

InfluxDB

Vincenzo Minolfi

Corso di laurea magistrale
in ingegneria informatica
Università di Verona

Progetto per Basi di Dati Avanzate

Table of Contents

- 1 Intro
- 2 InfluxDB
- 3 Arch.distribuita
- 4 VeronaCARD
- 5 Confronto anni
- 6 Tempi medi
- 7 Conclusioni

Table of Contents

- 1 Intro
- 2 InfluxDB
- 3 Arch.distribuita
- 4 VeronaCARD
- 5 Confronto anni
- 6 Tempi medi
- 7 Conclusioni

Scopo e scelte

- **Obiettivo:** Approfondire la conoscenza di un database non relazionale tra quelli visti a lezione
- **Sperimentazione:** Costruire una base dati reale ed eseguire una serie di query
- **Scelte:** la coppia database-dataset scelta è la seguente
 - InfluxDB
 - VeronaCARD

Table of Contents

- 1 Intro
- 2 **InfluxDB**
- 3 Arch.distribuita
- 4 VeronaCARD
- 5 Confronto anni
- 6 Tempi medi
- 7 Conclusioni

Introduzione

- Orientato alla gestione delle serie temporali
- Fa affidamento su tecnologie di terze parti solo per la gestione di utenti e configurazioni e per un protocollo di comunicazione tra nodi del cluster

Data model

- Un *bucket* è una raccolta di serie temporali a cui è associato un nome
- Ad ogni *bucket* è associata una retention policy che contiene:
 - un *retention period* ovvero la durata del dato
 - un *replication factor*
 - una *shard duration*
- Una *organization* è un gruppo di utenti ed ogni elemento di InfluxDB (*buckets*, *tasks* ed altro) appartiene ad un'organizzazione

Concetti chiave

- **Buckets:** contenitori per data points
 - Insiemi di dati
 - *Retention period*
- **Data points:** costituiti da quattro componenti
 - measurement
 - fieldset
 - tagset
 - timestamp

Modello dati

- **Measurement**

- nome della misura
- stringa

time	<u>measurement</u>	poi	dispositivo	field	value
2014-01-01T12:17:43Z	swipes	San Zeno	26	card	04E0F88ABF3180
2014-01-01T13:08:35Z	swipes	Santa Anastasia	29	card	0441E08ABF3180
2014-01-01T13:32:49Z	swipes	Castelveccchio	35	card	04CD648ABF3180
2014-01-01T13:33:01Z	swipes	Castelveccchio	35	card	04D5C88ABF3180

Modello dati

• Fieldset

- costituisce il dato vero e proprio
- è un insieme di coppie chiave:valore
- i valori possono essere stringhe, float, interi, o booleani
- non è indicizzato

time	measurement	poi	dispositivo	field	value
2014-01-01T12:17:43Z	swipes	San Zeno	26	card	04E0F88ABF3180
2014-01-01T13:08:35Z	swipes	Santa Anastasia	29	card	0441E08ABF3180
2014-01-01T13:32:49Z	swipes	Castelvechio	35	card	04CD648ABF3180
2014-01-01T13:33:01Z	swipes	Castelvechio	35	card	04D5C88ABF3180

Modello dati

- **Tagset**

- servono a classificare i dati
- insieme di coppie chiave:valore
- i valori sono sempre stringhe
- è indicizzato

time	measurement	<u>poi</u>	<u>dispositivo</u>	field	value
2014-01-01T12:17:43Z	swipes	San Zeno	26	card	04E0F88ABF3180
2014-01-01T13:08:35Z	swipes	Santa Anastasia	29	card	0441E08ABF3180
2014-01-01T13:32:49Z	swipes	Castelveccchio	35	card	04CD648ABF3180
2014-01-01T13:33:01Z	swipes	Castelveccchio	35	card	04D5C88ABF3180

Modello dati

- **Timestamp**

- è un istante temporale che viene mostrato all'utilizzatore secondo lo standard RFC3339 in UTC (ad esempio 2021-07-28T14:30:00.00Z)
- gestito internamente come epoch time con una precisione impostabile come secondi o nanosecondi
- è alla base della posizione fisica dei dati

time	measurement	poi	dispositivo	field	value
2014-01-01T12:17:43Z	swipes	San Zeno	26	card	04E0F88ABF3180
2014-01-01T13:08:35Z	swipes	Santa Anastasia	29	card	0441E08ABF3180
2014-01-01T13:32:49Z	swipes	Castelveccchio	35	card	04CD648ABF3180
2014-01-01T13:33:01Z	swipes	Castelveccchio	35	card	04D5C88ABF3180

