InfluxDB

Vincenzo Minolfi

Corso di laurea magistrale
in ingegneria informatica
Università di Verona

Progetto per Basi di Dati Avanzate

Table of Contents

Table of Contents

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

Introduzione

- Orientato alla gestione delle serie temporali
- Fa affidamento su tecnologie di terze parti solo per la gestione di utenti e configurazioni **(TODO: INSERIRE QUALE DB)**

Data model

- Un *bucket* è una raccolta di serie temporali a cui è associato un nome
- Ad ogni *bucket* è associato un *retention period* ovvero la durata del dato
- 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	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	Castelvecchio	35	card	04CD648ABF3180
2014-01-01T13:33:01Z	swipes	Castelvecchio	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	Castelvecchio	35	card	04CD648ABF3180
2014-01-01T13:33:01Z	swipes	Castelvecchio	35	card	04D5C88ABF3180

Modello dati

- **Tagset**
  - insieme
  - insieme di coppie chiave:valore
  - i valori sono sempre stringhe
  - 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	Castelvecchio	35	card	04CD648ABF3180
2014-01-01T13:33:01Z	swipes	Castelvecchio	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	Castelvecchio	35	card	04CD648ABF3180
2014-01-01T13:33:01Z	swipes	Castelvecchio	35	card	04D5C88ABF3180

Modello dati

- **Series**
  - è una collezione di dati che condividono measurement, tagset e fieldkey(s) e che costituisce quindi un raggruppamento logico

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

Modello dati

- TODO
  - Points
  - Series (dettagli)

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*

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

- Si possono raggruppare i dati temporalmente usando la funzione *aggregateWindow*

