ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA
SCUOLA DI INGEGNERIA E ARCHITETTURA

Corso di Laurea Magistrale in Ingegneria Informatica

PROGETTO DI UN'APPLICAZIONE WEB PER SERVIZI DI MOBILITÀ URBANA TRAMITE L'USO DI OPENDATA

Tesi in

Laboratorio di Reti di Telecomunicazioni LM

Relatore
Prof. Walter Cerroni

Presentata da
Andrea Giulia Cialotti

Correlatore
Prof. Franco Callegati

SESSIONE I
ANNO ACCADEMICO 2012/2013

Il calcolatore è straordinariamente veloce, accurato e stupido.
L'uomo è incredibilmente lento, impreciso e creativo.
L'insieme dei due costituisce una forza incalcolabile.
(Albert Einstein)

Indice

Sommario

Apertura, partecipazione, collaborazione e possibilità di creare una competenza collettiva sono i motivi che portano alla nascita e alla diffusione degli OpenData, i quali favoriscono l'interoperabilità e la trasparenza dei governi nei confronti dei cittadini, inoltre migliorano l'efficienza delle amministrazioni pubbliche, e mettono in grado le persone di affrontare meglio le decisioni che riguardano la loro vita potendo utilizzare informazioni che prima non erano disponibili.
Il lavoro svolto nell'elaborato si colloca nel settore della mobilità urbana e nasce dalla decisione dell'azienda Tper di mettere a disposizione i propri dati in formato OpenData sul sito web *http://www.tper.it/tper-open-data*. L'obiettivo principale è la realizzazione di un'applicazione in grado di fornire informazioni in tempo reale sulle linee di autobus, e relative fermate, in una determinata area di interesse.
Il primo capitolo definisce gli OpenData, cioè dati accessibili a chiunque senza vincoli di natura giuridica e tecnologica, sottolineandone l'importanza e la notevole diffusione che hanno avuto negli ultimi anni, infine presenta alcuni progetti attualmente attivi come *OpenStreetMap* e *wheredoesmymoneygo*.
Il secondo capitolo introduce i requisiti dell'applicazione in particolare illustrando le modalità con le quali l'utente può interagire con essa, ovvero inserendo nella richiesta la propria posizione nei formati di indirizzo postale o di coordinate geografiche, ed il tempo di percorrenza massimo che è disposto ad impiegare per raggiungere le fermate. Inoltre presenta gli OpenData di Tper e l'elaborazione necessaria per poterli utilizzare nell'applicazione.
Il terzo capitolo espone il processo di produzione dell'applicazione descrivendo nel dettaglio come vengono effettuate la traduzione da indirizzo postale a coordinate geografiche, la selezione delle fermate che rispettano i requisiti specificati dall'utente, e l'interazione con il Web Service *HelloBus*.
Il quarto capitolo mostra il funzionamento dell'applicazione utilizzando al-

cuni esempi che affrontano i principali casi d'uso.

Capitolo 1

Introduzione agli OpenData

L'idea che la conoscenza deve essere un bene comune era già nota ancor prima della nascita del World Wide Web. Nel 1942 il sociologo Robert King Merton discusse l'importanza di mettere i risultati della ricerca a disposizione di chiunque.
E' fondamentale che ogni ricercatore contribuisca alla conoscenza comune, rinunciando ai propri diritti di proprietà intellettuale sulla sua ricerca per permettere alla conoscenza di andare avanti. Il World Wide Web viene inventato proprio per questi motivi, e per favorire la comunicazione e la cooperazione tra i ricercatori del CERN da Tim Berners-Lee nel 1989.
Il primo ambito in cui è stata sperimentata questa cultura è l'Open Source che si fonda sui concetti di apertura, partecipazione e collaborazione. Si può imparare dal lavoro di altri ma in cambio si deve poi pubblicare il proprio: in questo modo si crea una competenza collettiva. Nel 2009 come si evince dall'Open Government Directive [1] emanata dal presidente Obama, la quale indica delle linee guida per favorire la trasparenza del governo, e dall'intervento di Tim Berners-Lee al TED (Technology, Entertainment, Design) [2], tali principi trovano utile applicazione anche nella raccolta e gestione dei dati.
I dati prodotti dalla pubblica amministrazione, in quanto finanziati da denaro pubblico, devono ritornare ai contribuenti, e alla comunità in generale, sotto forma di dati aperti e universalmente disponibili.
Questi sono i motivi che hanno portato alla nascita degli OpenData, i quali favoriscono l'interoperabilità, cioè la possibilità di elaborare dati provenienti da fonti diverse e utilizzarli per migliorare prodotti e servizi esistenti o per

svilupparne di nuovi. Inoltre favoriscono la trasparenza dei governi nei confronti dei cittadini, migliorano l'efficienza delle amministrazioni pubbliche, promuovono la collaborazione, mettono in grado le persone di affrontare meglio le decisioni che riguardano la loro vita, potendo utilizzare informazioni che prima non erano disponibili.
La diffusione degli OpenData non porta solo un aumento dei dati, ma un aumento della qualità dei dati in termini di valore economico, sociale e culturale che scaturisce dagli infiniti usi che possono esserne fatti.
Gli OpenData non focalizzano la loro utilità esclusivamente nel settore delle Pubbliche Amministrazioni, bensì trovano utile applicazione anche altri settori tra i quali: scientifico, finanziario, statistico, ambientale, culturale, geografico, meteorologico e trasportuale.
Esempi di applicazioni che utilizzano OpenData sono *"wheredoesmymoneygo"* un progetto inglese che permette di vedere come viene speso il denaro incassato dal governo mediante le tasse, *"openstreetmap"* il quale permette di visualizzare e modificare cartine geografiche dell'interno pianeta con un livello di dettaglio molto elevato, *"OpenParlamento"* che consente di monitorare le attività svolte nel parlamento e il processo di attuazione delle leggi.

1.1 Cosa sono gli OpenData

Secondo la `OpenDefinition` [3] gli OpenData sono:
"dati che possono essere utilizzati gratuitamente, riutilizzati e ridistribuiti da tutti. L'unico vincolo che possono avere nell'utilizzo, è al più quello di indicare la fonte e condividerli utilizzando la stessa licenza da questa adottata"
Tale definizione implica che gli *OpenData* sono dati accessibili a chiunque, senza vincoli di natura giuridica, ad esempio licenze che ne vincolino la vendita o la distribuzione a terzi, e di natura tecnologica, ad esempio dati in formati proprietari, o non provvisti di documentazione adeguata che ne consenta la comprensione. La licenza può tuttavia imporre l'attribuizione, cioè l'obbligo di citare la fonte dalla quale proviene il contenuto, e l'integrità, cioè nel caso in cui un database venga distribuito in forma modificata deve avere un nome o un numero di versione che lo distingua dall'opera originaria.

Gli standard internazionali per la trasparenza [4], hanno stabilito che gli OpenData sono:

1. **Completi**
   Tutti gli elementi appartenenti ad un database pubblico devono essere pubblicati, inclusi i contenuti archiviati in formato non digitale (eventualmente digitalizzandoli), ed i dati e le spiegazioni dei metodi utilizzati per generare i dati aggregati e derivati.

2. **Elementari**
   I dati devono essere raccolti alla fonte, e devono essere pubblicati con il livello di dettaglio con cui sono stati raccolti. I dati non sempre sono in un formato utilizzabile quando vengono raccolti, quindi in questi casi si può eseguire un'elaborazione post-raccolta, a condizione che i dati risultanti siano in formato che rifletta la capacità e la granularità del meccanismo di raccolta originale. I dati non devono MAI essere persi.

3. **Tempestivi**
   I dati devono essere resi disponibili il più rapidamente possibile al fine di massimizzare il valore per il pubblico. Gli aggiornamenti devono essere facilmente individuabili anche all'interno di database di grandi dimensioni, attraverso meccanismi come feed RSS, funzioni di ricerca in grado di filtrare in base alla data, e archivi che contengono snapshot del database prese a intervalli regolari.

4. **Accessibili**
   I dati devono essere a disposizione di chiunque, per la più vasta gamma di scopi possibili. Quindi devono essere condivisibili in modo semplice, ogni pagina o documento pubblicato deve essere identificato con un URI, che può essere diffuso via email, siti web, o social network. L'utente deve poter accedere al database completo senza dover effettuare nessuna registrazione o pagamento. Inoltre dovrebbe essere possibile il download dell'intero database attraverso protocolli come l'FTP, o l'r-sync (per sistemi Unix), e i dati dovrebbero essere forniti di API ben documentate, al fine di consentire anche elaborazioni automatizzate, ad esempio attraverso agenti software. L'accessibilità dovrebbe essere garantita anche per persone con disabilità, ad esempio per i non vedenti attraverso software screen reader che interpreta il dato e lo invia ad un monitor Braille, o ne effettua una riproduzione audio. Infine dovrebbe

essere presente un software che consenta di tradurre il documento nella lingua desiderata, per permettere di usufruire del dato anche a persone che non conoscono la lingua.

