Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Triennale in

Ingegneria Informatica

Il movimento NoSQL
Il caso di studio di CouchDB

Relatore: Prof. Giorgio Maria Di Nunzio

Laureando: Simone Marini

Anno Accademico 2012/2013

Indice

Elenco delle figure

Elenco dei listati codice

Capitolo 1

Introduzione

Il modello relazionale dei database, introdotto nel 1970 da Ted Codd e basato sul concetto matematico di relazione, è stato per quarant'anni quello dominante nell'ambito dei sistemi di gestione di basi di dati (DBMS, database management system). Attualmente, tra i prodotti proprietari di questo tipo più diffusi vi sono DB2[1] (di IBM), Oracle[2] (di Oracle), SQL Server[3] (di Microsoft); mentre tra quelli open source i maggiori sono MySQL[4] e PostgreSQL[5].
Le ragioni di questo successo sono dovute principalmente alla frequente presenza di dati omogenei e la capacità di immagazzinarli, garantendo un'elevata potenzialità delle interrogazioni: è quasi certamente possibile estrarre dal database le informazioni necessarie.
Inoltre, soddisfa altre necessità che sorgono, ad esempio, nella gestione di transazioni bancarie e finanziarie, grazie all'incapsulamento delle operazioni in transazioni che godono delle proprietà cosiddette ACIDe [1]:

- **Atomicità** (Atomicity): una transazione deve essere eseguita completamente, altrimenti non avviene;
- **Consistenza** (Consistency): le transazioni devono portare la base di dati da uno stato consistente ad un'altro, mantenedo così il rispetto dei vincoli di integrità dello schema;
- **Isolamento** (Isolation): ogni transazione viene eseguita senza interferire con altre transazioni che avvengono simultaneamente;
- **Persistenza** (Durability): le modifiche apportate sulla base di dati da una transazione andata a buon fine sono persistenti.

[1] http://www-01.ibm.com/software/data/db2/
[2] http://www.oracle.com/it/products/database/overview/index.html
[3] https://www.microsoft.com/italy/server/sql/default.mspx
[4] http://www.mysql.it/
[5] http://www.postgresql.org/

In alcuni contesti, tuttavia, i requisiti dei dati sono troppo flessibili e non risulta possibile definire uno schema prefissato. Inoltre, ci può essere l'esigenza, difficile da offrire con un database relazionale, di scalare orizzontalmente e di avere una rete soggetta a partizionamento. Ed è per rispondere a questi bisogni che sono sorti dei DBMS non relazionali e che sono stati inglobati sotto il termine NoSQL.

1.1 Il movimento NoSQL

Il termine NoSQL [2], acronimo di Not Only SQL, è stato usato per la prima volta nel 1998 da Carlo Strozzi per indicare il database relazionale, da lui realizzato, che non adoperava SQL. Successivamente, è stato ripreso nel 2009 da Eric Evans in un evento in cui si discuteva di basi di dati distribuite, non relazionali e che non offrono le classiche proprietà ACIDe delle transazioni.
Le ragioni [3] per adoperare un database NoSQL sono molteplici e possono essere così classificate:

1. Evitare complessità non necessaria: i database relazionali forniscono le proprietà ACIDe, varie funzioni e una serie di vincoli di coerenza sui dati. Tutto questo potrebbe essere superfluo per alcuni utilizzi.

2. Sono sufficienti le proprietà del teorema CAP [4]: in base a questo teorema dell'informatica teorica, un database distribuito non può fornire contemporaneamente tutte e tre le seguenti caratteristiche:

   - **Coerenza** (Consistency): dopo un'operazione di modifica sui dati, questa si propagherà di modo che ogni richiesta successiva di lettura ritorni il valore aggiornato;
   - **Disponibilità** (Availability): il database ritornerà sempre il risultato di una richiesta, purché almeno un server sia in esecuzione;
   - **Tolleranza al partizionamento** (Partition tolerance): il database, che viene eseguito su più nodi di una rete, continuerà a funzionare anche se la connessione tra due o più nodi verrà a mancare.

   Possono essere rispettate solamente due delle suddette proprietà, nel caso di un database distribuito, che deve essere tollerante al partizionamento, è necessario scegliere tra coerenza e disponibilità.
   Nell'eventualità di scegliere la disponibilità, si potrà solamente avere una coerenza eventuale, ovvero il sistema cercherà di propagare tra i nodi gli aggiornamenti ai dati quando sarà ristabilito il collegamento.

Questo può comportare dei conflitti, la presenza cioè della stessa informazione con valori differenti. Per risolvere questo problema si possono adoperare gli orologi vettoriali, un meccanismo che tiene traccia degli aggiornamenti avvenuti e del loro ordine temporale, scegliendo poi quale valore mantenere. Molto spesso, non essendovi un criterio generale per tale scelta, essa viene lasciata all'utente che potrà così effettuare il merge manuale dei dati.

3. Throughput elevato: alcuni database NoSQL forniscono un throughput dei dati molto superiore rispetto ai tradizionali DBMS relazionali. Per esempio, Google[6] è in grado di elaborare 20 petabyte al giorno memorizzati in BigTable[7] tramite l'utilizzo di MapReduce.

4. Scalabilità orizzontale e supporto hardware: la necessità di gestire enormi volumi di dati e di distribuirli su più server per aumentare le prestazioni e per non avere un singolo punto di fallimento. I database NoSQL possono infatti scalare orizzontalmente e sono tolleranti ai guasti, i server possono essere aggiunti o rimossi con relativa semplicità.
   Alcuni database NoSQL, come MongoDB[8] e CouchDB[9], forniscono automaticamente lo sharding, una tecnica con la quale un database viene suddiviso in parti e distribuito tra più macchine al fine di gestire un carico che un singolo server non sarebbe in grado.

5. Strutture di dati semplici: la maggior parte dei database NoSQL sono progettati per memorizzare strutture di dati che sono semplici, oppure più simili a quelle della programmazione orientata agli oggetti (OOP, object oriented programming). Ciò è particolarmente utile per applicazioni con strutture di dati di bassa complessità e che difficilmente possono beneficiare delle caratteritiche di un database relazionale.

1.2 Tassonomia dei database NoSQL

Basandosi su Eric Redmond, Jim R. Wilson (2012) *Seven Databases in Seven Weeks*, USA, Jackie Carter, i database NoSQL, in base a come vengono memorizzati i dati, vengono suddivisi in quattro categorie:

1. Chiave-valore (key-value).

2. Orientati alle colonne (columnar o column-oriented).

[6] www.google.com
[7] http://research.google.com/archive/bigtable.html
[8] http://www.mongodb.org/
[9] http://couchdb.apache.org/

3. Orientati al documento (document-oriented).

4. Grafo (graph).

1.2.1 Chiave-valore

Memorizzano coppie chiave-valore, accessibili tramite la chiave su cui sono anche indicizzati, mentre nel valore possono essere posti sia tipi di dato elementari, sia tipi avanzati.
Adatti in situazioni in cui non è possibile definire uno schema sui dati ed è necessario un accesso rapido alle singole informazioni. Consente di scalare orizzontalmente e di avere rapide operazioni di tipo CRUD (create, read, update, delete). Presenta forti limitazioni nelle capacità di interrogazione dei dati.
Utilizzato per memorizzare informazioni che non presentano correlazioni, ad esempio per il salvataggio delle sessioni degli utenti in ambito web.
Le funzionalità dipendono molto dal tipo di database, le versioni open source più diffuse sono Memcached[10], Voldemort[11], Redis[12] e Riak[13].

Redis
Redis [5] fornisce tipi di dato avanzati, come liste, tabelle hash, insiemi e insiemi ordinati. È basato sul modello master-slave, un server è eletto master per default e i dati vengono replicati su ogni server slave. Dispone di un meccanismo, detto *multi block*, simile alle transazioni.
Si presta ad essere adoperato come memoria cache per due motivi: permette di stabilire un tempo di vita alle coppie chiave-valore, al termine del quale verranno cancellate; consente di velocizzare le operazioni di scrittura, salvandole prima in memoria temporanea, a scapito di un incremento del rischio di perdita dei dati in caso di malfunzionamento.

Riak
Riak [5] è una fedele implementazione della Dynamo[14] di Amazon[15] ed è scritto in Erlang[16]. Il database è accessibile tramite URL, formattato secondo le linee guida di REST (REpresentational State Transfer), mentre il server risponde usando il formato HTTP. I valori inseriti possono essere qualunque cosa, dal semplice testo, al codice XML, fino alle immagini. Possono essere create relazioni tra le chiavi mediante strutture chiamate *link*, per-

[10] http://memcached.org/
[11] www.project-voldemort.com
[12] http://redis.io/
[13] http://basho.com/riak/
[14] http://aws.amazon.com/dynamodb/
[15] http://www.amazon.com/
[16] http://www.erlang.org/

corribili unidirezionalmente.
Le operazioni di scrittura vengono prima salvate in memoria temporanea, anche se è data la possibilità, rallentando l'operazione, di scrivere direttamente su hard disk.
È un database distribuito che fa uso degli orologi vettoriali per la risoluzione dei conflitti, supporta MapReduce come meccanismo per effettuare le query.

1.2.2 Orientati alle colonne

Così denominati poiché i dati, mantenendo la classica organizzazione in righe e colonne, sono memorizzati assieme per colonna. Ogni riga può avere un set diverso di colonne, poiché possono essere aggiunte quelle necessarie e tolte quelle inutilizzate, evitando così la presenza di valori null.
Progettato per scalare orizzontalmente, fino a migliaia di nodi, è particolarmente indicato per casi d'uso con un enorme volume di dati. Consente la compressione delle informazioni e il versioning.
Un utilizzo tipico è l'indicizzazione di pagine web: possiedono un testo, che può essere compresso, e cambiano nel tempo, beneficiando così del versioning.
Tra i database più diffusi vi sono HBase[17], Cassandra[18] e Hypertable[19].

HBase
HBase [5] si base su BigTable di Google, è progettato per scalare orizzontalmente in cluster e fornisce forti garanzie sulla coerenza dei dati. Utilizza Hadoop[20] per effettuare interrogazioni con MapReduce.
Viene adoperato, ad esempio, da Facebook[21] nell'infrastruttura che gestisce i messaggi.

