8Progetto **ADaP**

Specifica tecnica per lo sviluppo di procedura MongoDB per l’implementazione dei servizi **NP05 – Blocco Traffico e NP06 Sblocco Traffico**

Autore: A.Cecconi – Atlantica Digital

Data ultima revisione del documento: 27/05/2020

1. **Procedura per il Blocco del traffico**

Come descritto nel par. *NP\_05 Richiesta Blocco Traffico per Utenza* nel documento:

ADaP - MongoDB SIF - R58-19-1420 - Progetto TIM\_CO\_ADaP-Decommissioning\_NPAT.docx

una richiesta di **blocco** ha l’obiettivo di copiare il traffico compiuto da una certa utenza in un certo periodo temporale (informazioni contenute nella richiesta) dalla collection principale del db (LinesEnhanced), dove viene normalmente acquisito il traffico proveniente dai flussi (e che è oggetto del processo standard di svecchiamento a 180 gg del dato), verso una collection ad hoc (LinesEnquired) dove tale traffico sarà “freezato” a disposizione per analisi.

L’arrivo di una successiva richiesta di **sblocco** del traffico per tale utenza e periodo temporale fa sì che, trascorsi 60 gg dalla richiesta di sblocco, il relativo traffico sulla collection LinesEnquired venga eliminato.

I due processi di svecchiamento, quello standard a 180 gg sulla LinesEnhanced e quello a 60 gg dalla richiesta di sblocco sulla LinesEnquired, sono tra loro indipendenti.

Il servizio sarà gestito su MongoDB su due collection ad hoc:

*LinesEnquired* 🡪 tracciato identico alla solo shape TR di LinesEnhanced con in più alcuni campi di servizio

*LinesEnquiredLog* 🡪 un document rappresenta l’intero ciclo di blocco e sblocco di una richiesta di blocco

Per l’implementazione della fase di **blocco** del traffico, viene sviluppata da Atlantica la procedura seguente python (basata sul driver PyMongo):

**adapEnquiredTrBlock.py**

che, lanciata da crontab con periodicità configurabile, implementerà quanto segue:

* esamina il contenuto della collection LinesEnquiredLog in cerca di nuove richieste di blocco (document con data.blkStatus=1), se trovato:
  + effettua i controlli formali sul contenuto della richiesta di blocco e, in caso di anomalia, segnala un errore ponendo (update basata su campo LinesEnquiredLog.\_id):
    - LinesEnquiredLog.data.blkStatus: 5 (Error)

e, a seconda della tipologia di errore, valorizzi nel modo seguente i campi indicati:

|  |  |
| --- | --- |
| LinesEnquiredLog.data.blkErrCod | LinesEnquireedLog.data.blkErrDes |
| -502 | Formato input Errato (1) |
| -503 | Dati input errati: DataInizio maggiore di DataFine |
| -504 | Dati input errati: Intervallo di ricerca fuori range! |
| -505 | Dati input errati: DataInizio maggiore della Data Odierna |

(1) Per i campi data della request, il formato corretto è YYYY-MM-DD

* + superati i controlli formali, verifica se l’utenza esiste sulla collection LinesEnhanced, in caso negativo, segnala un errore ponendo (update basata su campo LinesEnquiredLog.\_id):
    - LinesEnquiredLog data.blkStatus: 5 (Error)
    - LinesEnquiredLog data.blkErrCode: -510
    - LinesEnquiredLog data.blkErrDes: Utenza da bloccare non trovata
  + in caso positivo, effettua la copia del traffico:
    - aprendo opportuno cursore sulla collection LinesEnhanced, recuperando i document per utenza (filtro basato sul campo data.LineNumber) e per intervallo temporale (filtro basato sul campo data.eventDate) e inserendo tali document sulla collection LinesEnquired ponendo, per ciascuno di essi, il campo data.blkLockId pari al valore del campo data.blkLockId del document della collection LinesEnquiredLog restituito dalla ricerca e relativo alla richiesta di blocco
    - se la precedente copia restituisce un errore, occorre porre (update basata su campo LinesEnquiredLog.\_id):
      * LinesEnquiredLog.data.blkStatus: 5 (Error)
      * LinesEnquiredLog.data.blkErrCode: -511
      * LinesEnquiredLog.data.blkErrDes: [descrizione dell’errore restituito dal driver PyMongo]
      * LinesEnquiredLog.data.blkCopyDate: data di sistema
      * LinesEnquiredLog.data.blkCopiedDocs: numero di docs copiati prima dell’errore
      * LinesEnquiredLog.data.blkCopiedCdrs: numero di cartellini copiati prima dell’errore
    - se invece la copia ha avuto successo, occorre porre (update basata su campo LinesEnquiredLog.\_id):
      * LinesEnquiredLog.data.blkStatus: 2 (Locked)
      * LinesEnquiredLog.data.blkCopyDateStart: data apertura cursore
      * LinesEnquiredLog.data.blkCopyDateEnd: data di sistema
      * LinesEnquiredLog.data.blkCopiedDocs: numero di docs copiati
      * LinesEnquiredLog.data.blkCopiedCdrs: numero di cartellini copiati

Lo script traccerà le principali operazioni eseguite nel file di log

**adapEnquiredTrBlock.log**

che viene fornito nella stessa directory che ospita lo script e il cui contenuto sarà alimentato in append mode dallo script stesso.