5. Utilizzabili da una macchina

I dati devono essere disponibili sia un formato leggibile dagli utenti, sia in uno strutturato in modo da permettere l'elaborazione automatizzata. Per esempio la pubblicazione di intervento ad una conferenza in formato video o audio, dovrebbe essere accompagnata dalla sua trascrizione. In particolare i database devono essere forniti in un formato semplice da processare, come ad esempio CSV, JSON o XML e devono sempre essere accompagnati da una documentazione chiara che spieghi dettagliatamente il significato di ogni campo.

6. Non proprietari

I dati devono essere disponibili in un formato aperto cioè un formato: sul quale nessun ente ha il controllo esclusivo; non soggetto a controlli di proprietà intellettuale in nessun paese; per il quale i documenti che ne definiscono la struttura sono liberamente accessibili. HTML e XML sono esempi di formati aperti.

7. Utilizzabili liberamente

I possibili utilizzatori o usi(ad esempio l'uso commerciale) dei dati non devono essere limitati da protezioni della proprietà intellettuale, come ad esempio copyright, marchi, brevetti. Questo deve essere specificato esplicitamente nella licenza (ulteriori dettagli nella sezione 1.2).

8. Sindacabili

Ogni ente pubblico o privato, quando pubblica dei dati deve fornire i contatti di una persona, che è stata incaricata di rispondere a domande e reclami sui contenuti pubblicati.

9. Individuabili

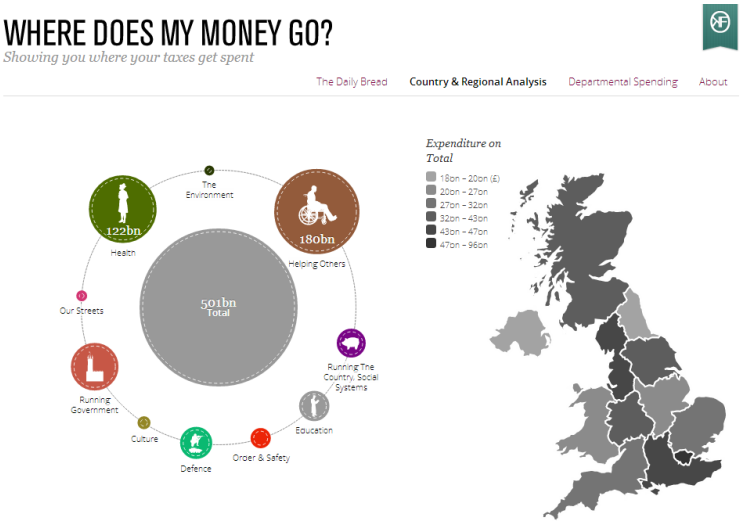
I dati devono essere trovati facilmente da chi li sta cercando. A tal fine devono essere inclusi in liste appropriate, le quali devono essere accessibile ai motori di ricerca, e mantenute aggiornate. Inoltre è opportuno fornire i siti web contenenti i dati di sitemap complete e includerle nei maggiori motori di ricerca.

10. Permanenti
Con il passare del tempo i dati devono essere archiviati in un modo che soddisfi i criteri precedenti.

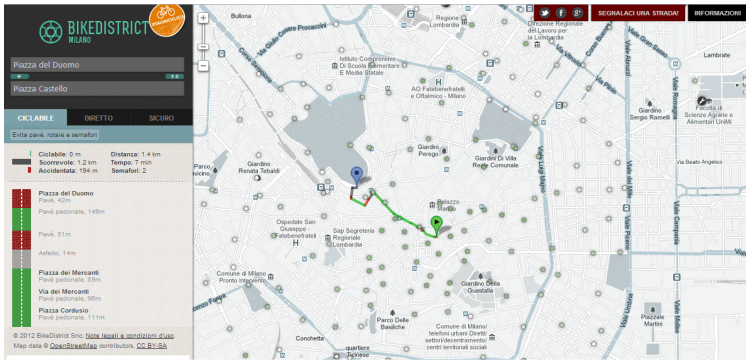
Gli OpenData favoriscono l'**interoperabilità**, cioè la possibilità di elaborare dati provenienti da fonti diverse e utilizzarli per migliorare prodotti e servizi esistenti o per svilupparne di nuovi. Inoltre favoriscono la trasparenza dei governi nei confronti dei cittadini, migliorano l'efficienza delle amministrazioni pubbliche, promuovono la collaborazione, mettono in grado le persone di affrontare meglio le decisioni che riguardano la loro vita, potendo utilizzare informazioni che prima non erano disponibili. La diffusione degli OpenData non porta solo un aumento dei dati, ma un aumento della qualità dei dati in termini di valore economico, sociale e culturale che scaturisce dagli infiniti usi che possono esserne fatti.
Esempi di progetti che utilizzano gli Open Data sono:

- *husetsweb.dk* [5] che permette alle famiglie di vedere come massimizzare l'efficienza dell'energia nella loro casa, includendo una pianificazione finanziaria e trovando costruttori che svolgano i lavori necessari. Esso utilizza le informazioni catastali e quelle relative ai sussidi governativi;

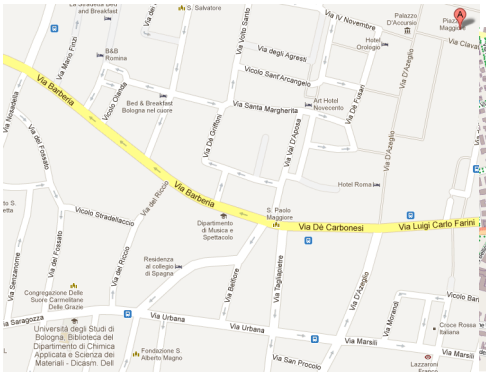
- *wheredoesmymoneygo* [6] un progetto inglese che permette di vedere come il governo spende il denaro incassato con le tasse.



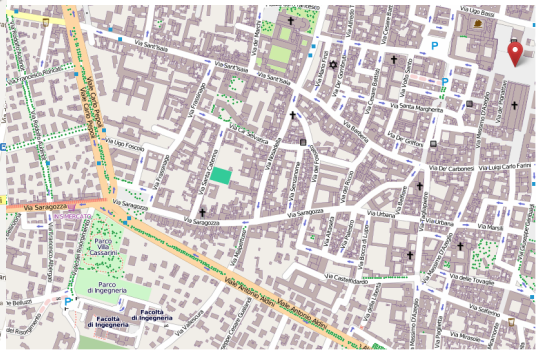
- *BikeDistrict* il quale calcola gli itinerari più adatti alle esigenze di chi utilizza la bici a Milano, permettendo di scegliere il percorso in base al tipo di terreno (asfaltato, sterrato), alla presenza di piste ciclabili, di semafori, e di rotaie.



- *OpenStreetMap* [7] equivalente di GoogleMaps con dati e codice in un formato *aperto*. E' molto più accurato di GoogleMaps in quanto è stato realizzato collettivamente da volontari, e chiunque può modificare le mappe e aggiungere dettagli.



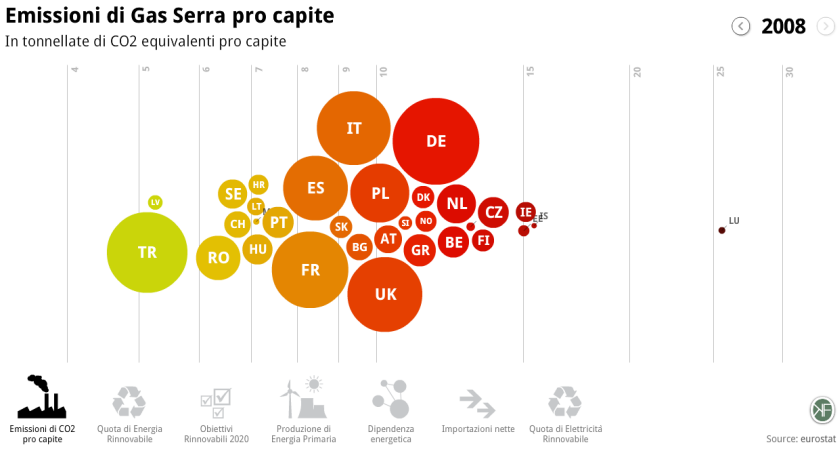
(a) *Piazza Maggiore, Bologna con GoogleMaps*



(b) *Piazza Maggiore, Bologna con OpenStreetMap*

- *Google Translate* il servizio di traduzione di Google, il quale utilizzando i numerosi documenti dell'unione europea disponibili in tutte le lingue per addestrare gli algoritmi di traduzione è migliorato notevolmente.

- *energy.publicdata.eu/ee/vis.html* [8] che permette di visualizzare delle statistiche sull'energia in europa in diversi anni, ad esempio percentuale di energia rinnovabile, dipendenza energetica dei paesi, emissione di anidride carbonica.

