribuito oltre che la disponibilità del nodo cluster.
- Nella famiglia dei sistemi Dynamo (come Cassandra) non ci sono nodi master specializzati. Tutti i nodi possono effettuare qualsiasi attività necessaria alle operazioni del cluster.
- I nodi in Cassandra assumono ruoli specializzati quando necessario.
- Quando avviene una richiesta dal client un nodo verrà considerato il coordinatore di quella richiesta.
- Quando un nuovo nodo è aggiunto al cluster un nodo verrà nominato come **seed node** da cui il nuovo nodo prenderà informazioni.
- Senza un nodo master Cassandra necessita di una nuova tecnica per memorizzare informazioni sullo stato del database distribuito e sulla disponibilità dei nodi.

## Gossip protocol

- Cassandra richiede che tutti i nodi rimangano aggiornati con le configurazioni del cluster e lo stato del database.
- Ogni secondo ciascun nodo trasmetterà informazioni sul proprio stato e sullo stato di ciascun altro nodo di cui è a conoscenza fino a un massimo di altre tre nodi. Detto **protocollo Gossip**.
- Lo stato del cluster è costantemente aggiornato da tutti i membri. La configurazione del cluster è archiviata nello spazio delle chiavi e in ciascun nodo ne ha una copia.
- Questa soluzione elimina ogni singolo punto di guasto, con la conseguenza che non possono essere applicate policy globali di coordinamento.
- Il protocollo gossip permette a Cassandra di implementare un sistema di ritrovazione del guasto dei nodi in maniera probabilistica.
- Inoltre, i nodi che non ricevono aggiornamenti sullo stato di un nodo diventano sempre più "preoccupati" per quel nodo e non gli inveranno alcuna operazione quando sono coordinatori.

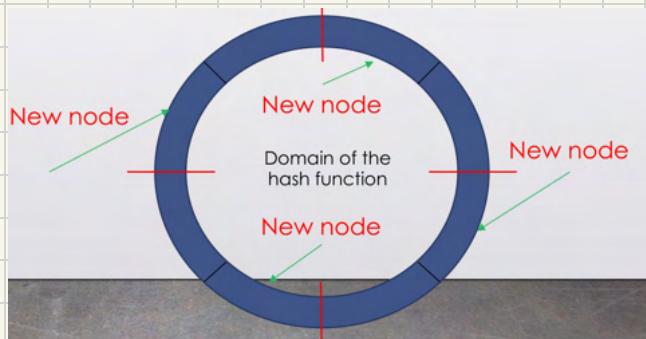
## Hashing Consistente

- I sistemi Dynamo-based distribuiscono i dati nel cluster utilizzando consistent hashing, ovvero quella chiave di riga ha un hash e l'elemento viene allocato al nodo corrispondente.
- Ogni nodo è responsabile per un dato range di valori hash. Tipicamente il range dei valori hash è da  $-2^{63}$  a  $2^{63}-1$ .



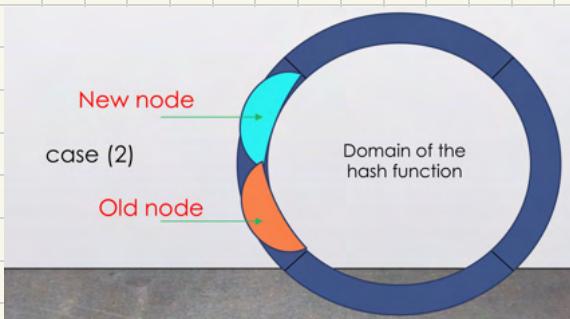
## Aggiunta di nuovi nodi

- Quando nuovi nodi vengono aggiunti al cluster rischiamo di sbilanciare lo storage dei dati dei nodi differenti.
- Un possibile approccio è di duplicare i nodi quando è necessario più spazio di memoria. Così facendo nuovi nodi possono trovare posto nell'anello tra due vecchi nodi.



Un'altra soluzione è aggiungere ogni volta, con due possibili approcci:

- Rimappare tutti i range degli hash → veramente costoso ma mantiene il bilanciamento dei nodi
- Mappare i nuovi nodi in un range esistente → molto semplice ma produce nodi sbilanciati.



## Virtual nodes

- Per evitare lo stato di sbilanciamento che può essere prodotto dalla crescita del cluster vengono introdotti i virtual nodes. Una soluzione semplice ed elegante.
- Vengono create in grande quantità, per esempio 256 nodi virtuali per nodo fisico.
- Quando viene aggiunto un nodo fisico al cluster alcuni nodi virtuali possono essere riallocati al nuovo nodo risultando in una configurazione bilanciata con overhead minimo.

## Virtual nodes



## Repliche

- Il nodo coordinatore che è responsabile per un dato hash range è responsabile che il numero N di copie della replica dei dati: richiesto sia scritto
- Di default il coordinatore scrive copie:

- Nei prossimi N-1 nodi sull'anello (*semplice strategia di replicazione*)
- In N-1 differenti nodi seguendo una strategia più complessa basata sulla *topologia della network del cluster*. In questo caso le copie sono memorizzate in altri server rack o datacenter

## Internals : modello di consistenza

- Uno dei fattori alla base della riaduzione dei database non relazionali è la necessità di abbandonare le restrizioni della stringente consistenza ACID
- I sistemi non relazionali offrono un range di livelli di consistenza, inclusa la *consistenza rigorosa*, anche se a livello di singolo oggetto.
- In assenza del modello di transazione ACID sono emersi molti approcci:

- Consistenza alla fine*
- Molti modelli di tunable consistency*

Coprire come un sistema gestisce la consistenza è essenziale per determinare come tale sistema può incontrare le necessità dell'applicazione

## Diversi tipi di consistenza:

- Consistenza con altri utenti*: se due utenti interrogano il database allo stesso momento vedono lo stesso dato

⇒ in un server singolo senza replicate ciò è sempre vero

- Consistenza a sessione singola* (o transazione): se modifichiamo una riga in una sessione e leggiamo nuovamente la stessa riga lo vedremo aggiornata

⇒ con le proprietà di atomicità e isolamento delle transazioni è sempre vero

- Consistenza con richiesta singola*: se accediamo a una tabella non vedremo inserimenti che accorrono dopo l'inizio della nostra query

⇒ con le proprietà di atomicità e isolamento delle transazioni è sempre vero

- Consistenza con la realtà*: durante la sessione lo stato del database deve sempre riflettere la situazione reale del sistema informatico

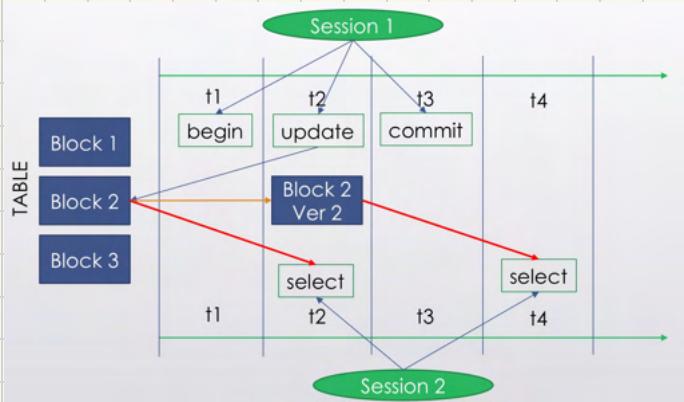
## RDBS approcci di consistenza

Il modo più semplice per implementare la consistenza ACID è con i Lock: protocollo di lock a due fasi

- Se una transazione sta leggendo un oggetto nessuna altra transazione può modificarlo
- Se una transazione sta modificando un oggetto nessuna altra transazione può leggere o modificarlo
- Se una transazione blocca un item non può fare il lock di altri oggetti

Abbiamo però bassa concorrenza

- Un'altra tecnica applicata agli RDBS è il modello multi-version concurrency control (MuCC) che evita i lock tra una scrittura e le letture di un dato pezzo di informazione.
- In MuCC copie multiple di dati sono etichettate da timestamps che permettono al sistema di avere uno snapshot del database a un dato istante. Ogni transazione di read accederà allo snapshot più recente che precede il timestamp della read
- Due write sullo stesso dato verranno gestite sempre con una strategia di lock



### Note:

- la sessione 2 può leggere la tabella anche se stava avvenendo una modifica: accederà il vecchio snapshot
- Dopo che la sessione 1 ha fatto il commit del suo aggiornamento la sessione 2 potrà vedere il nuovo stato della tabella
- La consistenza di sessione per la tabella 2 non è garantita

## Transazioni ACID negli RDBMS distribuiti

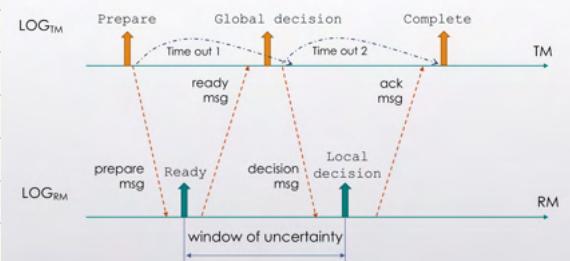
- Per garantire le proprietà ACID delle transazioni anche in ambiente distribuito è necessario introdurre un protocollo per gestire i COMMIT distribuiti.
- Viene perciò introdotto il protocollo di commit a due fasi
- Utilizzato quando le transazioni modificano dati su shard differenti del database distribuito

## 2PC

Nel protocollo c'è un nodo detto coordinatore, o transaction manager (TM) e gli altri nodi coinvolti detti resource manager (RM)

Il protocollo definisce nuovi record che devono essere scritti nel file di log

- Prepare record: scritto dal TM quando il protocollo inizia e contiene l'ID dei nodi coinvolti
- Global commit (o abort) record: scritto dal TM quando decide il risultato della transazione
- Complete record: è scritto dal TM quando il protocollo è terminato e le lock vengono rilasciate
- Ready record è scritto dal RM quando è pronto a partecipare al protocollo con un commit finale. Contiene l'ID del TM



- Se qualche messaggio preparato va perso dopo il time out la decisione globale sarà Global rollback
- Se qualche messaggio di ack va perso, dopo il timeout, il messaggio di decisione sarà quello di rimandare l'ack al nodo che lo sta aspettando

## Altri livelli di consistenza

### First read in consistency:

- I sistemi non relazionali limitano lo scope della consistenza a una singola operazione o oggetto
- Nell'RDMIS possiamo mantenere la consistenza tra statement multipli

### Livelli di consistenza a operazioni singole

**Strict Consistency:** una read ritornerà sempre il database più recente

**Eventual consistency:** il sistema può essere inconsistente in qualsiasi istante di tempo ma tutte le operazioni individuali verranno eventualmente applicate in maniera consistente alla fine. Se tutti gli aggiornamenti si fermano il sistema raggiungerà uno stato consistente

**Read your own writes:** è un eventual consistency dove si garantisce almeno di vedere qualsiasi operazione effettuata

**Weak consistency:** il sistema non dà garanzie che diventerà consistente. Per esempio se qualche nodo si guarda un aggiornamento potrebbe andare perso

## Mongo DB consistency

- In un architettura single server, Mongo DB fornisce una consistenza single document rigorosa.
- Ottiene ciò quando i lock, non implementa alcuna versione di MVCC
- Quando un documento verrà modificato, sarà bloccato sia per le letture che per le scritture di qualsiasi altra sessione
- Le granularità dei lock di Mongo DB sono cambiati durante la sua storia. All'inizio era l'intero sistema poi lo scope dei lock divenne il database, poi la collezione ed infine il document level

## Mongo DB eventual consistency

- In uno scenario multiserver, quando ci sono i replicaset, possiamo giungere ancora a una consistenza rigorosa fino a quando tutte le letture sono indirizzate al server primario
- Comunque possiamo configurare le preferenze di lettura in modo che permettano le letture dal server secondario introducendo così una forma di eventual consistency

## HBase consistency

- HBase prevede consistenza rigorosa per ogni riga (record) di una HBase table
- Durante un aggiornamento l'intera riga verrà bloccata dal Region server per prevenire un conflitto di aggiornamenti sulle altre colonne
- Le letture non sono bloccate dalle operazioni di scrittura
- Una forma di MVCC è applicata:
  - Quando letture e scritture accadono concorrentemente le operazioni di lettura leggeranno una versione del dato precedente della riga

## HBase MUCC

- HBase usa una variante dello schema System Change Number (SCN)
- All'inizio di un'operazione di write incrementa il write number (WN) che verrà memorizzato nella cella della riga aggiornata
- Quando parte un'operazione di read un read point number (RPN) viene assegnato al più alto write point completo
- L'operazione di lettura permette di accedere solo alle celle con WN minore uguale a RPN

## Eventual consistency delle regions replica

- Di default tutte le letture sono dirette al Region Server primario (RS), il cui risultato è un comportamento rigorosamente consistente.
- Tuttavia se la consistenza per le reads è impostata su coerenza temporale, allora una richiesta di lettura viene mandata prima al primary RS, seguita da una richiesta aggiuntiva al RS secondario. Il primo che darà risultato completa la richiesta.
- Il nome timeline consistency viene dal fatto che l'RS secondario riceve gli aggiornamenti nella stessa sequenza del primario.
- Eventual consistency è il risultato finale di questo approccio, quando l'RS primario non risponde, la richiesta di lettura può essere gestita dall'RS secondario senza valore aggiornato.