Modello dati

- Series

- è una collezione di dati che condividono measurement, tagset e fieldkey e che costituisce quindi un raggruppamento logico

<u>measurement</u>	<u>tag set</u>	<u>field key</u>
swipes	poi=San Zeno,dispositivo=26	card
swipes	poi=Santa Anastasia,dispositivo=26	card
swipes	poi=Castelvecchio,dispositivo=35	card

Query language

- Due linguaggi di interrogazione
 - **InfluxQL**: linguaggio simile a SQL supportato fino alla versione 1.8
 - **Flux**: nuova sintassi supportata dalla versione 1.8
- La versione 1.8 supporta quindi entrambi i linguaggi ma con limitazioni

Query language - Flux

- Una query è costituita da una sequenza di funzioni inserite in pipeline una dopo l'altra ed è quindi più programmatica che dichiarativa
- L'ordine è importante
- La selezione dei dati è effettuata indicando prima di tutto il bucket da cui estrarre i dati ed un intervallo temporale di interesse

```
from(bucket:"veronacard")
  |> range(
    start: 2019-01-01T00:00:00Z,
    stop: 2019-12-31T23:59:59Z
  )
```

Query language - Flux

- E' possibile filtrare i dati applicando la funzione *filter*
- Si possono raggruppare i dati temporalmente usando la funzione *aggregateWindow*

```
|> filter(fn:(r) => r._measurement == "swipes")
```

```
|> aggregateWindow(  
  every: 1d,  
  fn: count,  
  column: "_value",  
  timeSrc: "_start",  
  timeDst: "_time",  
  createEmpty: true  
)
```


Query language - Flux

- Per decidere di mantenere solo alcune colonne si applica *keep*
`|> keep(columns: ["_time", "_value", "poi"])`
- E' possibile anche aggiungere o modificare le colonne esistenti con la funzione *map*

```
|> map(fn: (r) => ({
  r with dayofyear:
    strings.substring(
      v: string(v:r._time),
      start: 5,
      end: 10
    )))
```

Query language - Tabelle in InfluxDB

- E' possibile raggruppare i dati in più tabelle usando la funzione *group* e indicando le colonne che guideranno il raggruppamento

```
|> group(columns=["_value"])
```

- E' possibile anche scegliere se raggruppare per le colonne indicate (default) o quelle non indicate

```
|> group(columns=["_value"], mode: "by")
```

```
|> group(columns=["_value"], mode: "except")
```

- E' possibile unire nuovamente i dati in un'unica tabella usando *group* senza parametri

```
|> group()
```

Query language - Tabelle in InfluxDB

- I risultati sono organizzati in una o più tabelle
- Il raggruppamento in InfluxDB non è distruttivo ed è possibile modificare o rimuovere il raggruppamento in qualunque momento in quanto non è altro che una riorganizzazione dei dati in tabelle. I gruppi collassano in un record per gruppo quando viene applicata una funzione di aggregazione

Tasks

- Possono essere creati dei task che consistono nell'esecuzione periodica di query che possono salvare i propri risultati in un *bucket* per una futura consultazione rapida

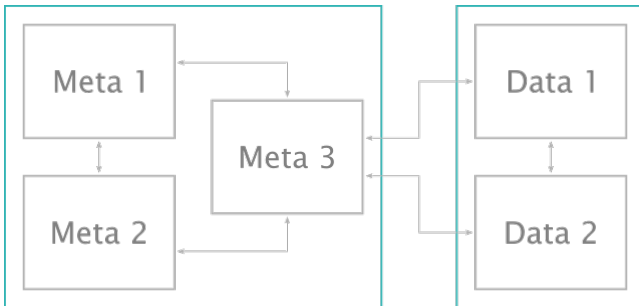
```
option task = {
  name: "task-name",
  every: 30m
}
from(bucket:"data-bucket")
[CORPO DELLA QUERY]
|> set(key: "_measurement", value: "measurement-name")
|> to(
  bucket: "results-bucket"
)
```

