UNIVERSITA' POLITECNICA DELLE MARCHE

FACOLTA' DI INGEGNERIA

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea magistrale in Ingegneria Informatica e dell'Automazione

RELAZIONE DI TECNOLOGIE PER I SISTEMI INFORMATIVI

PROGETTAZIONE DI UNA BASE DI DATI CON SISTEMI NOSQL

Anno accademico 2016-2017

SOMMARIO

INTRODUZIONE

L'esigenza di pubblicare un'innumerevole quantità di dati online, dovuta alla nascita dei sistemi cloud e all'evoluzione di internet, ha portato a dover gestire grandi quantità di dati garantendo l'affidabilità e la disponibilità dei servizi. I RDBMS risultano limitati per fronteggiare questo problema. Alcuni di questi limiti furono risolti dalla nascita e dall'utilizzo di una nuova tipologia di database: i NOSQL (not only sql) basati su un data model diverso da quello tabellare fino ad ora utilizzato per i database relazionali. Tra le caratteristiche principali dei DB NoSQL elenchiamo le seguenti:

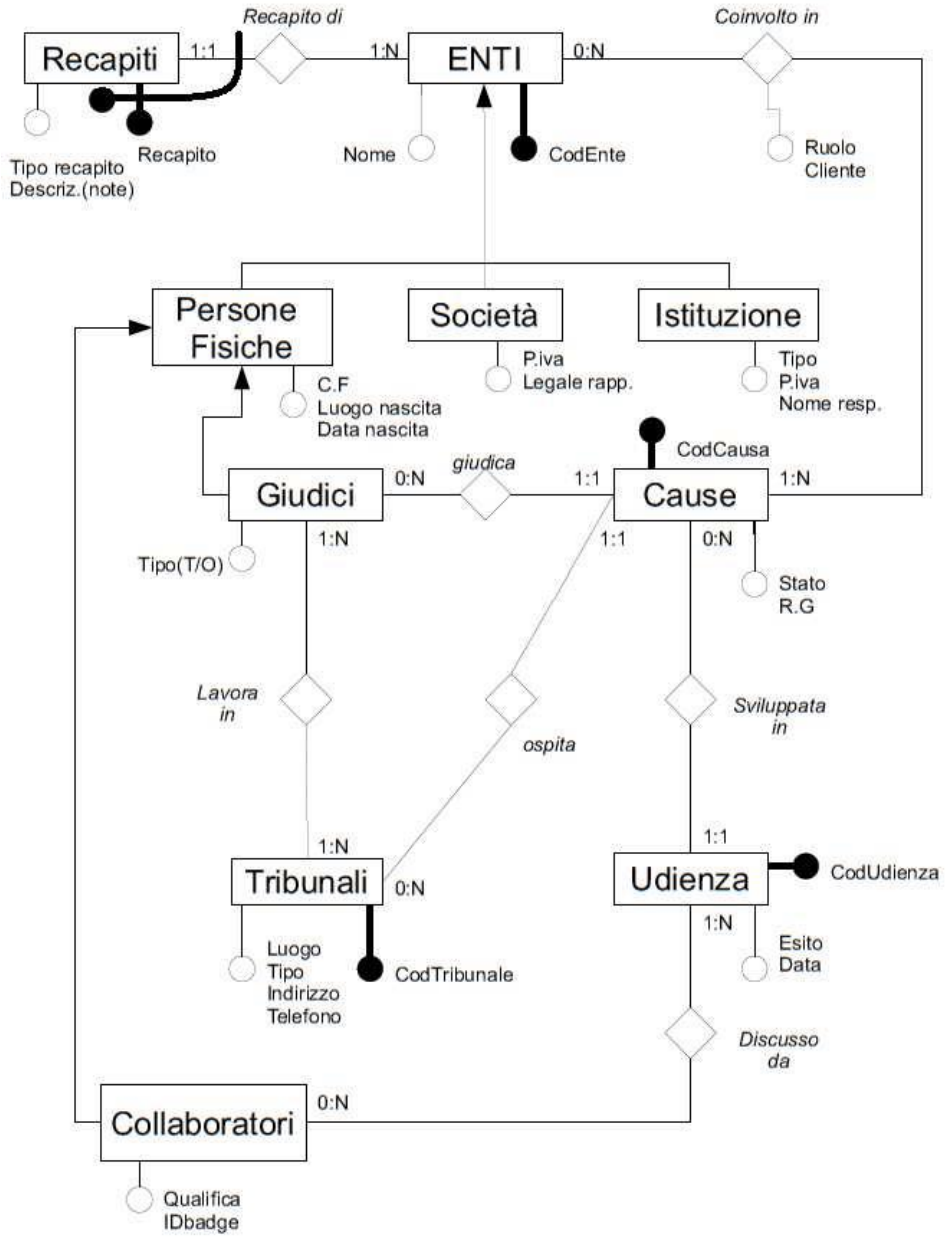
- Non ha uno schema predefinito
- Non utilizza l'SQL come linguaggio per la scrittura di query.

Col passare del tempo, questa famiglia di database, ha dimostrato di avere molti vantaggi rispetto ai RDBMS anche se in alcuni ambiti applicativi si preferisce continuare ad utilizzare i DB tradizionali per motivi elencati più avanti. **In questo elaborato viene illustrata la metodologia impiegata per effettuare una migrazione da un database relazionale in MySQL ad uno NOSQL; verranno poi illustrate le scelte e le metodologie di progettazione adottate per la realizzazione del database NOSQL.** In particolare verrà illustrata un'applicazione pratica della versione NOSQL di un database per la gestione di uno studio legale, basato su una rappresentazione dei dati document oriented e creato a partire dal diagramma E-R.

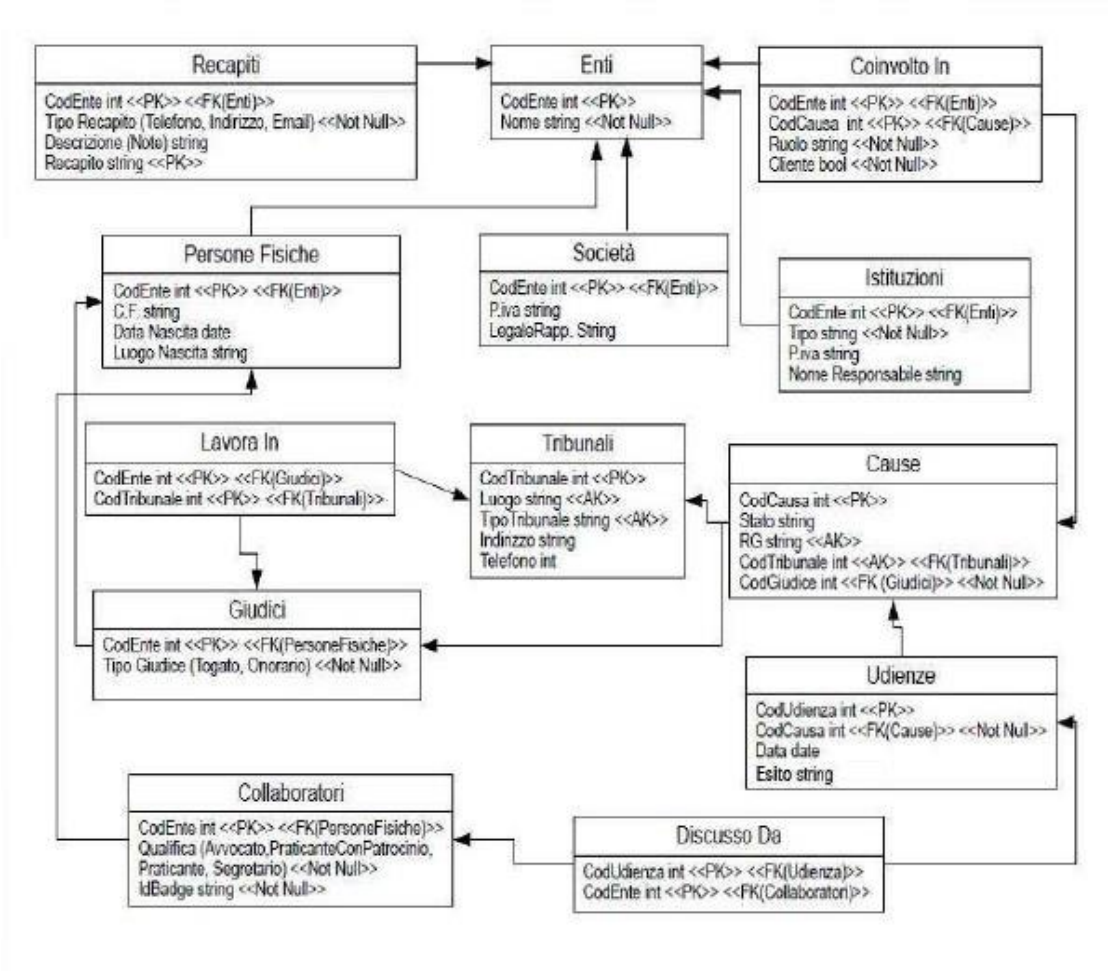
CAPITOLO 1: Il Problema affrontato

Il database relazionale di partenza, utilizzato per l'analisi, è una base di dati per la gestione di uno studio legale. Di seguito sono riportati il diagramma E-R, lo schema logico e le relative operazioni da eseguire sul database (per le specifiche complete del database relazionale vedere appendice 1):

Diagramma E-R:



Schema logico



Query (per visionare l'implementazione in SQL delle seguenti operazioni vedere appendice 1):

1. Dato un determinato ente trovare le cause in cui è coinvolto
2. Dato un determinato ente verificare il numero di udienze a cui ha partecipato in totale o per una determinata causa
3. Data una causa indicare il giudice e quali collaboratori/società sono coinvolti in essa.