## Cassandra consistency

- La tunable consistency è un approccio comune tra i sistemi della famiglia DynamoDB
- Anche Cassandra adotta un approccio simile e permette di impostare tre parametri:
  - Il fattore di replicazione: numero di copie dei dati che verranno mantenuti tra nodi multipli. Vengono memorizzati nel Keyspace (il catalogo condiviso di metadati che memorizza lo stato dei nodi)
  - Il write consistency level: definisce cosa deve accadere prima che il sistema possa completare una write request
  - Il read consistency level: definisce cosa deve accadere prima che il sistema possa completare una read request

### Write consistency level:

- ALL: la write deve propagare tutti i nodi
- ONE/TWO/THREE: la write deve essere propagata al numero di nodi specificato
- QUORUM: la write deve essere completata su un Quorum di replica nodes
- EACH\_Quorum: la write deve essere completata in un quorum di replica node in ciascun datacenter
- LOCAL\_QUORUM: la write deve essere completata a un quorum di replica nodes solo nel corrente data center
- ANY: la write avrà successo quando verrà scritta in qualsiasi node

### Read consistency level:

- ALL: tutte le repliche vengono lette
- One/two/three: la read request deve essere propagata al specifico numero di nodi
- QUORUM: La read sarà completata quando un quorum di nodi replica avrà risposto
- EACH\_QUORUM: La read sarà completata quando un quorum di nodi replica avrà risposto, in tutto il sistema
- LOCAL\_QUORUM: La read sarà completata quando un quorum di nodi replica avrà risposto, in questo datacenter
- LOCAL\_ONE: La read request verrà mandata al nodo più vicino nel datacenter attuale

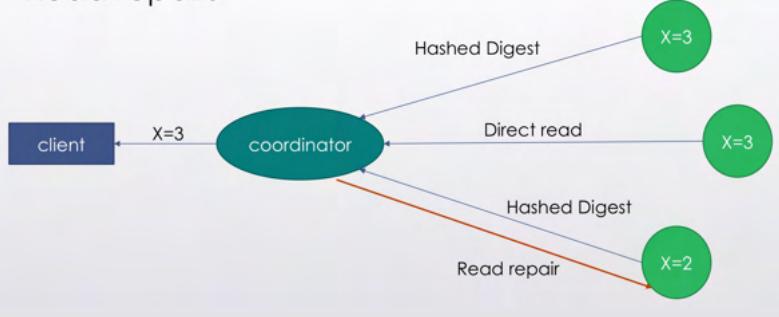
## Combinazione di livelli di consistenza read and write

READ \ WRITE	ONE	QUORUM	ALL
ONE	High performance and availability, but no consistency		Fast and highly available for writes; consistent but slow reads
QUORUM		Intermediate performance; good availability; strong consistency	
ALL	Slow and not highly available for write; fast and consistent reads		Strictest consistency, but lowest availability and performance

## Hinted handoff e read repair

- A meno che il write consistency level non sia ALL, avremo dell'inconsistenza nel Cassandra cluster.
- Hinted handoff permette a un nodo N di memorizzare un aggiornamento per un altro nodo Nb se Nb è momentaneamente fuoriuso. Se Nb riappaia disponibile in breve tempo (3 ore) la write verrà trasmessa, altrimenti andrà persa e avremo uno stato inconsistente.
- Read repair è un meccanismo che Cassandra usa per riparare le inconsistenze prodotte da un fallimento del procedure hinted handoff. Durante una read da più repliche, solo a un nodo viene emessa una lettura diretta, mentre agli altri nodi vengono inviate richieste per i digest hash dei dati. Se i risultati non sono coerenti vengono restituiti i dati più recenti e ai nodi non aggiornati viene inviata un istruzione di riparazione in lettura.

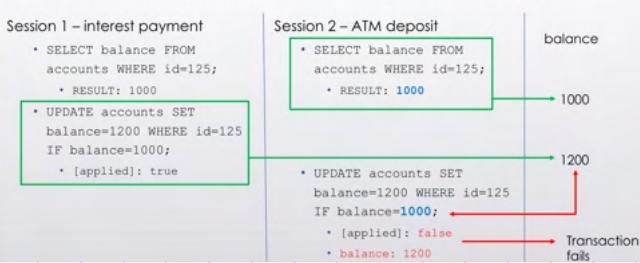
## Read repairs



## Transazioni leggere

- Cassandra è un sistema senza lock che usa la risoluzione dei conflitti: piuttosto che fare il locking per permettere alta disponibilità e performance.
- A volte le operazioni devono combinare atomicamente una lettura e una scrittura. Cassandra non supporta le transazioni tradizionali, ma introduce le transazioni leggere che si applicano a una sola operazione e supporta solo un compare and set pattern.
- Compare and set è un'operazione atomica che controlla un valore (read) e se il valore soddisfa un predicato, viene impostato un altro valore (write).

### Example: using the Cassandra Query Language



## L'esecuzione di una transazione leggera con replica - Protocollo Paradosso

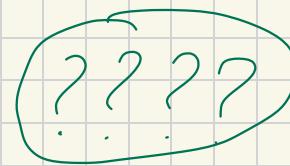
1<sup>a</sup> fase: il leader invia PREPARE → le repliche mandano promise

- La maggioranza di promise producono una decisione di continuazione, mentre in 2PC tutti gli RM devono accordarsi sul commit

2<sup>a</sup> fase: il leader chiede di leggere il valore corrente in ogni replica → le repliche ritornano il valore per fare il check delle IF clauses

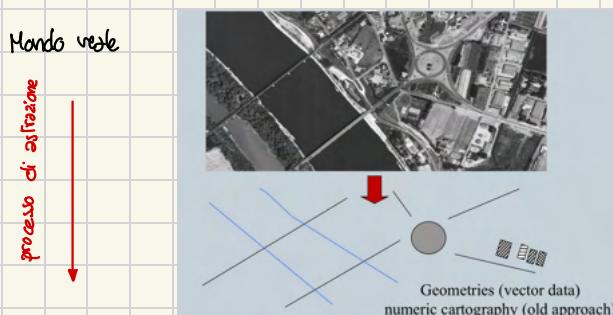
3<sup>a</sup> fase: il leader propone il nuovo valore → le repliche rispondono se sono in grado di applicare il nuovo valore

4<sup>a</sup> fase: il leader invia il commit → le repliche rispondono con l'ACK



## Spatial databases

### Geographical data: processo di astrazione



### Da raster a dati vettoriali



### Dati spaziali in formato digitale

• Un insieme di dati spaziali che descrivono una porzione della superficie terrestre può essere rappresentato in uno dei seguenti formati

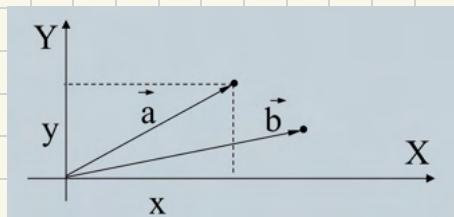
• **Formato raster:** è una griglia di celle dove i dati sono associati ad ogni cella della griglia. In questa categoria troviamo le fotografie aeree o le immagini satellitari e le rappresentazioni di misure di alcuni parametri Fisici effettuate in modo estensivo su una determinata area (modello "sul campo")

• **Formato vettoriale:** In questo caso la proprietà spaziale è rappresentata mediante un insieme di punti, curve e superfici utilizzando un insieme di coordinate e alcune equazioni. In questa categoria troviamo dati provenienti dalla cartografia digitale

Rappresentazione vettoriale basata sulla geometria lineare dello spazio euclideo a 2 dimensioni

Reference space: Piano Euclideo ( $\mathbb{R}^2$ )

Punto  $P = (x, y)$



$\vec{a}$  è un vettore che identifica il punto  $P$  in  $\mathbb{R}^2$ .

La distanza tra due punti  $P$  e  $Q$  identificati dai vettori  $a = (x_a, y_a)$  e  $b = (x_b, y_b)$

$$PQ = \|\vec{a} - \vec{b}\| = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$$

Operations on vectors:

- $\vec{a} + \vec{b} = \vec{c}$   $\vec{c} = (x_a + x_b, y_a + y_b)$
- $\vec{a} - \vec{b} = \vec{d}$   $\vec{d} = (x_a - x_b, y_a - y_b)$
- $\lambda \vec{a} = \vec{e}$   $\vec{e} = (\lambda x_a, \lambda y_a)$

Il segmento tra due punti  $P$  e  $Q$  identificati dai vettori  $a$  e  $b$ :

$$\vec{s} = \{\lambda \vec{a} + (1-\lambda) \vec{b} \mid \lambda \in [0,1]\}$$

Abbiamo poi i segment endpoint:

$$s.E_0 = P(\lambda=1) \text{ e } s.E_1 = Q(\lambda=0)$$

PolyLine: è una lista di  $n$  segmenti distinti

$$sp = (s_0, \dots, s_{n-1})$$

tale che tutti gli endpoint (si.  $E_0$ , si.  $E_1$ ) di ogni segmento si è concluso da almeno due segmenti, ad eccezione di due endpoint chiamati polyline endpoint



Proprietà della Polyline

Polyline Semplice

Dato la polyline  $sp = (s_0, \dots, s_{n-1})$ , se vale la condizione:

$$(\forall s_i, s_j \in sp)(i \neq j \Rightarrow (s_i \cap s_j = \emptyset \vee ((j = (i+1)_{mod n}) \wedge (s_i.E_0 = s_j.E_0 \vee s_i.E_0 = s_j.E_1 \vee s_i.E_1 = s_j.E_0 \vee s_i.E_1 = s_j.E_1)))$$

ovvero la polyline non si interseca con se stessa allora ho una polyline semplice

Abbiamo un **ciclo** quando data una polyline  $sp = (s_0, \dots, s_{n-1})$  vale la seguente condizione

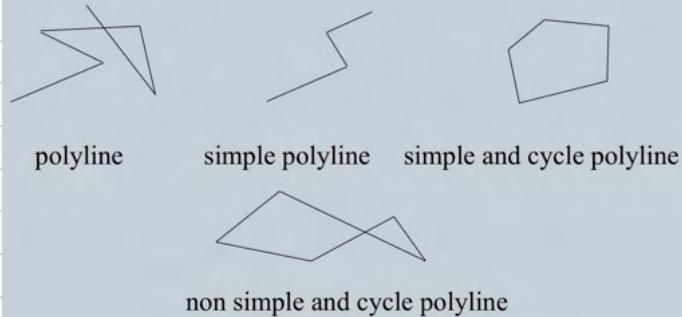
$$s_0.E_0 = s_{n-1}.E_1$$

Allora abbiamo una polyline chiusa (o ciclo)

### Orientazione

Ciascuna polyline  $sp = (s_0, \dots, s_{n-1})$  è orientata se è sempre definita come primo endpoint  $sp.E_0 = s_0.E_0$  e come secondo endpoint  $sp.E_1 = s_{n-1}.E_1$

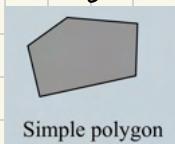
### Examples of polyline



### Poliгоно

Poliгоно semplice: dato un ciclo e una polyline semplice  $sp$ , è la porzione finita di spazio  $\mathbb{R}^2$  delimitata da  $sp$  (teorema delle curve di Jordan)

$sp$  è parte del poliгоно, il poliгоно è un insieme chiuso e连通的 di punti. Chiuso indica che il poliгоно contiene il suo bordo



Simple polygon

Poliгоно con buco: dato un ciclo e una polyline semplice  $sp$  e un insieme di cerchi e polylines semplici  $H = \{sp_1, \dots, sp_n\}$  è la porzione di spazio finita di  $\mathbb{R}^2$  che è delimitata da  $sp$  meno la porzione di  $\mathbb{R}^2$  delimitata da ogni  $sp_i \in H$

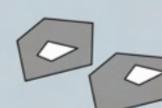
$sp$  e tutti i punti delle polylines di  $H$  sono contenuti nel poliгоно, ovvero il poliгоно è un insieme chiuso di punti.