1.2.3 Orientati al documento

Sono caratterizzati da una struttura fondamentale, detta *document*, scritto comunemente in Javascript Object Notation[22] (JSON), costituito da un identificatore univoco e da un qualsiasi altro numero di attributi. Questi possono essere qualunque tipo di dato, purché esprimibili come un documento, oppure possono contenere altri oggetti nidificati.
Adatti in situazioni in cui i dati variano nel tempo, si prestano inoltre a mappare correttamente gli oggetti nel modello OOP. Sono intrinsecamente

[17] http://hbase.apache.org/
[18] http://cassandra.apache.org/
[19] http://hypertable.org/
[20] http://hadoop.apache.org/
[21] https://it-it.facebook.com/
[22] http://www.json.org/

denormalizzati, presentano cioè al loro interno informazioni replicate.
I maggiori prodotti open source disponibili sono MongoDB e CouchDB.

MongoDB
MongoDB [5] è stato realizzato per contenere enormi volumi di dati. È eventualmente consistente, offre la possibilità di eseguire operazioni di lettura-scrittura atomiche e utilizza JavaScript come linguaggio per le interrogazioni.

CouchDB
CouchDB [5] si distingue per la robustezza, per la capacità di venire replicato tra più server di una rete e, in base al teorema CAP, di essere tollerante al partizionamento e disponibile alle richieste da parte dei client. Per evitare conflitti sui dati, adotta un meccanismo di versioning che impedisce le modifiche ad un client che fornisce un numero di versione non aggiornato, oppure è in grado di scegliere quale versione mantenere quando la stessa informazione è modificata su server differenti.
Anch'esso usa JavaScript come linguaggio di interrogazione.

1.2.4 Grafo

Basato sulla struttura di dati grafo, gli elementi fondamentali che lo costituiscono sono i nodi e i collegamenti tra questi, ed entrambi possono memorizzare coppie chiave-valore.
La peculiarità di questo modello è di essere adatto a dati fortemente interconnessi tra di loro, di effettuare le interrogazioni compiendo un'attraversamento efficiente del grafo, tramite cammini composti da nodi e collegamenti. Ad esempio, per velocizzare il cammino da un nodo ad un'altro può aggiungere un collegamento diretto tra i due, raggiungendo un costo unitario per l'operazione.

Neo4J[23]
Attualmente, Neo4J [5] è il database più diffuso e si contraddistingue per la capacità di attraversare rapidamente il grafo nelle interrogazioni. Utilizza REST come interfaccia per comunicare, ogni operazione è incapsulata in una transazione e gode delle proprietà ACIDe. Consente l'indicizzazione sulla chiave e la costruzione di indici invertiti per effettuare ricerche su testo memorizzato. È un database distribuito ad alta disponibilità, con un modello master-slave tra i server, diventa eventualmente consistente quando una modifica su uno slave non è sincronizzata con gli altri server.
Può essere utilizzato, ad esempio, nei social network: i nodi rappresentano gli utenti, mentre i collegamenti rappresentano le relazioni tra di essi.

[23] http://www.neo4j.org/

1.2.5 Modelli poliglotti persistenti

Come è avvenuto nel mondo della programmazione, l'utilizzo di un'unica tecnologia per risolvere un problema non rappresenta più l'unica soluzione adottabile.
Le differenti proprietà dei database hanno portato allo sviluppo dei *polyglot persistent model*, caratterizzati dalla presenza di differenti tipi di database, che cooperano tra loro sfruttando i punti di forza di ciascuno, per raggiungere un obiettivo comune.
Ad esempio, CouchDB può fungere come sorgente dei dati, Neo4J per gestire le relazioni tra di essi, mentre Redis può agire come memoria cache per i cambiamenti recenti e per effettuare ricerche rapide.

Capitolo 2

CouchDB

2.1 Introduzione

CouchDB [6,7] rappresenta uno dei nuovi tipi di database non relazionali appartenenti al movimento NoSQL. È un DBMS open source di tipo document-oriented, scritto in Erlang e sviluppato da Apache Software Foundation.
Rispetto ai DBMS relazionali, presenta un approccio diverso nel strutturare i dati, nel modo di salvarli e nel compiere operazioni di interrogazione, raggruppamento, filtraggio e replicazione. Il suo utilizzo è di facile comprensione principalmente per le seguenti ragioni:

- la comunicazione con il database avviene tramite il protocollo HTTP: CouchDB risulta quindi disponibile a tutti i linguaggi di programmazione che implementano questo protocollo. Data la sua semplicità di utilizzo, viene appreso velocemente dagli sviluppatori, specie da quelli che lavorano già in ambito Web;

- basato sui documenti: consente una elevata flessibilità alle informazioni che evolvono la loro struttura nel tempo, senza i vincoli di uno schema dei dati. Al contrario dei DBMS relazionali, che richiedono di modellare i dati in anticipo, con i documenti si possono aggregare i dati durante la fase del loro utilizzo;

- facile meccanismo di replicazione dei dati: consente di rispondere rapidamente alla necessità di scaling in richieste di lettura, scrittura e sulla quantità di dati;

- fault-tolerant: CouchDB è un ambiente controllato in cui ogni singolo problema rimane isolato nella sua operazione, senza propagarsi nel server. Gli errori sono gestiti cercando di dare sempre una risposta alle richieste, evitando interruzioni del servizio. È comunque possibile ottenere informazioni dettagliate sui malfunzionamenti interni e poter intervenire manualmente.

Ogni database è una collezione di documenti indipendenti, scritti nel formato JSON, interrogabili per mezzo di funzioni MapReduce, scritte in linguaggio JavaScript.
Raggiunge efficienza nelle operazioni interne grazie ai seguenti motivi:

1. struttura chiave-valore delle informazioni: l'utilizzo di una chiave consente accessi rapidi ai documenti sia in lettura che in scrittura;

2. utilizzo di strutture dati efficienti: dati, documenti e viste sono contenuti all'interno di B-Tree, consentendo ricerche veloci di un elemento o di un range di elementi;

3. accesso concorrente: nei DBMS tradizionali, quando vengono compiute operazioni di aggiornamento, viene bloccato l'accesso a un'intera tabella o al record interessato. Invece, CouchDB usa il modello MVCC (Multi-Version Concurrency Control), che in caso di aggiornamento inserisce una nuova versione di un record, permettendo così agli altri processi di accedere alla versione precedente dell'informazione senza venire bloccati in attesa.

2.2 Caratteristiche principali e funzionalità

2.2.1 Il modello dei dati

Le informazioni evolvono nel tempo, possono espandersi e aumentare, impedendo quindi di poter definire uno schema rigido per poterle contenere.
CouchDB si basa sui documenti, un formato per i dati molto flessibile che consente di poter modificare con semplicità la propria struttura, aggiungendo o rimuovendo campi. I documenti sono scritti nel linguaggio JSON e permettono, inoltre, di rappresentare meglio le informazioni del mondo reale: i dati, di diverso tipo, si presentano spesso aggregati tra di loro, proprietà che è possibile conservare con i documenti, mentre in un DBMS relazionale è necessario scindere le informazioni diverse in tabelle separate e successivamente unirle con operazioni di join.
La flessibilità di cui godono ha portato a definire alcune convenzioni, allo scopo di garantire un facile utilizzo delle informazioni:

- utilizzare singoli campi per i dati, usare campi composti per informazioni legate tra loro, sfruttare gli array per liste di valori;

- mantenere semplicità e coerenza nel formato tra i documenti, ad esempio usando lo stesso nome per i tipi di dato che contengono lo stesso tipo di informazione. A questo scopo, CouchDB dispone di validation function per controllare le informazioni all'interno di documenti inseriti o aggiornati;

- stabilire se usare un unico tipo di documento con tutti i dati, o più tipi di documento che poi verranno combinati assieme con le funzioni di aggregazione. In questo caso, è conveniente aggiungere un campo denominato type o schema, che consentirà di saper interpretare meglio il documento quando lo si andrà ad analizzare.

JSON (JavaScript Object Notation)

JSON è un formato semplice e chiaro, basato su un sottoinsieme della sintassi di JavaScript, facile da generare e analizzare con un elaboratore. In ambito Web è utilizzato per lo scambio di informazioni in applicazioni client-server, il MIME ufficiale che lo identifica è application/json.
CouchDB utilizza JSON per ogni comunicazione contenente strutture dati: i documenti, le richieste da parte del client, le informazioni di configurazione e i design documents, sono tutti rappresentati in questo formato.
JSON supporta gli stessi tipi di dati di JavaScript:

numbers: interi positivi e negativi, numeri in virgola mobile e in notazione scientifica;

boolean: possono assumere i valori true o false;

string: caratteri Unicode racchiusi tra doppie virgolette;

array: liste di valori separati da virgole e racchiusi da parentesi quadre. Ad esempio [“friday” , “saturday” , “sunday”]. Possono contenere qualsiasi tipo di dato, inclusi altri array;

object: lista di coppie chiave/valore, la chiave deve essere una stringa, mentre il valore può essere qualsiasi tipo di dato supportato. Ad esempio:

```
1        {
2            "title": "Eggs with bacon",
3            "cooktime": 4,
4            "ingredients": ["two eggs", "bacon"]
5        }
```

null: è possibile avere valori null. Ad esempio { “name” : null } .

2.2.2 Il protocollo HTTP

In CouchDB, ogni comunicazione tra client e server avviene tramite Hyper-Text Transfer Protocol (HTTP), un protocollo del livello applicazione che viene usato generalmente per la trasmissione di informazioni sul Web.