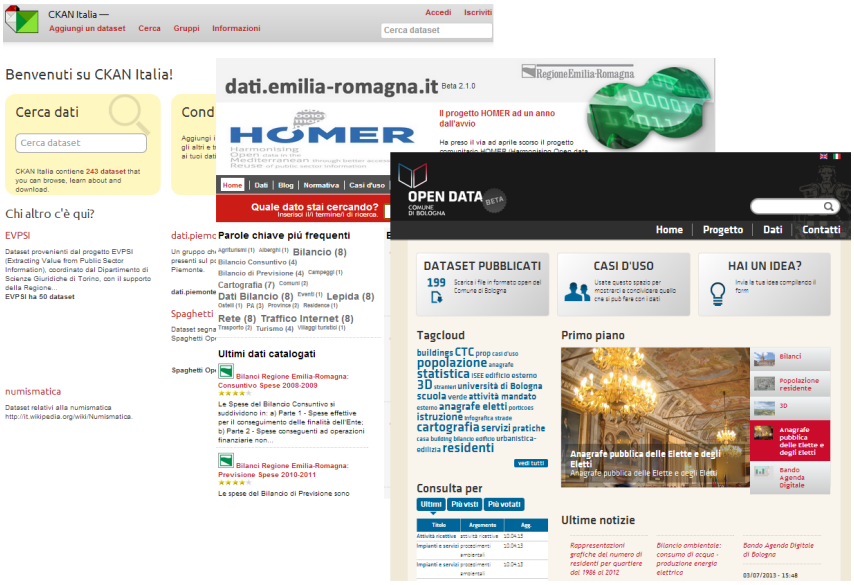


- *OpenParlamento* dove si possono monitorare le attività svolte nel parlamento e il processo di attuazione delle leggi, permettendo ai cittadini di sapere esattamente cosa sta succedendo e quali parlamentari sono responsabili delle decisioni prese.

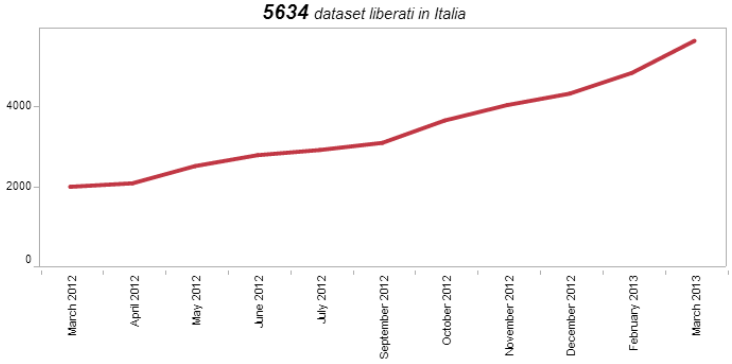
In Italia si iniziò a parlare di OpenData nel 2007 con il progetto OpenStreetMap, quando alcune amministrazioni locali grazie all'iniziativa di alcuni volontari hanno pubblicato con licenza aperta i dati riguardanti i propri

stradari. Dalla fine del 2011 è online *dati.gov.it*.

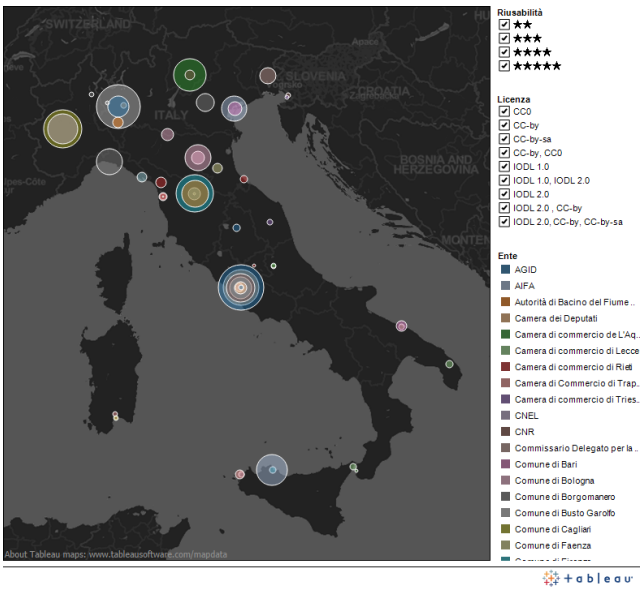
Poi in seguito sono stati pubblicati *dati.emilia-romagna.it*, *dati.comune.bologna.it*, *it.ckan.net* (sito per la catalogazione di dataset, basato sul software CKAN), e *datiopen.it* che offre gratuitamente tutti i servizi di caricamento, visualizzazione e segnalazione, e numerosi altri.



A marzo 2013, come riporta *dati.gov.it* in Italia si contano 5634 dataset resi disponibili.



Come si può vedere anche dalla cartina seguente, che mostra la distribuzione delle amministrazioni che rilasciano OpenData in Italia, la maggior parte dei dataset resi disponibili è localizzata nelle regioni settentrionali. Si noti che l'area della bolla è direttamente proporzionale al numero dei dataset rilasciati dalla specifica amministrazione.



Nonostante il movimento Open Data in Italia sia relativamente giovane, rappresenta una realtà vitale e attiva anche nei confronti delle altre realtà eu-

ropee, trascinato soprattutto dall'obbiettivo della trasparenza amministrativa. Infatti l'Agenzia per l'Italia Digitale ha stabilito dal 19 marzo 2013 l'*OpenData by default* cioè tutti i dati e documenti che le pubbliche amministrazioni pubblicano con qualsiasi modalità, senza l'espressa adozione di una licenza d'uso, si intendono rilasciati come dati aperti.

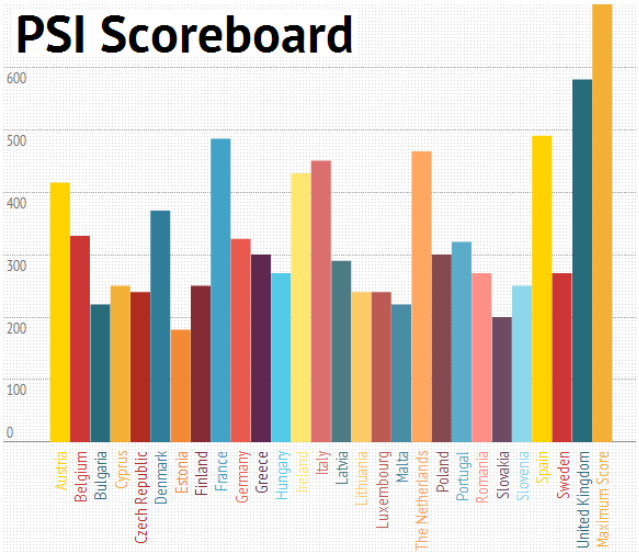


Figura 1.1: *Classifica elaborata dalla EPSI (European Public Sector Information [9]) ottenuta dalla media di diversi fattori: l'implementazione della direttiva PSI, la pratica di riuso, i formati, il prezzo, accordi esclusivi, PSI locale, eventi e attività. L'Italia si trova al* 5° *posto, e per gli eventi e attività al* 1° *posto con la Gran Bretagna.*

1.2 Come far diventare i dati *OpenData*

Quando un titolare di un database decide di farlo diventare *aperto* deve privilegiare la tempestività, ovvero preferire di pubblicare parti più piccole del database ma più velocemente. Non è vi è nessuna richiesta o obbligo di rendere tutti i dati *aperti*, anche perchè non è sempre detto che tutti i dati che vengono pubblicati suscitino reale interesse alle persone. Infatti è fondamentale essere sempre in contatto con i potenziali utilizzatori (cittadini,

imprenditori, sviluppatori) del dataset che è stato pubblicato, al fine di capire come aumentare il valore e la qualità dei dati pubblicati. Inoltre si devono tenere in considerazione tutte le conseguenze che comporta la pubblicazione dei dati, come ad esempio la possibilità di ricavare dati derivati mediante elaborazioni, oppure la possibilità di cambiare il modo di presentarli, oppure ancora che vengano utilizzati all'interno di sistemi più complessi insieme a dati provenienti da sorgenti diverse.
L'Open Knowledge Foundation ha identificato quattro grandi fasi per rendere i dati *aperti*, molti delle quali possono essere fatte contemporaneamente: scegliere il dataset, stabilire il tipo di licenza, rendere i dati disponibili, rendere i dati rintracciabili.