4. Trovare le cause ospitate in determinato tribunale
5. Considerato un tribunale determinare una lista dei giudici che vi lavorano
6. Data un'udienza determinare i collaboratori che l'hanno discussa
7. Trovare i recapiti per un determinato ente
8. Dato un giudice trovare una lista di collaboratori che hanno lavorato alle udienze delle cause presiedute da quel giudice
9. Trovare codice causa, RG per quelle cause le cui udienze si sono svolte in un determinato range temporale
10. Dati due enti specifici determinare se vi sono una o più cause in cui sono coinvolti
11. Aggiungere uno o più collaboratori ad un'udienza
12. Aggiornamento di varie entità
13. Inserire una nuova causa che coinvolge determinati enti
14. Inserire nuova udienza per una causa

15. Aggiornare lista di tribunali in cui lavora un determinato giudice

Migrare i dati di un'applicazione da una struttura relazionale ad una struttura NOSQL non è un'operazione banale. Innanzitutto è necessario sottolineare che le tecnologie NOSQL mettono a disposizione diversi modelli di dati; Prima di procedere, pertanto, offriamo una panoramica generale di questi sistemi, mettendo in risalto le differenze con il classico modello relazionale ed illustrando le motivazioni che hanno spinto a scegliere il modello a documenti.

CAPITOLO 2: I Database NOSQL

2.2 RDBMS vs. NoSQL

2.2.1 ACID vs. BASE

La caratteristica principale dei database relazionali è quella di garantire la **consistenza** dei dati mentre, al contrario, i database NoSQL garantiscono **una grande disponibilità di dat**i a discapito della consistenza. I database relazionali si basano sulle proprietà **ACID** mentre i database NoSql sulle proprietà **BASE**.

2.2.2 Proprietà ACID

Una transazione, è caratterizzata da un insieme di azioni che, partendo da un database in uno stato consistente, lo riportano in uno stato finale consistente.

Non sempre le transazioni vanno a buon fine, in questo caso è necessario fare "rollback" della transazione in modo da riportare il DB in uno stato consistente eliminando le modifiche effettuate. L'acronimo ACID indica le proprietà che ogni DBMS deve avere per assicurare la validità e il successo delle transazioni:

• **(A)tomicity - Atomicità:** il Dbms garantisce che la transazione venga eseguita fino alla fine, portando a termine tutti i cambiamenti effettuati, in caso contrario viene annullata;

• **(C)onsistency - Coerenza:** il Dbms garantisce i vincoli di integrità dei dati ovvero la transazione partendo da un database in stato coerente lo riporta in uno stato coerente;

• **(I)solation- Isolamento:** ogni transazione è indipendente dalle altre. Due transazioni distinte non devono interferire tra di loro;

• **(D)urability - Persistenza:** una volta effettuato il commit di una transazione, le modifiche effettuate devono permanere anche in caso di crash del sistema.

2.2.3 Teorema CAP

Il Teorema CAP fu teorizzato per la prima volta da Eric Brewer nel 2000 ed afferma che un sistema distribuito può fornire contemporaneamente solo due fra le tre proprietà descritte di seguito:

• **Consistency - (Consistenza):** tutti i nodi vedono gli stessi dati nello stesso momento;

• **Availability - Disponibilità:** ogni nodo di un sistema distribuito deve rispondere alle richieste a meno che non sia occupato;

• **Partition Tolerance - Tolleranza al Partizionamento:** definisce la capacità del sistema di tollerare l'aggiunta o la rimozione di un nodo nella rete(partizionamento) e la perdita di messaggi.

Secondo il teorema, un sistema distribuito è in grado di soddisfare al massimo due di queste caratteristiche allo stesso tempo, ma non tutte e tre contemporaneamente.

Vi sono, pertanto, tre casi possibili:

1. Sistemi che garantiscono coerenza e disponibilità normalmente non sono tolleranti alla partition tolerance. In pratica: se una componente di questi sistemi non è in linea, l'intero sistema non è in linea;
2. Sistemi che garantiscono tolleranza di partizione e disponibilità non possono garantire la coerenza in lettura, in quanto gli aggiornamenti del valore del dato – che per definizione è il distruttore della coerenza – possono essere effettuati su entrambi i lati della partizione;
3. Sistemi che garantiscono tolleranza di partizione e coerenza non possono garantire la disponibilità perché il sistema che gestisce questa base dati restituirebbe un errore fino a quando lo stato di anomalo partizionamento della propria integrità sarà risolto.

Le proprietà **BASE**, introdotte dall'autore dello stesso Teorema Cap, rappresentano una buona soluzione a questo limite.

Le caratteristiche che un sistema deve garantire partendo dalle considerazioni derivate dal teorema sono:

• **(B)asically (A)vailable**: il sistema deve garantire sempre la **disponibilità dei dati**;

• **(S)oft State**: il sistema può cambiare lo stato nel tempo anche se non sono state effettuate letture o scritture;

• **(E)ventual consistency:** il sistema può diventare consistente nel tempo, anche senza ricevere scritture grazie ai sistemi di recupero della consistenza.

2.1 I Database NoSql: Tassonomia

I database NoSql, come accennato nell'introduzione, vengono classificati sulla base del modello utilizzato per la memorizzazione dei dati. In particolare possiamo individuare 2 categorie differenti contenenti a loro volta modelli differenti:

1. Aggregate-oriented che si suddividono in:

a. Column Family Stores;
b. Document Oriented ;
c. Key Values Stores.

2. Modelli a grafo, detti anche **Graph Databases.**

La prima categoria di database NoSql si basa sul concetto di **"aggregato"** che indica una collezione di oggetti, raggruppati in un'entità, intesa come un insieme unico di singole parti non separabili.

2.1.1 Key Value Stores (Aggregate Oriented)

In questa tipologia di database vengono memorizzate coppie chiave valore e i tipi di dato possono essere sia elementari che avanzati. In Figura 2.1.1 è mostrato un esempio di key-value. Questa base di dati assomiglia molto ad una grande hash-table accessibile tramite la chiave primaria. Generalmente la scelta ricade su questa rappresentazione dei dati quando:

1. Non è possibile definire uno schema dei dati;
2. È necessario accedere velocemente alle informazioni.

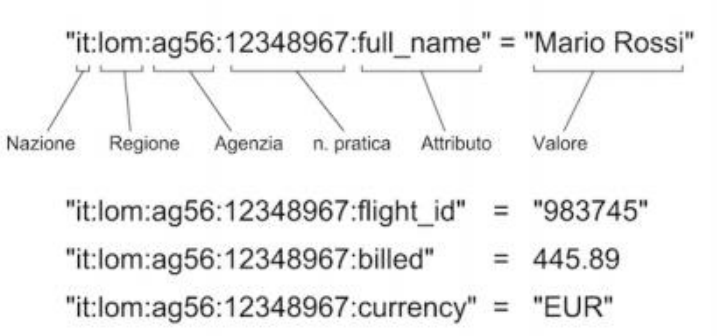


Figura2.1.1: Esempio key-value stores

La parte fondamentale per i database che utilizzano questa rappresentazione è la progettazione della chiave primaria, poiché si basano sull'indirizzamento diretto tipico delle chiavi hash. In molti casi i key-value store permettono la creazione di indici secondari ad esempio tramite B+Tree a discapito delle performance e della consistenza. Per utilizzare in modo performante questo modello è necessario conoscere i tipi di chiave che verranno utilizzate servendosi possibilmente di meccanismi distribuiti per far dipendere la loro generazione da un'autorità centrale.

I key-value store, utilizzati generalmente per salvare le sessioni degli utenti in ambito web, hanno come principale vantaggio **l'altissima disponibilità dei dati** e **altissime performance**

per quel che riguarda l'accesso ai dati, in quanto, l'indirizzamento di tipo hash ha un costo molto basso che va da O (1) a O (log (n)) dove n indica la quantità di dati presenti nel db.

2.1.2 Column Family Stores (Aggregate Oriented)

I Column Family Stores sono detti anche database Column Oriented ovvero "orientati alle colonne". Questo modello viene descritto come una mappa sparsa, distribuita, multidimensionale e persistente. In questo modello i dati si basano sempre sulla rappresentazione delle hash-table ma, diversamente dal modello key value, i dati possono arrivare a due o più livelli di indicizzazione. Ogni column-family contiene un insieme di dati che caratterizzano lo stesso tipo di informazione. Come si può vedere dall' esempio di Figura 2.1.2 , ogni unità informativa(UI) può avere un insieme diverso di colonne in quanto possono essere aggiunte quelle necessarie mentre vengono tolte quelle con valori a NULL, ottimizzando dunque la gestione della memoria, diversamente da quanto avviene nei database relazionali.

Figura 2.1.2: Esempio column family

Fondamentale per l'organizzazione e il partizionamento dei dati è il raggruppamento delle colonne in famiglie di colonne. Esse devono essere quasi sempre predefinite mentre le colonne e le righe possono essere aggiunte a runtime in maniera molto flessibile. Per questo motivo, il modello appena introdotto è **meno flessibile** dei **key-value stores** e dei **document stores**. I column family hanno una rappresentazione dei dati molto simile ai database relazionali, la differenza principale tra i due è la gestione dei valori NULL.

Se consideriamo un database relazionale, con diversi tipi di attributi, inserirà un NULL per ogni colonna dell'insieme priva di valori. Nel caso invece dei column family, per un dato con attributi vuoti verrà memorizzata solo una coppia chiave-valore in una riga. In Figura 2.1.3 possiamo notare la facilità con cui può essere aggiunta ed eliminata una column-family, cosa che nei DB relazionali avrebbe richiesto la ridefinizione dello schema con la conseguente perdita di tempo e spreco di memoria allocata.

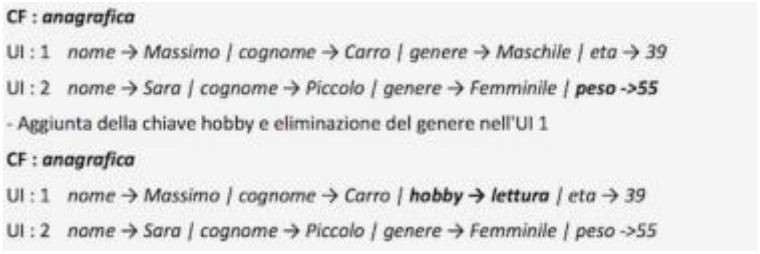


Figura 2.1.3: Esempio column family con modifica di un elemento

Questo modello di dati, grazie alla flessibilità e al partizionamento orizzontale, è molto utile nei casi in cui bisogna gestire **grandi quantità di dati con tanti attributi diversi**.