I metodi HTTP

HTTP possiede un'insieme di metodi che definiscono il tipo di operazione da eseguire, di cui CouchDB sfrutta solamente i seguenti:

GET: permette di accedere a una risorsa specifica, fornendone l'URL. In CouchDB è usato per richiedere database, documenti, design document, configurazioni e informazioni statistiche;

HEAD: permette di ottenere l'intestazione HTTP di una richiesta GET. In CouchDB consente di accedere ai metadati dell'oggetto richiesto. Ad esempio, per avere informazioni su un documento senza recuperare il documento stesso;

POST: consente di inviare dati all'URL specificato. All'interno di CouchDB è usato per inserire documenti il cui nome verrà generato automaticamente dal server, oppure per eseguire determinati comandi di amministrazione;

PUT: permette di effettuare l'upload di un file sul server. Usato da CouchDB per creare nuovi oggetti (quali database, documenti, design documents), con il nome specificato nell'URL;

DELETE: permette di cancellare una risorsa sul server. In CouchDB è usato per rimuovere database, documenti e design documents;

COPY: metodo non appartenente al protocollo standard, usato per copiare oggetti. CouchDB lo sfrutta per replicare i dati.

Gli header HTTP

Gli header HTTP contengono numerose informazioni per la trasmissione. In particolare, sono fondamentali per specificare il formato dei dati, consentendone perciò la corretta interpretazione.
Nei messaggi di richesta, gli header contengono a questo scopo i seguenti campi:

- content-type: specifica, in standard MIME, il tipo di contenuto dell'informazione nella richiesta;
- accept: specifica una lista di tipi MIME che il client è in grado di supportare. È un parametro opzionale, utile per garantire robustezza nel funzionamento.

Nei messaggi di risposta, gli header contengono in particolare i campi seguenti:

- content-type: specifica il tipo di MIME dei dati ritornati;

- cache-control: continene indicazioni utili al meccanismo di cache dei dati. Se il contenuto è must-revalidate, significa che l'informazione deve essere riconvalidata nella cache poiché è stata aggiornata;
- content-length: indica la lunghezza, in byte, del contenuto ritornato;
- etag: usato per contenere il numero di revisione di un documento.

Formattazione dei percorsi URL

All'interno di CouchDB, la struttura dei percorsi URL è formata da due parti, standardizzate secondo alcune regole.
Nella prima parte:

- ogni componente che ha il prefisso underscore, è usato per accedere a funzionalità interne o riguardanti l'intero sistema. Ad esempio, _uuids ritorna la lista di tutti gli UUID presenti in CouchDB;
- se il prefisso non contiene underscore, allora è il nome di un database.

Nella seconda parte:

- se inizia con underscore, allora può essere una funzionalità speciale riguardante il database specificato nella prima parte. Ad esempio, _compact consente la compressione di un database. Altrimenti si vuole accedere a un design document e recuperare informazioni dalle viste o da altri contenuti dinamici;
- se non contiene underscore, è il nome di un documento. Ogni ulteriore parte dell'URL si riferisce al documento stesso, per esempio ad un'allegato.

2.2.3 Replicazione dei dati

L'esigenza di avere copia delle informazioni, sorge in varie situazioni:

- la necessità di avere un database condiviso tra vari dispositivi, o tra utenti lontani geograficamente;
- disporre di una copia locale dei dati permette di potervi accedere sempre, sia in lettura che in scrittura, anche in assenza di connessione. La GUI (Graphical User Interface) ottiene dei rapidi tempi di risposta, mentre la sincronizzazione con gli altri nodi avviene in background;
- rimuovere il single point of failure, causato dalla presenza di un singolo server che per problemi hardware o software potrebbe interrompere il proprio servizio. Il sistema è reso quindi fault-tolerant;

- la replicazione è la tecnica fondamentale per fornire scaling. Distinguiamo tre tipi di scaling:

    1. scaling nelle richieste di lettura: quando un server, che può gestire un limitato numero di richieste di lettura, è sovraccarico, viene affiancato da un altro server che possiede la copia dei dati;
    2. scaling nelle richieste di scrittura: quando un server non è più sufficiente per gestire le richieste di scrittura sui dati, viene affiancato da un altro server;
    3. scaling sui dati: un server può contenere una certa quantità di dati. Superato tale limite, i dati devono essere suddivisi in parti e ogni porzione deve essere posta in un server distinto. L'insieme di questi server, che assieme possiedono tutte le informazioni, viene denominato cluster.

CouchDB soddisfa tutte queste esigenze. La replicazione delle informazioni è un processo unidirezionale, avviene cioè da un database sorgente a uno destinazione. È di tipo incrementale, consentendo, in caso di interruzione dell'operazione, di poter riprendere dal punto in cui il processo era stato interrotto. Inoltre, attraverso dei filtri, è possibile scegliere quali documenti replicare.
L'operazione è svolta in maniera efficiente, trasferisce solo i dati necessari per sincronizzare i database. CouchDB, infatti, confronta i due database per trovare quali documenti sono differenti, dopodiché trasmette solamente i documenti che hanno una nuova versione aggiornata. Questo grazie al numero di sequenza di ogni database, incrementato ad ogni cambiamento, e alla capacità di CouchDB di ritornare una lista di documenti aggiunti o aggiornati rispetto a un dato numero di sequenza precedente.
Considerati i database A, B, C, D ed E, che possono risiedere indifferentemente nello stesso server o in server diversi, i possibili scenari di replicazione sono:

- replicazione da A a B una volta soltanto, oppure sincronizzazione continua;
- replicazione da A a B e da B ad A continua;
- replicazione da A a B, a C, a D, a E, ad A;
- replicazione tra A, B, C, D ed E;
- replicazione tra A, B, C e D ad E.

Eventual consistency

Un database distribuito è costituito da nodi, sparsi nella rete, e collegamenti tra di essi. Ogni nodo, dopo che i propri dati sono stati modificati, propaga le modifiche negli altri nodi per mantenere la coerenza dei dati. Le connessioni tra i nodi, tuttavia, possono venire a mancare a causa di problemi nella rete, compromettendo perciò il funzionamento del sistema.
CouchDB interviene dando priorità alla disponibilità delle informazioni, consentendo ad un singolo server di continuare a fornire operazioni di lettura e scrittura sui dati. Al ripristino del collegamento, i dati saranno sincronizzati con gli altri server.
Secondo il teorema CAP, CouchDB rispetta le proprietà di disponibilità e tolleranza al partizionamento, ed è in cambio eventualmente coerente.

Gestione dei conflitti

I conflitti sui dati accadono quando, in presenza di informazioni replicate, lo stesso documento viene modificato in nodi diversi e, al momento della sincronizzazione, sorge la scelta di quale versione scegliere.
CouchDB rileva l'evento e lo segnala ponendo nel documento l'attributo {"_conflicts": true}. Dopodiché, con un algoritmo deterministico, ogni nodo è in grado di scegliere autonomamente la stessa versione del documento, mantenendo comunque quella scartata in caso la si voglia ripristinare. In questo modo, al termine tutti i nodi partecipanti hanno lo stesso documento, garantendo quindi uno stato coerente del dataset, e CouchDB è in grado di continuare a fornire risposte a richieste sul documento interessato.

Per analizzare il funzionamento di CouchDB in caso di conflitti, creare i database recipes e recipes-replica:

```
curl -X PUT http://127.0.0.1:5984/recipes
curl -X PUT http://127.0.0.1:5984/recipes-replica
```

Inserendo un documento nel database recipes:

```
curl -H 'Content-Type: application/json' \
-X PUT http://127.0.0.1:5984/recipes/eggsbacon \
-d '{"cooktime":3}'
```

Il server ritorna il documento:

```
{"ok":true,"id":"eggsbacon","rev":"1-b875366d104111e9da3060433222cb97"}
```

Successivamente, effettuare la replica di recipes in recipes-replica:

```
curl -X POST http://127.0.0.1:5984/_replicate \
-d '{"source":"recipes","target":"recipes-replica"}'
{"ok":true,...,"docs_written":1,"doc_write_failures":0}
```

Aggiornando il documento in recipes-replica con {"cooktime": 5}:

```
curl -H 'Content-Type: application/json' \
-X PUT http://127.0.0.1:5984/recipes-relica/eggsbacon \
-d '{"cooktime":5, "_rev":"1-b875366d104111e9da3060433222cb97"}'
{"ok":true,"id":"eggsbacon","rev":"2-ea292eccb3b43957279db046b6b80c37"}
```

Modificare il documento anche nel database recipes, ponendo {"cooktime": 4}:

```
curl -H 'Content-Type: application/json' \
-X PUT http://127.0.0.1:5984/recipes/eggsbacon \
-d '{"cooktime":4, "_rev":"1-b875366d104111e9da3060433222cb97"}'
{"ok":true,"id":"eggsbacon","rev":"2-2652495d506ba6cad9ac4eb56cca344d"}
```

Infine, per creare il conflitto sui dati, si ripete la replica :

```
curl -X POST http://127.0.0.1:5984/_replicate \
-d '{"source":"recipes-replica","target":"recipes"}'
{"ok":true,. . . ,"docs_read":1,"docs_written":1,"doc_write_failures":0}
```

CouchDB sceglie come vincitrice la versione contenente {"cooktime": 5} e avente {"rev": "2-ea292eccb3b43957279db046b6b80c37"}, rispetto a quella con {"cooktime": 4} e {"rev": "2-2652495d506ba6cad9ac4eb56cca344d"}. Per risolvere il conflitto, ognuno dei due database ha eseguito un algoritmo deterministico che sceglie:

- la versione che ha avuto il maggior numero di aggiornamenti;
- altrimenti, in caso di parità, quella che precede l'altra secondo l'ordine ASCII dei numeri di revisione.

Se la versione scelta non era quella corretta, è possibile rimediare aggiornando quella desiderata con i dati che contiene ed eliminando l'altra:

```
curl -X PUT  http://127.0.0.1:5984/recipes-relica/eggsbacon \
-d '{"cooktime":4,"_rev":"2-2652495d506ba6cad9ac4eb56cca344d"}'
curl -X DELETE \
http://127.0.0.1:5984/recipes-relica/eggsbacon? \
rev= 2-ea292eccb3b43957279db046b6b80c37
```

2.3 Le API di CouchDB

Le API, fornite dal DBMS per potervi interagire, possono essere suddivise in quattro categorie principali:

1. Server
2. Databases
3. Documents

4. Replication

Verranno di seguito analizzate utilizzando cURL, uno strumento a riga di comando per il trasferimento di dati con la sintassi URL e che supporta HTTP.