Fase 1 : Scegliere il dataset

Durante questa fase si deve capire quale dataset può essere reso pubblico, suscita interesse alla community, può essere pubblicato interamente così com'è, oppure è pubblicabile solo una sua parte, oppure è pubblicabile ma solo dopo alcune modifiche. In base a queste considerazioni si deve stilare una lista dei database candidati a diventare *aperti* e successivamente chiedere dei riscontri via web, tramite i social network, mailing list, forum, blog, evitando di richiedere registrazioni le quali ridurrebbero il numero delle risposte.
Inoltre una buona pratica è far riferimento ad eventuali OpenData già presenti, specialmente quelli nel settore a cui appartiene il dataset che si vuole pubblicare e divulgare la notizia della pubblicazione del dataset il più possibile, soprattutto alle autorità competenti e agli esperti del settore del dataset.

Fase 2 : Stabilire il tipo di licenza

Nella maggior parte delle giurisdizioni ci sono delle leggi sulla proprietà intellettuale dei dati che impediscono a terzi di utilizzare e distribuire i dati senza un permesso esplicito del titolare. Quando si decide si rendere i propri dati *aperti* è fondamentale assegnargli una licenza. Il progetto Open Data Commons [10] della Open Knowledge Foundation [11], ha specificato alcune linee guida da seguire per definire le Licenze riguardanti gli OpenData. In particolare ha definito alcune tipi di licenza, specificando che questi documenti non hanno alcun valore giuridico, ed è opportuno che siano analizzati

da un legale e confrontati con la legislatura del paese di interesse, prima di essere utilizzati:

ODC Public Domain Dedication and License

Consente di :

- copiare, distribuire, utilizzare il database per qualsiasi fine;
- elaborare il database per produrre una nuova opera;
- modificare, trasformare il database.

Non impone nessun vincolo.
Per attribuire ad un database tale licenza si deve copiare il testo integrale in un file `LICENSE.txt`, che dovrà essere messo all'interno del database, oppure riportare la seguente dichiarazione in tutti i siti o le locazioni in cui si trova il database:

```
This {DATA(BASE)-NAME} is made available under
the Public Domain Dedication and License
version v1.0 whose full text can be found at
http://opendatacommons.org/licenses/pddl/
```

ODC Open Database License

Si riferisce sia ai dati contenuti nel database, sia alla loro struttura, e alle modalità con le quali si può accedere ad essi.
Consente di :

- copiare, distribuire, utilizzare il database per qualsiasi fine;
- elaborare il database per produrre una nuova opera;
- modificare, trasformare il database.

Impone i seguenti vincoli:

- *Attribuzione*: si deve citare la fonte originale in ogni utilizzo pubblico del database, o opera prodotta con il database nel modo descritto nella licenza. Inoltre deve essere precisato che ogni informazione relativa al database originale deve rimanere intatta, ciò significa che se viene creata un'estensione del database con dei dati derivati, questa dovrà avere un nome e una versione che la distinguano dal database originale.

- *Condivisione allo stesso modo (Share-alike)*: se si vuole pubblicare una versione modificata del database, o un'opera prodotta da una versione modificata del database, lo si deve fare con lo stesso tipo di licenza del database originale.
- *Mantenerlo aperto*: se si distribuisce il database, o una sua versione modificata si possono usare delle misure tecnologiche per limitarne la diffusione (tipo Digital Rights Management ad esempio il codice seriale), a condizione di distribuirne anche una versione senza nessuna misura limitativa.

Per attribuire ad un database tale licenza si deve copiare il testo integrale in un file `LICENSE.txt`, che dovrà essere messo all'interno del database, oppure riportare la seguente dichiarazione in tutti i siti o le locazioni in cui si trova il database:

```
This {DATA(BASE)-NAME} is made available under
Open Database License whose full text can be found
at http://opendatacommons.org/licenses/odbl/.
Any rights in individual contents of the database
are licensed under the Database Contents License
whose text can be found at
http://opendatacommons.org/licenses/dbcl/
```

Open Data Commons Attribution License

Consente di:

- copiare, distribuire, utilizzare il database per qualsiasi fine;
- elaborare il database per produrre una nuova opera;
- modificare, trasformare il database.

Impone i seguenti vincoli:

- *Attribuzione*: si deve citare la fonte originale in ogni utilizzo pubblico del database, o opera prodotta con il database nel modo descritto nella licenza. Inoltre deve essere precisato che ogni informazione relativa al database originale deve rimanere intatta, ciò significa che se viene creata un'estensione del database con dei dati derivati, questa dovrà avere un nome e una versione che la distinguano dal database originale.

Per attribuire ad un database tale licenza si deve copiare il testo integrale in un file `LICENSE.txt`, che dovrà essere messo all'interno del database, oppure riportare la seguente dichiarazione in tutti i siti o le locazioni in cui si trova il database:

```
Contains information from DATABASE NAME which is made
available under the Attribution License
whose text can be found at
http://opendatacommons.org/licenses/by/1.0/
```

Creative Commons Zero License (CC0)

(a) *Marchio licenze CC*

(b) *Licenza CC0*

Fa parte delle licenze Creative Commons [12], le quali forniscono la licenza con un design a tre livelli:

1. *documento legale*: con strutturazione e termini linguistici legali;
2. *documento interpretabile da persone normali*: non ha valenza legale, serve per far comprendere a persone non competenti quali diritti da la licenza;
3. *documento machine-readable*: agevola la diffusione e la rintracciabilità della licenza, utilizzando il linguaggio Creative Commons Rights Expression Language (CC REL) il quale specifica come la licenza può essere descritta in RDF (ulteriori dettagli in 1.3.1).

La licenza CC0 [13] indica che l'autore ha reso la sua opera di dominio pubblico, rinunciando a tutti i suoi diritti e diritti connessi o simili che detiene su di essa, quali i suoi diritti morali (per quanto rinunciabili), i suoi diritti all'immagine o alla riservatezza, diritti che lo proteggono contro la concorrenza sleale, e diritti sulle banche di dati che limitino l'estrazione, la disseminazione ed il riuso dei dati. Inoltre egli non fornisce nessuna garanzia sull'opera, e declina ogni responsabilità per tutti gli usi che possono essere fatti dell'opera nella misura consentita dalla legge.

Per quanto riguarda l'Italia, è stata sviluppata l'**Italian Open Data License (IODL)**:

Esistono due versioni di questa licenza, ovvero IODL v1.0 [14] pubblicata nel 2010 e IODL v2.0 [15] pubblicata nel 2012, le quali consentono di:

- copiare, distribuire, utilizzare il database per qualsiasi fine;
- elaborare il database per produrre una nuova opera;
- modificare, trasformare il database.

L'Italian Open Data License v1.0 si distingue dalla v2.0 perchè la prima impone i vincoli di attribuizione e di condivisione allo stesso modo (compatibile con la licenza Open Database), mentre la seconda **solo** il vincolo di attribuizione (compatibile con la Open Data Commons Attribution).

Fase 3 : Rendere i dati disponibili

I database devono essere disposizione di chiunque, preferibilmente attraverso un download gratuito su un sito web, in un formato comprensibile dalle persone, ed anche in un formato che ne permetta l'utilizzo attraverso un processo automatico. Inoltre deve essere possibile accedervi da qualsiasi piattaforma. Il modo più semplice è permettere il download del database sul proprio sito web, ma in questo modo si limita l'utilizzo del database alle sole persone che sono a conoscenza del sito. Per questa ragione sono presenti diversi siti web che si occupano di raccogliere database relativi ad un particolare settore, ad esempio *cosm.com* è pensato per connettere dispositivi e apps, scambiarsi dati e idee tra sviluppatori. Altri modi per condividere il database sono attraverso un server FTP, o con BitTorrent.
I dati possono essere pubblicati anche con API (Application Programming Interface) le quali permettono di eseguire diverse elaborazioni sui di essi, ad esempio selezionare un sottoinsieme di dati dal database, con il vantaggio che poichè si connettono al database in tempo reale restituiranno sempre i dati più aggiornati. Un esempio di approccio via API è l'accesso ai dati mediante Web Services, con query la cui sintassi è descritta in formato XML all'interno della Web Service Description. I dati pubblicati con le API sono

strettamente dipendenti dal fornitore del database, perciò se questo effettua delle modifiche strutturali del database potrebbero non essere disponibili per un certo intervallo di tempo.
Diversamente quelli diffusi all'interno di file potrebbero non essere aggiornati, però rimangono sempre disponibili anche quando il fornitore effettua delle modifiche.

Fase 4 : Rendere i dati rintracciabili

Gli OpenData non hanno alcun valore se non vengono utilizzati da nessuno. E' fondamentale utilizzare tutti i mezzi possibili per divulgare il fatto che il database è diventato *aperto* ad esempio social network, mailing list, forum, conferenze, barcamps (conferenze online), e hackadays (hackathlon). Inoltre un approccio tipico è usare i tool esistenti che si occupano di raccogliere e indicizzare i database, come ad esempio *datahub.io* che contiene database provenienti da tutto il mondo. Oltre a questi sono disponibili online anche dei tool per realizzare queste raccolte di OpenData da zero, uno di questi è *ckan.org*.