2.1.3 Document Stores (Aggregate Oriented)

In questa categoria di database, i dati vengono archiviati in documenti in stile JSON (javascript object notation) o XML. Anche per questo data model esiste il concetto di "aggregato".

In questo caso la struttura d'aggregazione può essere suddivisa in più livelli gerarchici e i dati possono essere raggruppati in **"collection"** di vari tipi. Gli indici che identificano il singolo documento o attributo, vengono creati tramite B+Tree esattamente come accade nei database relazionali il che porta ad avere gli stessi vantaggi: gli indici su più attributi e ordinamento e ricerche su più intervalli.

Alcune delle caratteristiche principali dei **Document Stores** sono**:**

• la semplicità della sintassi;

• un numero di tipi molto ristretto;

• array rappresentati esplicitamente come liste ordinate;

• gerarchia esplicita;

• possibilità di aggiungere liberamente nuovi elementi.

Nei DB così strutturati è possibile effettuare ricerche oltre che sulle chiavi anche sui valori legati ad esse. Il vantaggio principale è dunque **la flessibilità e l'elevata performance nelle interrogazioni**, per questi motivi è spesso utilizzato nei siti web. Ogni documento all'interno di una collection ha una **key- id** univoca che identifica il documento. I DB più famosi che utilizzano questo modello di dati sono: CouchDB e MongoDB, utilizzato per il caso applicativo presentato in questo elaborato.

```
{
  _id: "mary",
  username: "mary",
  firstName: "Mary",
  lastName: "Wilson",
  games:
   [ { game: "Game:2345", opponent: "Player:rick"},
     { game: "Game:2611", opponent: "Player:ann"} ]
}
```

Figura 2.1.3: Rappresentazione di un documento in json

2.1.4 Graph Model (Graph Databases)

I tipi di Data Model descritti nei paragrafi precedenti si focalizzano sulla scalabilità dei dati e sulla flessibilità delle loro informazioni ma non sono molto adatti a gestire dati che hanno dei legami stretti tra loro. Questa problematica viene invece risolta dai Database Graph-Oriented che come dice il termine stesso si basano sulla struttura di dati a "grafo" e sono quindi composti da nodi collegati tra loro. Essi consentono le interrogazioni attraverso i cammini individuabili nel grafo. Considerando il numero di archi in uscita da ogni vertice che chiameremo M, il tempo che occorre per analizzare tutti questi archi è T= O(M). Quando viene utilizzata questo data model, la quantità totale di dati del database non influisce più sulle performance locali quindi diventa semplice gestire anche Graph Database di grandi dimensioni.

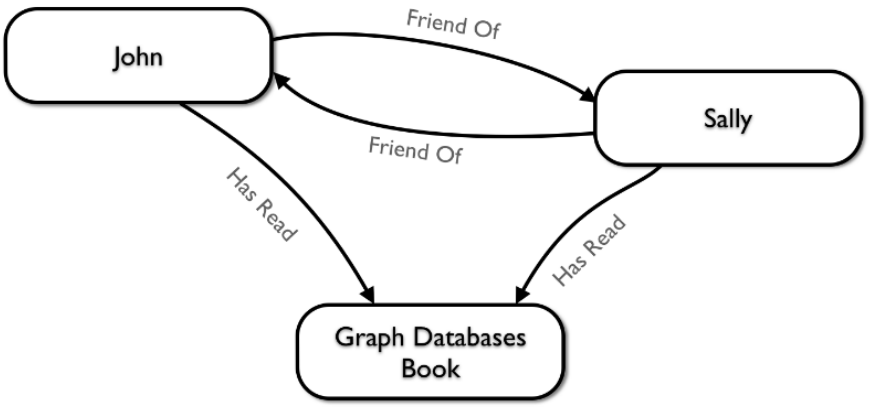


Figura 2.1.4: Esempio Graph Model

2.1.5 La nostra scelta

Osservando il diagramma E-R e quello logico del nostro database di partenza notiamo che i dati disposti nei vari oggetti sono fortemente collegati tra loro; il database è normalizzato in

quanto non presenta informazioni duplicate e l'aggiornamento dei dati avviene senza perdita d'integrità.

Per poter effettuare le interrogazioni elencate nel Capitolo 1 è necessario fare ampio uso del JOIN che rischia di non essere l'operatore ottimale: trattasi, infatti, dell'elemento più oneroso in termini di risorse del modello relazionale.

Dall'analisi che ha visto come oggetto principale il confronto tra vantaggi e svantaggi dei data model precedentemente elencati, si è scelto di utilizzare il **document oriented**.

Il motivo di questa scelta è dovuto alla possibilità, da esso offerta, di strutturare i dati in collezioni e documenti, al fine di massimizzare il contenuto informativo di ogni aggregato.

L' obiettivo, naturalmente, è fare in modo che ogni operazione coinvolga idealmente un solo aggregato offrendo un set di dati unico e completo per ogni operazione.

Ulteriori vantaggi:

- Rappresentazione di relazioni gerarchiche con un singolo record;
- **Modellazione flessibile dei dati:** non si basa su schemi predefiniti, le chiavi ed i valori di un documento non hanno dimensioni predefinite, il che rende la rimozione e l'aggiunta di campi molto semplice;
- **Prestazioni veloci delle query**: molti database di documenti dispongono di potenti motori e funzionalità di indicizzazione che consento di effettuare query rapide ed efficienti.
- **Prestazioni di scrittura veloci**: anche se a discapito della rigida coerenza dei dati.

CAPITOLO 3: La scelta degli aggregati

La filosofia che sta alla base del modello a documenti prevede che ogni documento aggregato raccolga **tutti i dati associati a un'entità**, in modo che qualsiasi applicazione possa trattare l'entità come oggetto e valutare in un sol colpo tutte le informazioni a essa correlate. In questo modo, si evitano i "fardelli" computazionali dovuti ai passaggi di aggregazione delle informazioni tipici del linguaggio SQL, in quanto tutti i dati necessari e corrispondenti a un medesimo oggetto sono già disponibili in un unico documento.

Risulta evidente, pertanto, che la scelta degli aggregati viene svolta in base alle operazioni che devono essere eseguite su di essi.

Le query sono le stesse definite per il database MySQL ed elencate nel Capitolo 1.

In base alle suddette operazioni è stata implementata la seguente scelta:

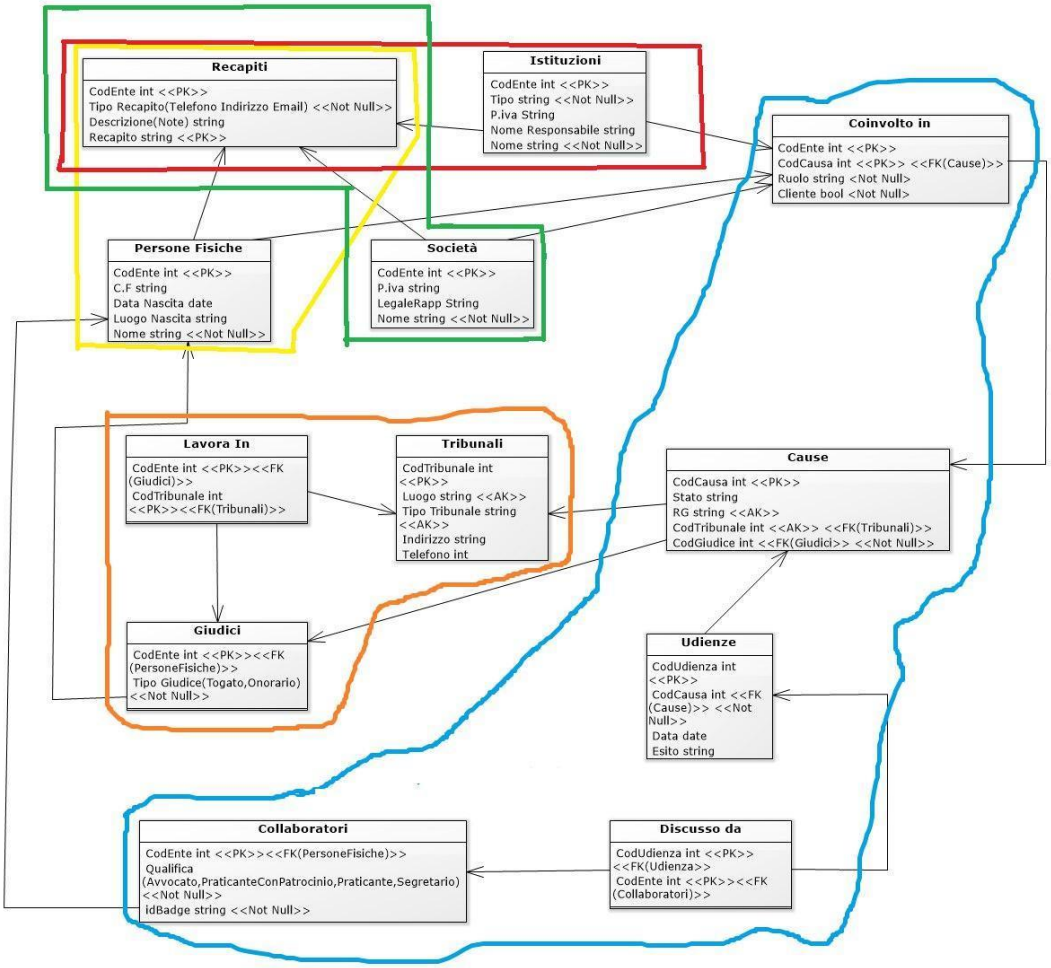


Figura 2.1.4: Scelta aggregati

Operazioni 1,2,3,4,6,8,9,10,11,12,13,14

Operazioni 3.2,5,12,13,15

Operazioni 7,12,13 -> aggregati in alto a sinistra

In **Figura 2.1.4** si individuano i seguenti aggregati:

- **"Causa"** contiene tutte le informazioni su entità coinvolte, tribunali, giudici e udienze raccolte all'interno di ulteriori oggetti innestati; Vi sono informazioni persino sui collaboratori rappresentati come oggetti innestati di udienze.
- **"Persone fisiche"** contiene i vari recapiti per ogni entità
- **"Società"** contiene i vari recapiti per ogni entità
- **"Istituzioni"** contiene i vari recapiti per ogni entità

Per quanto riguarda l'aggregato centrale ("giudici", "lavorain","tribunali") è stato deciso di creare due documenti separati (il perchè viene spiegato nel capitolo 3.1):

- **"Giudici"** in cui vengono replicate le informazioni relative ai tribunali nei quali lavorano
- **"Tribunali"** in cui vengono replicate le informazioni dei giudici che vi lavorano (codice, nome, cognome)

L'aggregazione dei dati risulta vantaggiosa in termini di efficienza, in quanto con il recupero di un documento avremo implicitamente il recupero delle informazioni relative necessarie per rispondere alle varie operazioni. Tuttavia, ciò potrebbe comportare la presenza di oggetti simili innestati in documenti diversi, con conseguente ridondanza dei dati, inconveniente che i database relazionali non presentano.