### Vincoli:

Ogni poliгоно soddisfa queste condizioni:

- Ciascuna polyline di  $H$  è contenuta dal poliгоно semplice delimitato da  $sp$ .
- Ciascuna polyline di  $H$  non interseca  $sp$ , è permessa solo l'intersezione in un singolo punto senza crossing
- I buchi non si intersecano  $\bullet \forall (sp_i, sp_j \in H) (i \neq j) \Rightarrow (\text{((}sp_i\text{ does not intersect }sp_j\text{)} \vee (sp_i \text{ intersects }sp_j \text{ only in one point with no crossing})) \wedge (sp_i \text{ is not contained in the polygon delimited by }sp_j))$

### Examples of polygon with holes



correct geometries



incorrect geometries

## Relazioni spaziali

- Per descrivere le proprietà spaziali di un oggetto, si possono usare le geometrie vettoriali per descrivere la forma, l'estensione e la posizione degli oggetti sulla superficie terrestre
- Un altro approccio può descrivere ciò descrivendo le relazioni spaziali che esistono tra un insieme di oggetti inseriti nello spazio di riferimento

Per esempio dire che Verona è adiacente

- a Vicenza descrive in maniera qualitativa una proprietà spaziale delle province di Verona e Vicenza



## Relazioni spaziali

### Relazioni binarie

- Una relazione binaria definisce una proprietà che collega due elementi di un insieme
- Nel nostro contesto le proprietà sono geometriche e le proprietà riguardano la loro posizione nello spazio

Abbiamo 3 categorie di relazioni binarie spaziali:

- Relazioni topologiche
- Relazioni basate sulla direzione
- Relazioni basate sulla distanza

### Relazioni topologiche

Definizione: una relazione  $R$  è topologica se è invariante rispetto a trasformazioni topologiche dello spazio

Hanno le seguenti proprietà:

- Non dipendono dalla distanza tra le geometrie e dalle loro estensioni
- Sono qualitative: descrivono come due geometrie sono in relazione senza specificare alcuna misura.
- Si riferiscono a concetti ad alto livello e frequentemente hanno una parola corrispondente nel linguaggio naturale

Differenti insiemi di relazioni topologiche possono essere definite:

- Molti modelli formali formali sono stati proposti cercando di identificare quello con le relazioni più utili

### Il modello di Max Egenhofer

Questo modello si applica agli insiemi di punti chiusi e regolari e si basa sulla divisione dello spazio prodotta da un valore geometrico  $\lambda$

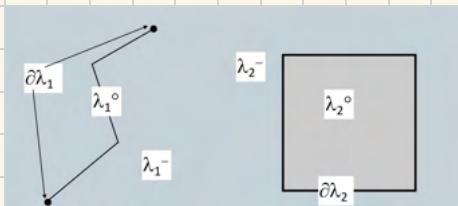
Questa suddivisione identifica:

- L'interno di  $\lambda$  (indicato da  $\lambda^+$ )
- Il bordo di  $\lambda$  (indicato da  $\partial\lambda$ )
- L'esterno di  $\lambda$  (indicato da  $\lambda^-$ )

Il modello può essere applicato a diversi tipi geometrici perché sia possibile definire il confine delle geometrie appartenenti al tipo

Applichiamo il modello alle seguenti geometrie in 2D e 3D

- Punto: hanno un confine vuoto
- Curva: hanno un confine combinatorio
- Superficie: ha un confine combinatorio



In questo modello l'insieme di tutte le relazioni topologiche che possono essere definite tra due geometrie A e B sono specificate dal risultato dell'intersezione di tutte le parti generate nella suddivisione dello spazio introdotte da  $A^\circ, \partial A, A^-$  e quelle introdotte da  $B^\circ, \partial B$  e  $B^-$ . Formalmente una relazione è riassunta dalla matrice

$$R(A,B) = \begin{pmatrix} A^\circ \cap B^\circ & A^\circ \cap \partial B & A^\circ \cap B^- \\ \partial A \cap B^\circ & \partial A \cap \partial B & \partial A \cap B^- \\ A^- \cap B^\circ & A^- \cap \partial B & A^- \cap B^- \end{pmatrix}$$

Il numero totale delle relazioni è 512, ma non tutte le combinazioni corrispondono a casi reali:

Il modello può essere applicato a:

- Due curve nel piano cartesiano
- Due poligoni nel piano cartesiano
- Una curva e un poligono nel piano cartesiano
- Una curva e un punto nel piano cartesiano
- Un poligono ed un punto nel piano cartesiano

Considerando il caso di due poligoni nel piano, è sufficiente considerare solo 4 intersezioni per ottenere tutti i casi rilevanti.

$$\begin{matrix} A^\circ \cap B^\circ & A^\circ \cap \partial B \\ \partial A \cap B^\circ & \partial A \cap \partial B \end{matrix}$$

## Relazioni topologiche (Clementini)

DATE due geometrie  $\lambda_1$  e  $\lambda_2$ , le seguenti relazioni topologiche possono essere definite

### DISJOINT

$\lambda_1$  DISJOINT  $\lambda_2 \Leftrightarrow (\lambda_1 \cap \lambda_2 = \emptyset)$

### TOUCH

$\lambda_1$  TOUCH  $\lambda_2 \Leftrightarrow (\lambda_1^\circ \cap \lambda_2^\circ = \emptyset) \wedge (\lambda_1 \cap \lambda_2 \neq \emptyset)$

Osservazione: Ogni relazione proposta da Clementini corrisponde ad un insieme di configurazioni della matrice di Egenhofer

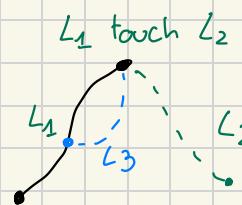
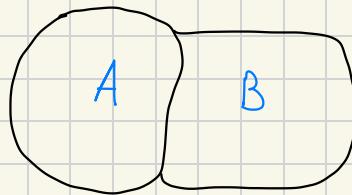
- DJ:  $a.DJ(b) \triangleq a.PS() \cap b.PS() = \emptyset$

$$\begin{aligned} R_{dj}(pt, pt) &= [FFT\ FFF\ TFT] \\ R_{dj}(pt, c/s) &= [FFT\ FFF\ T * T] \\ R_{dj}(c/s, pt) &= R_{dj}(pt, c/s)^T \\ R_{dj}(c/s, c/s) &= [FFT\ FF * T * T] \end{aligned}$$

T = true  
F = false  
\* = don't care

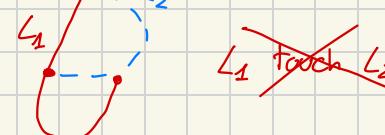
Esempi

A touch B

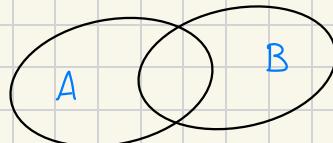


$L_3$  touch  $L_1$

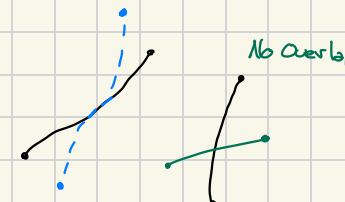
Il boundary di una linea sono i due estremi, perché considero : la boundary combinatoria



Due relazioni suddivisi per il tocco delle parti interne



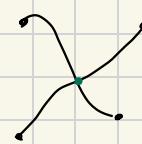
richiede dimensionalità, vale solo per linee e poligoni



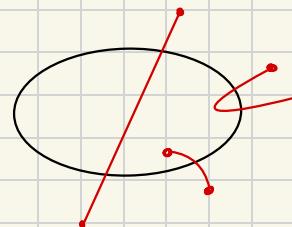
Cross

Cross

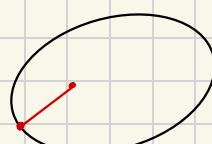
Cross tra due poligoni non esiste



Cross



In



### OVERLAP

$\lambda_1$  OVERLAP  $\lambda_2 \Leftrightarrow \dim(\lambda_1^\circ \cap \lambda_2^\circ) = \dim(\lambda_1) = \dim(\lambda_2) \wedge (\lambda_1 \cap \lambda_2 \neq \lambda_1) \wedge (\lambda_1 \cap \lambda_2 \neq \lambda_2)$

### CROSS

$\lambda_1$  CROSS  $\lambda_2 \Leftrightarrow \dim(\lambda_1^\circ \cap \lambda_2^\circ) \leq (\max(\dim(\lambda_1), \dim(\lambda_2)) - 1) \wedge (\lambda_1 \cap \lambda_2 \neq \lambda_1) \wedge (\lambda_1 \cap \lambda_2 \neq \lambda_2)$

### IN

$\lambda_1$  IN  $\lambda_2 \Leftrightarrow (\lambda_1^\circ \cap \lambda_2^\circ \neq \emptyset) \wedge (\lambda_1 \cap \lambda_2 = \lambda_1) \wedge (\lambda_1 \cap \lambda_2 \neq \lambda_2)$

### CONTAINS

$\lambda_1$  CONTAINS  $\lambda_2 \Leftrightarrow \lambda_2$  IN  $\lambda_1$

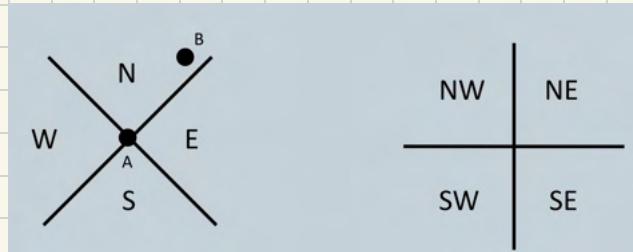
### EQUAL

$\lambda_1$  EQUAL  $\lambda_2 \Leftrightarrow (\lambda_1^\circ \cap \lambda_2^\circ \neq \emptyset) \wedge (\lambda_1 \cap \lambda_2 = \lambda_1) \wedge (\lambda_1 \cap \lambda_2 = \lambda_2)$

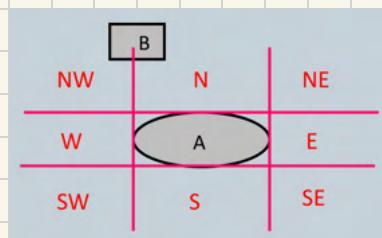
where the function dim returns the result dimension: o point, 1 curve, 2 surface or empty if the intersection is empty.

## Relazioni direction based

Le relazioni direction based sono definite sulla base della direzione rispetto a un sistema di riferimento che divide lo spazio in almeno 4 parti delle tiles. L'origine del sistema di riferimento si trova in uno delle geometrie considerate. Intersecando le tiles con l'altra geometria si possono derivare le relazioni tra le due. Considerando i punti come valori geometrici, le tessere definite dal sistema di riferimento possono essere le seguenti:



Considerando altri tipi di geometrie il cui valore ha un'estensione (come superficie e curve), il sistema di riferimento deve tener conto di tale estensione e così può essere definito come il seguente:



Dopo aver definito le piastrelle, si specifica la relazione direzionale tra A e B considerando le relazioni topologiche che esistono tra le piastrelle definite da A e la geometria B

In questo caso, se si prende A come geometria di riferimento e si utilizzano le relazioni topologiche definite dalla matrice di Egenhofer per descrivere la relazione tra le tessere di A e B, si ottengono un totale di 169 relazioni.

Queste relazioni richiedono la definizione di una funzione distanza in relazione allo spazio. Se adottiamo la distanza Euclidea allora la distanza tra due punti:  $P = (x_1, y_1)$  e  $Q = (x_2, y_2)$  sono definite come segue:

$$D(P, Q) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

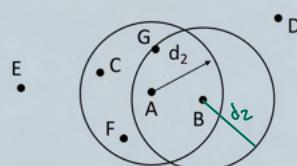
Se consideriamo geometrie con estensioni rilevanti è necessario ridefinire il concetto di distanza così:

$$\text{Dist}(A, B) = \min(\{D(P, Q) : P \in A, Q \in B\})$$

Data una coppia di distanze  $(d_1, d_2)$  possiamo definire una relazione basata sulle distanze tra due geometrie A e B come segue

$$R_{(d_1, d_2)}(A, B) \Leftrightarrow (d_1 < \text{Dist}(A, B) < d_2)$$

For example, considering the relation  $R_{(0, d_2)}$ :



A is in relation with the points: B, C, G, F.

B is in relation with the points: A and G.

## Spatial database

Modello del sistema che gestisce dati spaziali

Esistono vari approcci per rappresentare i dati spaziali:

- DBMS relazionale: estesi con specifiche rappresentazioni di geometrie in formato vettoriale (PostgreSQL+, PostGIS, Oracle 11g)

- Insiemi di file specifici: file di tipo shape + file DBF (ESRI)

- Dati semistrutturati: XML file nel formato GML

→ dati di natura  
spaziale e geografica  
↓  
oasi wa d' più GeoJSON

Database relazionali e dati spaziali

Il modello relazionale ancora oggi è un modello di riferimento per i sistemi che gestiscono database con informazioni alfaniumeriche

Le componenti geometriche degli oggetti geografici non possono essere rappresentati direttamente nel modello relazionale usando un insieme di attributi numerici contenenti le coordinate delle geometrie poiché:

- 1 L'utente non può interrogare con un interfaccia dove le geometrie sono rappresentate dalle loro coordinate, senza alcun supporto grafico
- 2 Le tecniche tradizionali che supportano l'accesso ai dati (indici) non sono efficaci per le query spaziali
- 3 Non c'è supporto esplicito per la specifica di query spaziali

L'RDBMS object oriented non può essere adattato per dati spaziali per le seguenti ragioni:

- Le performance di questi sistemi non sono soddisfacenti per avere a che fare con questi dati e non è fornito supporto esplicito per le query spaziali
- È difficile integrare questo sistema con i DBMS relazionali

Soluzione attualmente utilizzata

- DBMS specifici per dati spaziali (Geo-DBMS) o sistemi tradizionali DBMS con estensioni specifiche per avere a che fare con dati spaziali
- I sistemi adottano un modello georrelazionale

Definizione di GeoRDBMS

- Un sistema per la gestione dei dati spaziali è in primis un sistema per gestire i database (DBMS)
- È necessario fornire tipi predefiniti (tipi di dati spaziali SDT) per la rappresentazione di dati spaziali e per gestire i dati spaziali nello specifica della query
- È necessario fornire un implementazione degli SDT che includono l'accesso ai dati tramite una rete spaziale indice includendo anche un implementazione efficiente per l'operazione di join spaziale

## Geo DBMS data model

Gli attuali GeoDBMS adottano un data model geo-relazionale

- Questo modello è un'estensione del modello relazionale che include nuovi tipi e operazioni per rappresentare e manipolare i dati spaziali

Quali sono le estensioni del modello relazionale necessarie per permettergli di gestire dati spaziali?