1.3 Linked Open Data e Web Semantico

I Linked Open Data sono degli Open Data connessi tra loro tramite delle relazioni di semantica, le quali sono esprimibili nei dati stessi in un formato comprensibile dalle persone e dalle macchine che si chiama RDF (Resource Description Framework). In particolare viene utilizzato il modello RDF per descrivere il significato dei dati, e i link RDF per collegare i dati sulla base del loro significato. Infatti i Linked Open Data stanno alla base della teoria sul Web Semantico, un web nel quale si potrà navigare non solo sulla base della sintassi dei dati, ma anche sulla base del loro significato sfruttando i collegamenti RDF.
Ad esempio, quando un utente sta cercando dei dati relativi ad una persona da una sorgente, potrebbe essere interessato alle informazioni relative alla città dove abita quella persona. Queste informazioni possono essere ottenute seguendo un collegamento RDF dal dataset contenente i dati della persona, ad un altro dataset che le contiene. Quindi la ricerca di un documento non viene più fatta mediante collegamenti ipertestuali ma tramite collegamenti RDF. I collegamenti RDF possono essere interpretati da motori di ricerca

semantici, che permetteranno ricerche sofisticate con query che non restituiranno più dei semplici link a pagine HTML, ma dati strutturati che potranno essere utilizzati immediatamente all'interno di altre applicazioni.
Il Linking Open Data è un progetto del W3C che si occupa di pubblicare dataset in formato RDF e creare collegamenti RDF tra gli elementi provenienti da dataset diversi.
La figura 1.2 mostra i dataset resi aperti e collegati nel 2011, i quali sono circa 295, composti da oltre 31 miliardi di triple RDF, collegate da circa 504 milioni di link RDF.

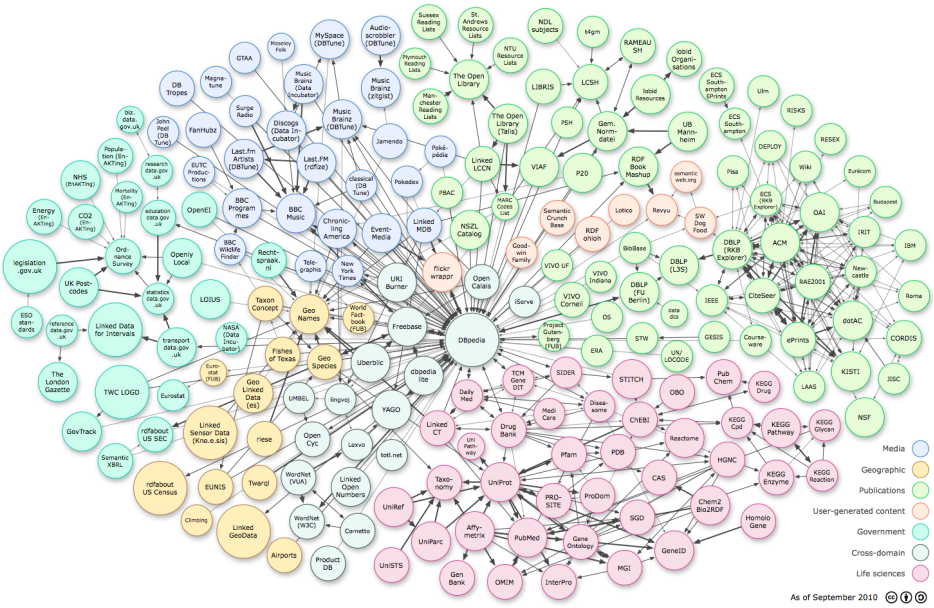


Figura 1.2: *Linking Open Data cloud (2011): dataset pubblicati e le loro relazioni.*

Tim Berners-Lee, l'inventore del World Wide Web, nel 2006 ha indicato un insieme di regole da seguire per pubblicare i dati sul web, in modo che essi diventino parte di uno spazio di dati globale:

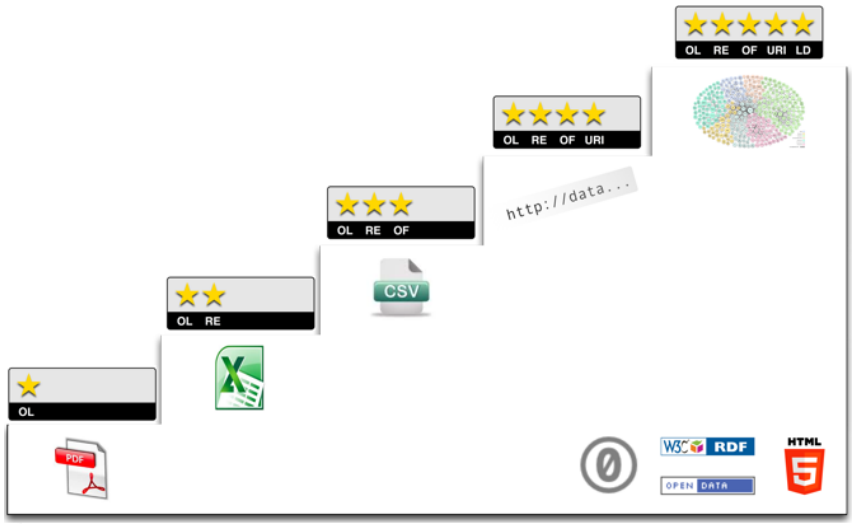
1. indentificare le risorse con Uniform Resource Identifier (URI);

2. usare HTTP URI in modo che si possano fare riferimenti facilmente e possano essere cercati sia da persone che da user agent;

3. quando qualcuno cerca una risorsa, fornirgli informazioni utili e collegate all'URI della risorsa, utilizzando i formati standard (RDF, SPARQL);

4. includere collegamenti ad altre URI collegate al contenuto mostrato, per migliorare la ricerca di altre informazioni utili nel Web.

Inoltre ha anche classificato il grado di apertura dei dati, in cinque livelli progressivi, associati al numero di stelle:



1 stella: il dato è disponibile sul web (in qualsiasi formato) ma con una licenza aperta;

2 stelle: il dato è disponibile in un formato strutturato che può essere interpretato da un software (per esempio un foglio di calcolo Microsoft Excel);

3 stelle: il dato è in un formato strutturato non proprietario (CSV);

4 stelle: oltre a rispettare tutti i criteri precedenti, il dato utilizza standard web interpretabili dalle applicazioni che ne descrivono il significato (RDF);

5 stelle: il dato rispetta tutti gli altri criteri e inoltre contiene collegamenti ad altri dati (linked data) al fine di fornire un contesto alle proprie informazioni.

1.3.1 Resource Description Framework

RDF è un framework per descrivere le risorse web e le loro relazioni semantiche attraverso grafici, in un modo interpretabile sia dalle persone che dalle macchine (con la sintassi XML). Il modello RDF codifica i dati nella forma di triple $< soggetto, predicato, oggetto >$.
Il *soggetto* di una tripla è l'URI che identifica la risorsa descritta. L'*oggetto* può essere sia un semplice valore letterale, tipo una stringa, un numero, o una data; o l'URI di un'altra risorsa che ha una certa relazione con il soggetto. Il *predicato* indica che tipo di relazione intercorre tra il *soggetto* e l'*oggetto*, ad esempio ″data di nascita″, ″datore di lavoro″. Il *predicato* è anch'esso un URI. Gli URI dei predicati si trovano nei *vocabolari*, i quali sono raccolte di URI che possono essere usate per rappresentare informazioni riguardanti un certo dominio. Tra i *vocabolari* well-known sviluppati dalla Semantic Web community si possono trovare:

- Friend-of-a-Friend (FOAF), per descrivere i legami tra le persone;
- Dublin Core (DC), per descrivere qualsiasi materiale digitale;
- Semantically-Interlinked Online Communities (SIOC), rappresenta le principali community online;
- Description of a Project (DOAP), per descrivere progetti;
- Simple Knowledge Organization System (SKOS) per rappresentare glossari, classificazioni, tassonomie e qualsiasi tipo di vocabolario strutturato;
- Geonames è un vocabolario geografico open con oltre 10 milioni di nomi di luoghi;
- Music Ontology, contiene termini per descrivere artisti, album e pezzi;
- Review Vocabulary, per descrivere le recensioni;
- Creative Commons (CC), contiene i vocaboli delle licenze.