3.1 Il problema della Ridondanza

Consideriamo le operazioni di inserimento e aggiornamento (11,...,15).

Ora prendiamo in esame un'entità, persona fisica, con tutti i suoi valori ed i suoi recapiti.

Al paragrafo precedente abbiamo effettuato la scelta di integrare le informazioni delle entità nell'aggregato causa.

A questo punto immaginiamo di dover cambiare o aggiungere un recapito ad una entità persona. Questo porterebbe a scorrere tutte gli aggregati di causa e aggiornare tutte le varie occorrenze di "persona" all'interno degli aggregati, operazione che potrebbe risultare onerosa.

In alternativa si potrebbe considerare l'ipotesi di aggiornare solo l'entità principale, ma ciò genererebbe problemi di consistenza in quanto le varie occorrenze non sarebbero aggiornate. **La soluzione adottata è stata dunque quella di effettuare una ridondanza parziale dell'informazione, andando a duplicare solo le informazioni strettamente necessarie allo svolgimento di ogni operazione e, soprattutto, con una bassa probabilità di variazione nel tempo.**

Si è scelto, ad esempio, di replicare valori come nome, cognome o codice fiscale che si suppone rimangano fissi nel tempo.

L'idea, infatti, di effettuare una ulteriore query di dettaglio qualora ce ne fosse bisogno sembra più conveniente del dover aggiornare tutte le repliche in caso di update.

CAPITOLO 4: Implementazione in Mongo

4.1 Le scelte di progettazione

Tra i database esistenti per la realizzazione di un data-model document oriented quello scelto per la realizzazione del NoSQL è **MongoDB** poiché:

- Memorizza i dati in documenti flessibili di tipo JSON, il che significa che i campi possono variare da documento a documento e la struttura dei dati può essere modificata nel tempo;

- Il modello di documento è associato agli oggetti nel codice dell'applicazione, rendendo i dati facili da utilizzare;

- L'indicizzazione e l'aggregazione in tempo reale offrono potenti strumenti per accedere e analizzare i dati;

- È un database distribuito al suo interno, quindi disponibilità elevata, ridimensionamento orizzontale e distribuzione geografica sono integrati e facili da usare;

- È open-source e facile da usare.

4.2 Mongo Overview

MongoDB è un database NoSQL orientato ai documenti, basato sul formato JSON/BSON per la memorizzazione e la rappresentazione dei dati.

I databases MongoDB risiedono in un server che può ospitare uno o più databases indipendenti tra di loro. I gruppi di documenti che contengono i dati sono chiamati **'collection'**. Una delle caratteristiche fondamentali che distingue MongoDB dai classici db relazionali è l'approccio *schemaless*, ovvero l'assenza di uno schema preciso, cosa che rende la struttura di questo DB molto flessibile; infatti, non è necessario come nei DB relazionali definire i campi a priori e rispettare poi con rigidità le caratteristiche dei vari tipi di dato, anzi nel caso di MongoDB gli attributi di un documento possono essere modificati in maniera dinamica senza modificare le proprietà dell'intera collezione.

MongoDB ha solo una chiave obbligatoria: id che corrisponde alla chiave primaria dei DB relazionali e serve per identificare in maniera univoca un documento. La suddivisione dei documenti in collezioni rende la ricerca molto più veloce.

Per la creazione di una nuova collezione è necessario selezionare il DB in cui vogliamo crearla con il comando: use (nomeDB) dopodiché basta scrivere l'istruzione

db.createCollection(nomeCollezione)

Per eliminare una collezione il comando da invocare è *drop()*, in questo modo vengono eliminati in automatico anche tutti i documenti in essa contenuti.
I dati in MongoDB vengono memorizzati come documenti in formato BSON dove ognuno di essi è un insieme di chiavi, ad ognuna delle quali viene associato un valore.
Per aggiungere un documento ad una collezione di MongoDB viene utilizzata la funzione insert:

db.nomecollection.insert(<documento o array di documenti>)

Una volta che un documento è stato inserito, può essere cercato con il comando find

db.nomecollection.find(<documento o array di documenti>)

Per l'aggiornamento le istruzioni utilizzate sono invece:

db.collection.update() oppure *db.collection.findAndModify()*

Per eliminare un documento di una collezione viene chiamata la funzione remove

db.collection.remove()

che, chiamata senza passare parametri, elimina tutti gli elementi della collezione.

4.3 Implementazione da MySQL a MongoDB

Per l'importazione dei dati in Mongo per prima cosa è stato necessario definire la struttura delle collection che appartengono al database.

Tale scelta è stata fatta sulla base degli aggregati definiti al capitolo 3.

Sono stati definite 7 collection la cui struttura generale viene presentata di seguito (viene presentato un esempio per ogni collection tratto dal db finale):

PERSONE FISICHE

```
{
        "_id" : ObjectId("5a14981492dd002d3d37f25c"),
        "CodEnte" : "90",
        "Nome" : "Rashad",
        "Cognome" : "Henson",
        "CodiceFiscale" : "1682082074399",
        "DataNascita" : "2009-09-03",
        "LuogoNascita" : "Cuenca",
        "recapiti" : [
                {
                        "TipoRecapito" : "Telefono",
                        "Descrizione" : "contatto del responsabile",
                        "Recapito" : "1-128-965-6128"
                }
        ]
}
```

SOCIETÀ

```
{
        "_id" : ObjectId("5a14983392dd002d3d37f776"),
        "CodEnte" : "40",
        "Nome" : "Phasellus LLP",
        "PartitaIVA" : "948859-9334",
        "LegaleRappresentante" : "Zachery Forbes",
        "recapiti" : [
                {
                        "TipoRecapito" : "Telefono",
                        "Descrizione" : "contatto del responsabile",
                        "Recapito" : "1-130-439-2625"
                }
```

```
13.             ]
14.     }
```

ISTITUZIONI


```
{
        "_id" : ObjectId("5a1497f092dd002d3d37efff"),
        "CodEnte" : "180",
        "Nome" : "Arcu Imperdiet Limited",
        "TipoIstituzione" : "Comune",
        "PartitaIVA" : "%B214978514",
        "NomeResponsabile" : "Allistair Spears",
        "recapiti" : [
                {
                        "TipoRecapito" : "Email",
                        "Descrizione" : "contatto del responsabile",
                        "Recapito" : "Phasellus@aliquetProinvelit.net"
                }
        ]
    }
```


TRIBUNALI


```
{
        "_id" : ObjectId("5a14984e92dd002d3d37fa69"),
        "CodTribunale" : "20",
        "Luogo" : "Ottawa",
        "TipoTribunale" : "Giudiziario",
        "Indirizzo" : "P.O. Box 109, 7443 Neque. Rd.",
        "Telefono" : "1-543-648-7947",
        "giudici" : [
                {
                        "CodEnte" : "71",
                        "Nome" : "Brennan",
                        "Cognome" : "Gillespie"
                },
                {
```

```
                        "CodEnte" : "359",
                        "Nome" : "Felix",
                        "Cognome" : "Kirkland"
                    },
                    {
                        "CodEnte" : "362",
                        "Nome" : "Sylvester",
                        "Cognome" : "Cabrera"
                    }
            ]
    }
```

GIUDICI


```
{
        "_id" : ObjectId("5a1497cf92dd002d3d37eef0"),
        "CodEnte" : "90",
        "Nome" : "Rashad",
        "Cognome" : "Henson",
        "CodiceFiscale" : "1682082074399",
        "DataNascita" : "2009-09-03",
        "LuogoNascita" : "Cuenca",
        "TipoGiudice" : "Togato",
        "recapiti" : [
                {
                        "TipoRecapito" : "Telefono",
                        "Descrizione" : "contatto del responsabile",
                        "Recapito" : "1-128-965-6128"
                }
        ],
        "tribunali in cui lavora" : [
                {
                        "CodTribunale" : "3",
                        "Luogo" : "Cardiff"
                },
                {
                        "CodTribunale" : "64",
                        "Luogo" : "Schiltigheim"
                },
                {
                        "CodTribunale" : "72",
                        "Luogo" : "Kitchener"
                },
                {
                        "CodTribunale" : "99",
                        "Luogo" : "Lauterach"
```

```
                }
            ]
    }
```

COLLABORATORI

```
{
            "_id" : ObjectId("5a1497b292dd002d3d37ed54"),
            "CodEnte" : "110",
            "Nome" : "Yuli",
            "Cognome" : "Norman",
            "CodiceFiscale" : "1665010684899",
            "DataNascita" : "2013-09-30",
            "LuogoNascita" : "San Lazzaro di Savena",
            "Qualifica" : "Segretario",
            "IdBadge" : "QQM10YGW9SC",
            "recapiti" : [
                    {
                            "TipoRecapito" : "Email",
                            "Descrizione" : "contatto del responsabile",
                            "Recapito" : "vitae.aliquet.nec@ac.edu"
                    }
            ]
    }
```