1. Introduzione di tipi specifici per la rappresentazione di dati spaziali
2. Introduzione di estensioni al linguaggio di interrogazione per esprimere condizioni di selezione basate sulle proprietà spaziali

## Data model attuali

- Data model di Oracle 11g
- Data model di ESRI Shapefile
- Data model di ESRI Geodatabase
- Database di vecchi sistemi (MGE Intergraph, MapInfo, Informix)

## Feature Tables

→ caratteristiche

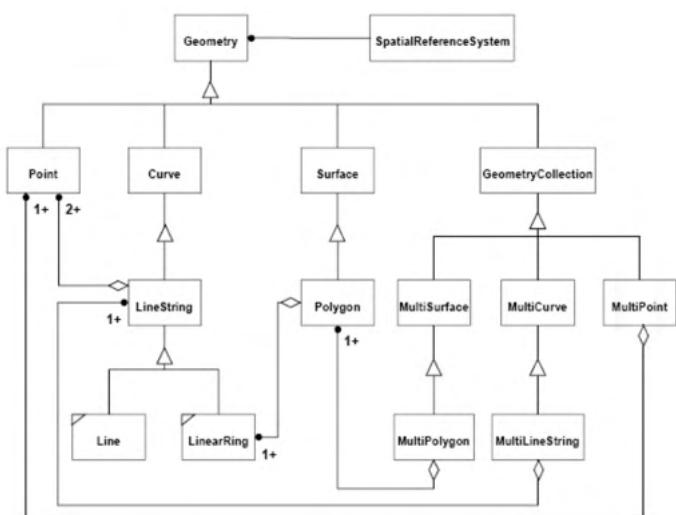
Le raccolte di feature geospatiali semplici verranno concettualmente archiviate come tabelle con colonne di voci geometriche in un DBMS relazionale (RDBMS), ciascuna feature verrà archiviata come riga in una tabella.

Gli attributi non spaziali delle feature verranno mappati in colonne cui tipi sono estratti dall'insieme di SQL-92 standard. Gli attributi spaziali delle feature verranno mappati sulle colonne i cui tipi di dati SQL si basano su tipi di dati geometrici aggiuntivi per SQL. Una tabella le cui righe rappresentano le feature verrà chiamata **feature table**

## Feature Tables

Le implementazioni delle feature table sono descritte con due environment: SQL 92 e SQL 92 con geometry types, che è l'environment SQL esteso con i geometry types

## Geometry Model



## Geometry Class

Geometry è la classe principale della gerarchia ed è una classe astratta (non istanziabile)

Le sottoclassi istanziabili si restringono a oggetti a 0, 1 e 2 dimensioni che esistono nello spazio bidimensionale  $R^2$ . Tutte le istanze geometriche valide sono definite topologicamente come un insieme chiuso di punti (ovvero le geometrie definite includono il loro bordo).

Metodi base sulle geometrie

- **GeometryType ()**:String — Returns the name of the instantiable subtype of Geometry of which *this* Geometry instance is a member.
- **Dimension ()**:Integer — The inherent dimension of *this* Geometry object, which must be less than or equal to the coordinate dimension.
- **SRID ()**:Integer — Returns the Spatial Reference System ID for *this* Geometry.
- **Envelope()**:Geometry — The minimum bounding box for *this* Geometry.
- **AsText()**:String — Exports *this* Geometry to a specific well-known text representation of Geometry.
- **IsEmpty()**:Integer — Returns 1 (TRUE) if *this* Geometry is the empty geometry. If true, then *this* Geometry represents the empty point set,  $\emptyset$ , for the coordinate space.
- **IsSimple()**:Integer — Returns 1 (TRUE) if *this* Geometry has no anomalous geometric points, such as self intersection or self tangency.
- **Boundary()**:Geometry — Returns the closure of the combinatorial boundary of *this* Geometry.
- **Distance(anotherGeometry:Geometry)**:Double — Returns the shortest distance between any two points in the two geometries as calculated in the spatial reference system of *this* Geometry.
- **Buffer(distance:Double)**:Geometry — Returns a geometry that represents all points whose distance from *this* Geometry is less than or equal to distance. Calculations are in the Spatial Reference System of *this* Geometry.
- **ConvexHull()**:Geometry — Returns a geometry that represents the convex hull of *this* Geometry.
- **Intersection(anotherGeometry:Geometry)**:Geometry — Returns a geometry that represents the point set intersection of *this* Geometry with anotherGeometry.
- **Union(anotherGeometry:Geometry)**:Geometry — Returns a geometry that represents the point set union of *this* Geometry with anotherGeometry.
- **Difference(anotherGeometry:Geometry)**:Geometry — Returns a geometry that represents the point set difference of *this* Geometry with anotherGeometry.
- **SymDifference(anotherGeometry:Geometry)**:Geometry — Returns a geometry that represents the point set symmetric difference of *this* Geometry with anotherGeometry.

Un database Georrelazionale è composto da:

- La tabella del sistema di riferimento spaziale nominata `spatial_ref_sys` memorizza informazioni sui sistemi di riferimento spaziali usati nel database
- La tabella delle colonne che contengono i metadati geometrici, fornisce i metadati sulle reference spaziali delle colonne geometriche nel database
- I tipi geometrici di SQL estendono l'insieme dei tipi disponibili con SQL 92 includendo i tipi geometrici
- Un insieme di Feature Table, ogni Feature Table è una tabella contenente uno o più attributi geometrici modellati usando una colonna il cui tipo SQL corrisponde a un tipo geometrico SQL. Le relazioni tra Feature sono definite con riferimenti a **Foreign Key**
- Un insieme di tavole tradizionali

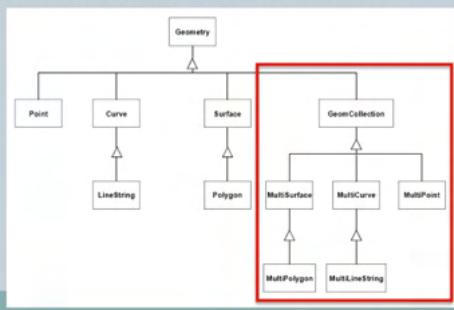
## Componenti di un database georelazionale

### Geometric Columns Metadata table or view

```
CREATE TABLE GEOMETRY_COLUMNS (
    F_TABLE_CATALOG VARCHAR(256) NOT NULL,
    F_TABLE_SCHEMA VARCHAR(256) NOT NULL,
    F_TABLE_NAME VARCHAR(256) NOT NULL,
    F_GEOMETRY_COLUMN VARCHAR(256) NOT NULL,
    COORD_DIMENSION INTEGER,
    SRID INTEGER REFERENCES SPATIAL_REF_SYS,
    CONSTRAINT GC_PK PRIMARY KEY (F_TABLE_CATALOG,
        F_TABLE_SCHEMA,
        F_TABLE_NAME, F_GEOMETRY_COLUMN)
)
```

### SQL Geometry Types

Geometry, Point, Curve, LineString, Surface, Polygon, GeomCollection, MultiPoint, MultiCurve, MultiLineString, MultiSurface, MultiPolygon



### Collezione geometrica

Una collezione geometrica è una geometria che è una collezione di 1 o più geometrie.

Tutti gli elementi in una collezione geometrica devono essere nello stesso sistema di riferimento spaziale, che è anche il sistema di riferimento della collezione geometrica.

Metodi:

- **NumGeometries()**:Integer—Returns the number of geometries in this GeometryCollection.
- **GeometryN(N:integer)**:Geometry—Returns the Nth geometry in this GeometryCollection.

### Multipunto

• Una multipoint è una collezione geometrica zero dimensionale. Gli elementi della multipoint si restringono ai punti.

• I punti non sono né connessi né ordinati.

• Una multipoint è semplice se non ci sono al suo interno due punti equivalenti (con le stesse coordinate).

• Il confine di una multipoint è l'insieme vuoto.

### Multicurva

• Una multicurve è una collezione geometrica a una dimensione i cui elementi sono curve.

• La multicurve è una classe non istanziabile.

• Una multicurve è semplice se tutti i suoi elementi sono semplici e l'unica intersezione tra due elementi concessa è quella tra punti sul bordo di ciascuno dei due elementi.

• Una multicurve è chiusa se tutti i suoi elementi sono chiusi.

• Una multicurve è definita come topologicamente chiusa.

Il bordo di una multicurve è ottenuto applicando la regola di unione 'mod 2'. Un punto nel confine di una multicurve se è nel confine di un numero dispari di elementi di una multicurve.

- **IsClosed()**:Integer—Returns 1 (TRUE) if this MultiCurve is closed (StartPoint () = EndPoint () for each curve in this MultiCurve).

- **Length()**:Double—The Length of this MultiCurve which is equal to the sum of the lengths of the element Curves.

## Esempio di multicurva

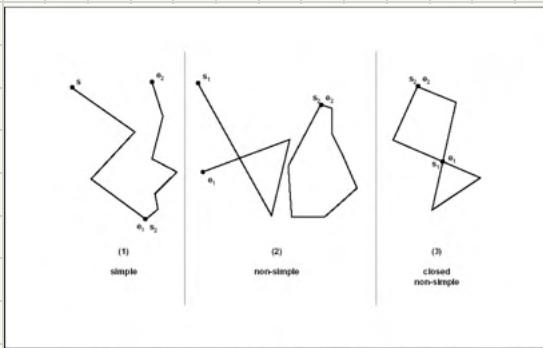


Figure 2.3—(1) a simple MultiLineString, (2) a non-simple MultiLineString with 2 elements, (3) a non-simple, closed MultiLineString with 2 elements

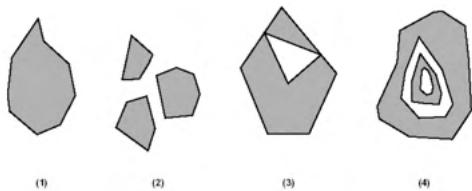
## Una multisuperficie

- È una collezione geometrica bidimensionale i cui elementi sono superfici.
  - Gli interni di qualsiasi due superfici in una multisuperficie non possono intersecarsi.
  - I bordi di due qualsiasi elementi in una multisuperficie possono intersecarsi al più in un numero finito di punti.
  - Le multisuperficie non sono classi istanziabili con questa specifica.
  - Le sottoclassi istanziabili delle multisuperficie sono i multipoli, che corrispondono a una collezione di Poligoni.
- **Area()**:Double—The area of *this* MultiSurface, as measured in the spatial reference system of *this* MultiSurface.
  - **Centroid()**:Point—The mathematical centroid for *this* MultiSurface. The result is not guaranteed to be on *this* MultiSurface.
  - **PointOnSurface()**:Point—A Point guaranteed to be on *this* MultiSurface.

## Multipoli:

- Una multipoli è una multisuperficie i cui elementi sono poligoni.
- Il confine di un multipoli è un insieme di curve chiuse che corrispondono ai bordi dei suoi poligoni.
- Ogni curva nel confine del multipoli è nel confine di esattamente un elemento poligono, e ogni curva nel confine di un elemento poligono è nel confine del multipoli.

## Esempio di multipoli



## Rappresentazione testuale in SQL della geometria

Geometry Type	SQL Text Literal Representation	Comment
Point	'POINT (10 10)'	a Point
LineString	'LINESTRING (10 10, 20 20, 30 40)'	a LineString with 3 points
Polygon	'POLYGON ((10 10, 10 20, 20 20, 20 15, 10 10))'	a Polygon with 1 exterior ring and 0 interior rings
Multipoint	'MULTIPOINT (10 10, 20 20)'	a MultiPoint with 2 point
MultiLineString	'MULTILINESTRING ((10 10, 20 20), (15 15, 30 15))'	a MultiLineString with 2 linestrings
MultiPolygon	'MULTIPOLYGON (((10 10, 10 20, 20 20, 20 15, 10 10)), ((60 60, 70 70, 80 60, 60 60)))'	a MultiPolygon with 2 polygons
GeomCollection	'GEOMETRYCOLLECTION (POINT (10 10), POINT (30 30), LINESTRING (15 15, 20 20))'	a GeometryCollection consisting of 2 Point values and a LineString value

## Riassunto dei costrutti di un database geo-relazionale

### Dominio base

- Dominio di SQL 92 + Tipi geometrici di SQL + Specifica delle simple feature

### Relazione (tabella)

- Permette di definire insiemi di tuple omogenei, dove ogni tupla ha un insieme di attributi che ha come dominio uno dei domini base

### Indice spaziale

- È una struttura per accedere al database spaziale sulle sue locazioni nello spazio delle reference (R-tree, Quad-tree, ...)