Table of Contents

- 1 Intro
- 2 InfluxDB
- 3 Arch.distribuita**
- 4 VeronaCARD
- 5 Confronto anni
- 6 Tempi medi
- 7 Conclusioni

Nodi

- Due tipi di nodi: data nodes e meta nodes



Tipi di nodi (1)

- **Meta nodes:**
 - Lavoro di amministrazione (aggiunta/rimozione server al cluster, movimento dati)
 - Ciascun nodo tiene traccia dello stato dell'intero cluster (nodi, buckets, utenti, posizione shards, tasks) e comunica con tutti gli altri meta nodes
 - Numero magico 3 (o massimo 5)
 - Viene utilizzato un *consensus protocol* chiamato **Raft**

Tipi di nodi (2)

- **Data nodes:**
 - Salvataggio dati (measurement, tagset, fieldset, timestamp), esecuzione letture e scritture
 - Ciascun *data node* comunica con tutti gli altri nodi del cluster, di entrambi i tipi
 - Numero multiplo del *replication factor*, dipendente dal carico di lavoro

Shards

- Una *retention policy* consiste nella definizione della durata del dato, di un intervallo temporale (*shard duration*) ed un numero di copie (*replication factor*) per la gestione degli *shard*
- Viene creato uno *shard group* per ogni *shard duration* di dati
- Ogni *shard* raccoglie i dati di una o più serie per la durata temporale dello *shard group* di appartenenza e viene duplicato in base al *replication factor* ed esse sono distribuite su diversi *data nodes*
- **Scritture**
 - In base al datetime viene calcolato lo *shard group* di appartenenza del dato, lo *shard* dipende da measurement e tagset:

```
// key is measurement + tagset
// shardGroup is the group for the values based on timestamp
// hash with fnv and then bucket
shard := shardGroup.shards[
    fnv.New64a(key) % len(shardGroup.Shards)]
```

Consistenza delle scritture

- Quando un *data node* riceve un comando di scrittura determina quali *data nodes* gestiscono le copie dello *shard* di riferimento
- Il sistema cerca di scrivere su tutte le copie dello *shard*
- La *consistency* può essere impostata per quella specifica scrittura, in caso contrario verrà usato il default. Le opzioni sono:
 - *any*: almeno una scrittura persistente o *hinted handoff*
 - *one*: almeno una scrittura persistente
 - *quorum*: la maggioranza delle scritture persistenti
 - *all*: tutte le scritture persistenti
- Nota: potrebbe accadere che il dato venga persistito anche per un'operazione di scrittura fallita

Hinted handoff

- Meccanismo di gestione della momentanea indisponibilità di un *data node*
- Le scritture non soddisfabili immediatamente vengono scritte in una coda persistente che il nodo, una volta tornato disponibile, cercherà di smaltire
- Se il nodo A deve scrivere sul nodo B ma la coda di B non è vuota A scriverà direttamente nella coda per garantire l'ordine e risparmiare tentativi
- La coda è persistente

Lecture

- Per ogni intervallo temporale coperto dalla query viene calcolato lo shard group di riferimento.
- Per ciascuno shard group se il nodo coordinatore contiene uno shard esegue la query su esso altrimenti sceglie casualmente uno degli shard per l'esecuzione della query
- Raccolti i risultati il nodo coordinatore li assembla e restituisce il risultato

Table of Contents

- 1 Intro
- 2 InfluxDB
- 3 Arch.distribuita
- 4 VeronaCARD**
- 5 Confronto anni
- 6 Tempi medi
- 7 Conclusioni

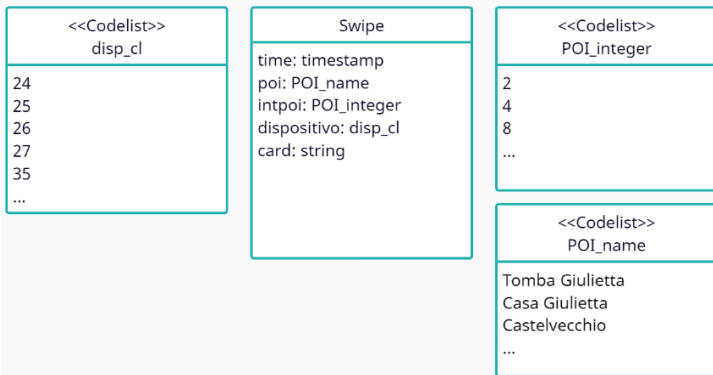
Cos'è VeronaCARD

- *VeronaCARD* per il turismo a Verona
- **Utilizzo** della tessera:
 - Accesso a prezzo ridotto o nullo in punti di interesse
 - Trasporto
- **Durata**: Validità di 24, 48 o 72 ore
- **Funzionamento**: in ciascun punto di interesse (POI) la tessera viene letta da un dispositivo