CAUSE

```
 {
            "_id" : ObjectId("5a1342af9092c71ed82591d7"),
            "CodCausa" : "20",
            "Stato" : "InCorso",
            "RG" : "BZC21NNS2UH",
            "CodTribunale" : "29",
            "CodGiudice" : "80",
            "tribunale" : [
                    {
                            "CodTribunale" : "29",
                            "luogoTribunale" : "Valleyview",
                            "TipoTribunale" : "Giudiziario"
                    }
            ],
            "giudice" : [
```

```
                {
                        "CodEnte" : "80",
                        "Nome" : "Zahir",
                        "Cognome" : "Gillespie"
                }
        ],
        "udienze" : [
                {
                        "CodUdienza" : "331",
                        "La_Data" : "2004-02-05",
                        "Esito" : "Sospesa",
                        "Collaboratori" : [
                                {
                                        "CodCollaboratore" : "120",
                                        "nomeCollaboratore" : "Carter",
                                        "cognomeCollaboratore" : "Smith"
                                }
                        ]
                },
                {
                        "CodUdienza" : "607",
                        "La_Data" : "1994-02-03",
                        "Esito" : "Rinviata",
                        "Collaboratori" : [
                                {
                                        "CodCollaboratore" : "108",
                                        "nomeCollaboratore" : "Hayden",
                                        "cognomeCollaboratore" : "Harrell"
                                }
                        ]
                },
                {
                        "CodUdienza" : "1065",
                        "La_Data" : "2004-02-05",
                        "Esito" : "Terminato",
                        "Collaboratori" : [ ]
                }
        ],
        "entita coinvolte" : [
                {
                        "CodEnte" : "163",
                        "nome" : "Melvin",
                        "cognome" : "Mccullough",
                        "Cliente" : "1"
                },
                {
                        "CodEnte" : "1010",
                        "nome" : "Deanna",
                        "cognome" : "Horn",
                        "Cliente" : "1"
                },
                [
```

```
                                {
                                        "CodEnte" : "25",
                                        "nome" : "Mauris PC",
                                        "Cliente" : "0"
                                }
                        ]
                ]
```

Per la migrazione dei dati dal database in MySQL a quello in Mongo si è deciso di procedere con la seguente metodologia in 2 passi:

1. Esportare i dati in formato JSON (secondo le strutture appena definite) dal DBMS[1]

2. Importare i dati in Mongo utilizzando mongoimport[2]

4.2 Query in MongoDB

Le query definite per il database relazionale possono essere eseguite nel nostro database mongo digitando gli statement riportati di seguito.

Si può notare come ogni query faccia uso di **una sola collection**:

1. Dato un determinato ente trovare le cause in cui è coinvolto
```
db.cause.find({"entitacoinvolte.CodEnte":"70"},
{_id:0,CodCausa:1,Stato:1,RG:1,"entita coinvolte.nome":1})
```

2. Data una causa indicare il giudice e quali collaboratori/società sono coinvolti in essa.
```
db.cause.find({"udienze.Collaboratori.CodCollaboratore":"95"}).count()
```

3. Dato un determinato ente verificare il numero di udienze a cui ha partecipato in totale o per una determinata causa
```
db.cause.find({"CodCausa":"9"},
{_id:0,"giudice.Nome":1,"giudice.Cognome":1,"entita coinvolte.nome":1,
"entita coinvolte.cognome":1})
```
4. Trovare le cause ospitate in determinato tribunale
```
db.cause.find({"tribunale.luogoTribunale":"Valleyview"},
```

1 gli script in php per l'esportazione in formato json sono riportati in appendice 3

2 Mongoimport è un programma che permette di importare dati in formato JSON o CSV

```
{_id:0,CodCausa:1,Stato:1,RG:1,"tribunale.luogoTribunale":1})
```

5. Considerato un tribunale determinare una lista dei giudici che vi lavorano

```
db.tribunali.find({"Luogo":"Cardiff"},
{_id:0,"giudici.Nome":1,"giudici.Cognome":1})
```

6. Data un'udienza determinare i collaboratori che l'hanno discussa

```
db.cause.find({"udienze.CodUdienza":"1090"},
{_id:0, "udienze.CodUdienza":1,"udienze.Collaboratori.nomeCollaboratore":1,"
udienze.Collaboratori.cognomeCollaboratore":1})
```

7. Trovare i recapiti per un determinato ente

```
db.giudici.find({"CodEnte":"70"},
{_id:0,"giudici.Nome":1,"giudici.Cognome":1,"recapito.TipoRecapito":1,
"recapito.Descrizione":1})
```

8. Dato un giudice trovare una lista di collaboratori che hanno lavorato alle udienze delle cause presiedute da quel giudice

```
db.cause.find({"giudice.CodEnte":"71"},
{_id:0,"udienze.Collaboratori.nomeCollaboratore":1,
"udienze.Collaboratori.cognomeCollaboratore":1})
```

9. Trovare codice causa, rg per quelle cause le cui udienze si sono svolte in un determinato range temporale

```
db.cause.find({"udienze.La_Data" :{$gt:"1998-12-31",$lt:"2003-12-31"}},
{_id:0,CodCausa:1,RG:1})
```

10. Dati due enti specifici determinare se vi sono una o più cause in cui sono coinvolti

```
db.cause.find( { $and: [ { "entita coinvolte":"161" },
{"entita coinvolte":"50"} ] } ).count()
```

11. Aggiungere uno o più collaboratori ad un'udienza

```
db.cause.update({"udienze.CodUdienza":"447"},
        {$push : {Collaboratori :{CodCollaboratore:"97",
        nomeCollaboratore:"Nicholas"cognomeCollaboratore:"Cook"}}})
```

12. Aggiornamento di varie entità

```
db.societa.update({ "CodEnte":"22"},{"PartitaIVA": "534555"})
```

13. Inserire una nuova causa che coinvolge determinati enti

```
db.cause.insert([{"CodCausa":"1000","Stato":"Sospesa","RG":"FIS71AZU8GY",
```

```
        "CodTribunale":"71","CodGiudice":"71",
"tribunale":[{"CodTribunale":"71","luogoTribunale":"Manisa",
            "TipoTribunale":"Civile"}],
"giudice":[{"CodEnte":"71","Nome":"Brennan",
            "Cognome":"Gillespie"}],
"udienze":[{"CodUdienza":"278","La_Data":"1994-02-03",
            "Esito":"Terminato",
            "Collaboratori":[{"CodCollaboratore":"98","
                        nomeCollaboratore":"Hilel",
                        "cognomeCollaboratore":"Goodman"}
                ]},
            {"CodUdienza":"447","La_Data":"1992-04-06",
            "Esito":"Rinviata",
            "Collaboratori":[{"CodCollaboratore":"114",
                        "nomeCollaboratore":"Elijah",
                        "cognomeCollaboratore":"Prince"}]
                }]]})
```

14. Inserire nuova udienza per una causa

```
db.cause.update({"CodCausa":"1000"},
{$push :{udienze: { CodUdienza:"1000",La_Data:"2017-11-
                    17",Esito:"Rinviata"} } }  )
```

15. Aggiornare lista di tribunali in cui lavora un determinato giudice

```
db.giudici.update( { "CodEnte": "71" },
{ "CodEnte":"71", "tribunali in cui lavora":{ CodTribunale: "12",
                                Luogo: "Court-Saint-Etienne" } } )
```

4.3 MongoDB vs MySQL

Analizzando il database implementato in mongo possiamo notare alcuni vantaggi (e svantaggi):

- **Leggerezza computazionale**: il database in mongo non prevede operazioni di aggregazione sui dati, in quanto tutte le informazioni sono già raccolte in un unico documento associato all'oggetto da trattare. Nel db in mysql la complessità dell'operatore di join, e quindi il peso computazionale, cresce con l'ingigantirsi della base di dati, del numero di tabelle e delle informazioni da trattare. Il NoSQL, invece, non ha limiti di dimensioni in questo senso. Così si ottengono migliori prestazioni e performance anche in ambienti di Big Data. Può essere necessario, tuttavia, la duplicazione delle informazioni in collection diverse con problemi di integrità in caso di aggiornamento.

- **Assenza di schema**: i database NoSQL sono privi di schemi in quanto il documento JSON contiene tutti i campi utili, senza necessità di definizione. In questo modo, le applicazioni possono essere arricchite di nuovi dati e informazioni, definibili liberamente all'interno dei documenti JSON. I database non relazionali, a differenza di quelli SQL, si rivelano quindi adatti ad inglobare velocemente nuovi tipi di dati e a conservare dati semi-strutturati o non strutturati. Lo svantaggio è che non essendoci dei controlli formali sull'integrità dei dati, il compito è svolto totalmente dall'applicativo che dialoga con il database.

- **Scalabilità orizzontale garantita**: l'aggregazione dei dati e l'assenza a priori di uno schema definito offrono l'opportunità di scalare orizzontalmente i database NoSQL senza difficoltà e senza rischi operativi.

CAPITOLO 5: Analisi dei Risultati

Uno degli obiettivi principali, che ci siamo posti per la realizzazione del progetto riportato nel seguente elaborato è il confronto dei due database dal punto di vista della velocità d'esecuzione delle query.

A tale proposito per ogni query MySQL, eseguita da riga di comando è stato calcolato il tempo d'esecuzione e confrontato con la rispettiva query eseguita in MongoDB.

Per le query in MySQL il tempo è stato calcolato lanciando di volta in volta uno script PHP da riga di comando, mentre per MongoDB è stato sufficiente inserire un'istruzione in più per ogni query.

