Lezione 4 18/03/2024

Ci sono alcuni casi in cui il modello relazionale non è il massimo.

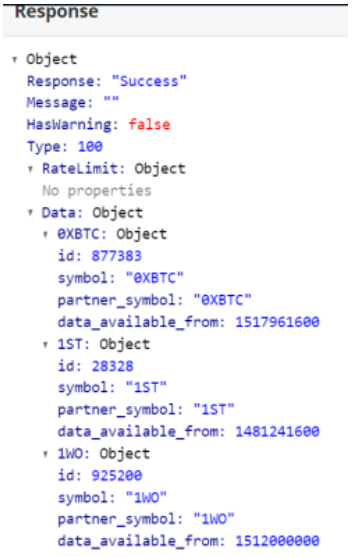
Use case NoSQL (use case poliglotta)

Use case 1

L'idea è di catturare le informazioni sulle criptovalute, perché sembra ci sia una correlazione tra le informazioni sui social e il valore delle criptovalute.

Si vuole costruire un DB per memorizzare le risposte di API in ambito crypto, per ogni API response voglio costruire una tabella per gestire i dati. Data la risposta API devo costruire lo schema.

Quindi potremmo pensare ad un DB relazionale con una tabella che ha questi 3 campi e la data.



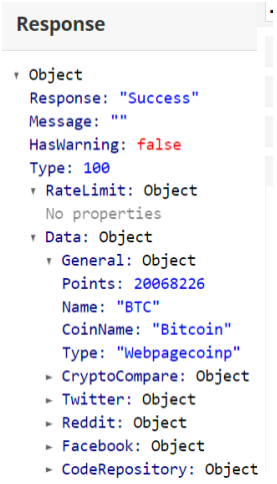
Ora vediamo una risposta json diversa.

Vediamo che Data è un oggetto complesso, che contiene un array di oggetti. La risposta contiene le criptovalute presenti sull'API.

Quindi questa risposta ha una parte di log (success...) e un array di elementi.

La parte di log ci può essere utile a livello applicativo o anche no, può essere salvato in un file di log, ma non ci interessa.

Possiamo fare una tabella singola per contenere i dati.



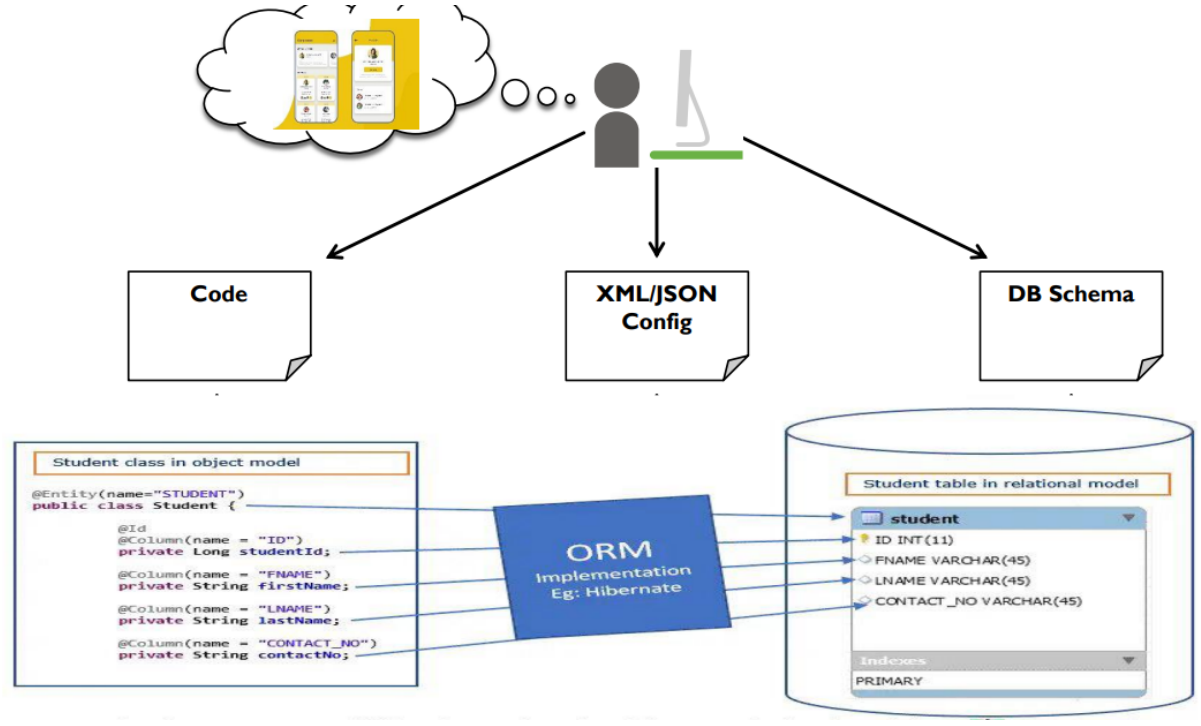
Questa risposta contiene molti oggetti complessi, che contengono molti attributi.