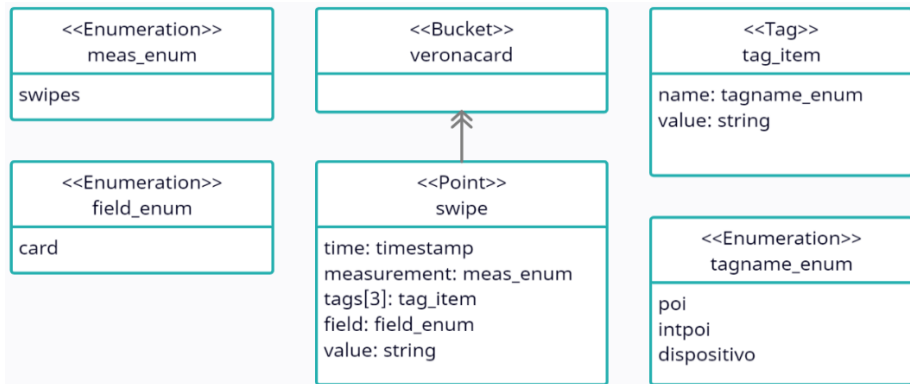
Struttura e descrizione del dataset

- **Cosa:** dati delle strisciate delle tessere nei vari POI
- **Formato dati:** elenco di CSV divisi per anno dal 2014 al 2020
- **Informazioni disponibili**
 - Data della strisciata
 - Ora della strisciata
 - Punto di interesse (POI)
 - Identificativo del lettore di tessere
 - Identificativo della tessera
 - Altri dati

UML schema



Physical schema



Riempimento ed interrogazione

- Python tramite `influxdb_client` (lib ufficiale)
 - **Riempimento:** API di scrittura
 - **Interrogazione:** API di lettura per l'esecuzione di query scritte in Flux
- Interfaccia web integrata
- Interfaccia testuale accessibile da terminale

Table of Contents

- 1 Intro
- 2 InfluxDB
- 3 Arch.distribuita
- 4 VeronaCARD
- 5 Confronto anni**
- 6 Tempi medi
- 7 Conclusioni

Obiettivo

- Il primo obiettivo è stato ottenere un confronto tra l'anno pre-covid (2019) e l'anno caratterizzato dalla pandemia (2020)
- Si è cercato di preparare la query in modo che desse i dati in maniera il più possibile adatta all'obiettivo
- Il programma scritto in Python si connette al database, avvia l'esecuzione della query e visualizza i dati utilizzando matplotlib

Passaggi (ultima versione)

- Caricamento dati dei due anni di interesse 2019 e 2020
 - **Range:** selezione dell'intervallo di interesse
 - **Filtri:** selezione della misura e delle colonne di interesse
 - **Aggregazione:** aggregazione temporale con applicazione del conteggio strisciate
- Calcolo delle due colonne per giorno dell'anno e anno
- Operazione di pivot per accorpare i dati dei due anni

Codice query in Flux

```
import "strings"
from(bucket:"veronacard")
|> range(start: 2019-01-01T00:00:00Z, stop: 2020-12-31T23:59:59Z)
|> filter(fn:(r) => r._measurement == "swipes")
|> keep(columns:["_time", "_value", "poi"])
|> aggregateWindow(every: 1d, fn: count, column: "_value",
    timeDst: "_time", createEmpty: true)
|> map(fn: (r) => ({r with
    dayofyear: strings.substring(v: string(v:r._time), start: 5, end: 10),
    year: strings.substring(v: string(v:r._time), start: 0, end: 4)
}))
)
|> pivot(
    rowKey:["dayofyear", "poi"], columnKey: ["year"], valueColumn: "_value"
)
```

Passaggi (versione precedente, con join)