## Example of spatial database schema

### Relations

WOOD(Type: STRING, Extension: MULTIPOLYGON)  
 MEADOW(Pasture: BOOLEAN, Extension: MULTIPOLYGON)

### Spatial indices

WOOD\_spatial\_idx ON WOOD(Extension);  
 MEADOW\_spatial\_idx ON MEADOW (Extension);

## Example of an spatial database instance

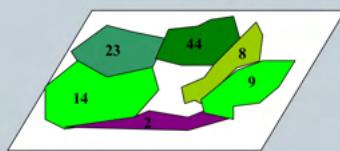
### WOOD

Tipo	Extension
Copse	23,44
Broadleaf	8,44

### MEADOW

Pasture	Extension
True	14,9
False	2

Graphical representation of the geometries



## Creare un database spaziale in SFM (PostGIS)

- PostGIS è un'estensione di PostgreSQL per gestire dati spaziali
- Introduce tipi geometrici e funzioni per i dati spaziali
- Per illustrare le sue caratteristiche descriviamo i passi necessari per effettuare le seguenti task:

- 1 Creare una tabella in PostGIS con attributi spaziali e non spaziali
- 2 Caricare dati spaziali da shapefile a PostGIS
- 3 Conicare dati nella tabella creata al punto 1

### Step 1 : Creazione di una tabella del database

- Crea la tabella senza attributi spaziali usando il comando SQL CREATE TABLE
- Aggiunta di uno o più attributi spaziali attraverso la funzione di PostGIS

AddGeometryColumn (schema, table, attributeGeo, srid, geoType, dimension)

```
CREATE TABLE S.prova (ID int, NAME varchar(20));  
SELECT AddGeometryColumn ('S', 'prova', 'geom', -1, 'LINESTRING', 2);
```

- Sono disponibili i tipi spaziali (geoType): POINT, LINESTRING, POLYGON, MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION
- srid: codice che identifica il sistema di riferimento (-1 se sconosciuto)

### Effetti delle funzioni AddGeometryColumn():

- Aggiunge un attributo spaziale alla tabella con i vincoli di integrità che garantiscono la correttezza dei valori che assumerà.
- Genera i metadati necessari nella tabella "geometry\_columns" seguendo lo standard OGC

### Step 2: caricare dati spaziali da un shapefile

- Per caricare una grande quantità di dati è disponibile un'applicazione: shp2pgsql. Converti uno shapefile in un file SQL. Il file SQL contiene la lista di comandi SQL che crea la tabella e carica il contenuto del shapefile in essa:

```
shp2pgsql <options> <shapefile> <table> <fileSQL>  
<options>:  
-c: create a new table  
-D: use the format PostgreSQL "dump" for the output.  
-s: allows to specify the SRID  
-I: create an index on the spatial attribute
```

### Step 3: Il trasferimento dei dati dalla tabella generata da shp2pgsql alla tabella finale

Il seguente comando SQL verrà usato per il data transfer:

```
INSERT INTO <DBtable> (<attributeList>) (SELECT  
<attributeList> FROM <TEMPtable> WHERE <cond>)
```

- Gli errori possono essere generati dalle operazioni di insert se le geometrie caricate dal shapefile non sono corrette rispetto al tipo geometrico associato all'attributo spaziale nella tabella finale del database

## Step 4: fare il check del contenuto finale della tabella

- Come possiamo vedere il contenuto di un attributo spaziale?

Possiamo usare la funzione che converte un valore geometrico dalla sua rappresentazione interna in una leggibile e testuale come WKT (well known Text representation) o EWKT per le geometrie in 3D.

- ST\_AsText(geom), ST\_AsEWKT(geom)
- Inverse functions: ST\_GeomFromText(WKT),  
ST\_GeomFromEWKT(EWKT)
- WKT example: LINESTRING(1 1, 3 5)

da testo a  
geometria

Altimenti è possibile visualizzare queste geometrie attraverso altri GIS software come OpenJump o QGIS

## SQL in GIS

- PostGIS fornisce le seguenti funzioni che implementano le relazioni topologiche di Clementini:

- boolean ST\_Disjoint(geometry A, geometry B);

Descrizione: Overlap, Touches, Within implicano tutte che le geometrie non siano disgiunte nello spazio. Se una delle precedenti ritorna true, allora le geometrie non sono spazialmente disgiunte. Disjoint implica il falso per l'intersezione spaziale

- boolean ST\_Touches(geometry g1, geometry g2);

Descrizione: ritorna vero se i soli punti in comune tra g1 e g2 si trovano nell'unione dei punti dei bordi di g1 e g2.

La relazione ST\_touches può essere applicata a: Area/Area, Linea/Linea, Area/Linea, Punto/Area, ma non a Punto/Punto



- boolean ST\_Within(geometry A, geometry B);

Descrizione: ritorna vero se la geometria A è completamente all'interno della geometria B. Perché questa funzione abbia senso, le geometrie devono essere della stessa proiezione delle coordinate e avere uguali SRID.

Se A è Within a B e viceversa allora A e B sono spazialmente uguali.

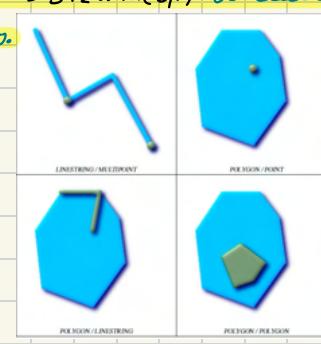


- boolean ST\_Contains(geometry A, geometry B);

Descrizione: geometry A contiene geometry B se e solo se non ci sono punti di B che si trovano nell'exterior di A e almeno un punto di B giace nell'interno di A.

Ritorna TRUE se la geometria B è completamente all'interno di A. Perché questa funzione abbia senso, le geometrie devono essere della stessa proiezione delle coordinate e avere uguali SRID.

ST\_Contains è la contrapposita di ST\_Within.  $ST\_Contains(A,B) \rightarrow ST\_Within(B,A)$  ad eccezione di grammatiche non valide in cui ritorna sempre falso o non definito.



## • boolean ST\_Equals(geometry A, geometry B)

**DESCRIZIONE:** ritorna vero se le geometrie date sono "uguali spazialmente". Viene usata per avere una risposta 'migliore' rispetto all'operatore '='.

Ugualanza spaziale implica  $ST\_Within(A,B) = \text{true}$  e  $ST\_Within(B,A) = \text{true}$  e significa anche che l'ordine dei punti può essere diverso ma rappresenta la stessa struttura geometrica

Per verificare se l'ordine dei punti è consistente, usa  $ST\_OrderingEquals$

## • boolean ST\_Overlaps(geometry A, geometry B)

**DESCRIZIONE:** ritorna vero se le geometrie si sovrappongono spazialmente. Con ciò intendiamo che esse si intersecano, ma una non contiene interamente l'altra.

## • boolean ST\_Crosses(geometry A, geometry B)

**DESCRIZIONE:**  $ST\_Crosses$  ritorna true se le loro intersezioni si "incrociano spazialmente", le geometrie hanno alcuni punti interni in comune ma non tutti. L'intersezione degli interni delle due geometrie non deve essere vuota e deve avere dimensioni inferiori alla massima dimensione delle due figure in input. Inoltre l'intersezione delle due geometrie non deve equivalere neanche alle due geometrie. Sorgenti o ritorna FALSE

Altre funzioni interessanti sono:

- **geometry ST\_Buffer(geometry g1, float radius\_of\_buffer);**

**Description:** Returns a geometry/geography that represents all points whose distance from this Geometry/geography is less than or equal to radius\_of\_buffer. Geometry: Calculations are in the Spatial Reference System of the geometry.
- **float ST\_Distance(geometry g1, geometry g2);**

**Description:** For geometry type returns the 2-dimensional minimum Cartesian distance between two geometries in projected units (spatial ref units).

## Query

1. **Municipality**(ID, Name, Inhabitants, Extension: MULTIPOLYGON)  
**Province**(ID, Name, Extension: MULTIPOLYGON)  
**River**(ID, Name, Path: MULTILINESTRING)  
**Road**(ID, Name, Type, Path: MULTILINESTRING)

**Q:** Find the name of the rivers that cross at least two different provinces.

```
SELECT R.Name
FROM River R, Province P1, Province P2
WHERE P1.ID <> P2.ID AND
      ST_Crosses(R.Path, P1.Extension) AND
      ST_Crosses(R.Path, P2.Extension)
```

2. **Find the name of the municipalities that are located at a distance less than 20Km from the river 'Adige'.**

```
Municipality (ID, Name, Inhabitants, Extension: MULTIPOLYGON)
River(ID, Name, Path: MULTILINESTRING)

SELECT M.Name
FROM Municipality M, River R
WHERE R.Name = 'Adige' AND
      ST_Overlaps(M.Extension,
                  ST_Buffer(R.Path, 20.000));
```

3. **Find for every road the rivers that it intersects returning in the result the name of the road, the name of the river and the geometry WKT of the road.**

```
River(ID, Name, Path: MULTILINESTRING)
Road(ID, Name, Type, Path: MULTILINESTRING)

SELECT RD.Name, RV.Name, ST_AsText(RD.Path)
FROM Road RD, River RV
WHERE NOT ( ST_Disjoint(RD.Path, RV.Path))
```

4. **Find for every province the number of municipalities it contains returning in the result the name of the province, the number of its municipalities and the average inhabitants of its municipalities.**

```
Municipality(ID, Name, Inhabitants, Extension: MULTIPOLYGON)
Province(ID, Name, Extension: MULTIPOLYGON)

SELECT P.Name, count(*) As #mun, avg(M.Inhabitants)
FROM Municipality M, Province P
WHERE ST_Contains(P.Extension, M.Extension)
GROUP BY P.ID, P.Name
```

5. **Find the municipalities with less than 10.000 inhabitants which are adjacent to a municipality with more than 1.000.000 inhabitants, returning in the result the name of the municipality and the name of its province.**

```
Municipality(ID, Name, Inhabitants, Extension: MULTIPOLYGON)
Province(ID, Name, Extension: MULTIPOLYGON)

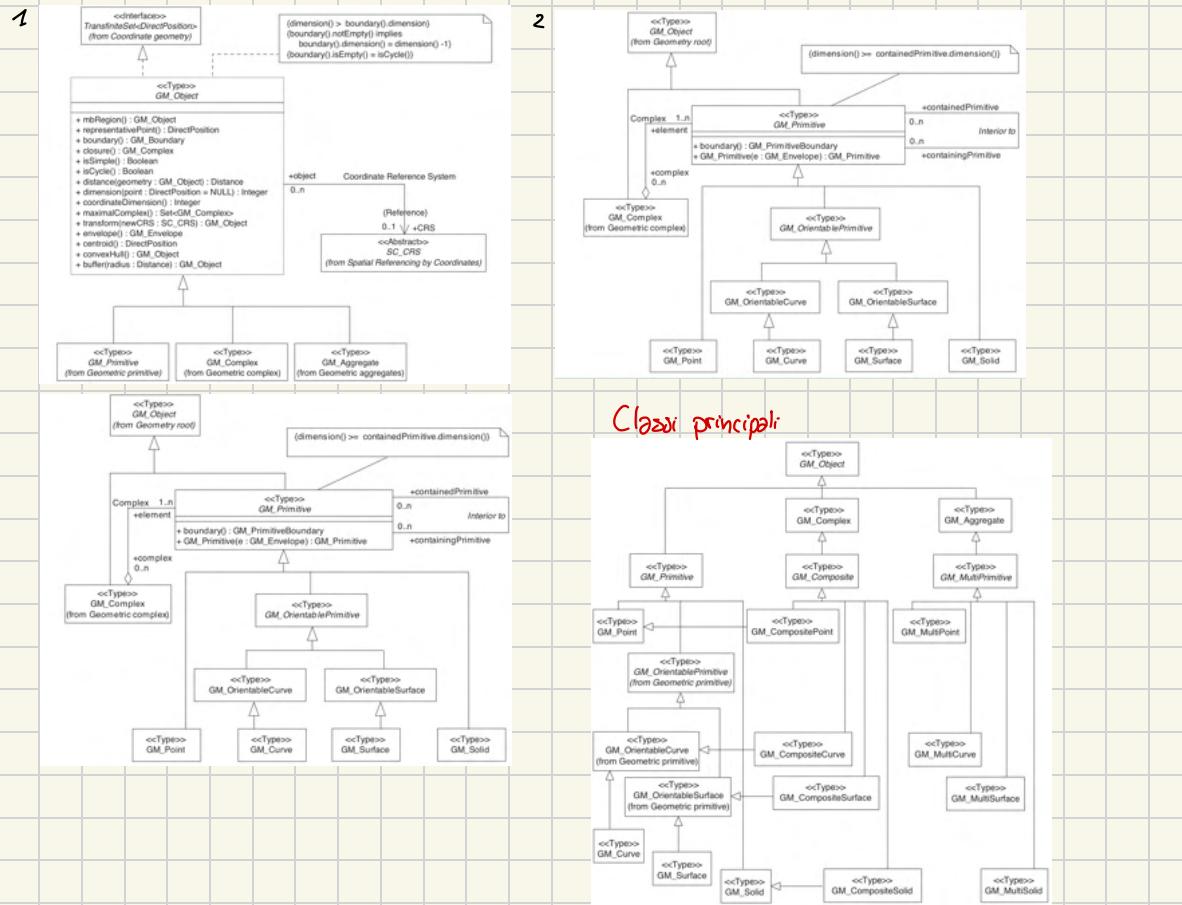
SELECT M.Nome, P.Nome
FROM Municipality M, Province P
WHERE M.Inhabitants < 10.000 AND
      ST_Contains(P.Extension, M.Extension) AND
      EXISTS (SELECT 1 FROM Municipality M1
              WHERE M1.Inhabitants > 1.000.000 AND
                    S_Touches(M.Extension, M1.Extension));
```

E' possibile fare il design di un database spaziale a livello concettuale applicando lo stesso approccio basato sullo UML già presentato per le tecnologie NoSQL.