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

Query language - Flux

- Per decidere di mantenete 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 *group* 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"
)
```

Distributed architecture

- Due tipi di nodi: data nodes e meta nodes
- **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)

Distributed architecture

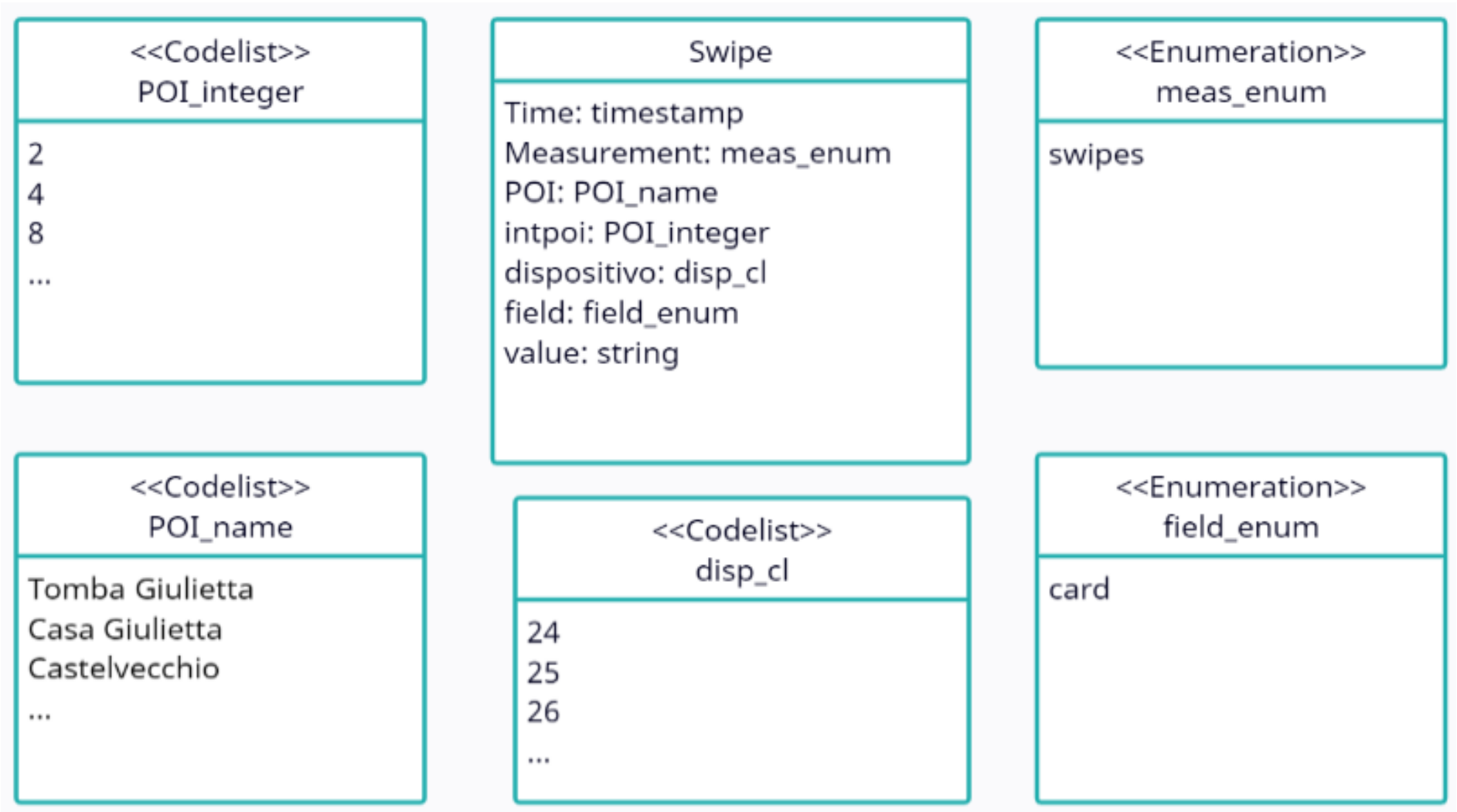
- **Data nodes**:
  - Salvataggio dati (measurement, tagset, fieldset), esecuzione letture e scritture
  - Ciascun nodo comunica con tutti gli altri nodi del cluster, di entrambi i tipi
  - Numero multiplo del *replication factor*, dipendente dal carico di lavoro

Table of Contents

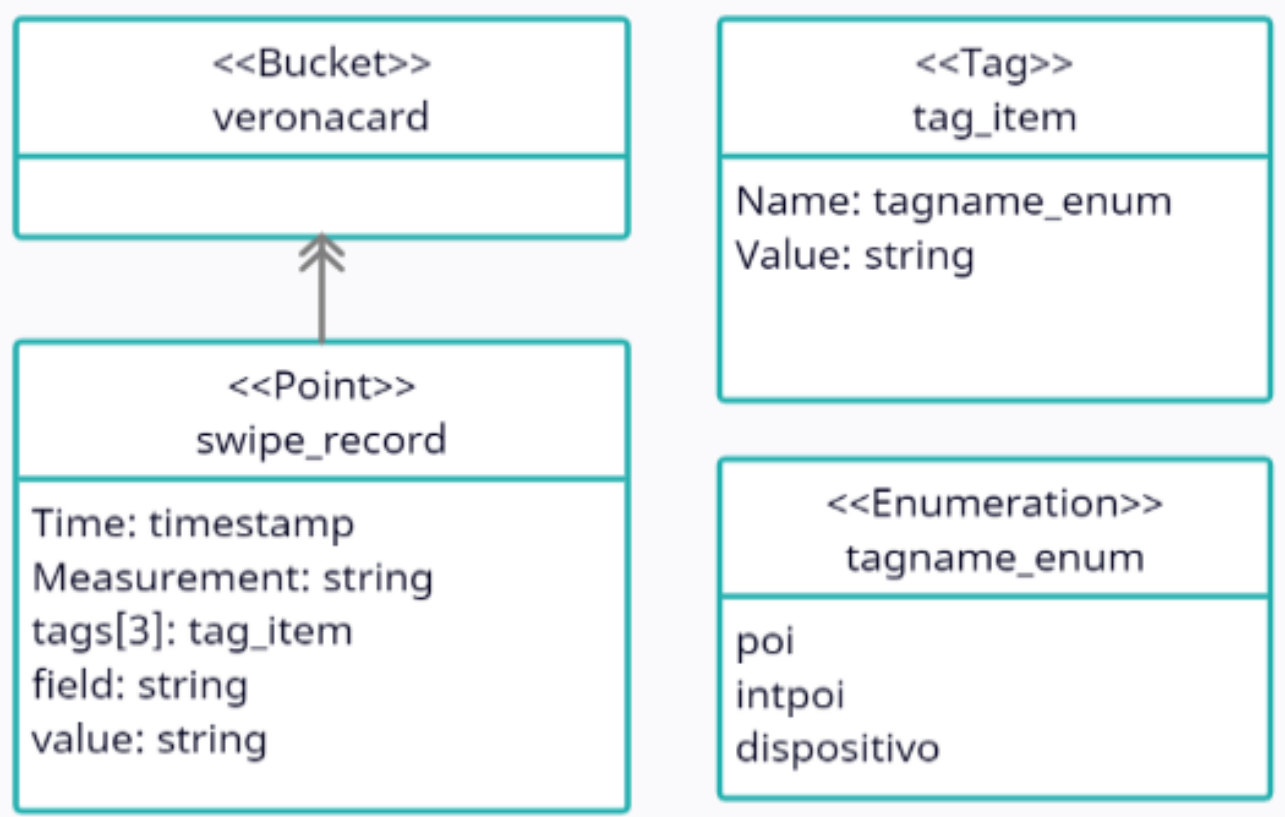
Cos'è VeronaCARD

- *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 tessere viene letta da un dispositivo

UML schema


<<Codelist>>
POI_integer
2
4
8
...
Swipe
Time: timestamp
Measurement: meas_enum
POI: POI_name
intpoi: POI_integer
dispositivo: disp_cl
field: field_enum
value: string
<<Enumeration>>