- Caricamento dati del 2019
 - **Range:** selezione dell'intervallo di interesse
 - **Filtri:** selezione della misura e delle colonne di interesse
 - **Aggregazione:** aggregazione temporale con applicazione del conteggio strisciate
- Caricamento dati del 2020, stessi passaggi del 2019
- Operazione di **join** tra i due insiemi di dati basata sul giorno dell'anno
 - **Opzione 1:** timeShift di un anno -> problema dell'anno bisestile (doppio 28 febbraio 2019)
 - **Opzione 2:** parsing del datetime per estrarre dd-MM

Problema dell'operazione di join

- Un solo tipo disponibile: inner join
- Conseguente esclusione del 29 febbraio dall'analisi se uno dei due anni non è bisestile
- Soluzione accettabile?

Codice query in Flux (parte 1)

```
import "strings"

manageData = (tables=<-) =>
  tables
  |> filter(fn:(r) => r._measurement == "swipes")
  |> keep(columns: ["_time", "_value", "poi"])
  |> aggregateWindow(
    every: 1d, fn: count, column: "_value",
    timeSrc: "_start", timeDst: "_time", createEmpty: true
  )
  |> map(fn: (r) => ({
    r with dayofyear:
      strings.substring(v: string(v:r._time), start: 5, end: 10)
  })
  )
```

Codice query in Flux (parte 2)

```
from2019 = from(bucket:"veronacard")
|> range(start: 2019-01-01T00:00:00Z, stop: 2019-12-31T23:59:59Z)
|> manageData()

from2020 = from(bucket:"veronacard")
|> range(start: 2020-01-01T00:00:00Z, stop: 2020-12-31T23:59:59Z)
|> manageData()

join(
  tables: { fr19: from2019, fr20: from2020 }, on: ["dayofyear", "poi"]
)
|> keep(
  columns: ["_time", "_value_fr19", "_value_fr20", "dayofyear", "poi"]
)
```

Risultati del confronto

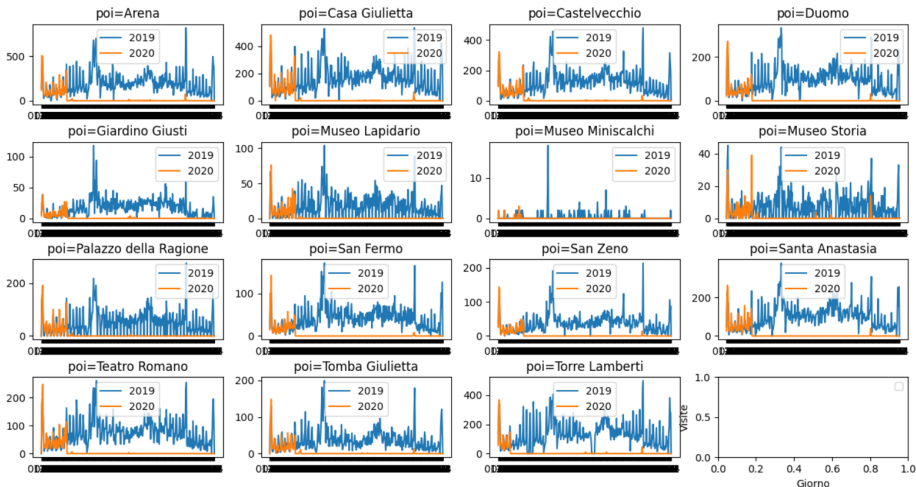


Table of Contents

- 1 Intro
- 2 InfluxDB
- 3 Arch.distribuita
- 4 VeronaCARD
- 5 Confronto anni
- 6 Tempi medi**
- 7 Conclusioni

Obiettivo

- Il secondo obiettivo è stato ottenere i tempi medi tra ciascuna strisciata in un POI e la strisciata della medesima tessera nel POI seguente
- Si è cercato di preparare la query in modo che desse i dati in maniera il più possibile adatta all'obiettivo
- Il programma scritto in Python si connette al database, avvia l'esecuzione della query, raccoglie i risultati che vengono poi esportati per la visualizzazione della matrice