Dobbiamo solo estendere UML con i tipi spaziali adeguati.

I tipi spaziali che vogliamo usare nella modellazione UML sono le classi appartenenti allo standard ISO 19107 "Spatial Schema".

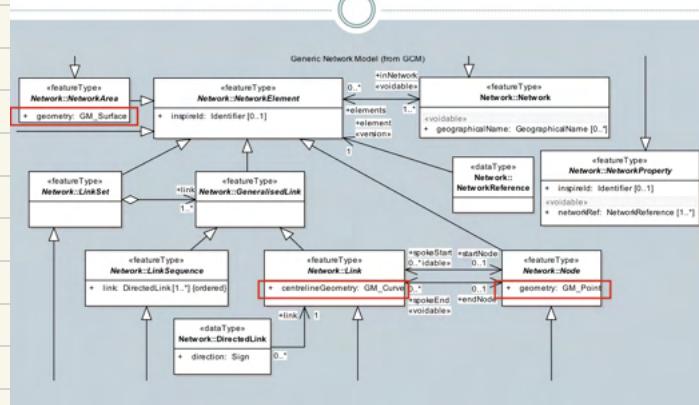
## Spatial schema ISO 19107



## Classi principali

### Examples of UML class diagrams with spatial attributes (from INSPIRE directive)

UML (ISO spatial schema)	OGC (simple feature specification)
<b>GM_Object</b>	
<b>GM_Aggregate</b>	→ <b>GEOMETRYCOLLECTION</b>
<b>GM_Curve</b>	→ <b>LINESTRING</b>
<b>GM_MultiCurve</b>	→ <b>MULTILINESTRING</b>
<b>GM_Point</b>	→ <b>POINT</b>
<b>GM_MultiPoint</b>	→ <b>MULTIPOINT</b>
<b>GM_Surface</b>	→ <b>POLYGON</b>
<b>GM_MultiSurface</b>	→ <b>MULTIPOLYGON</b>





## Database bi-temporali:

Valid e Transaction Time possono essere legati in diverse maniere

Sulla base delle relazioni che esistono tra:

- L'inizio e/o la fine della validità
- Il transaction Time di ciascuna singola tupla di una tabella

• Ci sono diverse classi di relazioni:

- retroattive, retroattiva delayed, predittive, predittiva anticipata
- retroattive bounded, fortemente retroattive delayed, fortemente retroattive bounded delayed
- generale, degenerata ...

Dato il punto di partenza VT<sub>s</sub> del valid time e il punto di partenza TT<sub>s</sub> del transaction time, una relazione è retroattiva se, per ogni tupla della relazione vale  $VT_s \leq TT_s$

• Una relazione è retroattiva delayed con bound  $\Delta t \geq 0$  se per ogni tupla della relazione vale  $VT_s \leq TT_s - \Delta t$

• In accordo coi criteri citati la relazione dei sintomi del paziente è retroattiva, perché  $VT_s < TT_s$  per tutte le tuple. Questi dati vengono inseriti dopo che sono sorti i sintomi.

## Database temporali:

• Possono essere definiti i seguenti tipi di database temporali

- Snapshot: rappresentano solo lo stato corrente del contesto dell'applicazione modellata
- Valid Time database: (conosciuto anche come Historical Database) supportano solo il tempo di Validità
- Transaction time database: supportano solo il transaction time
- Bitemporal time: supportano sia tempo di validità che transaction time L'esempio precedente era un database bitemporale

Ogni relazione in un database bi-temporale è estesa con la specifica di tempo di validità e transaction time con granularità che può essere

- a livello di tupla
- a livello di attributo

e specifica il tempo come:

- Istante
- Intervallo
- Insieme di intervalli

Consideriamo ora database bi-temporali con tempo specificato a livello di tupla come intervallo, o solo in certi casi come insieme di intervalli

Se il tempo di validità e il transaction time sono rappresentati come intervalli ciò indica che a ciascuna tupla sono associati due intervalli temporali, ognuno definito da due istanti di tempo: inizio e fine. Ogni intervallo è specificato come segue

[ start, end ]

Dove gli istanti sull'asse del tempo sono rappresentati con la granularità scelta per il tempo, che può essere millisecondi, giorni, mesi o anni. È possibile anche il valore  $\infty$  che indica un intervallo aperto a destra

Per esempio: [ 1/1/2000,  $\infty$  ]

• Può rappresentare il VT di una tupla che memorizza un oggetto nato l' 1/1/2000 e che oggi è ancora vivo

• Può rappresentare il TT di una tupla aggiunta l' 1/1/2000 e che ad oggi non è stata ancora eliminata

Nelle query sarà necessario specificare condizioni sugli intervalli temporali, è importante introdurre allora un insieme di relazioni

A  -----  B  -----	A IS EQUAL TO B B IS EQUAL TO A
A  -----  B  ----- -----	A IS BEFORE B B IS AFTER A
A  -----  B  ----- -----	A MEETS B B IS MET BY A
A  -----  B  ----- -----	A OVERLAPS B B IS OVERLAPPED BY A
A  -----  B  ----- -----	A STARTS B B IS STARTED BY A
A  -----  B  ----- -----	A FINISHES B B IS FINISHED BY A
A  -----  B  ----- -----	A IS DURING B B CONTAINS A

## Integrazione del tempo e degli spazi in Postgres

• Per rappresentare in maniera valida il tempo in PostgreSQL introduciamo queste regole:

• Quando una tabella deve specificare il valid time delle sue tuple, aggiungiamo due attributi alla tabella

- Start VT di tipo TIMESTAMP
- End VT di tipo TIMESTAMP

• Nel scrivere le query possiamo usare la funzione OVERLAPS per comparare due VT come segue:

- (StartVT1, EndVT1) OVERLAPS (StartVT2, EndVT2) or
- (StartVT1, length1) OVERLAPS (StartVT1, length2)
- length1/2 can be an interval, like: INTERVAL '100 days'

## Nel predicato

(StartVT<sub>1</sub>, EndVT<sub>1</sub>) OVERLAPS (StartVT<sub>2</sub>, EndVT<sub>2</sub>)

Ogni periodo (*start*, *end*) viene considerato rappresentare l'intervallo semiaperto  $start \leq time < end$   
 a meno che *start* e *end* non siano uguali e allora rappresentano lo stesso istante

Overlap ritorna vero se i due intervalli hanno almeno un istante in comune, compreso se uno è contenuto nell'altro

- Other operations with DATE, INTERVAL and TIMESTAMP

```

date + integer → date
Add a number of days to a date
date '2001-09-28' + 7 → 2001-10-05

date + interval → timestamp
Add an interval to a date
date '2001-09-28' + interval '1 hour' → 2001-09-28 01:00:00

date + time → timestamp
Add a time-of-day to a date
date '2001-09-28' + time '03:00' → 2001-09-28 03:00:00

interval + interval → interval
Add intervals
interval '1 day' + interval '1 hour' → 1 day 01:00:00

timestamp + interval → timestamp
Add an interval to a timestamp
timestamp '2001-09-28 01:00' + interval '23 hours' → 2001-09-29 00:00:00

time + interval → time
Add an interval to a time
time '01:00' + interval '3 hours' → 04:00:00

```

- Other operations with DATE, INTERVAL and TIMESTAMP

```

date - date → integer
Subtract dates, producing the number of days elapsed
date '2001-10-01' - date '2001-09-28' → 3

date - integer → date
Subtract a number of days from a date
date '2001-10-01' - 7 → 2001-09-24

date - interval → timestamp
Subtract an interval from a date
date '2001-09-28' - interval '1 hour' → 2001-09-27 23:00:00

time - time → interval
Subtract times
time '05:00' - time '03:00' → 02:00:00

time - interval → time
Subtract an interval from a time
time '05:00' - interval '2 hours' → 03:00:00

timestamp - interval → timestamp
Subtract an interval from a timestamp
timestamp '2001-09-28 23:00' - interval '23 hours' → 2001-09-28 00:00:00

interval - interval → interval
Subtract intervals
interval '1 day' - interval '1 hour' → 1 day -01:00:00

timestamp - timestamp → interval
Subtract timestamps (converting 24-hour intervals into days, similarly
to justify_hours())
timestamp '2001-09-29 03:00' - timestamp '2001-07-27 12:00' → 63 days 15:00:00

```

CREATE TABLE Patient (

```

    PID INTEGER, Name VARCHAR(30),
    Surname VARCHAR(30), Dept VARCHAR(30),
    BirthDate DATE, StartVT TIMESTAMP, EndVT TIMESTAMP);
(VT is the period of stay at the hospital)

CREATE TABLE Symptom (
    Patient INTEGER, Symptom VARCHAR(30), StartVT TIMESTAMP,
    EndVT TIMESTAMP);
(VT is the period during which the symptom occurs)

CREATE TABLE Therapy (
    Patient INTEGER, Therapy VARCHAR(30),
    Quantity FLOAT, StartVT TIMESTAMP, EndVT TIMESTAMP);
(VT is the period during which the therapy is given to the patient)

```

query Temporal:

- Find the surname and the name of the patients that on 3/2/23 have been staying at the department of Medicine for at least 3 days and they had a therapy of 'paracetamol' on the same day.

```

SELECT P.Surname, P.Name
FROM PATIENT P, THERAPY T
WHERE T.Patient = P.PID AND P.Dept = 'Medicine' AND
P.StartVT <= DATE '3/2/2023' - INTERVAL '3' DAY
AND P.EndVT > DATE '3/2/2023'
AND T.Therapy = 'paracetamol'
AND (T.StartVT, T.EndVT+1) OVERLAPS
(DATE '3/2/23', DATE '3/2/23');

```

- Find the surname and the name of the patients that were in some department on 1/5/22 e that during the stay have had fever followed by a symptom of abdominal pains.

```

SELECT P.Surname, P.Name
FROM PATIENT P, SYMPTOM Sf, SYMPTOM Sd
WHERE (P.StartVT, P.EndVT+1) OVERLAPS
(DATE '1/5/22', DATE '1/5/22');
AND Sf.Patient = P.PID AND Sf.End_VT <= Sd.Start_VT
AND Sf.Symptom = 'fever' AND Sd.Patient = P.PID
AND Sd.Symptom = 'abdominal pains'
AND (P.StartVT, P.EndVT+1) OVERLAPS (Sf.StartVT, Sf.StartVT)
AND (P.StartVT, P.EndVT+1) OVERLAPS (Sf.EndVT, Sf.EndVT)
AND (P.StartVT, P.EndVT+1) OVERLAPS (Sd.StartVT, Sd.StartVT)
AND (P.StartVT, P.EndVT+1) OVERLAPS (Sd.EndVT, Sd.EndVT)

```

Note bene

- In the previous query if we only write:

```

(P.StartVT, P.EndVT+1) OVERLAPS (Sf.StartVT, Sf.EndVT)
instead of
(P.StartVT, P.EndVT+1) OVERLAPS (Sf.StartVT, Sf.StartVT)
(P.StartVT, P.EndVT+1) OVERLAPS (Sf.EndVT, Sf.EndVT)

```

We are accepting also the case in which the VT of the symptom is not contained in the VT of the stay, and this is explicitly excluded by the query description ("... during the stay ...")!