Esempi di tracciamento:

[2019-09-30 10:00:00] ADaP – Servizio di Blocco Traffico v.1.0 – inizio:

[2019-09-30 10:00:00] LinesEnquiredLog: non trovata alcuna richiesta di blocco

[2019-09-30 10:00:00] ADaP – Servizio di Blocco Traffico – fine

[2019-09-30 10:00:05] ADaP – Servizio di Blocco Traffico v.1.0 – inizio:

[2019-09-30 10:00:05] LinesEnquiredLog: trovata richiesta di blocco valida \_id: 5d52afcf7bbc7596386df6b0 (utenza: 3351028124, data dal: 01/06/2019, data al: 31/08/2019)

[2019-09-30 10:00:05] blocco traffico eseguito con successo

[2019-09-30 10:00:05] ADaP – Servizio di Blocco Traffico – fine

[2019-09-30 10:00:10] ADaP – Servizio di Blocco Traffico v.1.0 – inizio:

[2019-09-30 10:00:10] LinesEnquiredLog: trovata richiesta di blocco NON valida \_id: 5d52afcf7bbc7596386df677 ERRORE: Dati input errati: DataInizio maggiore di DataFine

[2019-09-30 10:00:10] ADaP – Servizio di Blocco Traffico – fine

La manutenzione del file di log è delegata all’amministratore di sistema.

1. **Procedura per lo Sblocco del traffico**

Per l’implementazione della fase di **sblocco** del traffico, viene sviluppata la seguente procedura python (basata sul driver PyMongo):

**adapEnquiredTrRelease.py**

che, lanciata da crontab con periodicità configurabile, implementerà quanto segue:

* esamina il contenuto della collection LinesEnquiredLog in cerca di:
  + nuove richieste da sbloccare (document con data.blkStatus=3): se trovata e se sono trascorsi 60 giorni dalla data LinesEnquiredLog data.rlsDate, procede con la cancellazione del traffico bloccato, ossia:
    - cancella tutti i document della tabella LinesEnquired caratterizzati dal valore del campo data.blkLockId uguale al valore del campo LinesEnquiredLog.\_id relativo alla richiesta di blocco/sblocco;
    - se la precedente cancellazione restituisce un errore, occorre porre (update basata su campo LinesEnquiredLog.\_id):
      * LinesEnquiredLog.data.blkStatus: 5 (Error)
      * LinesEnquiredLog data.rlsDelDate: data di sistema
      * LinesEnquiredLog.data.rlsErrCode: -512
      * LinesEnquiredLog.data.rlsErrDes: : [descrizione dell’errore restituito dal driver PyMongo]
    - se invece la cancellazione ha avuto successo, occorre porre (update basata su campo LinesEnquiredLog.\_id):
      * LinesEnquiredLog.data.blkStatus: 4 (Deleted)
      * LinesEnquiredLog data.rlsDelDate: data di sistema
* richieste da cancellare (document con data.blkStatus=4 e la cui richiesta di sblocco è più vecchia di un anno dalla data di sistema): se trovata, occorre cancellare il document contenente la richiesta di blocco e di sblocco – la query per recuperare i documenti da cancellare è la seguente:

db.LinesEnquiredLog.find({data.rlsDate : {$lte : new Date(ISODate().getTime()-1000 \* 3600 \* 24 \* 366)}, data.blkStatus : 4})

la cancellazione procede quindi sulla base dei documenti, in particolare dei loro campi “\_id”, recuperati, ad es.:

db.LinesEnquiredLog.remove({ "\_id" : ObjectId("5d52afcf7bbc7596386df6b0"))

Tutte le operazioni di aggiornamento (update, remove) sulla collection LinesEnquiredLog vanno basate sul campo \_id: potrebbero esistere nello stesso momento più richieste di blocco/sblocco per la stessa utenza (su periodo più o meno sovrapposti

Lo script traccerà le principali operazioni eseguite nel file di log

**adapEnquiredTrRelease.log**

che viene fornito nella stessa directory che ospita lo script e il cui contenuto sarà alimentato in append mode dallo script stesso.

Esempi di tracciamento:

[2019-09-30 12:00:00] ADaP – Servizio di Sblocco Traffico v.1.0 – inizio:

[2019-09-30 12:00:00] LinesEnquiredLog: non trovata alcuna richiesta di sblocco

[2019-09-30 12:00:00] ADaP – Servizio di Sblocco Traffico – fine

[2019-09-30 12:15:00] ADaP – Servizio di Sblocco Traffico v.1.0 – inizio:

[2019-09-30 12:15:00] LinesEnquiredLog: trovata richiesta di sblocco valida \_id: 5d52afcf7bbc7596386df6b0 (utenza: 3351028124, data dal: 01/06/2019, data al: 31/08/2019)

[2019-09-30 12:15:05] sblocco traffico eseguito con successo

[2019-09-30 12:15:00] ADaP – Servizio di Sblocco Traffico – fine

[2019-09-30 12:30:00] ADaP – Servizio di Sblocco Traffico v.1.0 – inizio:

[2019-09-30 12:30:00] LinesEnquiredLog: trovata richiesta di blocco sblocco DA CANCELLARE \_id: 5d52afcf7bbc7596386df6b0 (utenza: 3351028124, data dal: 01/06/2019, data al: 31/08/2019)

[2019-09-30 12:30:05] cancellazione richiesta eseguita con successo

[2019-09-30 12:30:00] ADaP – Servizio di Sblocco Traffico – fine

La manutenzione del file di log è delegata all’amministratore di sistema.

**Installazione dello script**

Per poter eseguire gli script è necessario installare Python 3 (ultima versione) e alcune librerie:

* Pip
* Pip install Pymongo
* Pip install Pprint
* Pip install dnspython
* Pip install python-dateutil