Come possiamo modellare un oggetto così complicato nel modello relazionale?

Qui potrei avere per esempio una tabella principale (general) con i riferimenti ad altre 5 tabelle per i social. Però poi i join sono tanti.

Cosa succede se l'API viene aggiornata, aggiungendo instagram? Dovrei fermare tutto, cambiare la base di dati.

Alternativamente potrei anche fare una tabella unica visto che i dati dentro a ciascun oggetto sono simili, ma lo svantaggio è quello che ho tanti campi vuoti.



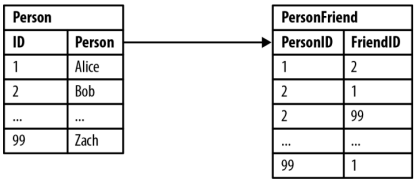
è necessario un ORM (object relational mapping) che fa il collegamento tra il programma a oggetti e il database che ha le tabelle.

Però in questo caso ho ricevuto json, l'ho tradotto in relazionale, e poi l'ho ri trasformato in json.

Potrei anche salvare una cella che contiene json (non standard in SQL), con tipo di dato json.

Use case 2

Si vuole creare un nuovo social network.
Si progetti la base di dati, relazionale, per poter memorizzare le relazioni «segue» e «è seguito da» di tutti gli utenti.



Scriviamo la query per trovare tutti gli amici degli amici di un utente. Scriviamo quindi un doppio join.

```
SELECT p1.Person AS PERSON, p2.Person AS FRIEND_OF_FRIEND
FROM PersonFriend pf1 JOIN Person p1
  ON pf1.PersonID = p1.ID
JOIN PersonFriend pf2
  ON pf2.PersonID = pf1.FriendID
JOIN Person p2
  ON pf2.FriendID = p2.ID
WHERE p1.Person = 'Alice' AND pf2.FriendID <> p1.ID
```

Ora vogliamo scrivere la query per verificare se il principio del «six separation degree» è verificato nel social network (con 6 passi di amicizie si può raggiungere ...), quindi devo scrivere una query con 6 join.

Forse il modello relazionale non è in grado di modellare in modo efficace ed efficiente i nuovi contesti applicativi nati negli ultimi 20 anni, diventa complicato con i database relazionali, c'è un grosso problema di scalabilità.

Modello relazionale

Positivo: il modello relazionale è molto rigido, semplice, molto conosciuto. Tutto quello che serve è nella base di dati relazionale, se un valore non è presente c'è null. Ci sono dietro quasi 40 anni di ricerca e sviluppo. Ci sono le proprietà ACID, si può fare il rollback, anche distribuito. Molti dati sono in modello relazionale e probabilmente rimarranno così perché fare data porting è pericoloso.

Negativo: il modello relazionale è molto rigido, devo mettere il valore null dove non so un valore (che può essere interpretato in modi diversi), un attributo è un valore, non sa gestire gli array, non è compatibile con i linguaggi di programmazione moderni, è difficile fare modifiche a tabelle, non è facilmente scalabile.

NoSQL

è un insieme di modelli di rappresentazione dei dati e relativi software di gestione.

Tutti questi modelli sono schema free (schemaless). Non c'è più il passaggio logico: definisco lo schema, inserisco i dati. I dati sono lo schema.

Spesso non è prevista la costruzione di uno schema prima dell'inserimento dei dati. L'inserimento dei dati (descrizione estensionale) implica per quel dato lo schema associato. Il successivo dato inserito può avere uno schema implicito diverso.

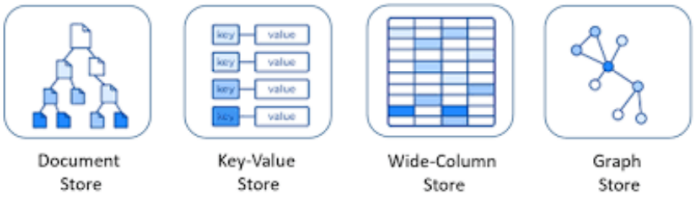
Il vantaggio e lo svantaggio di non avere lo schema è che non c'è una regola fissa, non sai cosa c'è dentro al db, ogni istanza può essere diversa.

Tutti i modelli hanno l'assunzione di mondo aperto, cioè metto solo i dati che conosco, non metto quelli che non conosco.

Modelli NoSQL

- Key-Value Stores
- Wide Column Stores
- Document Databases
- Graph Databases

RDF databases as well as Tuple stores



Key value

Il key value è un modello chiave-valore. Key-Value sono tabelle di hash dove la chiave punta a un particolare valore. Il mapping chiave-valore è supportato da meccanismi di hash per massimizzare le performance.