Si possono anche definire dei vocaboli nuovi, che non sono presenti nelle ontologie già definiti attraverso RDF Vocabulary Description Language 1.0: RDF Schema. I link RDF sono fondamentali per il Web dei dati (o web

semantico) perchè permettono di collegare isole di dati in uno spazio globale di dati interconnessi, e di scoprire sorgenti di dati aggiuntive.
I collegamenti RDF si dividono in collegamenti *interni* e collegamenti *esterni*: i primi sono triple RDF nelle quali il soggetto e l'oggetto sono riferimenti URI che puntano alla stessa sorgente di dati, mentre i secondi sono triple RDF nelle quali il soggetto è un riferimento URI nello spazio dei nomi di un dataset, mentre il predicato e/o l'oggetto sono riferimenti URI che puntano a spazi di nomi di altri dataset. Dereferenziando queste URI si ottiene la descrizione delle risorse collegate fornita dal server remoto. Si noti che per dereferenziare si intende l'atto di recuperare una rappresentazione di una risorsa o la descrizione semantica di una risorsa creata dal possessore dell'URI. Se la rappresentazione della risorsa recuperata è un RDF contenente ulteriori URI, questa operazione può essere fatta ricorsivamente. Questo è il meccanismo con il quale si naviga nel web semantico.
Ci sono 3 tipi di collegamenti RDF:

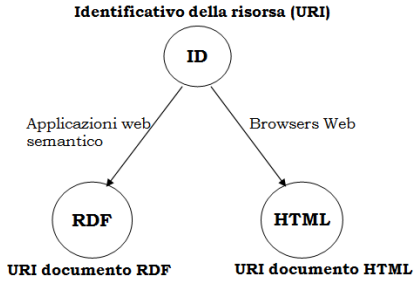
- **relazionali:** puntano alle cose connesse in altre sorgenti di dati, per esempio, altre persone, luoghi o generi. Ad esempio i collegamenti di relazioni permettono alle persone di ottenere informazioni riguardanti il luogo dove vivono, o dati bibliografici riguardanti le pubblicazioni che hanno fatto.

- **di identità:** puntano agli alias URI usati da altre sorgenti di dati per identificare lo stesso oggetto reale o concetto astratto. I collegamenti di identità permettono ai client di recuperare ulteriori descrizioni riguardanti un'entità da altre sorgenti di dati. Inoltre hanno un'importante funzione sociale perchè permettono di esprimere diversi punti di vista riguardo alle entità e ai concetti.

- **di vocabolario:** puntano dai dati alle definizioni nei vocabolari dei termini che sono usati per rappresentare i dati, così come da queste definizioni alle definizioni dei relativi termini negli altri vocabolari. I collegamenti di vocabolario rendono i dati autodescrittivi e permettono alle applicazioni di comprendere e integrare i dati attraverso l'uso dei vocabolari.

Non si devono confondere gli identificatori di documenti Web, con gli identificatori di altre risorse. Un determinato URI può identificare un documento Web oppure una risorsa (anche esterna al Web).

La domanda sorge spontanea: ″Come si recuperano le rappresentazioni delle risorse mediante i loro URI?″.
La figura seguente riporta la relazione tra una risorsa e i documenti che la rappresentano.



Il protocollo HTTP quando l'accesso ad una pagina Web va a buon fine restituisce il codice di stato `200`, però quando si pubblicano URI che identificano entità che non sono necessariamente documenti Web è necessario un approccio specifico. Il W3C ha proposto due soluzioni: utilizzare hash URI, oppure il codice di stato `303`.

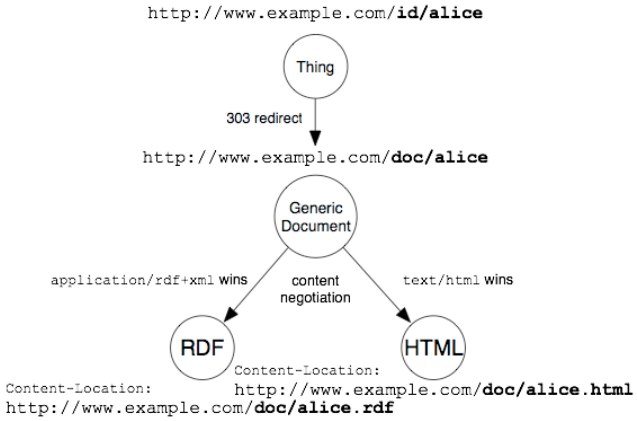
Hash URI

Questa soluzione consiste nell'utilizzare URI che contengono una parte speciale che è separata dal resto dell'URI da un simbolo hash (″#″). Quando un client vuole recuperate un URI hash, la parte dell'URI che si trova dopo il simbolo hash viene tagliata prima di effettuare la richiesta HTTP al server. Questo significa che un hash URI non può essere recuperata direttamente, di conseguenza si intuisce che può non identificare necessariamente un documento Web. Infatti le URI hash possono essere usate per identificare documenti Web e risorse senza creare ambiguità. Se l'URI richiesta corrisponde ad un documento web, e ad un documento RDF il server deciderà quale restituire. Tipicamente questa decisione viene presa tenendo conto delle preferenze del client, e della configurazione del server. Successivamente utilizzerà l'header `Content-Location` per indicare l'esito della scelta.

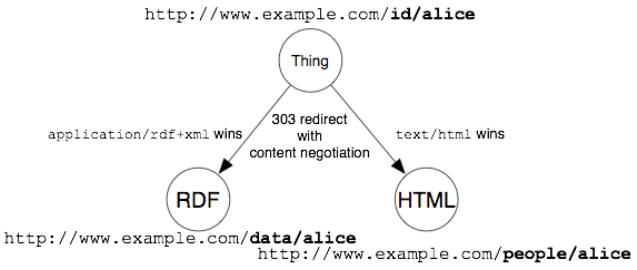
Codice di stato 303

Questa soluzione consiste nell'utilizzare il codice di stato speciale `303 See Other`, per indicare che la risorsa richiesta non è un normale documento Web, quindi il server restituirà la locazione di un documento

contenente informazioni sulla risorsa richiesta. In questo modo si evita ambiguità tra oggetti del mondo reale e risorse che li rappresentano. In altre parole, se il server risponde con un normale codice di stato nel range `2XX`, tipo `200 OK`, allora il client sa che l'URI identifica un documento Web, altrimenti se risponde con un codice `303` allora significa che l'URI identifica una risorsa reale e il client verrà reindirizzato ad un documento che rappresenta la risorsa. Nel caso in cui il documento RDF e la sua rappresentazione HTML coincidono, allora il client viene reindirizzato su un documento generico dove potrà recuperare il documento RDF, o la sua rappresentazione HTML.



Avere dei documenti generici che contengono URI di rappresentazioni della risorsa più specifiche, permettono un maggior numero di utilizzi ed elaborazioni della risorsa. Ad esempio un documento generico riguardante un libro, può contenere URI di documenti contenti il libro nelle varie traduzioni. Se invece l'URI del documento RDF della risorsa differisce da quello della sua rappresentazione HTML, il server restituisce immediatamente il documento RDF o la sua rappresentazione HTML.



Capitolo 2

Analisi ed elaborazione degli OpenData di Tper

2.1 Requisiti dell'applicazione

L'applicazione ha lo scopo principale di fornire informazioni in tempo reale sui servizi di mobilità urbana, riguardanti linee di autobus e relative fermate in una determinata zona di interesse all'interno dell'area di Bologna.
L'utente interagisce con l'applicazione inserendo inizialmente la posizione in cui si trova utilizzando, come possibili formati, l'indirizzo postale o le coordinate geografiche, ed inserendo poi il massimo tempo di percorrenza a piedi che è disposto a impiegare per raggiungere la fermata. Il server seleziona tra i dati forniti da Tper tutte le fermate che soddisfano i requisiti della richiesta, e le linee che transitano su di esse. Le fermate scelte, con le rispettive linee, verranno restituite all'utente, che dovrà specificare il suo interesse verso tutte le possibilità o parte di esse. Infine l'applicazione restituirà gli orari reali dei prossimi due autobus in arrivo per ciascuna linea indicata, i quali sono ottenuti dal Web Service Hello Bus di Tper.
Il servizio dovrà presentare le informazioni con un livello di dettaglio tale da consentire la comprensione a chiunque, sia pendolari che usufruiscono abitualmente dei servizi di mobilità urbana di Bologna, sia utenti con poca confidenza con la rete di trasporti di Tper o con la città in generale.
Inoltre dovrà gestire un numero elevato di utenti che effettuano richieste eterogenee, tra le quali però potranno emergere categorie di utenti con una richiesta uguale o con caratteristiche simili, come ad esempio l'insieme de-

gli studenti di una scuola, ognuno dei quali richiede informazioni inerenti le fermate nei pressi della scuola. In quest'ottica si è poi ipotizzata la possibilità di inserire un tablet all'interno dell'atrio scolastico in modo da fornire le informazioni agli studenti in maniera automatica.

