

Progetto di Basi di Dati 2

|  |  |
| --- | --- |
| Versione | 1.0 |
| Data | 24/06/2022 |
| Destinatario | Prof. Genoveffa Tortora |
| Presentato da | Aliberti Rocco, Tiziano La Monica |

Composizione Team e Acronimi

|  |  |  |
| --- | --- | --- |
| **Nome e Cognome** | **Matricola** | **Acronimo** |
| Aliberti Rocco |  | AR |
| Tiziano La Monica | 0522501258 | TLM |

Scopo del documento

Il presente documento descrive il lavoro svolto da noi 2 studenti Aliberti Rocco e Tiziano La Monica per l’esame di Basi di Dati 2.

Gli obiettivi da raggiungere erano molteplici, per dare un visione generale e più completa possibile dei database non relazionali, partiremo dallo studio dello stato dell’arte e delle varie caratteristiche e sfaccettature che si trovano nei Database documentali per poi sviluppare un piccolo prototipo di data warehouse.

Il motore di information retrieval è basato su database non relazionale o No-SQL. I database non relazionali sono un “nuova frontiera” di aggregazione dati che soprattutto negli ultimi anni con il sopravvento di numerosi social possono dar sfogo a tutti i loro punti di forza ed alla loro velocità. L’obiettivo infatti dei database non relazionali non è l’immagazzinare dati in tabelle rigide e prestabilite ma puntano a dare flessibilità e soprattutto velocità nella lettura dati presenti e nella scrittura di nuovi dati.

Database No-SQL

Negli ultimi anni , aumentando sempre di più il volume di dati da memorizzare e avendo la necessità di elaborare grandi quantità di dati in poco tempo, si sta andando incontro ad un nuovo modello di gestione dei dati che si allontana dal modello relazionale. Questo nuovo modello prende il nome di NoSQL, che sta per “Not Only SQL” o “Not Relational”, proprio per sottolineare il distacco dai database relazionali (SQL è il più comune linguaggio di interrogazione dei dati nei database relazionali). Questi archivi di dati, a differenza di quelli costruiti basandosi sul modello relazionale, non presuppongono una struttura rigida o uno schema dove si descrivono le proprietà che i dati dovranno avere e le relazioni tra di essi. I database non relazionali puntano infatti ad essere più flessibili, in quanto ormai i flussi di dati che arrivano dalle sorgenti massive del web presentato notevoli irregolarità e sono più sparpagliati e sparsi (Big Data). Per aumentare il livello della prestazione nella gestione e interrogazione dei dati si punta dunque ad un sistema distribuito.

MongoDB

MongoDB (il cui nome deriva da “humongous”) è un DBMS non relazionale, diventato open source nel 2009 con licenza AGPL, document-oriented, scalabile e altamente performante. La struttura di MongoDB è ottimizzata per offrire ottime prestazioni in lettura e scrittura. Per la lettura di dati si ha la possibilità di  distribuire i vari dati in più server replicati, le interrogazioni sono più semplici e veloci grazie all'assenza di join e l'approccio ai documenti rende possibile la rappresentazione di relazioni gerarchiche complesse attraverso documenti nidificati e array.

Punti fondamentali di questo DBMS sono sicuramente il Database document-oriented che permette l’archiviazione di dati sotto forma di documenti JSON, quindi strutture più semplici, gestibili facilmente e facilmente leggibili.

Il supporto agli indici è un altro punto a favore perché se non utilizziamo gli

indici, MongoDB deve esaminare ogni documento presente nella collezione e

confrontare i campi indicati nelle query fornita. Le prestazioni dell’interrogazione

degradano, quindi, man mano che il nostro database si consolida acquisendo

sempre più dati. Indicizzare una collezione consente di ridurre drasticamente

numero di documenti esaminati da MongoDB durante una query, con una

conseguente riduzione dei tempi di esecuzione. D’altra parte, la decisione di

indicizzare uno o più campi deve essere ponderata, perché l’indicizzazione ha

comunque l’effetto di occupare spazio su disco e rallenta l’inserimento e la

modifica dei documenti.[

Altro punto a favore è la replicazione, sostanzialmente lo scopo di rafforzare

l’affidabilità del sistema in caso di fallimenti hardware, software o infrastrutturali.

Storicamente MongoDB permette due meccanismi per implementare la

replicazione: Master-Slave e Replica Set.

Lo sharding è uno dei metodi cardine di questo DBMS, utilizzato per distribuire

i dati su più macchine. MongoDB utilizza lo sharding per supportare le

distribuzioni con set di dati molto grandi e operazioni a throughput elevato.

Ultima caratteristica degna di nota è  la Query document-based che permette la creazione di query dinamiche ad hoc.

Studio di fattibilità

La fase di studio della fattibilità è parte dello sviluppo di un progetto, essa risulta essere fondamentale in quanto bisogna analizzare la fattibilità della realizzazione, in termini di semplicità di realizzazione, semplicità d’uso, efficacia di funzionamento e tanti altri parametri di misura qualitativa.  
In dettaglio, come detto in precedenza, in fase di analisi abbiamo dovuto scegliere quale DataSet utilizzare, valutandone quindi vari parametri, come la completezza dei dati, la chiarezza, il possibile utilizzo in termine di query ed altri fattori come la quantità di dati ed il gradimento personale verso la collezione di dati trattata.