2.3.1 Server API

Per avere informazioni sull'istanza di CouchDB in esecuzione, o semplicemente per assicurarsi che sia presente, digitare:

```
curl http://127.0.0.1:5984
```

Nell'URL è specificato l'indirizzo IP del server e la porta, che per default è la 5984. In caso di successo, il server risponde con una stringa in formato JSON contenente varie informazioni, ad esempio:

```
{"couchdb":"Welcome","version":"1.3.1"}
```

2.3.2 Databases API

CouchDB è un DBMS, ed in quanto tale è in grado di gestire più database al suo interno.

Creare un database

Per creare un database occorre usare l'opzione -X, che consente di usare il metodo PUT per comunicare con il server, e specificare il nome del database nella seconda parte dell'URL:

```
curl -X PUT http://127.0.0.1:5984/recipes
```

Se l'operazione ha esito positivo, il server risponde con:

```
{"ok":true}
```

Informazioni su un database

Ottenere informazioni su un database, specificato nell'URL, è possibile con il metodo GET:

```
curl -X GET http://127.0.0.1:5984/recipes
```

Il server ritorna una stringa contenente varie informazioni, tra cui il numero di documenti contenuti, la dimensione dei dati e il momento in cui è stato creato. In questo modo, inoltre, è possibile constatare l'esistenza del database.

Eliminare un database

Per eliminare un database, si deve fornire il percorso e usare il metodo DELETE:

```
curl -X DELETE http://127.0.0.1:5984/recipes
```

Se l'operazione ha successo, il server risponde ancora con:

```
{"ok":true}
```

2.3.3 Documents API

Il documento è la struttura dati centrale di CouchDB, possiede un identificatore univoco e un numero di revisione che dipende dagli aggiornamenti subiti. È possibile accedere solo ai documenti del database utilizzato in un dato momento.
Le operazioni possibili sui documenti sono:

1. inserire un documento;
2. ottenere un documento;
3. aggiornare un documento;
4. eliminare un documento.

Inserire un documento

Per inserire un nuovo documento, senza specificarne l'identificatore, si usa il metodo POST, mentre con l'opzione -d si indica il documento da inserire nel body della richiesta HTTP:

```
curl -H 'Content-Type: application/json' \
-X POST http://127.0.0.1:5984/recipes \
-d '{"title": "Lasagne"}'
```

La risposta del server contiene l'identificatore e il numero di revisione del documento, generati automaticamente. Entrambi i parametri sono degli Universally Unique Identifier (UUID), numeri casuali aventi una probabilità di ripetersi molto bassa.
Nell'eventualità si voglia specificare l'identificatore, deve essere inserito alla fine dell'URL e si deve usare il metodo PUT:

```
curl -H 'Content-Type: application/json' \
-X PUT http://127.0.0.1:5984/recipes/lasagne \
-d '{"title": "Lasagne"}'
```

Ottenere un documento

Usando il metodo GET e specificando l'identificatore nell'URL, è possibile ottenere un documento:

```
curl -X GET http://127.0.0.1:5984/recipes/lasagne
```

Il server ritorna il documento con tutti i suoi campi, l'identificatore e il numero di revisione.

Aggiornare un documento

Per aggiornare un documento non si possono modificare solo alcuni campi, ma è necessario aggiornarlo per intero. Si deve definire una nuova versione, riscrivendo tutti i campi, mantenendo solo l'identificatore originale. È indispensabile specificare anche il numero di revisione, al fine di aggiornare l'informazione più recente:

```
curl -H 'Content-Type: application/json' \
-X PUT http://127.0.0.1:5984/recipes/lasagne \
-d '{"title": "Lasagne␣al␣forno", \
"␣rev":"1-f07d272c69ca1ba91b94544ec8eda1b6"}'
{"ok":true,"id":"lasagne","rev":"2-77b8d2ee630bd017122ea2fe0b10a8b4"}
```

In caso di successo, il server ritorna un documento con {"ok": true}, l'identificatore e il nuovo numero di revisione.
I documenti possono contenere allegati di qualsiasi tipo, come immagini, musica o film. Gli allegati sono identificati da un nome e dal tipo di contenuto, per consentirne la corretta interpretazione.
Ad esempio, per aggiungere l'immagine image.jpg, salvata nella directory corrente, al documento creato in precedenza:

```
curl -X PUT http://127.0.0.1:5984/recipes/lasagne \
image.jpg?rev=2-77b8d2ee630bd017122ea2fe0b10a8b4 \
--data-binary @image.jpg -H 'Content-Type: image/jpg'
```

L'opzione data-binary consente di inviare i dati esattamente come sono, senza nessuna elaborazione aggiuntiva. Nel documento vengono salvati i metadati relativi all'immagine, quali, ad esempio, il tipo di contenuto e la dimensione in byte.
In seguito, l'immagine è accessibile con l'indirizzo:

```
http://127.0.0.1:5984/recipes/lasagne/image.jpg
```

Eliminare un documento

La cancellazione di un documento avviene con DELETE, specificando il percorso e il numero di revisione:

```
curl -H 'Content-type: application/json' -X DELETE \
http://127.0.0.1:5984/recipes/lasagne? \
rev=2-77b8d2ee630bd017122ea2fe0b10a8b4
{"ok":true,"id":"lasagne","rev":"3-3ba3659cc3189cc87bb070cf5568ea39"}
```

Il server, nonostante l'eliminazione, aggiorna il numero di revisione. Questo perché in presenza di dati replicati è indispensabile sapere che un documento è stato rimosso in un nodo.

2.3.4 Replication API

La replicazione dei dati è effettuata usando il metodo POST, definendo un database sorgente, di cui fare la replica, e un database destinazione, su cui copiare i dati. I database sorgente e destinazione possono essere in locale o in remoto, i casi rilevanti sono i seguenti:

- sorgente in locale e destinazione in remoto, detta push replication;
- sorgente in remoto e destinazione in locale, detta pull replication;
- sorgente e destinazione in locale, per funzioni di backup del database.

In quest'ultimo caso, dapprima si crea un database destinazione:

```
curl -X PUT http://127.0.0.1:5984/recipes-replica
```

Dopodiché, si può usare recipes-replica come destinazione per l'operazione:

```
curl -X POST http://127.0.0.1:5984/_replicate \
-d '{"source":"recipes","target":"recipes-replica"}'
```

L'operazione può richiedere tempo e mantiene aperta la connessione tra sorgente e destinazione per tutta la durata, fatto che può essere rilevato entrando in modalità verbose con l'opzione -v.
Aggiungendo {"continuous": true} è possibile mantenere attivo il processo di replicazione: quando avvengono modifiche nel database sorgente, queste, in un momento di massima efficienza scelto da CouchDB, vengono trasmesse in quello di destinazione. Ad esempio:

```
curl -X POST http://127.0.0.1:5984/_replicate \
-d '{"source":"recipes", "target":"recipes-replica", "continuous":true}'
```

2.4 Design Document

I design documents sono tra le componenti più importanti di CouchDB, il loro uso è di fondamentale importanza per l'utilizzo delle informazioni salvate. Consentono di imporre un controllo sulla struttura dei documenti, gestire la convalida dei documenti, ricercare e recuperare informazioni nel

database.
Sono costituiti da documenti contenenti codice JavaScript, o in qualsiasi altro linguaggio supportato da CouchDB, ad esempio Erlang, che vengono eseguiti all'interno del server. Possiedono, come ogni altro documento, un identificatore univoco e un numero di revisione, vengono replicati tra i database. Ogni database può avere nessun, uno o più design document, che devono essere collocati nell'area apposita _design.
Design document comprende i seguenti tipi di funzioni:

view functions: sono delle funzioni che permettono di accedere ai documenti e di ricercare informazioni;

show functions: a partire da un singolo documento, convertono il contenuto in un'altro formato, come HTML, JSON, XML;

list functions: analogamente alle show functions, converte il contenuto di una view in vari tipi di formati;

document validation: interviene quando viene aggiunto o aggiornato un documento, per controllarne il contenuto;

update handlers: quando un documento viene aggiornato, devono essere esplicitamente chiamati per compiere delle azioni sul documento. Usati, ad esempio, per incrementare valori, aggiungere timestamp;

filters: consentono di filtrare le informazioni che vengono replicate tra database.

Un design document può contenerne vari tipi, ad esempio:

```
{
    "language" : "javascript",
    "_id" : "_design/recipes_d",
    "_rev" : "1-b41a673ce62351a2c629734d4dc220f9",

    "views": {...},
    "shows": {...},
    "lists": {...},
    "validate_doc_update": "function( newDoc, oldDoc, userCtx)
       {...}"
}
```

Supponendo di salvare il design document nel file recipes_desing.js, per caricarlo nel database si usa il metodo PUT e gli si assegna il nome recipes_d:

```
curl -X PUT -d @recipes_design.js \
'http://localhost:5984/recipes/_design/recipes_d'
```

Le operazioni di recupero, aggiornamento e cancellazione si effettuano, come per ogni documento, usando rispettivamente i metodi GET, PUT e DELETE.

2.4.1 Show functions

A partire da un documento, questo tipo di funzioni permettono di convertire il contenuto in vari formati, ad esempio HTML. Sono side effect free, non apportano cioè nessuna modifica ai dati, ed il loro output può essere salvato in memorie cache, in modo da alleggerire di richieste il server che ospita CouchDB. Un design document può contenere più di una show function.
La funzione ha il prototipo function(doc, req) {...} e riceve in ingresso due parametri:

1. il documento da processare;
2. l'oggetto che ha generato la richiesta, contenente tutte le informazioni della richiesta iniziale (quali l'header HTTP, l'indirizzo del client, i parametri dell'URL). In base a queste informazioni si possono compiere scelte all'interno della funzione.

Ritorna in output i dati sotto forma di stringa, o in alcuni formati, tra i quali HTML, XML, CSV.
Considerando, ad esempio, il design document recipes_d, vi si aggiunge la seguente show function:

```
"shows": {
    "recipe" : "function (doc,req) {
        return '<h1>' + doc.title + '</h1>'
    }"
}
```

L'accesso avviene specificando, nell'ultima parte dell'URL, il nome del documento su cui applicare la funzione:

```
curl -X GET \
http://127.0.0.1:5984/recipes/_design/recipes_d/_show/recipe/Lasagne
```

2.4.2 Views

Le views sono uno strumento fondamentale per la ricerca e l'aggregazione di informazioni all'interno di un database.
Gli usi principali delle views sono i seguenti:

- fornire indicizzazione per la ricerca efficiente di documenti;

- creare tabelle e liste di informazioni che riassumono i dati dei documenti;
- estrarre o filtrare informazioni salvate nei documenti;
- effettuare calcoli su informazioni presenti in un insieme di documenti.