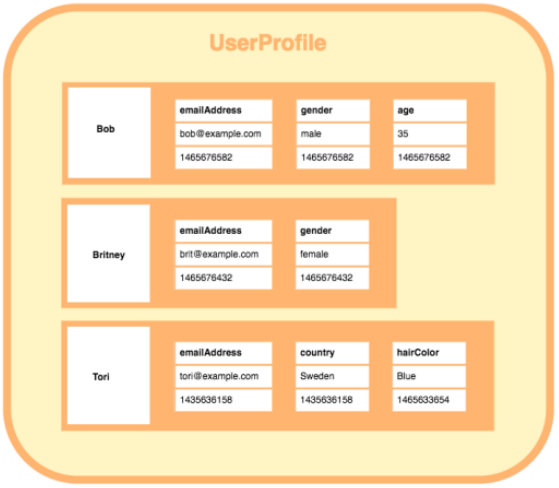
Wide column

La chiave non punta più ad un oggetto informe, ma punta ad una serie di colonne.

Nell'esempio "Bob" "Britney" "Tori" sono le chiave, che puntano alle 3 colonne.

Schema free perché appunto ciascun record è diverso, gli attributi sono diversi.

Quindi come una tabella questo ha il concetto di righe e di colonne, ma le colonne possono essere diverse.



Document based

I documenti sono indirizzati nel db tramite una chiave unica.

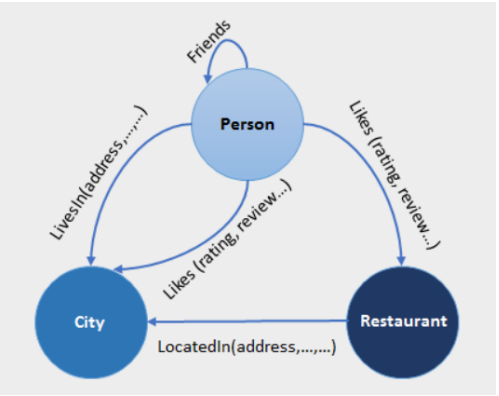
Gli oggetti possono essere delle coppie chiave valore, array di valore o reference esterna.

Questo è tipo un json.

Graph store

è quello più complicato.

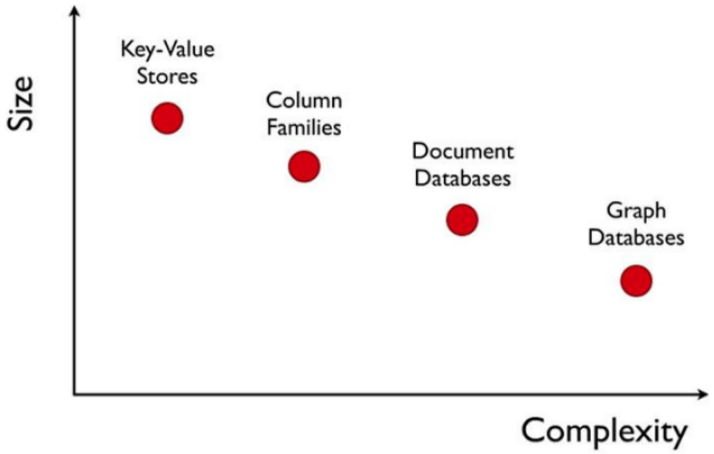
Graph Databases sono costruiti da nodi e relazioni fra nodi (archi).
I nodi hanno proprietà: rappresentano le entità (e.g."Bob" or "Alice"). Le proprietà sono informazioni pertinenti ai nodi (e. g. età:18).



Comparazione

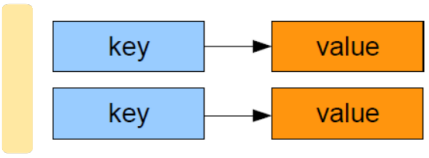
Type	Examples
Key-Value Store	redis, riak
Wide Column Store	HBASE, Cassandra
Document Store	mongoDB, CouchDB
Graph Store	Neo4j,

Più il modello è semplice, più dati posso gestire.

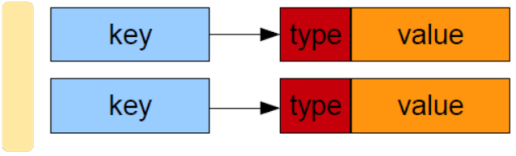


Key value

Nella sua formulazione più semplice è tra i modelli più elementari.



È possibile associare a un valore un suo tipo



è usato per esempio per salvare il session id per ricordare il carrello amazon.

Permette un accesso veloce tramite strutture di hash, anche distribuite.

è di fatto un caching, che ha il grande vantaggio di essere scalabile, veloce ed efficiente.

Operazioni elementari: inserire una coppia, cancellare una coppia, modificare una coppia, trovare un valore associato a una coppia.