Di seguito una breve descrizione del DataSet utilizzato:

ATP DATASET: collezione di dati con i nomi di tutti i tennisti che dal 1950 al 2022 hanno vinto o sono arrivati in una finale di uno dei 4 slam annuali.

In particolare, questo DATABASE contiene circa 3000 record con tutte le informazioni relative ad i vari tornei per ogni anno.

I motivi di questa scelta possono essere descritti in piccole considerazioni che rispecchiano il concetto di studio della fattibilità:  
Il DataSet scelto appunto, risulta essere particolarmente completo, ben popolato, di semplice lettura (chiaro ed intuitivo), ritenuto quindi abbastanza manutenibile .

Studio del DataSet

Il DataSet scelto è composto da varie caratteristiche, vediamo le principali in dettaglio:

* YEAR = anno svolgimento torneo
* TOURNAMENT= nome del torneo
* WINNER= vincitore torneo
* RUNNER\_UP= finalista
* WINNER\_NATIONALITY= nazionalità vincitore
* WINNER\_ATP\_RANKING= ranking atp vincitore
* RUNNER\_UP\_ATP\_RANKING= ranking finalista
* WINNER\_LEFT\_OR\_RIGHT\_HANDED= mano utilizzata dal tennista
* TOURNAMENT\_SURFACE= superfice torneo
* WINNER\_PRIZE= vincita in dollari

Progettazioni possibili Query

Trovare tutti gli slam:

***router***.get('/getSlams',function (req, res) {  
 Slam.find({},function (err, Slams) {  
 if (err) {  
 res.json({  
 status: "error",  
 message: err,  
 });  
 }  
 res.json({  
 status: "success",  
 message: "Slams retrieved successfully",  
 data: Slams }); });});

Calcolare il totale delle vincità per tennista:

***router***.get('/getTotalRevenueForPlayer/:tennista', function (req, res) {  
 Slam.aggregate([  
 {$match: {WINNER: req.params.tennista}},  
 { $group : {\_id: "$WINNER", totale: { $sum : "$WINNER\_PRIZE" }}}],function (err, total\_revenue) {  
 if (err) {   
 res.json({   
 status: "error",   
 message: err,   
 });   
 }   
 res.json({   
 status: "success",   
 message: "Slams retrieved successfully",  
 data: total\_revenue }); }); });

Trovare il peggior tennista per ranking ATP:

***router***.get('/get\_worst\_winner/:tennista/:max\_ranking', function (req, res) {  
Slam.find({WINNER:req.params.tennista,WINNER\_ATP\_RANKING:{$gte:req.params.max\_ranking}},function (err, Slams) {  
 if (err) {   
 res.json({   
 status: "error",   
 message: err,   
 });   
 }   
 res.json({   
 status: "success",   
 message: "Slams retrieved successfully",   
 data: Slams }); }); });  
  
***router***.get('/get\_worst\_runner\_up/:tennista/:max\_rating', function (req, res) { Slam.find({RUNNER\_UP:req.params.tennista,RUNNER\_UP\_ATP\_RANKING:{$gte:req.params.max\_rating}},function (err, Slams) {  
 if (err) {  
 res.json({  
 status: "error",  
 message: err,  
 });  
 }  
 res.json({  
 status: "success",

message: "Slams retrieved successfully",  
 data: Slams }); });});

Trovare uno slam per nome:

***router***.get('/getSlamsForName/:name', function (req, res) {  
 Slam.find({TOURNAMENT:req.params.name},function (err, Slams) {  
 if (err) {  
 res.json({  
 status: "error",  
 message: err,  
 });  
 }  
 res.json({  
 status: "success",  
 message: "Slams retrieved successfully",  
 data: Slams }); });});

Trovare il ranking ATP per tennista:

***router***.get('/getATPRankingForPlayers/:tennista', function (req, res) {  
 Slam.aggregate([  
 {$match: {WINNER: req.params.tennista}},  
 { $group : {\_id: "$WINNER", media: { $avg : "$WINNER\_ATP\_RANKING" }}}],function (err, Slams) {  
 if (err) {   
 res.json({   
 status: "error",   
 message: err,   
 });   
 }   
 res.json({   
 status: "success",   
 message: "Slams retrieved successfully",  
 data: Slams }); }); });

Trovare gli slam per tennisti vincitori:

***router***.get('/getWin/:tennista', function (req, res) {  
 Slam.find({WINNER:req.params.tennista},function (err, Slams) {  
 if (err) {  
 res.json({  
 status: "error",  
 message: err,  
 });  
 }  
 res.json({  
 status: "success",  
 message: "Slams retrieved successfully",  
 data: Slams }); });});

Trovare gli slam per anni:

***router***.get('/getSlamsYearRange/:from\_to', function (req, res) {  
 var range=req.params.from\_to.split('\_');  
 var from=parseInt(range[0]);  
 var to=parseInt(range[1]);  
 Slam.find({$and: [{YEAR:{$gte: from}}, {YEAR:{$lte: to}}]},function (err, Slams) {  
 if (err) {  
 res.json({  
 status: "error",  
 message: err,  
 }); }  
 res.json({  
 status: "success",  
 message: "Slams retrieved successfully",  
 data: Slams });});});

Trovare gli slam per la nazionalità del vincitore:

***router***.get('/getSlamsYearsNationality/:nazione/:from\_to', function (req, res) {  
 var range=req.params.from\_to.split('\_');  
 var from=parseInt(range[0]);  
 var to=parseInt(range[1]);  
 Slam.aggregate([  
 {$match:{WINNER\_NATIONALITY:req.params.nazione}},  
 {$match:{$and:[{YEAR:{$gte: from}}, {YEAR:{$lte: to}}]}},  
 {  
 $group: {  
 \_id: "$YEAR",  
 total: { $sum: 1 }  
 }  
 },{$sort : { \_id : 1} }],function (err, Slams) {  
 if (err) {  
 res.json({  
 status: "error",  
 message: err,  
 });  
 }  
 res.json({  
 status: "success",  
 message: "Slams retrieved successfully",  
 data: Slams});});});

Calcolare la vincita di un tennista per anni:

***router***.get('/getRevenue/:from\_to/:tennista, function (req, res) {  
 var range=req.params.from\_to.split('\_');  
 var from=parseInt(range[0]);  
 var to=parseInt(range[1]);  
 Slam.aggregate([  
 {$match:{ $and: [{YEAR:{$gte: from}}, {YEAR:{$lte: to}}], WINNER:req.params.tennista}},  
 {$group:{ \_id:"$YEAR", somma:{ $sum : "$WINNER\_PRIZE" }}},  
 {$sort : { \_id : 1} }],function (err, Slams) {  
 if (err) {   
 res.json({   
 status: "error",   
 message: err,   
 });   
 }   
 res.json({   
 status: "success",   
 message: "Slams retrieved successfully",   
 data: Slams }); }); });

Trovare i vincitori:

***router***.get('/getTennisPlayers',function(req, res){  
 Slam.aggregate([  
 {$group:{ \_id:"$WINNER"}}],function(err, Tennisti) {  
 if (err) {   
 res.json({   
 status: "error",   
 message: err,   
 });   
 }   
 res.json({   
 status: "success",   
 message: "Slams retrieved successfully",  
 data: Tennisti  
 }); }); });

Trovare i finalisti:

***router***.get('/getRunner\_Up',function(req, res){  
 Slam.aggregate([  
 { "$group": { \_id: "$RUNNER\_UP" } }],function(err, Tennisti) {  
 if (err) {   
 res.json({   
 status: "error",   
 message: err,   
 });   
 }   
 res.json({   
 status: "success",   
 message: "Slams retrieved successfully",   
 data: Tennisti  
 }); }); });

Trovare i vincitori per nazionalità:

***router***.get('/getNationality',function(req, res){  
 Slam.distinct("WINNER\_NATIONALITY",function(err, Nazione) {  
 if (err) {   
 res.json({   
 status: "error",   
 message: err,   
 });   
 }   
 res.json({   
 status: "success",   
 message: "Nation retrieved successfully",  
 data: Nazione  
 });   
 });   
});

Trovare il miglior ranking ATP per anno:

***router***.get('/get\_best\_ranking\_for\_winner\_year/:anno/:ranking', function (req, res) {  
 Slam.find({YEAR:anno\_inserito, WINNER\_ATP\_RANKING:{$lte:rating\_inserito}},function (err, Slams) {  
 if (err) {  
 res.json({   
 status: "error",   
 message: err,   
 }); }  
 res.json({   
 status: "success",   
 message: "Slams retrieved successfully",   
 data: Slams }); });});

***router***.get('/get\_best\_ranking\_for\_runner\_up\_year/:anno/:ranking', function (req, res) {  
 Slam.find({YEAR:anno\_inserito, RUNNER\_UP\_ATP\_RANKING:{$lte:rating\_inserito}},function (err, Slams) {  
 if (err) {  
 res.json({  
 status: "error",  
 message: err,  
 }); }  
 res.json({  
 status: "success",  
 message: "Slams retrieved successfully",  
 data: Slams }); });});

Implementazione e realizzazione del progetto

Abbiamo sviluppato le Query proposte in JavaScript, utilizzando come Ide IntelliJ , importando il driver apposito per utilizzare MongoDb.

Per permettere la connessione abbiamo realizzato la classe Api.JS

Per le varie Query abbiamo realizzato la classe Api.JS

Ed infine abbiamo realizzato con HTML delle pagine per la visualizzazione grafica dei risultati delle varie Query.

Abbiamo realizzato un’apposita GUI che mostrasse a schermo il risultato di una data Query al DB.