CouchDB consente di definirne più di una all'interno del campo "views" di design document. Per la sua robustezza, CouchDB tenta di creare una view nonostante possano essere presenti errori nel contenuto o nel formato dell'output. Gli errori vengono comunque salvati nel file di log, accessibile tramite il percorso http://host_address:5984/_log, oppure possono essere monitorati con il metodo log(), che invia informazioni alla view e le inserisce nel file di log.

Funzioni Map e Reduce

Una view è composta da due funzioni: la funzione Map e la funzione Reduce.
La funzione Map itera su ogni documento del database e ne mappa le informazioni contenute, emettendo per ciascuno:

- automaticamente l'dentificatore;
- una chiave e un valore con il metodo emit(key, value).

Ad esempio, la seguente funzione Map, denominata "by_title", emette, per ogni documento che possiede un titolo, una riga avente per chiave il titolo e per valore null:

```
{
    "views": {
        "by_title": {
            "map": "function(doc) {
                if (doc.title != null)
                    emit(doc.title, null)
            }"
        }
}
```

L'accesso avviene tramite il seguente URL:

`http://127.0.0.1:5984/recipes/_design/recipes_d/_view/by_title`

La funzione Reduce riceve in ingresso le righe generate dalla funzione Map, le raggruppa in base al valore della chiave ed esegue quindi delle operazioni di riduzione. Ritorna, infine, un'insieme di chiavi e valori associati, in cui la chiave è quella del raggruppamento e il valore è il risultato della funzione

di riduzione.
CouchDB dispone di alcune funzioni Reduce già implementate:

_count: fornisce il numero di righe del raggruppamento;

_sum: somma i valori di ogni riga, che devono essere necessariamente dei tipi numerici;

_stats effettua calcoli statistici su valori numerici, tra i quali la somma (sum), il conteggio (count), il valore minimo (min), il valore massimo (max), e la somma quadratica (sumsqr).

Nella seguente view, ad esempio, vengono mostrate le ricette in base alle parole chiave e successivamente viene fatto il loro conteggio con la funzione _count:

```
"by_keyword": {
    "map": "function(doc) {
        if (doc.keywords) {
            for(i=0;i<doc.keywords.length;i++){
                emit(doc.keywords[i], null);
            }
        }
    }",
    "reduce": "_count"
}
```

È altresì possibile definire delle funzioni Reduce personalizzate, in questo caso la funzione riceve in ingresso tre parametri:

1. key, un array di coppie key e document ID generati dalla funzione Map;
2. values, un array di valori generati dalla funzione Map e che appartengono ai documenti identificati dagli elementi key;
3. rereduce, un valore booleano che indica quando la funzione Reduce richiama sé stessa.

Nel seguente esempio è stata ricreata la funzione _sum:

```
function(keys,values,rereduce)
{
    var sum = 0;
    for(var idx in values) {
        sum = sum + values[idx];
    }
    return sum;
}
```

Gli indici

La costruzione di una view implica la creazione di un indice, questo consente di velocizzare le operazioni di recupero delle informazioni.
Gli indici si basano sui B-Tree e contengono le informazioni fornite in output dalle funzioni Map e Reduce, cosicché possono essere riutilizzate senza dover essere ricalcolate di volta in volta.
Durante il primo accesso ad una view, l'intero indice viene costruito. Se, successivamente, avvengono cambiamenti nei documenti, l'indice viene aggiornato al suo primo utilizzo e solamente nelle parti in cui sono accaduti dei cambiamenti. L'aggiornamento coinvolge tutte le views presenti, anche se non sono ancora state utilizzate.
Questo tipo di aggiornamento differito garantisce l'efficienza del database quando avvengono operazioni sui documenti.

Interrogazione delle views

I risultati forniti dalle views, che sono automaticamente ordinati crescentemente secondo la codifica UTF-8, possono essere manipolati specificando nell'URL alcuni parametri aggiuntivi.
Tra questi, i principali sono i seguenti:

group_level: specifica il livello del raggruppamento. Per default vale 0, avviene cioè riga per riga. Usando un array di chiavi, con 1 il raggruppamento è sul primo elemento, con 2 sul primo e il secondo, e così via;

startkey e endkey: il primo consente di definire l'elemento da cui iniziare a ritornare i risultati, il secondo l'ultimo elemento da ritornare. Usandoli assieme è possibile definire un intervallo di elementi. Ad esempio, per avere le ricette il cui titolo inizia solamente con "Apricot":

```
http://127.0.0.1:5984/recipes/_design/recipes_d/_view/
by_title?startkey=%22Apricot%22&endkey=%22Apricot%007F%22
```

key: permette di specificare la key degli elementi da ritornare. Ad esempio, per avere le ricette che hanno per key "lemon sole":

```
http://127.0.0.1:5984/recipes/_design/simple/_view/
by_ingredient?key=%22lemon%20sole%22
```

descending: specificando **descending=true** i risultati vengono ritornati in ordine decrescente;

limit: consente di specificare il numero di righe contenute nei risultati;

skip: consente di definire il numero di righe, a partire dall'inizio, da non includere nei risultati;

stale: quando si accede ad una view, per evitare il ritardo introdotto dall'aggiornamento dell'indice in seguito a modifiche sui documenti, è possibile specificare **stale=true**. In questo modo si accede alle informazioni obsolete dell'indice già presente.

Views temporanee

Per testare il funzionamento delle views è conveniente utilizzare le views temporanee. Queste consentono di inviare il contenuto della view nel body HTTP, usando il percorso speciale _temp_view. L'esempio di inizio paragrafo diviene:

```
curl -X POST -H 'Content-type: application/json' \
http://127.0.0.1:5984/recipes/_temp_view \
-d '{"map": "function(doc) { if (doc.title != null) \
emit(doc.title,null)}"}'
```

Richiedono l'immediata esecuzione e non utilizzano indici, questo le rende inefficienti e quindi sconvenienti da utilizzare per scopi diversi dal testing.

2.4.3 List functions

La list function trasforma ogni riga generata da una view in un tipo di formato, ad esempio HTML, XML, JSON, CSV. Ogni design document può averne una o più, contenute all'interno dell'oggetto con chiave "list".
Il prototipo della funzione è function(head, req), i parametri di ingresso sono:

- head, contiene informazioni sulla view oltre i dati delle righe (i. e., il numero totale di righe, il numero di quelle saltate e l'offset);
- req, contiene tutte le informazioni sulla richiesta (ad esempio, lo header HTTP, informazioni su client e server, il percorso della richiesta).

Per esemplificare, la seguente list function crea un' elenco di titoli di ricette con il link alla ricetta specifica, visualizzata tramite la show function recipe:

```
1 "lists": {
2     "list_title": "function (head,req) {
3         start({ "headers": {
4                 "Content-type" : "text/html"
5             }
6         });
7         send('<ul>');
```

```
        var row;
        while (row = getRow()) {
            send('<li><a href="/recipes/_design/detail/_show/
               recipe/'+ row.id + '\">' + row.key + '</a></li
               >');
        }
        send('</ul>');
    }"
}
```

La funzione start è usata per mandare in output lo header HTTP, specificando il tipo di contenuto come HTML. La list function viene chiamata sul gruppo di righe, ciascuna accessibile con getRow(), iterate con un ciclo while.
Per accedervi, specificare nell'ultima parte dell'URL il nome della list function e della view da cui riceve i dati:

```
http://127.0.0.1:5984/recipes/_design/recipes_d/_list/list_title/by_title
```

2.4.4 Validation functions

Le validation functions vengono eseguite nel momento in cui un documento nuovo viene aggiunto o un documento salvato viene aggiornato, per controllarne il contenuto, riformattare i dati, e impedire operazioni non autorizzate.
Il loro scopo è di imporre delle regole di coerenza sui dati, a causa della flessibilità che JSON concede nella costruzione di documenti.
Vengono usate, ad esempio, per semplificare le funzioni MapReduce, in modo che non debbano più controllare in ogni documento la presenza di determinati campi.
Ne può essere definita una per ogni design document e all'interno del campo validate_doc_update. Nell'eventualità siano presenti più validation functions, vengono eseguite tutte e ogni operazione deve superare tutti i controlli per essere salvata.
Esemplificando, la seguente funzione controlla la presenza del titolo e degli ingredienti in una ricetta:

```
function(newDoc, saveDoc, userCtx){
    function require(beTrue, message){
        if(!beTrue) throw({forbidden : message});
    };

    if(type=='recipe'){
        validate([
```

```
8           newDoc.title,       "Recipe must have a title",
9           newDoc.ingredients, "Recipe must have a ingredient"
10          ]);
11      }
12  }
```

I parametri di ingresso sono:

1. newDoc, il documento nuovo aggiornato o aggiunto;
2. saveDoc, il documento già presente nel caso di un'operazione di aggiornamento;
3. userCtx, l'utente e i ruoli che possiede.

Se non sono presenti il titolo o gli ingredienti, viene lanciata un'eccezione e inviato un messaggio recante il motivo per il quale l'operazione è stata bloccata.

Capitolo 3

Realizzazione di un database in CouchDB e PostgreSQL

3.1 Introduzione

In questo capitolo è affrontata la progettazione e l'implementazione di una base di dati per un blog, prendendo spunto dall'esempio presente a pagina 32 in MC Brown (2012) *Getting started with CouchDB*, USA, O'Reilly Media.
Il blog si occupa di ricette, ognuna composta di titolo, sottotitolo, tempo di preparazione, tempo di cottura, numero di porzioni, ingredienti, procedimento e data di pubblicazione. Le ricette possono essere commentate dagli utenti, che devono specificare il proprio nome e indirizzo email.
Nella prima parte del capitolo, il database viene realizzato con CouchDB tramite la struttura basata sui documenti; nella seconda parte, lo stesso database viene realizzato con il DBMS relazionale PostgreSQL.
Al termine, verranno analizzate le differenze tra i due approcci adoperati e, tramite il processo di denormalizzazione, il database relazionale raggiungerà alcune delle proprietà di quello orientato ai documenti.