Passaggi - primo tentativo

- Caricamento dati del 2019
 - **Filtri**: selezione della misura e delle colonne di interesse
 - **Raggruppamento** dei dati per numero tessera
 - **Enumerazione** dei valori
- Ripetizione per ottenere una seconda lista di record simile
 - Scostamento di uno dell'enumerable
- Operazione di **join** tra i due insiemi basata sull'enumerable

Passaggi - problema e nuovo approccio

- Quasi 400.000 record sono una mole di dati difficilmente gestibile in join
- `lead()` e `lag()` sono le due funzioni che annullerebbero la necessità di una join ma non sono supportate
- Sono supportate `elapsed()` e `difference()`
- **Idea**
 - Assegnare un valore numerico a ciascun POI creando un tag *intpoi*
 - Applicare `difference()` su *intpoi* creando una nuova colonna
 - Applicare `elapsed()` su `_time`

Passaggi - Affinamento e difetti

- Scegliere valori numerici tali che la differenza tra ciascuna coppia sia univoca ed evitare la duplicazione della colonna.
- Nei risultati della query ogni record riporta l'identificativo della coppia di POI e la differenza temporale tra le relative strisciate della tessera
- Questo nuovo metodo non permette di distinguere i casi di due strisciate consecutive nello stesso POI

Codice query in Flux

```
from(bucket:"veronacard_intpoi")
|> range(start: 2019-01-01T00:00:00Z, stop: 2019-12-31T23:59:59Z)
|> filter(fn:(r) => r._measurement == "swipes")
|> keep(columns: ["_value", "_time", "intpoi"])
|> group(columns:["_value"])
|> sort(columns:["_time"])
// |> duplicate(column: "intpoi", as: "destpoi")
|> map(fn: (r) => ({ r with intpoi: uint(v: r.intpoi) }) )
|> difference(
    nonNegative: false, columns: ["intpoi"], keepFirst: true
)
|> elapsed(unit: 1m, timeColumn: "_time", columnName: "elapsed")
|> keep(columns: ["elapsed", "intpoi"])
|> group(columns:["intpoi"])
|> mean(column: "elapsed")
```

Risultati permanenza più spostamento

	Museo Miniscalchi Casa Giulietta Castelvechio Teatro Romano Santa Anastasia Arena Palazzo della Ragione Duomo San Zeno Torre Lamberti San Fermo Museo Storia Giardino Giusti POI														
Tomba Giulietta		115	312	372	429	169	464	361	430	287	331	102	132	60	446
Museo Miniscalchi	476		86	372	192	158	57	62	60	128	438	75	1197		309
Casa Giulietta	314	473		287	276	125	176	196	103	340	95	113	204	101	362
Castelvechio	283	596	422		426	191	449	249	363	147	325	392	432	48	475
Teatro Romano	380	88	305	472		459	138	101	298	374	246	289	191	194	124
Arena	310	207	326	334	498		485	327	363	314	260	276	292	44	540
Santa Anastasia	454	69	125	333	146	249		72	89	332	81	141	289	182	192
Duomo	326	89	232	258	101	313	69		169	233	141	247	175	137	278
Palazzo della Ragione	305	554	92	316	299	213	125	190		343	47	179	135	104	449
San Zeno	394	246	734	216	611	393	695	413	624		463	348	518	156	586
Torre Lamberti	335	78	106	263	288	192	169	173	43	271		145	317	95	389
San Fermo	112	54	147	399	266	132	308	258	209	303	167		77	73	194
Museo Storia	149	1037	224	473	197	261	337	308	172	415	226	60		94	108
Museo Lapidario	430	1174	820	499	950	203	998	860	637	387	816	747	879		1030
Giardino Giusti	379		382	647	170	456	403	407	401	480	402	237	244	277	
POI															204

Table of Contents

- 1 Intro
- 2 InfluxDB
- 3 Arch.distribuita
- 4 VeronaCARD
- 5 Confronto anni
- 6 Tempi medi
- 7 Conclusioni**

Conclusioni e considerazioni

- Il sistema è effettivamente molto orientato alla gestione delle serie temporali
- L'assenza di funzioni come `lead()` e `lag()` presenti in altri sistemi non è trascurabile, è presente una feature request su github
- La scelta di un database di questo tipo costringe a ragionare diversamente (ad esempio "ungroup")
- Le join sono ora presenti ma limitate (type: "inner")
- In molti casi di errore nella query il sistema non segnala errori ma non dà risultati