- Find the surname and the name of the patient that have had a therapy of paracetamol and the first symptom after the therapy started was of abdominal pains.

```
SELECT P.Surname, P.Name
FROM PATIENT P, THERAPY T, SYMPTOM S
WHERE S.Patient = P.PID AND T.Patient = P.PID
AND T.Therapy = 'paracetamol'
AND S.Symptom = 'abdominal pains'
AND T.StartVT <= S.StartVT AND
NOT EXISTS (SELECT 1 FROM SYMPTOM S1
WHERE S1.Patient = P.PID AND
T.StartVT <= S1.StartVT AND
S1.StartVT < S.StartVT);
```

- Find the surname and the name of the pairs of patients that had exactly the same symptoms on 3/2/14.

```
SELECT P1.Surname, P1.Name, P2.Surname, P2.Name
FROM PATIENT P1, PATIENT P2, SYMPTOM S
WHERE S.Patient = P1.PID AND
(S.StartVT, S.EndVT+1) OVERLAPS ('3/2/14', '3/2/14') AND
NOT EXISTS (SELECT 1 FROM SYMPTOM S1
WHERE S1.Patient = P1.PID
AND (S1.StartVT, S1.EndVT+1) OVERLAPS ('3/2/14', '3/2/14'))
AND NOT EXISTS (SELECT 1 FROM SYMPTOM S2
WHERE S2.Patient = P2.PID AND
(S2.StartVT, S2.EndVT+1) OVERLAPS ('3/2/14', '3/2/14')
AND S1.Symptom = S2.Symptom)
```

Non voglio  
trovare un  
simbolo  
diverso

## query Spazio-temporali

```
CREATE TABLE Epidemic (
    Code CHAR(3), FirstCase POINT, Disease VARCHAR(50),
    StartVT TIMESTAMP, EndVT TIMESTAMP)
```

(VT is the period in which the epidemic exists)

```
CREATE TABLE NumberOfCases (
    EpidCode CHAR(3), Number INTEGER,
    StartVT TIMESTAMP, EndVT TIMESTAMP)
```

(VT is the period in which the number of cases is valid)

```
CREATE TABLE Evolution (
    EpidCode CHAR(3), Extension MULTIPOLYGON,
    StartVT TIMESTAMP, EndVT TIMESTAMP)
```

(VT is the period in which the epidemic is extended on the polygon contained in the attribute Extension)

- Find the code and the disease of the epidemics currently in progress that intersect (even only in a small part) the territory described by the geometry with WKT representation:

```
POLYGON((1200 45000, 1300 45500, 1250 46000, 1200 45000))
```

```
SELECT E.Code, E.Disease
FROM Epidemic E, Evolution EV WHERE E.Code = EV.EpidCode AND
(E.StartVT, E.EndVT+1) OVERLAPS (DATE 'today', DATE 'today') AND
(EV.StartVT, EV.EndVT+1) OVERLAPS (DATE 'today', DATE 'today')
AND NOT ST_Disjoint(EV.Extension, ST_GeomFromText(
    'POLYGON((1200 45000, 1300 45500,
    1250 46000, 1200 45000))'));
```

- Find the evolution of the extension of the epidemic having code E345 from 1/1/2015 to 30/4/2015 returning the geometry as WKT and the instant of start and end of the validity of each geometry of the evolution.

```
SELECT ST_AsText(EV.Extension), EV.StartVT as start,
EV.EndVT as end
FROM Evolution EV
WHERE EV.EpidCode = 'E345' AND
NOT (EV.EndVT < DATE '1/1/2015' OR
DATE '30/4/2015' < EV.StartVT)
ORDER BY EV.StartVT
```

- Find the code and the starting instant and the ending instant of the epidemic whose extension has never intersected the territory

```
POLYGON((1200 45000, 1300 45500, 1250 46000, 1200 45000))
```

```
SELECT E.Code, E.StartVT as start, E.EndVT as end
FROM Epidemic E
WHERE NOT EXISTS (SELECT 1 FROM Evolution EV
WHERE E.Code = EV.EpidCode AND
NOT ST_Disjoint(EV.Extension,
ST_GeomFromText(
    'POLYGON(1200 45000, 1300 45500, 1250 46000, 1200 45000))));
```

## Normalization theory

L'obiettivo della teoria della normalizzazione, che è nata nel contesto del modello relazionale è di fornire metodi formali per modello relazionale, è quello di fornire metodi formali per:

- 1 Descrivere le connessioni semantiche tra i dati: (dipendenza funzionale)
- 2 Scoprire la ridondanza nella rappresentazione dei dati: (verifica dello schema con le forme normali)
- 3 Valutare la qualità dello schema (forme normali)
- 4 Suggerire miglioramenti alla progettazione dello schema (schema decomposition)

### Ridondanza

Le seguenti tabelle memorizzano gli esami dei pazienti in un determinato ospedale. Gli attributi che descrivono le informazioni che vogliamo rappresentare sono: codice, nome, cognome e data di nascita del paziente, insieme alla data, al tipo e all'esito dei suoi esami.

Example 1 (Example of redundancy)

Code	EType	Surname	Name	BDate	Result	EDate
00100	glycemia	Rossi	Paolo	1.1.1980	120	17.7.13
00200	glycemia	Bianchi	Mario	10.10.1981	90	18.06.14
00100	hemoglobin	Rossi	Paolo	1.1.1980	13	5.9.14

Si noti che nella prima e nella terza riga i dati relativi al paziente "Rossi Paolo" sono duplicati, producendo così una ridondanza nella tabella.

La presenza di ridondanza nella rappresentazione dei dati può essere volutamente non sempre come un aspetto negativo. In alcuni contesti la ridondanza infatti è necessaria per garantire i requisiti di performance o le richieste di disponibilità.

Tuttavia, nelle applicazioni tradizionali, quando l'identità degli oggetti (o delle istanze informative) e la coerenza sono importanti per il sistema informativo, la ridondanza può diventare un problema, poiché produce anomalie nella modifica dei dati.

Abbiamo 3 tipi di anomalie:

Anomalie di aggiornamento: si verifica quando per aggiornare il valore di una proprietà di un oggetto è necessario modificare più tuple in una o più tabelle.

Anomalie di inserimento: si verifica quando per inserire un nuovo oggetto è necessario aggiungere valori per proprietà sconosciute (valori nulli) anche per attributi che fanno parte di una chiave primaria della tabella.

Anomalie di cancellazione: si verifica quando per cancellare un oggetto è necessario cancellare i valori di altri oggetti che si vogliono lasciare nel database, oppure introdurre valori nulli, per evitare questo problema. Questo anche per gli attributi di una chiave primaria della tabella.

Per poter descrivere in modo formale le caratteristiche delle informazioni da rappresentare, consentendo l'individuazione automatica della ridondanza e delle sue conseguenze viene introdotto il concetto di dipendenza funzionale.

Questo concetto permette di specificare un vincolo di integrità che descrive un legame funzionale tra due insiemi di attributi.

Le dipendenze funzionali possono essere utilizzate come strumenti formali per progettare i dati in forma astratta, in maniera indipendente rispetto alla tecnologia usata per l'implementazione. Tuttavia, sono state adottate principalmente per analizzare solo gli schemi dei database relazionali.

## Dipendenze funzionali

Le dipendenze funzionali possono essere usate per:

- Specificare un vincolo di integrità sul contenuto del database
- Descrivere che alcuni attributi determinano il valore assunto da altri attributi
- Generalizzare il concetto di chiave

Definizione 1 (dipendenza funzionale da un'istanza di relazione r)

Sia  $r$  un'istanza di uno schema di relazioni  $R(x)$ , e siano  $\alpha \subseteq X$  e  $\beta \subseteq X$ . Diciamo che  $\alpha$  determina funzionalmente  $\beta$  ( $\alpha \rightarrow \beta$ ) in  $r$ , se per ogni coppia di tuple  $t_1, t_2 \in r$  che hanno gli stessi valori in  $\alpha$  presenteranno gli stessi valori in  $\beta$ . Più formalmente:

$$\alpha \rightarrow \beta \text{ è soddisfatta in } r \text{ se } \forall t_1, t_2 \in r: ((t_1[\alpha] = t_2[\alpha]) \Rightarrow (t_1[\beta] = t_2[\beta]))$$

Definizione 1 (dipendenza funzionale su uno schema relazionale  $R(x)$ )

Una dipendenza funzionale  $\alpha \rightarrow \beta$  è valida sullo schema  $R(x)$ , se per ogni istanza  $r$  di  $R(x)$  vale che  $r$  soddisfa  $\alpha \rightarrow \beta$ .

Alcuni esempi di dipendenza funzionale validi sullo schema:

EXAM(Code, EType, Surname, Name, BDate, Result, EDate)

Example 2

D<sub>1</sub>: Code → Surname Name BDate  
D<sub>2</sub>: Code EType EDate → Result  
D<sub>3</sub>: Surname Name → Name

- D<sub>1</sub>: indica che gli attributi Surname, Name e BDate sono determinati da Code
- D<sub>2</sub>: indica che gli attributi Code, EType, EDate determinano il risultato dell'esame
- D<sub>3</sub>: è detta banale, poiché è ovvio che se due tuple sono uguali sugli attributi Cognome e Nome, allora lo saranno anche sul singolo attributo Name

Derivazione di nuove dipendenze funzionali

Si noti che, a partire da alcune dipendenze funzionali valide nel contesto applicativo, altre possono essere derivate per inferenza da esse.

Diciamo che le ultime sono logicamente implicite dalle prime, formalmente dato un insieme di dipendenze funzionali F:

$$F \models \alpha \rightarrow \beta$$

significa che la dipendenza funzionale  $\alpha \rightarrow \beta$  è logicamente implicata da F

Example 3 (Examples of derivation)

Starting from

D<sub>1</sub>: Code → Surname Name BDate  
D<sub>2</sub>: Code EType EDate → Result  
D<sub>3</sub>: Surname Name → Name

D<sub>1</sub>, D<sub>2</sub>  $\models$  Code Etype EDate → Surname Name BDate Result  
moreover :

D<sub>1</sub>, D<sub>2</sub>  $\models$  Code Etype EDate → Code Etype EDate Surname Name BDate Result

Ovvero, (Code, Etype, Date) compongono una super chiave per la tabella Esame poiché determinano tutti gli attributi della tabella

### Definizione 3 ( Chiusura di un insieme di dipendenze funzionali)

Dato un insieme di dipendenze funzionali  $F$ , l' insieme di tutte le dipendenze che sono logicamente implicate da  $F$  è chiamato Chiusura di  $F$  ( $F^+$ )

$$F^+ = \{\alpha \rightarrow \beta \mid F \models \alpha \rightarrow \beta\}$$

Rappresenta tutta la "conseguenza" che abbiamo

### Definizione 4 ( dipendenze funzionali banali)

Una dipendenza funzionale  $\alpha \rightarrow \beta$  definita su una relazione  $R(x)$  è banale se è sempre soddisfatta da qualsiasi istanza  $r$  della relazione  $R(x)$ . In generale  $\alpha \rightarrow \beta$ , è banale se  $\beta \subseteq \alpha$

### Definizione 5 ( Superchiavi)

Data una relazione dello schema  $R(x)$ ,  $K \subseteq X$  è una superchiave di  $R(x)$ , se  $K$  determina funzionalmente  $X$  ovvero  $K \rightarrow X$

### Definizione 6 ( Chiave)

Data una relazione con Schema  $R(x)$ ,  $K \subseteq X$  è una chiave di  $R(x)$ , se  $K$  determina funzionalmente  $X$  ( ovvero  $K \rightarrow X$  ) e non esiste un sottinsieme proprio  $\alpha \subseteq X$  tale che  $\alpha$  determini funzionalmente  $X$  ( ovvero  $\alpha \rightarrow X$  )

## Normalizzazione dei database relazionali

Il processo di normalizzazione mira a ridurre la ridondanza in uno schema relazionale applicando le seguenti fasi:

- Definire l'insieme delle dipendenze funzionali valide per lo schema relazionale considerato
- Applicare il test per determinare se lo schema relazionale soddisfa una forma normale o meno
- In caso contrario, decomporre lo schema, cioè dividere le relazioni, in modo che la forma normale sia soddisfatta e le relazioni non contengano più ridondanza

### Forma normale di Boyce e Codd

E' la forma normale più restrittiva. Infatti: essa richiede che una relazione non contenga alcuna ridondanza

#### Definizione 1 (Forma normale di Boyce e Codd)

Boyce and Codd Normal Form

Una relazione  $R(X)$  è in forma normale "Boyce e Codd", o soddisfa la BCNF, rispetto a un insieme di dipendenze funzionali  $F$ , se per ogni dipendenza funzionale non banale  $\alpha \rightarrow \beta \in F^+$  valida su  $X$ :

$$\alpha \rightarrow X \in F^+ \text{ (o } \alpha^+ = X\text{)}$$

Cioè,  $\alpha$  è una superchiave per  $R(X)$

Vista la precedente definizione di BCNF applicata a una relazione, possiamo estendere questa definizione a uno schema relazionale come segue:

uno schema relazionale  $R_1(X_1), \dots, R_n(X_n)$  è in BCNF  
se per ogni  $R_i(X_i)$  con  $i \in \{1, \dots, n\}$  è dimostrato essere in BCNF

Se uno schema relazionale è in BCNF, allora possiamo dire che è corretto, cioè non contiene ridondanze, viceversa è necessario applicare un processo di normalizzazione a tale schema per rimuovere le ridondanze non necessarie

#### Definizione 2 (Chiusura di un insieme di attributi)

Dato un insieme di dipendenze funzionali  $F$  su un insieme di attributi  $X$  e sia  $\alpha \subseteq X$ . Chiamiamo chiusura di  $\alpha$  rispetto a  $F$  il seguente insieme di attributi:

$$\alpha^+ = \{ A \mid \alpha \rightarrow A \in F^+ \text{ e } A \text{ è un attributo singolo} \}$$

→ indicato con la maiuscola

### Testing BCNF

#### Algorithm 1 (BCNF Test)

In:  $S = \{R_1(X_1), \dots, R_n(X_n)\}$ , a set of functional dependencies  $F$   
Out: TRUE/FALSE

```
Begin
  ForEach  $R_i(X_i) \in S$  do
    ForEach  $Y \subseteq X_i$  do
      compute  $Y^+$ 
      If  $X_i \subseteq Y^+$  or  $(Y^+ \cap (X_i - Y) = \emptyset)$ 
        continue;
      Else
        Return FALSE;
      End if
    End do
  End do
Return TRUE;
```

' mi dice che c'è qualcosa in più ma che non fa parte della tabella

If  $X_i \subseteq Y^+$  or  $(Y^+ \cap (X_i - Y) = \emptyset)$

continue;

Else

Return FALSE;

End if

End do

End do

Return TRUE;

## Esempio di Test BCNF

Let  $X$  be a set of attributes:  $X = \{A, B, C, D, E\}$   
and let  $\mathcal{F}$  be a set of functional dependencies on  $X$ :  
 $\mathcal{F} = \{A \rightarrow B, B \rightarrow C, B \rightarrow D\}$ .