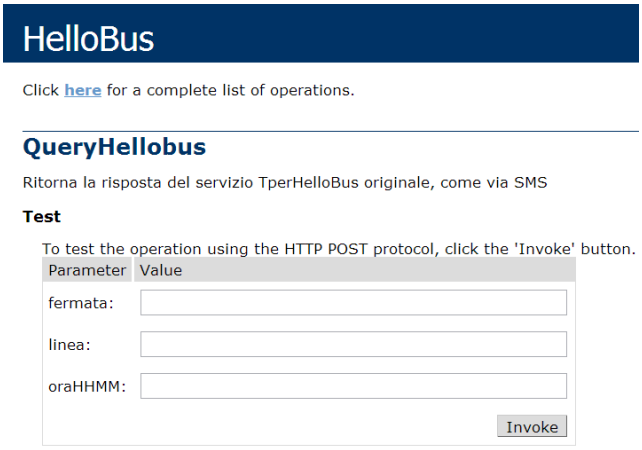


Figura 2.1: *Form dell'Hello Bus Web Server di Tper*

2.2 Database di Tper

L'azienda Tper fornisce due tipologie di OpenData: una riguarda le informazioni *statiche* (ad esempio dove sono collocate le fermate, quali tragitti effettuano le linee), le quali sono ottenibili scaricando i file in formato xml o csv dal loro sito; e l'altra riguarda le informazioni in tempo reale (ritardi degli autobus), le quali vengono restituite mediante query al Web Service Hello Bus.
Poichè il sito di Tper non presenta nessuna documentazione in merito agli OpenData pubblicati, è stato necessario uno studio per comprendere il significato dei dati mediante la tecnica del reverse engineering.
Dallo studio è emerso che all'interno del database vi sono quattro entità fondamentali: Fermata, Linea, Percorso e Arco.

Fermata

Pensilina caratterizzata da un codice che la identifica univocamente e informazioni relative alla sua collocazione geografica. Due pensiline

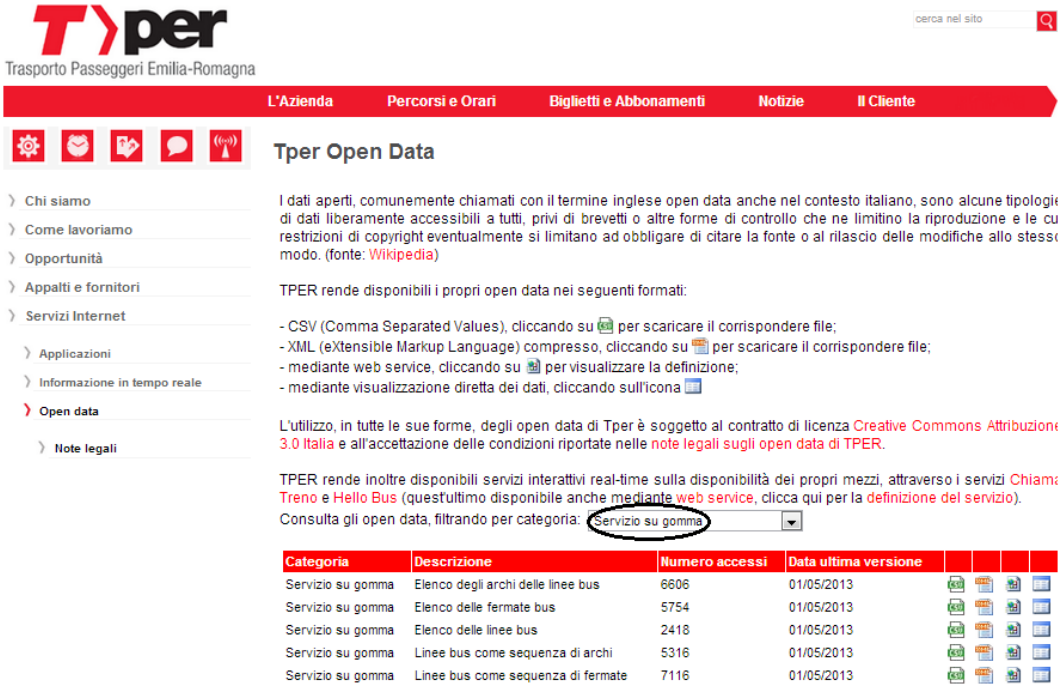


Figura 2.2: *Pagina web degli OpenData di Tper. Vengono posti in evidenza quelli utilizzati nell'applicazione.*

poste sui lati opposti di una strada sono rappresentate da due Fermate diverse con codici di identificazione diversi.

Linea

Tratta principale ed eventuali sottotratte secondarie. Ad esempio con linea *14* si intende la linea *14* che effettua la tratta principale e le linee *14A*, *14B*, *14C* che seguono parte della tratta principale e poi si scostano da questa fino a terminare in capolinea diversi. Oppure al contrario partono da capolinea diversi e poi si immettono nella tratta principale. Essa nei giorni feriali effettua la tratta da *Piazza Giovanni XXIII* e arrivato alla fermata *Tangenziale di San Vitale* si suddivide nella linea *14A* che prosegue per la fermata *Antolini* fino al capolinea *Deposito Due Madonne*, e nelle linee *14B* e *14C* che proseguono insieme nelle fermate *Larga* e *Larga Stazione*.
Successivamente il *14B* passa per la fermata *Carrozzaio* con destinazione *Rot. Negroni*, mentre il *14C* transita per la fermata *Innocenti*

Figura 2.3: *Linea 14 Barca - Ospedale S. Orsola - Due Madonne/Pilastro*

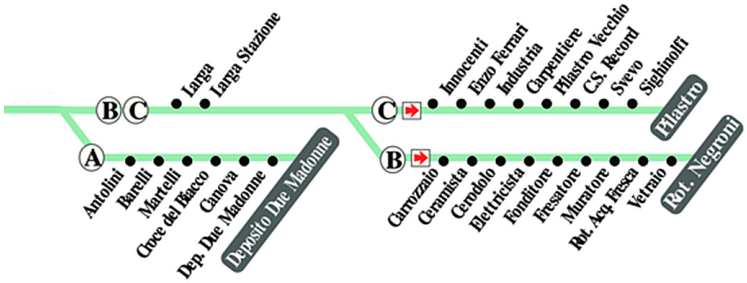


Figura 2.4: *Linee 14A, 14B, 14C Barca - Ospedale S. Orsola - Due Madonne/Pilastro*

con destinazione finale *Pilastro*.

Percorso

Tratta o sottotratta, intesa come insieme di fermate, effettuata da una determinata linea in un giorno feriale o festivo. Un percorso viene identificato univocamente dalla linea, un numero (unico all'interno della linea), ed un verso. Ogni linea ha almeno quattro percorsi:

- la sequenza delle fermate sulla tratta principale in una direzione effettuata nei giorni feriali;
- la sequenza delle fermate sulla tratta principale nella stessa direzione effettuata nei giorni festivi;
- la sequenza delle fermate sulla tratta principale nella direzione opposta effettuata nei giorni feriali;
- la sequenza delle fermate sulla tratta principale nella stessa direzione effettuata nei giorni festivi.

Ad esempio la linea *14* ha in totale 12 percorsi.

Arco

Sequenza ordinata di punti in una certa direzione situati in corrispondenza di una fermata oppure fra due fermate consecutive. Ogni percorso è espresso come sequenza di archi che sono a loro volta una sequenza di fermate e punti tra di esse. Inoltre lo stesso arco può appartenere a più percorsi, nel caso in cui abbiano parte di una tratta in comune. Ad esempio, come si vede in fig. 2.5, presa una linea L e una specifica fermata F_4 di un percorso essa presenta quattro archi incidenti: un arco entrante che ha come ultima fermata F_4, e un arco uscente, che ha come prima fermata F_4 per ambedue le direzioni.

Le informazioni sono contenute all'interno delle seguenti tabelle:

- `Elenco delle fermate bus`, contiene informazioni sulle fermate e sulla loro collocazione geografica nei formati di indirizzo postale e coordinate geografiche (latitudine e longitudine).

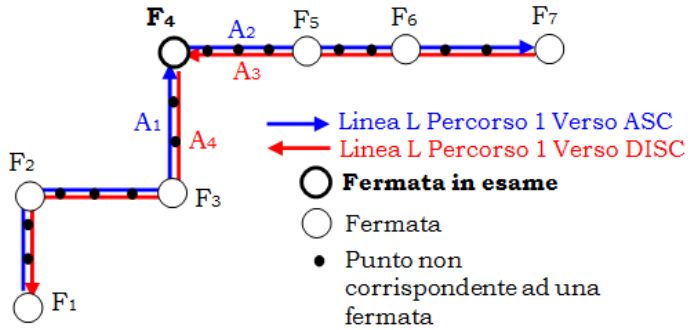
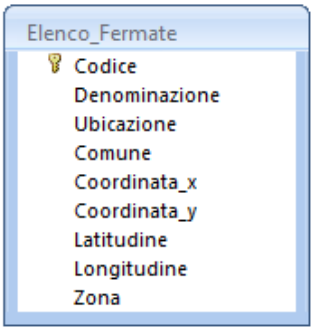
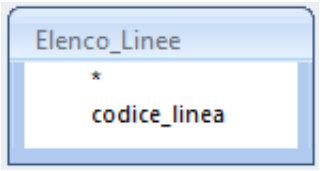


Figura 2.5: *Esempio di fermata con 4 archi incidenti su di essa.*

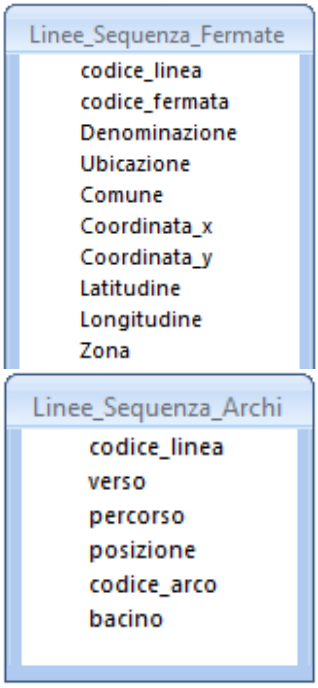


- `Elenco delle linee bus`, contiene una lista delle linee bus.