In particolare per le query di update e find sui documenti, lo statement utilizzato è explain("executionStats") dunque avremo:

db.<nome collection>.explain("executeStats").update()

db.<nome collection>.explain("executeStats").find()

Siccome il comando precedentemente descritto non è supportato per le query di inserimento in tal caso,per il calcolo dei tempi d'esecuzione, lo statement eseguito dopo l'esecuzione della query è:

db.system.profile.find({ ns: 'nomedb.collection'}).pretty()

Di seguito, sono riportati in una tabella i risultati relativi ai tempi d'esecuzione in Mongo e in MYSQL di ognuna delle query elencate nel capitolo 1,

Numero Query	Tempi d'esecuzione in MYSQL	Tempi d'esecuzione in MongoDB
1	150ms	91 ms
2	338ms	155 ms
3	142ms	42 ms
4	1.2ms	13 ms
5	32ms	2 ms
6	175ms	14 ms
7	20ms	17 ms
8	0.4ms	12 ms
9	14ms	61 ms
10	1.3 ms	7 ms
11	150 ms	11 ms
12	73 ms	35 ms
13	113 ms	30 ms
14	141 ms	2 ms
15	0.5 ms	2 ms

Come è possibile notare, eccetto rare eccezioni le query calcolate su Mongo hanno dei tempi d'esecuzione più rapidi rispetto a quelli MYSQL. Possiamo notare però che i casi in cui si ottengono tempi migliori per MYSQL sono in corripondenza di query su tabelle con un numero di dati nettamente inferiore di quelli contenuti nell'aggregato corrispondente. Inoltre, a causa della dimensione ridotta del database, (massimo 5.000 record per la tabella cause ma 150 per tribunali o giudici) le query sul RDBMS vengono eseguite in RAM e non sul disco fisso, rendendo l'esecuzione più veloce.

Un fattore che invece sicuramente rende più onerose le query, peggiorando dunque i tempi d'esecuzione in MYSQL è l'operatore di JOIN, che invece, come spiegato nel corso di questo elaborato, in NOSQL non è adoperato anche perché gli aggregati sono costruiti in modo da poter soddisfare al meglio le query.

Per quanto riguarda le query di inserimento invece (11,13,14) i tempi di elaborazione sono molto differenti tra i due DB. I motivi principali di questa differenza sono:

1. MySQL durante l'inserimento dei dati ha l'obbligo di controllare la correttezza e la validità dei NOT NULL, delle PRIMARY KEY, delle FOREIGN KEY e del vincolo UNIQUE definiti per le tabelle. Questi controlli aumentano i tempi di risposta.
2. MySQL supporta le transazioni ACID enunciate nel capitolo 2 di questo elaborato che, tramite la tecnica del *doublewrite*, garantiscono la consistenza dei dati anche nel caso in cui le transazioni non andassero a buon fine a discapito della velocità d'esecuzione.

La tecnica del ***doublewrite*** si basa su una doppia scrittura, ovvero, i dati vengono prima scritti su un doublewrite buffer e solo se questa scrittura va a buon fine vengono scritti sul file di dati. MongoDb, al contrario, si basa sulla tecnica ***fire and forget***, tutte le operazioni di scrittura vengono inviate al database senza la necessità che ci sia una risposta da parte del DB, velocizzando le operazioni di scrittura rispetto a MySQL.

Per effettuare un test di maggiore consistenza, si è deciso di inserire una quantità maggiore di record e documenti nei rispettivi database. L'inserimento dei dati è stato effettuato con 10.000 valori generati in maniera random per la tabella cause. Dall'inserimento dei record nella tabella e nella collection relativa si ottengono dei risultati che avvalorano ancora di più l'ultima tesi in quanto si ottiene 1470 ms per MySQL e 150 ms per MongoDB

CAPITOLO 6: Conclusioni

Il cuore di questo elaborato è rappresentato dalla metodologia e dalle scelte di progettazione adottate per la realizzazione del database NOSQL e il confronto con l'RDBMS di partenza, soprattutto dal punto di vista dell'esecuzione delle query. Dai risultati ottenuti si evince che MongoDB ha prestazioni migliori rispetto a MySQL soprattutto per grandi quantità di dati, grazie all'assenza di vincoli e uno schema fisso. La flessibilità di MongoDB è un fattore importante nelle applicazioni moderne in cui l'esigenza di avere tempi di risposta brevi è maggiore della sicurezza e consistenza dei dati. Un'altra caratteristica è la tecnica dello *"sharding"*. Essa garantisce la scalabilità orizzontale suddividendo i dati in modo da memorizzarli in macchine diverse distribuendo il carico di lavoro ed evitando l'impiego di macchine più potenti. Nonostante ciò, quando si ha a che fare con operazioni e dati che richiedono consistenza e sicurezza è preferibile utilizzare MySQL (e.g. database di una banca).

Appendice 1: Requisiti database mysql

Si vuole realizzare una base di dati che modelli alcune classi coinvolte nella gestione di uno studio legale, non destinate ai clienti. L'entità principale che si vuole modellare sono le cause. Di ognuna di esse interessa il numero di Registro Generale (RG), che identifica la causa all'interno di ogni tribunale, lo stato attuale, gli enti coinvolti come parti nella causa, il loro ruolo nella causa e se siano o meno clienti dello studio in relazione a quella causa; inoltre, altri elementi d'interesse sono il giudice e il tribunale incaricati della causa e le (zero o più) udienze in cui è stata discussa.

Una causa, per come è stata definita, può essere affidata solo ad un giudice che lavora presso il tribunale dove essa si svolge. Per ogni ente l'attributo d'interesse comune è il nome. Essi si suddividono in persone fisiche, società o istituzioni; delle prime gli attributi da considerare sono codice fiscale, data e luogo di nascita; delle seconde partita IVA e nome del legale rappresentante; delle ultime tipo di istituzione (comune, provincia e così via) e nome del responsabile. Inoltre, ad ogni ente sono associati uno o più recapiti.

Ogni recapito è caratterizzato da una descrizione(Nota) che può essere un numero di telefono, un indirizzo fisico o di posta elettronica. Per i giudici, oltre agli attributi relativi alle persone fisiche, gli elementi di interesse sono il tipo (togato o onorario) e i tribunali in cui lavorano (un giudice lavora in uno o più tribunali e un tribunale ha uno o più giudici). Un giudice togato non può lavorare presso un ufficio del Giudice di Pace. Per i tribunali sono rilevanti il tipo, il luogo, l'indirizzo in cui risiede e il numero di telefono. In ogni luogo c'è al più un tribunale di ogni tipo. Delle udienze ci interessano la data e l'esito, oltre ai collaboratori (almeno uno) che le hanno discusse. Per i collaboratori sono rilevanti, oltre agli attributi relativi alle persone fisiche, la qualifica (Avvocato, Praticante con patrocinio, Praticante senza patrocinio e Segretario) e ad ognuno di essi è inoltre associato un IdBadge per l'accesso al sistema. Le operazioni tipiche sono creazione, aggiornamento ed eliminazione delle entità sopra nominate. Di seguito sono riportare le query per il db in questione in MYSQL

1) Dato un determinato ente(società|istituti|persone_fisiche|giudici|collaboratori) trovare le cause in cui è coinvolto

```
SELECT   coinvoltoin.CodCausa,cause.Stato,personefisiche.Nome

FROM coinvoltoin JOIN cause ON cause.CodCausa=coinvoltoin.CodCausa

JOIN enti ON coinvoltoin.CodEnte= enti.CodEnte
JOIN personefisiche ON enti.CodEnte=personefisiche.CodEnte WHERE enti.CodEnte = 70

UNION SELECT   coinvoltoin.CodCausa,cause.Stato,istituzioni.Nome

FROM coinvoltoin JOIN cause ON cause.CodCausa=coinvoltoin.CodCausa JOIN enti ON

coinvoltoin.CodEnte= enti.CodEnte  JOIN istituzioni ON
```

```
enti.CodEnte=istituzioni.CodEnte WHERE enti.CodEnte = 70

UNION SELECT   coinvoltoin.CodCausa,cause.Stato,societa.Nome FROM coinvoltoinJOIN c
ause ON cause.CodCausa=coinvoltoin.CodCausa

JOIN enti ON coinvoltoin.CodEnte= enti.CodEnte

JOIN societa ON enti.CodEnte=societa.CodEnte WHERE enti.CodEnte = 70
```

Oppure separate per ogni ente e cercate in base al nome/cognome

```
SELECT   coinvoltoin.CodCausa,cause.Stato,personefisiche.Nome,

personefisiche.CodEnte,personefisiche.Cognome FROM coinvoltoin JOIN cause ON

cause.CodCausa=coinvoltoin.CodCausa JOIN personefisiche ON

coinvoltoin.CodEnte= personefisiche.CodEnte

WHERE personefisiche.Nome = 'Olympia' AND personefisiche.Cognome ='Carlson'
```

2) Dato un determinato ente verificare il numero di udienze a cui ha partecipato in totale o per una determinata causa

```
SELECT personefisiche.Nome, personefisiche.Cognome,

COUNT(discussada.CodUdienza)AS NUMUDIENZE FROM enti JOIN discussada ON

(discussada.CodEnte = enti.CodEnte)

JOIN personefisiche ON (enti.CodEnte = personefisiche.CodEnte)

WHERE personefisiche.Nome = 'Carter' AND personefisiche.Cognome = 'Smith'
```

3) Data una causa indicare il giudice e quali collaboratori sono coinvolti in essa.

```
SELECT   giudici.CodEnte, personefisiche.Nome, personefisiche.Cognome

FROM cause JOIN giudici  ON (cause.codGiudice = giudici.CodEnte)

JOIN personefisiche ON (giudici.CodEnte=personefisiche.CodEnte)

WHERE cause.CodCausa 9 UNION SELECT   collaboratori.CodEnte,

personefisiche.Nome ,personefisiche.Cognome  FROM cause

JOIN coinvoltoin ON (cause.CodCausa=coinvoltoin.CodCausa)
```

```
JOIN collaboratori ON (coinvoltoin.CodEnte=collaboratori.CodEnte)

JOIN personefisiche ON (collaboratori.CodEnte=personefisiche.CodEnte)

WHERE cause.CodCausa = 9

UNION SELECT   coinvoltoin.CodEnte,societa.Nome,societa.LegaleRappresentante FROM c
ause JOIN coinvoltoin ON(cause.CodCausa=coinvoltoin.CodCausa)

JOIN societa ON (coinvoltoin.CodEnte=societa.CodEnte) WHERE cause.CodCausa=9

UNION SELECT   coinvoltoin.CodEnte, istituzioni.Nome,

istituzioni.TipoIstituzione FROM cause

JOIN coinvoltoin ON (cause.CodCausa=coinvoltoin.CodCausa)

JOIN istituzioni ON (coinvoltoin.CodEnte= istituzioni.CodEnte)

WHERE cause.CodCausa=9
```