3.2 Modellazione dell'esperimento con CouchDB

Anzitutto, si è scelto di organizzare le informazioni in due tipi di documenti:

1. documento ricetta, rappresenta una ricetta ;
2. documento commento, rappresenta un commento ad una ricetta.

Questa suddivisione semplifica la struttura interna dei documenti, consentendo una migliore leggibilità e interpretazione dei contenuti. Inoltre, separando i commenti in un documento a parte, è ridotto il numero di modifiche

che accadrebbero se fosse utilizzato un unico documento per contenere una ricetta e i relativi commenti.
Viene quindi creato il database recipes con il comando:

```
curl -X PUT http://127.0.0.1:5984/recipes
```

3.2.1 Documento ricetta

Il documento ricetta è costituito da un identificatore stabilito dall'utente, dai campi title, subtitle, ingredients, cooktime, preparationtime, publicationdate, servings, directions e type.
Quest'ultimo, che facilita l'utilizzo del documento perché permette di conoscere le informazioni contenute, è posto al valore {"type": "recipe"}.
Gli ingredienti sono stati salvati in un array di oggetti, dato che un ingrediente e la sua quantità costituiscono un tipo di dato composto. Il procedimento è stato salvato in un array in modo da preservare l'ordine di esecuzione delle varie fasi.

Codice 3.1: Esempio di documento ricetta

```
{
    "_id": "baconeggs",
    "title": "Bacon and eggs",
    "subtitle": "Bacon and eggs casserole recipe",
    "ingredients": [
        {
            "ingredient": "bacon",
            "measure": "4 strips"
        },
        {
            "ingredient": "eggs",
            "measure": "18"
        },
        {
            "ingredient": "milk",
            "measure": "1 cup"
        },
        {
            "ingredient": "shredded cheese",
            "measure": "1 cup"
        },
        {
            "ingredient": "sour cream",
            "measure": "1 cup"
        },
```

```
26          {
27              "ingredient": "sliced green onions",
28              "measure": "1/4 cup"
29          },
30          {
31              "ingredient": "salt",
32              "measure": "1 teaspoons"
33          },
34          {
35              "ingredient": "pepper",
36              "measure": "1/2  teaspoon"
37          }
38      ],
39      "publicationdate": "2013-08-12T17:25",
40      "cooktime": 40,
41      "preparationtime": 20,
42      "servings": 8,
43      "directions": [
44          "In a large skillet, cook bacon over medium heat until
               crisp. Remove to paper towel to drain.",
45          "In a large bowl, beat eggs. Add milk, cheese, sour
               cream, onions, salt and pepper.",
46          "Pour into a greased 13-in. x 9-in. baking dish.
               Crumble bacon and sprinkle on top. Bake, uncovered,
               at 325 grades for 40-45 minute or until knife
               inserted near the center comes out clean. Let stand
               for 5 minutes."
47      ],
48      "type": "recipe"
49  }
```

Supponendo che il documeto in Codice 3.1 sia salvato nel file bacon_eggs.js, per inserirlo nel database digitare il comando:

```
curl -H 'Content-Type: application/json' \
-X PUT http://127.0.0.1:5984/recipes/baconeggs \
-d @bacon_eggs.js
```

Il simbolo @ indica di inserire il contenuto del file bacon_eggs.js nel body della richiesta HTTP.

3.2.2 Documento commento

Il documento commento comprende i campi author, email, comment, datetime, type e recipedocid.

L'attributo che ne identifica il tipo è posto al valore {"type": "comment"}, mentre l'attributo **recipedocid** contiene l'identificatore della ricetta alla quale il commento si riferisce. In questo caso l'identificatore è creato automaticamente dal DBMS.

Codice 3.2: Esempio di documento commento

```
{
    "author": "Bob",
    "email": "bob@couchdb.com",
    "comment": "Good recipe!",
    "datetime": "2013-08-12T20:25",
    "type": "comment",
    "recipedocid": "baconeggs"
}
```

Supponendo che il documento in Codice 3.2 sia salvato nel file bob_comment.js, per inserirlo nel database utilizzare il comando:

```
curl -H 'Content-Type: application/json' \
-X POST http://127.0.0.1:5984/recipes \
-d @bob_comment.js
```

3.3 Modellazione dell'esperimento con un DBMS relazionale

In questo paragrafo viene realizzato con PostgreSQL lo stesso database realizzato precedentemente con CouchDB. Saranno percorse le classiche fasi per la realizzazione di un database relazionale:

1. raccolta dei requisiti strutturati;
2. progettazione concettuale e relativo schema concettuale;
3. progettazione logica e relativo schema logico;
4. progettazione fisica.

3.3.1 Requisiti strutturati

In questa fase vengono individuate le entità, che rappresentano un oggetto fisico o un concetto del mondo reale [1], e le proprietà che le descrivono.

Frasi per Ricetta

Ogni ricetta è pubblicata in una data e orario da memorizzare, ha un titolo, sottotitolo, tempo di preparazione, tempo di cottura, numero di porzioni e una lista di ingredienti. Per la sua preparazione occorre seguire un procedimento per fasi.

Frasi per Commento

Di ogni commento si devono memorizzare il momento in cui è rilasciato, il testo, il nome dell'autore e il suo indirizzo email. Un commento è riferito ad un'unica ricetta e ogni ricetta ne può possedere più di uno.

3.3.2 Progettazione concettuale

In questa fase vengono formalizzate le informazioni raccolte nella fase precedente, definendo la struttura ad alto livello della base di dati. Questa è composta dalle entità con i relativi attributi, cioè le proprietà che la caratterizzano, e associazioni [1], cioè i legami tra le entità. Inoltre, vengono definiti i vincoli che non sono esprimibili tramite il modello utilizzato.
In figura 3.1 è rappresentato lo schema Entità-Associazione della realtà di interesse, ottenuto tramite il modello E-R.

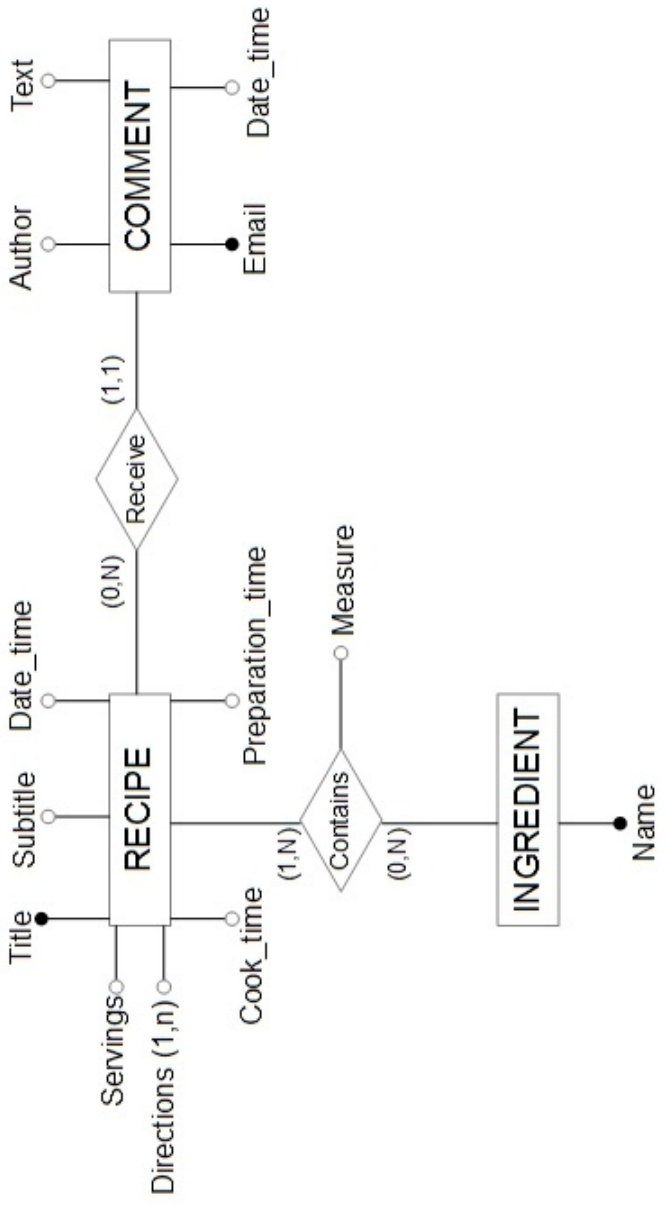

Figura 3.1: Schema E-R di Recipes

A causa dell’attributo multivalore “Directions”, per poter passare alla fase successiva lo schema di figura 3.1 è stato ristrutturato. L’attributo è diventato l’entità “Direction”, collegata con l’entità “Recipe” dall’associazione “Follow”. Il risultato è il nuovo schema E-R di figura 3.2.