meas_enum
swipes
<<Codelist>>
POI_name
Tomba Giulietta
Casa Giulietta
Castelvecchio
...
<<Codelist>>
disp_cl
24
25
26
...
<<Enumeration>>
field_enum
card

Physical schema



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

Table of Contents

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

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

- 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

TODO INSERIRE TESTO QUERY 1

Risultati del confronto

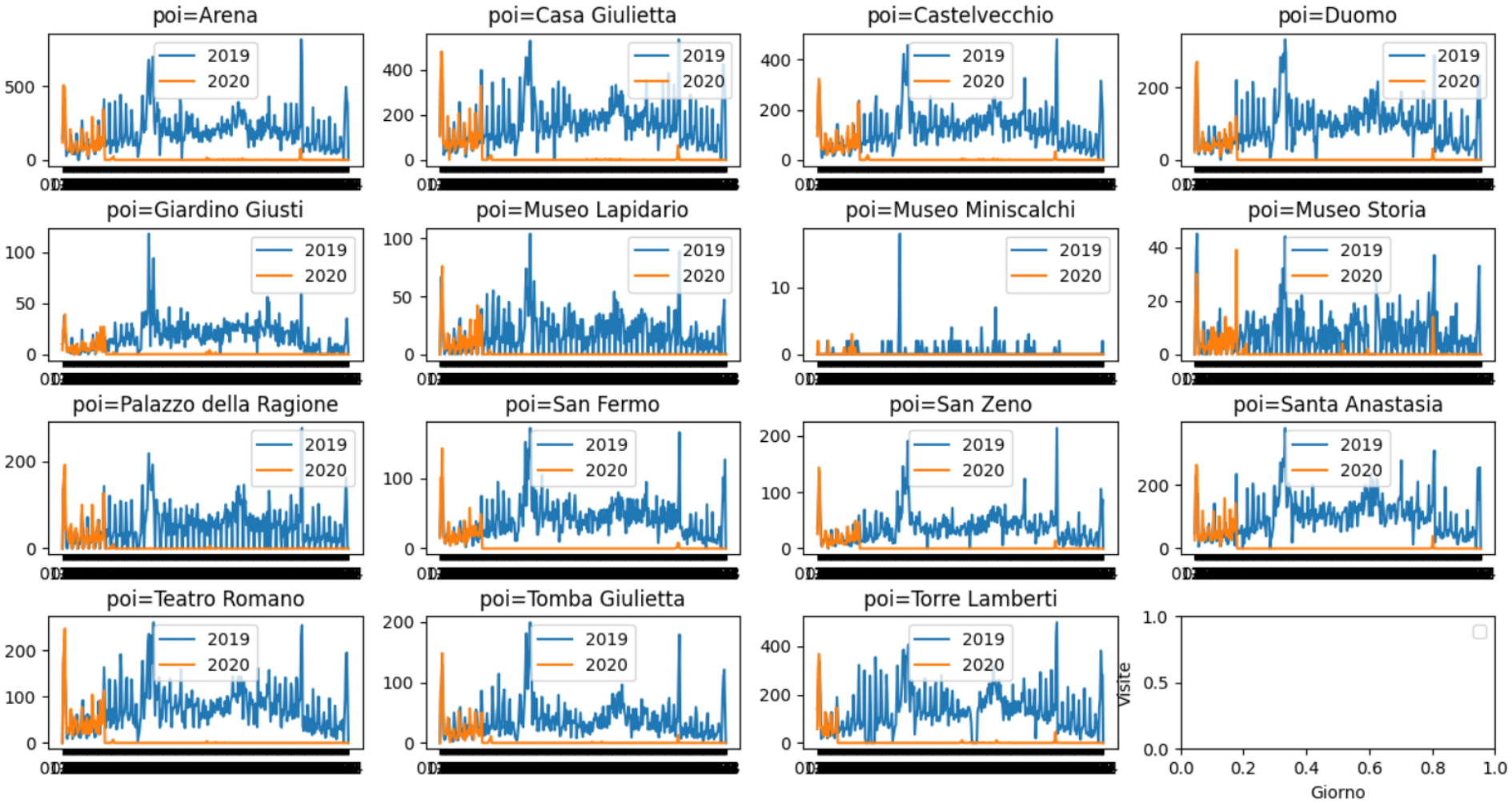


Table of Contents

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

TODO INSERIRE TESTO QUERY 2

Risultati permanenza più spostamento

	Tomba Giulietta	Museo Miniscalchi	Casa Giulietta	Castelvecchio	Teatro Romano	Arena	Santa Anastasia	Duomo	Palazzo della Ragione	San Zeno	Torre Lamberti	San Fermo	Museo Storia	Museo Lapidario	Giardino Giusti	POI
Tomba Giulietta		115	307	368	430	169	475	363	430	281	337	101	135	61	456	
Museo Miniscalchi	476		86	372	137	111	589	64	60	128	438	75	1197		309	
Casa Giulietta	279	473		272	278	104	172	200	103	330	93	110	205	101	319	
Castelvecchio	280	596	420		433	157	452	250	365	135	321	381	428	49	497	
Teatro Romano	385	88	305	470		416	137	102	298	378	245	285	189	193	124	
Arena	199	207	318	249	502		461	333	363	277	256	262	290	49	531	
Santa Anastasia	443	69	124	318	143	198		72	91	303	80	133	286	178	197	
Duomo	325	87	243	254	102	274	69		167	219	140	230	174	135	275	
Palazzo della Ragione	298	554	92	292	291	159	131	192		333	46	174	134	104	449	
San Zeno	392	246	732	214	620	334	672	415	606		471	349	495	151	585	
Torre Lamberti	326	78	102	265	274	143	159	169	44	258		137	315	103	364	
San Fermo	108	54	146	404	263	105	301	257	207	298	166		76	75	206	
Museo Storia	145	1037	229	453	197	261	337	308	171	415	223	60		94	108	
Museo Lapidario	369	1174	770	390	941	103	999	870	637	347	812	740	879		1038	
Giardino Giusti	361		389	673	171	463	405	406	395	489	398	229	240	311		
POI																190

Table of Contents

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
- Lavorando con dataset di buone dimensioni si può facilmente andare incontro a rallentamenti **(TOGLIERE?)**