4) Trovare le cause ospitate in determinato tribunale

```
SELECT   cause.CodCausa, tribunali.Luogo FROM cause

JOIN tribunali ON cause.CodTribunale = tribunali.CodTribunale

 WHERE tribunali.Luogo = 'Cardiff'
```

5) Considerato un tribunale determinare una lista dei giudici che vi lavorano

```
SELECT   lavorain.CodEnte, personefisiche.Nome, personefisiche.Cognome , tribunali.
Luogo, tribunali.Indirizzo giudici.TipoGiudice FROM lavorain

JOIN tribunali ON (lavorain.CodTribunale= tribunali.CodTribunale)

JOIN giudici ON (giudici.CodEnte = lavorain.CodEnte)

JOIN personefisiche ON (giudici.CodEnte = personefisiche.CodEnte)

WHERE tribunali.Luogo= 'Cardiff'
```

6) Data un’udienza determinare i collaboratori che l'hanno discussa

```
SELECT   collaboratori.CodEnte, personefisiche.Nome, personefisiche.Cognome, udienz
e.CodUdienza, udienze.Esito, udienze.La_Data FROM udienze

JOIN discussada ON (udienze.CodUdienza= discussada.CodUdienza)

JOIN collaboratori ON (discussada.CodEnte= collaboratori.CodEnte)

JOIN personefisiche ON (collaboratori.CodEnte = personefisiche.CodEnte)

 WHERE udienze.CodUdienza = 1510
```

7) Trovare i recapiti per un determinato ente

```
SELECT   recapiti.TipoRecapito, recapiti.Descrizione, recapiti.Recapito,enti.CodEnt
e, personefisiche.Nome, personefisiche.Cognome FROM recapiti
JOIN enti ON(enti.CodEnte= recapiti.CodEnte)
JOIN personefisiche ON(recapiti.CodEnte=personefisiche.CodEnte)
WHERE personefisiche.Nome = 'Malik' AND personefisiche.Cognome = 'Berger'
```

8) Dato un giudice trovare una lista di collaboratori che hanno lavorato alle udienze delle cause presiedute da quel giudice

```
SELECT   cause.CodCausa , udienze.CodUdienza , collaboratori.CodEnte,
cause.CodGiudice , personefisiche.Nome, personefisiche.Cognome,
collaboratori.Qualifica FROM giudici
JOIN cause ON (giudici.CodEnte= cause.CodGiudice)
JOIN udienze ON (cause.CodCausa = udienze.CodCausa)
JOIN discussada ON (udienze.CodUdienza=discussada.CodUdienza)
JOIN collaboratori ON(discussada.CodEnte = collaboratori.CodEnte)
JOIN personefisiche ON (collaboratori.CodEnte=personefisiche.CodEnte)
WHERE giudici.CodEnte= 90
```

9) Trovare codice causa, rg per quelle cause le cui udienze si sono svolte in un determinato range temporale

```
SELECT cause.CodCausa, cause.rg , udienze.CodUdienza FROM udienze
JOIN cause ON(udienze.CodCausa=cause.CodCausa)
WHERE udienze.La_Data BETWEEN '1998-12-31' AND '2003-12-31'
```

10) Dati due enti specifici determinare se vi sono una o più cause in cui sono coinvolt

```
SELECT count(*) FROM
(SELECT coinvoltoin.*  FROM coinvoltoin  WHERE coinvoltoin.CodEnte='32') as tb1
JOIN (SELECT coinvoltoin.* FROM coinvoltoin  WHERE coinvoltoin.CodEnte='147')
as tb2 ON tb1.CodCausa = tb2.CodCausa
```

Oppure specificando generalità degli enti:

```
SELECT count(*) FROM

(SELECT coinvoltoin.* FROM coinvoltoin JOIN societa ON societa.CodEnte = coinvoltoi
n.CodEnte WHERE societa.Nome='Tincidunt Neque Vitae PC') as tb1 JOIN

(SELECT coinvoltoin.* FROM coinvoltoin JOIN personefisiche ON personefisiche.CodEnt
e = coinvoltoin.CodEnte WHERE personefisiche.Nome='Basil' AND personefisiche.Cognom
e='Barron') as tb2 ON tb1.CodCausa = tb2.CodCausa
```

11) Aggiungere uno o più collaboratori ad un'udienza

```
INSERT INTO discussada (CodUdienza, CodEnte) VALUES (10,91)
```

12) Aggiornare dati enti(società|istituti|persone_fisiche|giudici|collaboratori)

```
UPDATE societa SET  PartitaIVA = '534555' WHERE CodEnte = 22
```

13) Inserire nuova causa che coinvolge determinati enti

```
INSERT INTO `coinvoltoin`(`CodEnte`, `CodCausa`, `Ruolo`, `Cliente`)

VALUES (71,150,'difesa',1)
```

14) Inserire nuova udienza per una causa

```
INSERT INTO udienze (CodCausa, La_Data,Esito) VALUES( '325' ,'1992-12-
13','Sospesa')
```

15) Aggiornare lista di tribunali in cui lavora un determinato giudice

```
UPDATE lavorain SET CodTribunale=’11’ WHERE CodEnte=’71’
```

Appendice 2: Script PHP per creare le collection

Giudici:

```php
<?php

$server = 'localhost';
$user = 'root';
$pass = '';
$dbname = 'database_studio_vecchio';
$mysqli = new mysqli($server, $user, $pass, $dbname);
//$con = mysql_connect($server, $user, $pass) or die("Can't connect");
//mysql_select_db($dbname);
if ($mysqli->connect_error) {
    die('Connect Error (' . $mysqli->connect_errno . ') '
            . $mysqli->connect_error);
}
$result = $mysqli->query(' SELECT p.*, g.TipoGiudice FROM  personefisiche AS p inner join giudici as g ON p.CodEnte = g.CodEnte');

if(!$result) {
    die('Database error: ' . $mysqli->error);
}

$data = array();
// use fetch_array instead of fetch_assoc as the column
while($row = $result->fetch_assoc()) {

   $recapiti = $mysqli->query("SELECT TipoRecapito,Descrizione1,Recapito
        FROM recapiti where CodEnte='".$row['CodEnte']."'");
   if($recapiti)
   {
       $contacts=array();
       while($recapito = $recapiti->fetch_assoc()){
           $contacts[] = $recapito;
       }
       $row['recapiti'] = $contacts;
   }
```

```
33.
34. $tribunali = $mysqli->query("SELECT t.CodTribunale,t.Luogo FROM tribunali as t inner join lavorain as lv ON t.CodTribunale = lv.CodTribunale where lv.CodEnte='".$row['CodEnte']."'");
35.
36.    if($tribunali)
37.    {
38.        $data_to_add=array();
39.        while ($tribunale = $tribunali->fetch_assoc()){
40.            $data_to_add[] = $tribunale;
41.        }
42.        $row['tribunali in cui lavora'] = $data_to_add;
43.    }
44.
45.     $data []= $row;
46.
47.
48. }
49.
50. echo json_encode($data);
51.
52.
53.
54. ?>
```

Società:

```
1.  <?php
2.
3.  $server = 'localhost';
4.  $user = 'root';
5.  $pass = '';
6.  $dbname ='database_studio_vecchio';
```

```
$mysqli = new mysqli($server, $user, $pass, $dbname);
//$con = mysql_connect($server, $user, $pass) or die("Can't connect");
//mysql_select_db($dbname);
if ($mysqli->connect_error) {
    die('Connect Error (' . $mysqli->connect_errno . ') '
            . $mysqli->connect_error);
}
$result = $mysqli->query(' SELECT i.* FROM  societa AS i'


);

if(!$result) {
    die('Database error: ' . $mysqli->error);
}

$data = array();
// use fetch_array instead of fetch_assoc as the column
while($row = $result->fetch_assoc()) {

   $recapiti = $mysqli->query("SELECT TipoRecapito,Descrizione,Recapito FROM recapiti where CodEnte ='".$row['CodEnte']."'");
   if($recapiti)
   {
       $recapitiToAdd= array();
       while($recapito = $recapiti->fetch_assoc()){
           $recapitiToAdd[] = $recapito;
       }
       $row['recapiti'] = $recapitiToAdd;

    }

    $data []= $row;