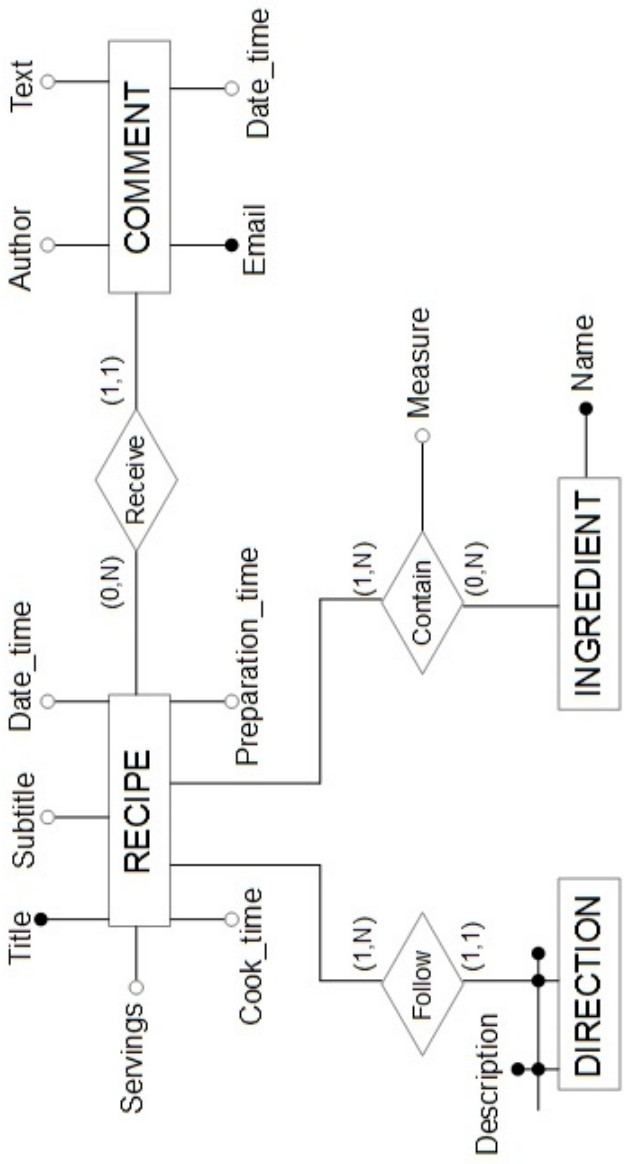

Figura 3.2: Schema E-R ristrutturato di Recipes

È necessario introdurre la seguente regola di vincolo per rappresentare correttamente la realtà di interesse:

- i passaggi da seguire per una ricetta hanno un ordine di esecuzione.

3.3.3 Progettazione logica

Questa fase prevede la traduzione dello schema concettuale nel corrispondente schema logico, utilizzando il modello relazionale che rappresenta la base di dati come una collezione di relazioni matematiche.
Quindi, tramite opportune trasformazioni, dallo schema E-R di figura 3.2 si è ottenuto lo schema logico di figura 3.3.

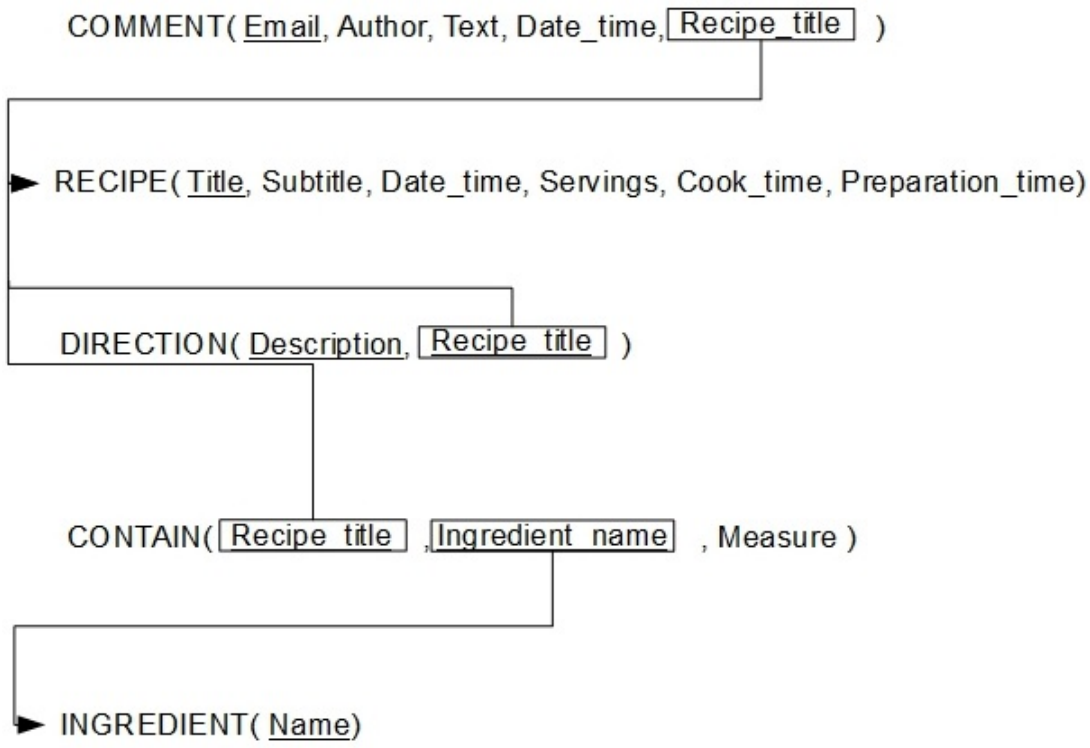

Figura 3.3: Schema logico di Recipes

L'ultimo passaggio, la progettazione fisica, è riportato in Appendice A.

3.4 Confronto tra i due modelli

3.4.1 Differenze di progettazione

La progettazione e l'implementazione del database con il DBMS PostgreSQL ha un modus operandi ben definito, che si è sviluppato e consolidato nei quarant'anni di utilizzo del modello relazionale.
Invece, CouchDB non dispone di un metodo di progettazione preciso, ma si affida a regole empiriche sviluppate col tempo o da scoprire con l'esperienza personale.
Una di queste è la differenziazione dei documenti: questa scelta è dettata soprattutto dall'uso che si fa di queste informazioni. Se devono essere manipolate separatamente, ad esempio se si vuole dare un ordine ai commenti, è conveniente utilizzare un documento separato in modo da semplificare le operazioni.
La possibilità di accedere ad un solo database per volta e il costo computazionale che comporta l'iterazione sui documenti, ha portato alla definizione di un'ulteriore regola: mantenere in un database solamente i dati utilizzati dall'applicazione, ponendo in un database separato tutte le altre informazioni non utilizzate.

3.4.2 Normalizzazione e denormalizzazione

Il database relazionale è stato progettato secondo i principi della normalizzazione, è privo cioè di dati ridondanti e le relazioni sono scomposte in modo tale che ognuna rappresenti un singolo concetto.
Questa proprietà garantisce l'univocità dei dati, che comporta una minore occupazione di memoria. Inoltre, le operazioni di aggiornamento, inserimento e cancellazione, vengono eseguite in un solo punto, facilitando la loro gestione e diminuendo la possibilità di errore sulla coerenza dei dati.
Conservare la normalizzazione è utile in contesti nei quali le prestazioni nelle operazioni di scrittura sono più importanti di quelle in lettura. Se avvengono più frequentemente scritture sui dati, mentre le query avvengono più raramente, allora conviene adottare la normalizzazione.
Invece, nei database denormalizzati [8] viene introdotta volontariamente ridondanza nelle informazioni, ad esempio raggruppando gli attributi appartenenti a relazioni diverse. Permette di migliorare il recupero delle informazioni poiché queste si trovano già in forma aggregata, invece di dover ricostruire le informazioni originarie scomposte precedentemente in relazioni diverse.

Ad esempio, la denormalizzazione è adottata dai DBMS del movimento NoSQL per migliorare la lettura di elevate quantità di informazioni, aspetto ritenuto più importante rispetto all'organizzazione dei dati.
Per contro, la denormalizzazione porta ad un aumento dell'occupazione di memoria a causa dei dati ripetuti, ed un aumento della complessità nelle operazioni di scrittura: la modifica di un'informazione deve essere sincronizzata tra tutte le copie presenti, in modo da mantenere lo stato coerente della base di dati. Questo onere è affidato all'applicazione che ne fa uso, complicandone perciò la realizzazione.

3.4.3 Denormalizzazione del database relazionale

Considerare la seguente query: calcolare per ogni ricetta il numero di ingredienti e il numero di passaggi che la compongono.
La sua realizzazione in CouchDB avviene con la view riportata in Codice 3.3, che consiste nello scorrere la lista di documenti di tipo ricetta e, grazie alla presenza delle informazioni ricercate in ogni singola ricetta, ritornare per ognuno la lunghezza degli array degli ingredienti e dei passaggi.

Codice 3.3: View per il calcolo degli ingredienti e dei passaggi

```
{
    "views": {
        "#_ingredients_and_steps": {
            "map": "function(doc) {
                if(doc.type=="recipe"){
                    var num_ingredients=0;
                    var num_steps=0;
                    if (doc.ingredients)
                        num_ingredients=doc.ingredients.length;
                    if(doc.directions)
                        num_steps=doc.directions.length;
                    emit(doc.title, [num_ingredients, num_steps
                       ]);
                }
            }",
        }
    }
}
```

Invece, la query realizzata con PostgreSQL è riportata in Codice 3.4. Per realizzarla è stato necessario definire le viste "Counts_ingredients", di Codice 3.5, e "Counts_steps", di Codice 3.6.
Per la sua esecuzione sono necessarie tre operazioni di join: uno tra le tabelle "Recipe" e "Contain", un'altro tra "Direction" e "Recipe", l'ultimo tra

le viste “Counts_Directions” e “Count_steps”.
La presenza di operazioni di join, necessaria conseguenza di un database normalizzato, può comportare un rallentamento nell’esecuzione della query, la cui determinazione in termini di tempo è lasciata a ricerche future. Supponendo di dare priorità alla velocità di esecuzione della query e di considerare meno importante l’organizzazione dei dati, accettando che aumenti la complessità delle operazioni di scrittura per mantenere la coerenza delle informazioni e non avendo problemi sulla quantità di memoria necessaria, si può procedere alla denormalizzazione della base di dati relazionale.

Codice 3.4: Query per il calcolo degli ingredienti e dei passaggi

```
SELECT Recipe_title, Ingredients_num, Steps_num
FROM COUNTS_INGREDIENTS AS I JOIN COUNTS_STEPS AS S
ON I.Recipe_title=S.Recipe_title;
```

Codice 3.5: View per il calcolo degli ingredienti

```
CREATE VIEW COUNTS_INGREDIENTS(Recipe_title, Ingredients_num)
AS SELECT R.title, COUNT(*)
FROM RECIPE AS R JOIN CONTAIN AS C ON R.title=C.recipe_title
GROUP BY R.title;
```

Codice 3.6: View per il calcolo dei passaggi

```
CREATE VIEW COUNTS_STEPS(Recipe_title, Steps_num)
AS SELECT R.title, COUNT(*)
FROM RECIPE AS R JOIN DIRECTION AS D ON R.title=D.recipe_title
GROUP BY R.title;
```

Riprogettazione del database PostgreSQL denormalizzato

Per fare a meno dell’uso di join tra tabelle, è stato modificato lo schema concettuale originario di figura 3.1 introducendovi dati ridondanti, ottenendo quello di figura 3.4. All’entità “Recipe” è stato aggiunto un nuovo identificatore di nome ID, un intero autoincrementante per distinguere le tuple. Sono stati poi trasferiti nella suddetta entità gli attributi:

- description, appartenente all’entità “Direction”;
- name, appartenente all’entità “Ingredient”;
- measure, appartenente alla relazione “Contains”.