Test if the following schema is in BCNF:

$R_1(A, C, E)$   
 $R_2(B, D)$

- Remembering that  $\mathcal{F} = \{A \rightarrow B, B \rightarrow C, B \rightarrow D\}$ :

- $R_1(A, C, E)$ :  
 $\{A\}^+ = \{A, B\}$   
 $= \{A, B, C, D\}$   
-  $X_i \subseteq Y^+$ ?  
 $\{A, C, E\} \not\subseteq \{A, B, C, D\}$   
-  $Y^+ \cap (X_i - Y) = \emptyset$ ?  
 $\{A, B, C, D\} \cap \{\{A, C, E\} - \{A\}\} = \{C\} \neq \emptyset$

AE è la chiave

$R_1$  is not in BCNF. Nessuna delle due condizioni vale.

- Remembering that  $\mathcal{F} = \{A \rightarrow B, B \rightarrow C, B \rightarrow D\}$ :

- $R_2(B, D)$   
 $\{B\}^+ = \{B, C, D\}$   
-  $X_i \subseteq Y^+$ ?  
 $\{B, D\} \subseteq \{B, C, D\} \checkmark$   
 $\{D\}^+ = \{D\}$   
-  $X_i \subseteq Y^+$ ?  
 $\{B, D\} \not\subseteq \{D\}$   
-  $Y^+ \cap (X_i - Y) = \emptyset$ ?  
 $\{D\} \cap \{\{B, D\} - \{D\}\} = \emptyset \checkmark$

$R_2$  is in BCNF.

## Decomposizione

Quando uno schema relazionale non è in BCNF, è possibile applicare una decomposizione dello schema per produrre uno schema che soddisfi BCNF.

### Definizione 3 (Decomposizione)

Dato uno schema relazionale  $R(x)$ , una decomposizione di  $R(x)$  è un insieme di relazioni:

$$\{R_1(x_1), R_2(x_2), \dots, R_n(x_n)\}$$

tale che  $X = X_1 \cup X_2 \cup \dots \cup X_n$

Si noti che gli  $x_i$  non devono per forza essere disgiunti.

Non tutte le decomposizioni possono essere accettabili. Due proprietà definiscono le decomposizioni accettabili.

La prima proprietà che definisce le decomposizioni accettabili riguarda la conservazione del contenuto del database.

#### Definizione 4 (Lossless join decomposition)

Data una relazione  $R(x)$  e un insieme di dipendenze funzionali  $F$ , la decomposizione  $R_1(x_1), \dots, R_n(x_n)$  è una decomposizione di lossless join, se per tutte le istanze  $r$  di  $R(x)$  che soddisfano  $F$ , vale la seguente condizione:

$$\Pi_{x_1}(r) \bowtie \dots \bowtie \Pi_{x_n}(r) = r$$

Questa proprietà richiede che il contenuto della relazione  $r$  possa essere ricostruito esattamente per mezzo di un join naturale applicato alle relazioni appartenenti alla decomposizione. Ciò implica che la decomposizione non può essere una partizione degli attributi in  $x$ .

Il seguente teorema fornisce un test per la proprietà di Lossless join decomposition.

#### Teorema 1 (Lossless join decomposition di due relazioni)

Una decomposizione  $R_1(x_1), R_2(x_2)$  di una relazione  $R(x)$  è una Lossless join decomposition rispetto a un insieme di dipendenze funzionali  $F$ , se una delle seguenti condizioni è valida:

$$\begin{aligned} x_0 \rightarrow x_1 &\in F^+ \\ x_0 \rightarrow x_2 &\in F^+ \end{aligned}$$

$$\text{con } x_0 = x_1 \cap x_2$$

#### Example

A possible decomposition of the following relation:

$EXAM(PatCode, Surname, Name, BirthDate, ExamType, ExamDate, Value)$

with the following valid functional dependencies:

$PatCode \rightarrow Surname\ Name\ BirthDate$

$PatCode\ ExamDate\ ExamType \rightarrow Value$

could be:

$R_1(PatCode, Surname, Name, BirthDate)$

$R_2(PatCode, ExamDate, ExamType, Value)$

Notice that, the decomposition  $R_1, R_2$  is lossless join w.r.t. the above listed functional dependencies, indeed,  $X_0 = \{PatCode\}$  and:

$PatCode \rightarrow Surname\ Name\ BirthDate$  is valid.

La seconda proprietà che definisce le decomposizioni accettabili riguarda la conservazione dei vincoli di integrità.

#### Definizione 5 (Conservazione delle dipendenze funzionali)

Data una relazione  $R(x)$  e un insieme di dipendenze funzionali  $F$ , la decomposizione  $R_1(x_1), \dots, R_n(x_n)$  preserva le dipendenze funzionali, se vale la seguente condizione:

$$(\Pi_{x_1}(F) \cup \dots \cup \Pi_{x_n}(F))^+ = F^+$$

$$\text{dove } \Pi_{x_i}(F) = \{\alpha \rightarrow \beta \mid \alpha \rightarrow \beta \in F^+ \wedge \alpha \cup \beta \subseteq X_i\}$$

Questa proprietà richiede che la decomposizione non possa dividere in relazioni diverse attributi che sono coinvolti nella stessa dipendenza funzionale.

### Example 5 (Decompositions)

The relation  $R = (A, B, C)$  with its functional dependencies:

$F = \{A \rightarrow B, B \rightarrow C\}$  can be decomposed in two different ways:

1<sup>st</sup> DECOMPOSITION:

$R_1(A, B)$   
 $R_2(B, C)$

$R_1(A, B)$   
 $R_2(A, C)$

- lossless join
  - preserves functional dependencies
- lossless join
  - does not preserve functional dependencies

Si noti che, in una decomposizione che preserva le dipendenze funzionali, i vincoli di integrità possono essere verificati su ogni singola relazione senza alcuna join aggregativo.

### Algorithm 2 (BCNF Decomposition)

In: a schema  $S = \{R_1(X_1), \dots, R_m(X_m)\}$ , a set of functional dependencies  $F$

Out: a decomposition  $\{R'_1(X'_1), \dots, R'_n(X'_n)\}$  with  $m \leq n$  that satisfies BCNF

Begin

    Result :=  $\{X_1, \dots, X_m\}$ ;

    Repeat

        ForEach  $X_i \in \text{Result}$  do

            If  $R_i(X_i)$  is not in BCNF

                ForEach  $\alpha \rightarrow \beta \in \Pi_{X_i}(F^+)$  do

                    If  $\alpha \rightarrow \beta$  is non trivial AND  $\alpha \cap \beta = \emptyset$  AND  $X_i \not\subseteq \alpha^+$

                        Result := ( $\text{Result} - \{X_i\}$ )  $\cup \{(X_i - \beta)\} \cup \{(\alpha, \beta)\}$ ; break;

                    End if

                End do

            End if

        End do

    Until Result changes

    Return Result;

End

### Example 6

Given the following schema and set of functional dependencies:

$R = (\text{doctor}, \text{town}, \text{address}, \text{patient}, \text{drug}, \text{period})$

$F = \{\text{doctor} \rightarrow \text{town address}, \text{drug} \rightarrow \text{period doctor}\}$

A key of  $R$  is  $\{\text{patient}, \text{drug}\}$ . By applying the previous algorithm 2, the following decomposition is obtained:

$R_1 = (\text{doctor}, \text{town}, \text{address})$  from the  $\text{doctor} \rightarrow \text{town address}$

$R_2 = (\text{drug}, \text{period}, \text{doctor})$  from the  $\text{drug} \rightarrow \text{period doctor}$

$R_3 = (\text{patient}, \text{drug})$

### Teorema 2 (BCNF, lossless join e conservazione delle dipendenze)

Dato una relazione  $R(x)$  e un insieme di dipendenze funzionali  $F$ , supponiamo che  $R(x)$  non sia in BCNF, allora non è sempre possibile generare una decomposizione di  $R(x)$  tale che:

- Sia nella BCNF
- Sia una lossless join decomposition
- Preservi tutte le dipendenze funzionali di  $F$

### Terza forma normale

Poiché non è sempre possibile ottenere una decomposizione che sia allo stesso tempo in BCNF e con le due proprietà richieste, è necessario definire una forma normale meno restrittiva.

Venne introdotta la terza forma normale con le seguenti caratteristiche:

- Permette una certa ridondanza nelle relazioni
- Conserva sempre le dipendenze forense dopo la decomposizione

## Definizione 6 (terza forma normale)

Una relazione  $R(x)$  è in terza forma normale (3NF) rispetto a un insieme di dipendenze funzionali  $F$  se, per ogni dipendenza non banale  $\alpha \rightarrow \beta \in F^*$  valida su  $X$ , vale una delle seguenti condizioni:

- $\alpha$  è una superchiave di  $R(x)$  (come richiesto da BCNF)
- ogni attributo  $A \in (\beta - \alpha)$  è contenuto in una chiave di  $R(x)$

## 3NF vs BCNF

### Teorema 3

Sia  $R(x)$  uno schema relazionale. Se  $R(x)$  è in BCNF, allora è anche in 3CNF. Non vale il contrario.

$$\text{BCNF}(R(x)) \Rightarrow \text{3NF}(R(x))$$

### Example 7

Let Therapy be a relation with the following schema:

$$\begin{aligned} &\text{Therapy(patient, symptom, therapy)} \\ &F = \{\text{patient symptom} \rightarrow \text{therapy} \\ &\quad \text{therapy} \rightarrow \text{symptom}\} \end{aligned}$$

Considerando il test BCNF, possiamo vedere che calcolando la chiusura del sottoinsieme  $\{\text{terapia}\}$  otteniamo  $(\text{terapia}, \text{sintomo})$ , che non contiene l'intero schema della relazione Terapia(), ma contiene alcuni attributi aggiuntivi, cioè sintomo, quindi vede il test BCNF.

Tuttavia considerando la 3NF, possiamo vedere che la dipendenza  $\text{terapia} \rightarrow \text{sintomo}$  non è un problema per 3NF, poiché l'attributo sintomo fa parte di una chiave della relazione Terapia()

### Algorithm 3 (3NF Test)

```
In: a schema  $S = \{R_1(X_1), \dots, R_m(X_m)\}$ , a set of functional dependencies  $F$ 
Out: TRUE/FALSE
Begin
  ForEach  $R_i \in S$  do
    compute the keys of  $X_i$ :  $K_{i,1}, \dots, K_{i,k_i}$ 
    ForEach  $Y \subseteq X_i$  do
      compute  $Y^+$ ;
      If  $X_i \subseteq Y^+$  or  $(Y^+ \cap (X_i - Y) = \emptyset)$  or  $(Y^+ \cap (X_i - Y)) \subseteq \bigcup_{j=1}^{k_i} K_j$ 
        continue;
      Else
        Return FALSE;
      End if
    End do
  End do
  Return TRUE;
End
```

## Decomposizione della 3NF

L'algoritmo per verificare la soddisfazione della 3NF ha un'elevata complessità; infatti calcolare le chiavi di uno schema relazionale dato un insieme di dipendenze funzionali ha una complessità esponenziale.

Tuttavia l'algoritmo per produrre una decomposizione che sia in 3NF è più semplice e può essere calcolato in tempo polinomiale.

### Algorithm 4 (Decomposizione di uno schema relazionale in 3NF)

**In:** a set of attributes  $X$  to decompose, a set of functional dependencies  $\mathcal{F}_c$  which is a minimal cover  
**Out:** a decomposed schema  $\{R_1(X_1), \dots, R_n(X_n)\}$   
**Begin**  
     $i := 0; S = \emptyset;$   
    **ForEach**  $\alpha \rightarrow \beta \in \mathcal{F}_c$  **do**  
        **If** it does not exist  $R_j \in S : \alpha\beta \subseteq R_j$   
             $i := i + 1; S := S \cup \{R_i(\alpha\beta)\};$   
        **End if**  
    **End do**  
    compute a key  $K$  for  $X$  w.r.t.  $\mathcal{F}_c$ ;  
    **If** it does not exist  $R_j \in S : K \subseteq R_j$   
         $i := i + 1; S := S \cup \{R_i(K)\};$   
    **End if**  
    **Return**  $S$ ;  
**End**

### Example 8

Given the following relational schema:

$R(\text{hospital}, \text{patient}, \text{doctor}, \text{department})$

$\mathcal{F} = \{\text{doctor} \rightarrow \text{hospital department}; \text{patient hospital} \rightarrow \text{doctor}\}$

The keys of  $R$  are:  $\{\text{patient, hospital}\}$  and  $\{\text{doctor, patient}\}$ . The decomposition resulting from the application of the algorithm presented in the previous slide is:

$R_1(\underline{\text{doctor}}, \text{hospital}, \text{department})$

$R_2(\underline{\text{patient}}, \text{hospital}, \text{doctor})$

## Seconda Forma Normale (2NF)

Una relazione  $R(x)$  è in seconda forma normale (2NF) rispetto a un insieme di dipendenze funzionali  $F$  se per ogni dipendenza non banale  $\alpha \rightarrow \beta \in F^+$  che è valida su  $x$ , vale una delle seguenti condizioni:

- Tali gli attributi  $A \in (\beta - \alpha)$  sono contenuti in una chiave di  $R(x)$
- $\alpha$  non è un sottoinsieme proprio di una chiave di  $R(x)$

Può essere provato che:  $3NF \Rightarrow 2NF$