```

```
40.
41. }
42.
43. echo json_encode($data);
44.
45.
46.
47. ?>
```

Cause

```
1.  <?php
2.
3.  $server = 'localhost';
4.  $user = 'root';
5.  $pass = '';
6.  $dbname = 'database_studio_legale';
7.  $mysqli = new mysqli($server, $user, $pass, $dbname);
8.  //$con = mysql_connect($server, $user, $pass) or die("Can't connect");
9.  //mysql_select_db($dbname);
10. if ($mysqli->connect_error) {
11.     die('Connect Error (' . $mysqli->connect_errno . ') '
12.             . $mysqli->connect_error);
13. }
14. $result = $mysqli->query(' SELECT c.*  FROM  cause AS c '
15. );
16.
17. if(!$result) {
18.     die('Database error: ' . $mysqli->error);
19. }
20.
21. $data = array();
22.
23. while($row = $result->fetch_assoc()) {
24.
25.    // per ogni causa aggiungiamo il tribunale o i tribunali
26.
27.    $tribunali = $mysqli-> query
    ("SELECT CodTribunale,Luogo as luogoTribunale,TipoTribunale FROM tribunali w
    here CodTribunale='".$row['CodTribunale']."'");
28.    if($tribunali)
29.    {
30.        $tribunaleToAdd=array();
31.        while($tribunale = $tribunali->fetch_assoc()){
32.            $tribunaleToAdd[] = $tribunale;
33.        }
34.        $row['tribunale'] = $tribunaleToAdd;
35.    }
36. // per ogni causa aggiungiamo i giudici che l'hanno presieduta
37. $giudici = $mysqli-> query
    ("SELECT p.CodEnte,p.Nome,p.Cognome FROM personefisiche as p
38.                             inner join giudici as g ON g.CodEnte = p.CodEnte

39.                              where p.CodEnte='".$row['CodGiudice']."'");
```

```
40.
41. if($giudici)
42. {
43.     $giudiciToAdd = array();
44.     while($giudice = $giudici->fetch_assoc()){
45.         $giudiciToAdd[] = $giudice;
46.     }
47.     $row['giudice'] = $giudiciToAdd;
48. }
49.
50. // aggiungiamo le udienze per ogni causa
51. $udienze =$mysqli->query
    ("SELECT u.CodUdienza,u.La_Data,u.Esito FROM udienze as u
52.                             where u.CodCausa='".$row['CodCausa']."'");
53. if($udienze)
54. {
55.     $udienzeToAdd=array();
56.     while($udienza = $udienze->fetch_assoc()){
57.         //per ogni udienza aggiungiamo i collaboratori
58.             $collaboratori = $mysqli-> query
    ("SELECT p.CodEnte as CodCollaboratore,p.Nome as nomeCollaboratore,p.Cognome
     as cognomeCollaboratore FROM personefisiche as p inner join collaboratori
59. as c ON c.CodEnte = p.CodEnte inner join discussada as d ON
60. d.CodEnte = c.CodEnte
61. where d.CodUdienza='".$udienza['CodUdienza']."'");
62.             if($collaboratori)
63.             {
64.                 $collaboratoriToAdd= array();
65.                 while($collaboratore = $collaboratori->fetch_assoc()){
66.                     $collaboratoriToAdd[] = $collaboratore;
67.                 }
68.                 $udienza['Collaboratori'] = $collaboratoriToAdd;
69.             }
70.
71.         $udienzeToAdd[] = $udienza;
72.
73.     }
74.     $row['udienze'] = $udienzeToAdd;
75. }
76.
77. // aggiungiamo le persone coinvolte per ogni causa
78. $personeCoinvolte= $mysqli-> query
    ("SELECT p.CodEnte as CodEnte ,p.Nome as nome,p.Cognome as cognome,c.
79. FROM coinvoltoin as c inner join personefisiche as p ON
80. c.CodEnte = p.CodEnte where c.CodCausa='".$row['CodCausa']."'");
81. if($personeCoinvolte)
82. {
83.     $personeCoinvolteToAdd = array();
84.     while ($personeCoinvolta = $personeCoinvolte->fetch_assoc()){
85.         $personeCoinvolteToAdd[] = $personeCoinvolta;
86.     }
87.     $row['entita coinvolte'] = $personeCoinvolteToAdd;
88. }
89.
90. $societàCoinvolte= $mysqli-> query
    ("SELECT s.CodEnte as CodEnte ,s.Nome as nome,c.Cliente
91.  FROM coinvoltoin as c inner join societa as s ON c.CodEnte = s.CodEnte
92.   where c.CodCausa='".$row['CodCausa']."'");
93.
94. if($societàCoinvolte)
95. {
96.     $societàCoinvolteToAdd = array();
97.     while ($societàCoinvolta = $societàCoinvolte->fetch_assoc()){
98.         $societàCoinvolteToAdd[] = $societàCoinvolta;
```

```
    }
            if(count($societàCoinvolteToAdd)>0)
            $row['entita coinvolte'][] = $societàCoinvolteToAdd;
        }

        $istituzioniCoinvolte= $mysqli-> query
    ("SELECT i.CodEnte as CodEnte ,i.Nome as nome,i.TipoIstituzione, c.Cliente
        FROM coinvoltoin as c
        inner join istituzioni as i ON c.CodEnte = i.CodEnte
         where c.CodCausa='".$row['CodCausa']."'");

        if($istituzioniCoinvolte)
        {
        $istituzioniCoinvolteToAdd = array();
        while ($istituzioniCoinvolta = $istituzioniCoinvolte-> fetch_assoc())
        {
        $istituzioniCoinvolteToAdd[] = $istituzioniCoinvolta;
        }
        if(count($istituzioniCoinvolteToAdd)>0)
        $row['entita coinvolte'][] = $istituzioniCoinvolteToAdd;
        }

        $data []= $row;
        }
        echo json_encode($data);
        //echo bin2hex( MongoDB\BSON\fromPHP($data));

        ?>
```

Collaboratori

```
<?php

$server = 'localhost';
$user = 'root';
$pass = '';
$dbname = 'database_studio_vecchio';
$mysqli = new mysqli($server, $user, $pass, $dbname);
//$con = mysql_connect($server, $user, $pass) or die("Can't connect");
//mysql_select_db($dbname);
if ($mysqli->connect_error) {
    die('Connect Error (' . $mysqli->connect_errno . ') '
            . $mysqli->connect_error);
}
$result = $mysqli-
>query(' SELECT p.*,c.Qualifica,c.IdBadge FROM  personefisiche AS p
```

```
15.
16.     inner join collaboratori as c ON p.CodEnte = c.CodEnte'
17. );
18.
19. if(!$result) {
20.     die('Database error: ' . $mysqli->error);
21. }
22.
23. $data = array();
24. // use fetch_array instead of fetch_assoc as the column
25. while($row = $result->fetch_assoc()) {
26.
27.     $recapiti = $mysqli-
    >query("SELECT TipoRecapito,Descrizione,Recapito FROM recapiti where CodEnte
    ='".$row['CodEnte']."'");
28.     if($recapiti)
29.     {
30.         $recapitiToAdd = array();
31.         while($recapito = $recapiti->fetch_assoc()){
32.             $recapitiToAdd[] = $recapito;
33.         }
34.         $row['recapiti'] = $recapitiToAdd;
35.
36.      }
37.
38.      $data []= $row;
39.
40. }
41.
42. echo json_encode($data);
43. ?>
```

Tribunali

```php
<?php

$server = 'localhost';
$user = 'root';
$pass = '';
$dbname = 'database_studio_vecchio';
$mysqli = new mysqli($server, $user, $pass, $dbname);
if ($mysqli->connect_error) {
    die('Connect Error (' . $mysqli->connect_errno . ') '
            . $mysqli->connect_error);
}
$result = $mysqli->query(' SELECT t.* FROM  tribunali AS t' );

if(!$result) {
    die('Database error: ' . $mysqli->error);
}

$data = array();
// use fetch_array instead of fetch_assoc as the column
while($row = $result->fetch_assoc()) {

$giudici = $mysqli>query
("SELECT p.CodEnte,p.Nome,p.Cognome FROM personefisiche as p inner join
lavorain as lv ON lv.CodEnte=p.CodEnte where lv.CodTribunale='".$row['CodTri
bunale']."'");
    if($giudici)
    {
        $giudiciToAdd = array();
        while($giudice = $giudici->fetch_assoc()){
            $giudiciToAdd[] = $giudice;
        }
        $row['giudici'] = $giudiciToAdd;

```

```
32.      }
33.
34.      $data []= $row;
35.      }
36. echo json_encode($data);
37. ?>
```

Persone Fisiche

```
1.  <?php
2.
3.  $server = 'localhost';
4.  $user = 'root';
5.  $pass = '';
6.  $dbname = 'database_studio_vecchio';
7.  $mysqli = new mysqli($server, $user, $pass, $dbname);
8.  if ($mysqli->connect_error) {
9.      die('Connect Error (' . $mysqli->connect_errno . ') '
10.             . $mysqli->connect_error);
11. }
12. $result = $mysqli->query(' SELECT p.* FROM  personefisiche AS p'
13. );
14.
15. if(!$result) {
16.     die('Database error: ' . $mysqli->error);
17. }
18.
19. $data = array();
20. // use fetch_array instead of fetch_assoc as the column
21. while($row = $result->fetch_assoc()) {
22.
23.     $recapiti = $mysqli-
    >query("SELECT TipoRecapito,Descrizione,Recapito FROM recapiti where CodEnte
    ='".$row['CodEnte']."'");
24.     if($recapiti)
25.     {
```

```
        $recapitiToAdd = array();
        while($recapito = $recapiti->fetch_assoc()){
            $recapitiToAdd[] = $recapito;
        }
        $row['recapiti'] = $recapitiToAdd;

    }

    $data []= $row;


}

echo json_encode($data);
 ?>
```

Istituzioni

```
<?php
$server = 'localhost';
$user = 'root';
$pass = '';
$dbname = 'database_studio_vecchio';
$mysqli = new mysqli($server, $user, $pass, $dbname);
if ($mysqli->connect_error) {
    die('Connect Error (' . $mysqli->connect_errno . ') '
            . $mysqli->connect_error);
}
$result = $mysqli->query(' SELECT i.* FROM  istituzioni AS i'


);
```

```
15.
16. if(!$result) {
17.     die('Database error: ' . $mysqli->error);
18. }
19.
20. $data = array();
21. // use fetch_array instead of fetch_assoc as the column
22. while($row = $result->fetch_assoc()) {
23.
24.     $recapiti = $mysqli->query("SELECT TipoRecapito,Descrizione,Recapito
    FROM recapiti where CodEnte='".$row['CodEnte']."'");
25.     if($recapiti)
26.     {
27.         $recapitiToAdd = array();
28.         while($recapito = $recapiti->fetch_assoc()){
29.             $recapitiToAdd[] = $recapito;
30.         }
31.         $row['recapiti'] = $recapitiToAdd;
32.      }
33.      $data []= $row;
34. }
35. echo json_encode($data);
36. ?>
```