In seguito, data l’assenza di attributi, sono state tolte le entità “Direction” e “Ingredient”. Di conseguenza anche la relazione “Contains”.

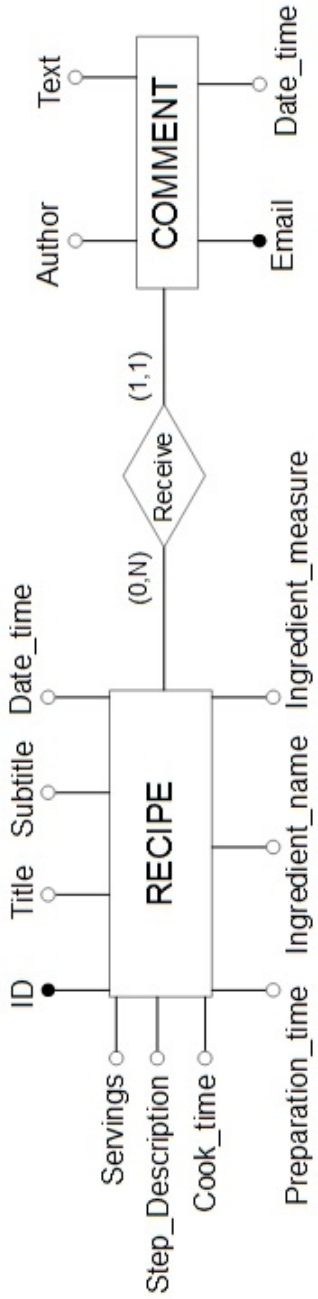

Figura 3.4: Schema E-R denormalizzato di Recipes

Successivamente, dallo schema concettuale denormalizzato di figura 3.4 è stato ottenuto il nuovo schema logico di figura 3.5.

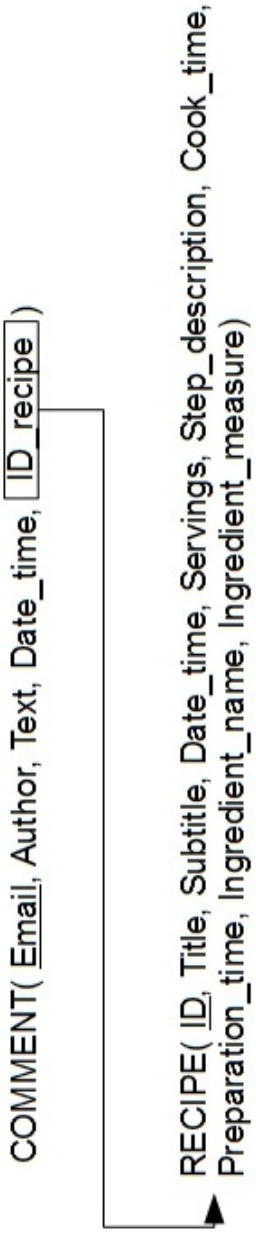

Figura 3.5: Schema logico denormalizzato di Recipes

Infine, la progettazione fisica della basi di dati denormalizzata è riportata in Appendice A.
La query iniziale è stata riscritta in Codice 3.7.

Codice 3.7: Query aggiornata per il calcolo degli ingredienti e dei passaggi

```
SELECT title, COUNT(DISTINCT step_description) AS Steps_num,
COUNT(DISTINCT ingredient_name) AS Ingredients_num
FROM RECIPE
GROUP BY title;
```

Capitolo 4

Conclusioni

In questa tesi abbiamo descritto le ragioni del dominio che hanno avuto i DBMS relazionali e i motivi che hanno portato alla nascita del movimento NoSQL. In particolare, la necessità di immagazzinare elevate quantità di informazioni e la flessibilità della loro struttura nel tempo, garantendo comunque efficienza nelle operazioni di inserimento, modifica e recupero dei dati. In seguito è stata fatta una classificazione di questi DBMS non relazionali, accennando al loro funzionamento e ad alcuni degli esemplari attualmente più rilevanti.
Abbiamo approfondito il DBMS document-oriented CouchDB, analizzandone il modello dei dati, l'utilizzo del protocollo HTTP per la comunicazione, la capacità di replica delle informazioni e i metodi per manipolare i database e i documenti. Sono stati spiegati i design document, un tipo speciale di documento che contiene le funzioni fondamentali all'interrogazione dei dati, alla loro convalida e altre importanti funzionalità.
È stata affrontata la progettazione e l'implementazione dello stesso database in CouchDB e con il DBMS relazionale PostgreSQL, analizzando poi le differenze emerse nella realizzazione e la presenza di dati ridondanti in CouchDB, che ne favorisce le prestazioni. Infine, è stata denormalizzata la base di dati relazionale per raggiungere le prestazioni di quella realizzata con CouchDB.
In conclusione, in un contesto nel quale servono dati replicati è vantaggioso utilizzare CouchDB, che grazie al suo potente meccanismo di replicazione può svolgere con facilità questo compito. È per di più in grado di mantenere robustezza sui dati grazie al versioning dei documenti, che protegge pienamente dai possibili conflitti. Un'altra proprietà per la quale conviene usarlo è la capacità di immagazzinare milioni di informazioni e, nonostante questo, eseguire interrogazioni efficienti grazie all'uso implicito di indici basati su B-Tree.
Lo studio dei tempi di inserimento e di recupero dei dati viene lasciato come sviluppo futuro della tesi. In particolare, adoperare varie quantità di informazioni sui database realizzati con CouchDB e con PostgreSQL, con-

frontare i risultati e verificare se la presenza di join implica un rallentamento dell'interrogazione in PostgreSQL, e la variazione dei tempi con la denormalizzazione del database.

Appendice A

Progettazione fisica

Utilizzando il DBMS PostgreSQL 9.2.4, si crea dapprima lo schema “Recipes” con il comando:

```
CREATE DATABASE recipes
```

Successivamente, dopo aver ottenuto l’accesso al database, si creano le tabelle con il linguaggio SQL di:

- Codice A.1, per il database normalizzato;
- Codice A.2, per il database denormalizzato.

A.1 Database normalizzato

Codice A.1: Codice SQL della struttura del Database

```
CREATE TABLE RECIPE(
    title VARCHAR(32),
    subtitle VARCHAR(64),
    date_time TIMESTAMP,
    servings SMALLINT,
    cook_time SMALLINT,
    preparation_time SMALLINT,
    PRIMARY KEY (title)
);

CREATE TABLE COMMENT(
    email VARCHAR(64),
    author VARCHAR(32),
    text VARCHAR(512),
    data_time TIMESTAMP,
    recipe_title VARCHAR(32),
    PRIMARY KEY (email),
```

```
    FOREIGN KEY(recipe_title) REFERENCES RECIPE(title)
    ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE INGREDIENT(
    name VARCHAR(64),
    PRIMARY KEY(name)
);

CREATE TABLE DIRECTION(
    recipe_title VARCHAR(32),
    phase_num SMALLINT,
    description VARCHAR(512),
    PRIMARY KEY(recipe_title, phase_num),
    FOREIGN KEY(recipe_title) REFERENCES RECIPE(title)
    ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE CONTAIN(
    recipe_title VARCHAR(32),
    ingredient_name VARCHAR(64),
    measure VARCHAR(16),
    PRIMARY KEY(recipe_title, ingredient_name),
    FOREIGN KEY(recipe_title) REFERENCES RECIPE(title)
    ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY(ingredient_name) REFERENCES INGREDIENT(name)
    ON UPDATE CASCADE ON DELETE RESTRICT
);
```

Nella tabella “Direction” è stata inserita la colonna phase_num per contenere il numero del passagio all’interno della ricetta a cui è associato.

A.2 Database denormalizzato

Codice A.2: Codice SQL della struttura del Database denormalizzato

```
CREATE TABLE RECIPE(
    id SERIAL,
    title VARCHAR(32),
    subtitle VARCHAR(64),
    date_time TIMESTAMP,
    servings SMALLINT,
    cook_time SMALLINT,
    preparation_time SMALLINT,
    step_description VARCHAR(512),
```

```
    step_num SMALLINT,
    ingredient_name VARCHAR(64),
    ingredient_measure  VARCHAR(16),
    PRIMARY KEY (id)
);

CREATE TABLE COMMENT(
    email VARCHAR(64),
    author VARCHAR(32),
    text VARCHAR(512),
    data_time TIMESTAMP,
    id_recipe INT,
    PRIMARY KEY (email),
    FOREIGN KEY(id_recipe) REFERENCES RECIPE(id)
    ON UPDATE CASCADE ON DELETE CASCADE
);
```

Nella tabella “Recipe” è stata aggiunta la colonna step_num per contenere il numero del passaggio all’interno della ricetta a cui appartiene.

Bibliografia

[1] Ramez Elmasri, Shamkant B. Navathe (2011) *Sistemi di Basi di Dati Fondamenti*, Italia, Pearson

[2] http://en.wikipedia.org/wiki/NoSQL

[3] http://www.christof-strauch.de/nosqldbs.pdf

[4] http://en.wikipedia.org/wiki/CAP_theorem

[5] Eric Redmond, Jim R. Wilson (2012) *Seven Databases in Seven Weeks*, USA, Jackie Carter

[6] MC Brown (2012) *Getting started with CouchDB*, USA, O'Reilly Media

[7] J. Chris Anderson, Jan Lehnardt, Noah Slater (2010) *CouchDB: The Definitive Guide*, USA, O'Reilly Media

[8] http://it.wikipedia.org/wiki/Denormalizzazione