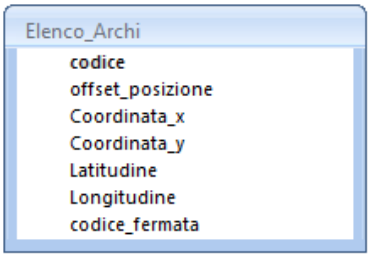


- `Linee bus come sequenza di fermate`, contiene informazioni su tutte le fermate effettuate da ciascuna linea, senza però indicarne l'ordine.

- `Linee bus come sequenza di archi`, contiene tutti i percorsi effettuati da ciascuna linea, descritti come sequenza ordinata di archi in una direzione. L'ordine degli archi viene indicato mediante il campo `posizione`.

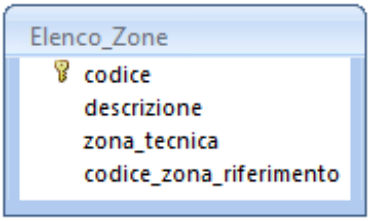


- `Elenco degli archi delle linee bus`, contiene una lista di archi descritti come sequenza ordinata di punti collocati o meno in corrispondenza di una fermata. L'ordine è indicato dall' `offset_posizione`, che vale 0 per il punto iniziale dell'arco e cresce per i punti successivi, mentre il `codice_fermata` vale 0 se il punto non è collocato su una fermata, altrimenti assume il codice della fermata.



- `Elenco delle zone`, contiene i dati relativi alle zone.

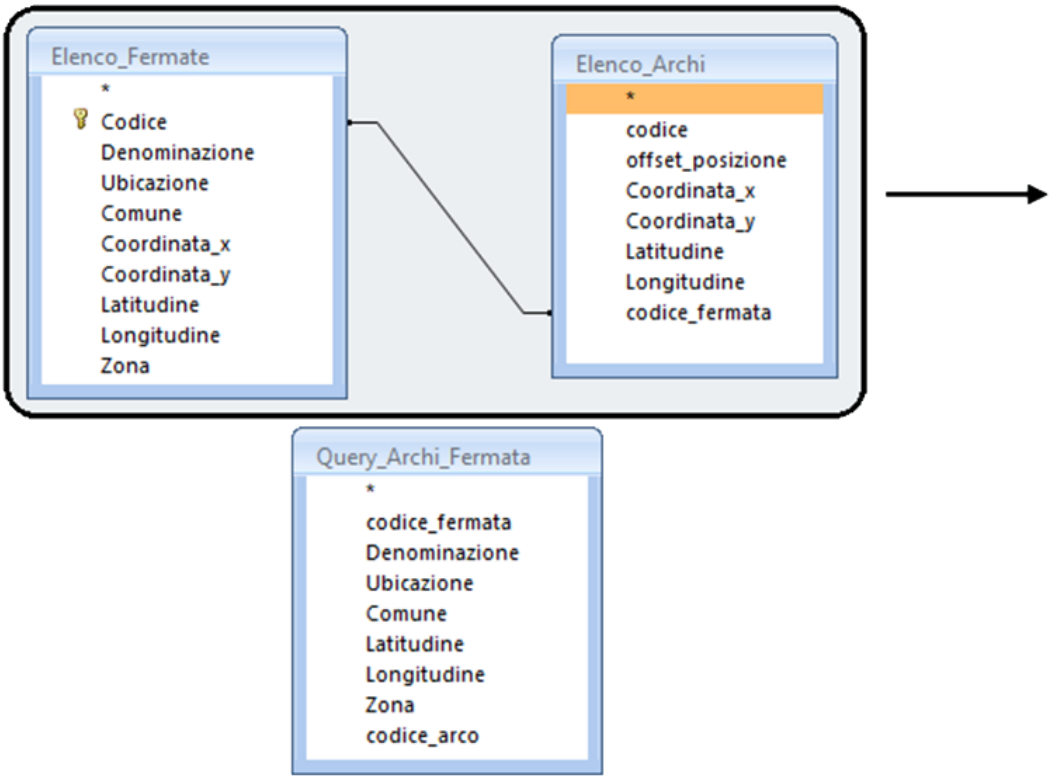
L'applicazione si propone di fornire all'utente informazioni sulle fermate posizionate all'interno dell'area di interesse e relative linee che transitano su di



esse. In particolare, per quanto concerne le linee, è opportuno indicarne il numero e la destinazione, ovvero il capolinea, in modo che l'utente comprenda su quale lato della strada transiterà l'autobus diretto alla sua destinazione. Come si può notare dalla descrizione precedente, tale informazione non è presente esplicitamente all'interno del database, ma può essere ricavata eseguendo alcune queries.
Data una fermata F ed una linea L che transita su di essa, il capolinea si ricava come ultima fermata dell'ultimo arco del percorso di L di cui fa parte F. Segue una descrizione dettagliata della query, implementata in mysql, che restituisce la destinazione delle linee di ogni fermata, e delle sottoqueries che la compongono, partendo dalla sottoquery più innestata a quella generale.

1) Query_Archi_Fermata

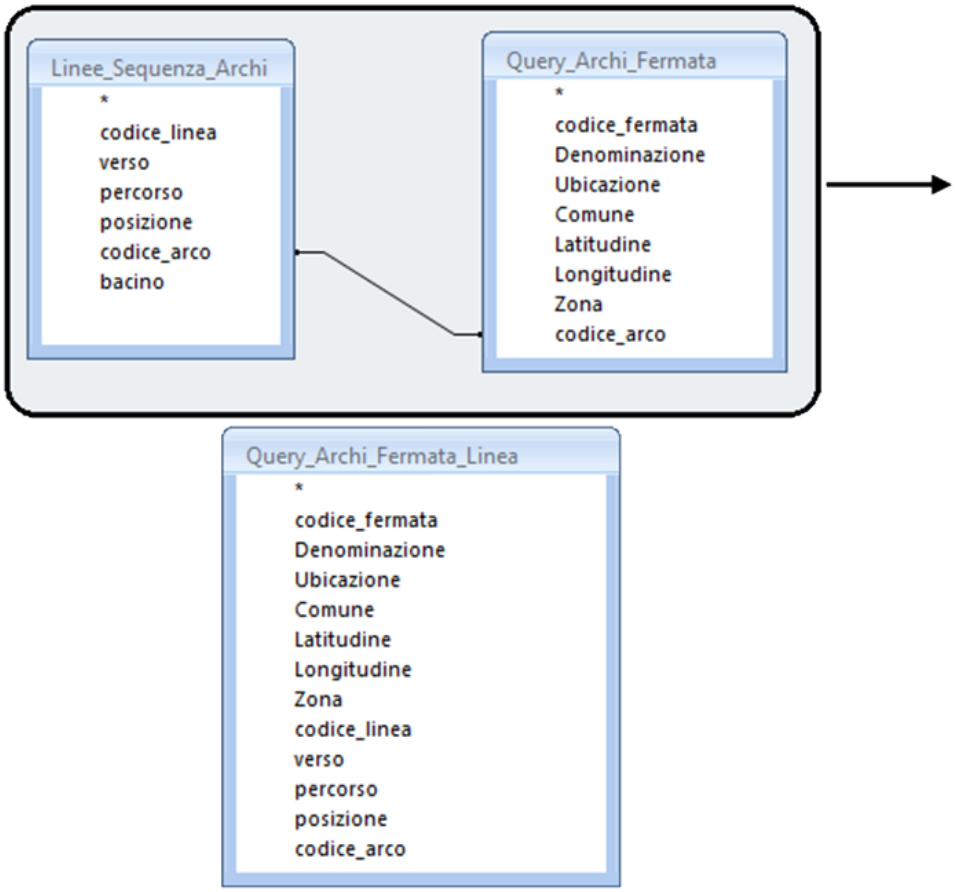
Seleziona per ogni fermata tutti gli archi incidenti su di essa.




```
CREATE VIEW Query_Archi_Fermata AS
SELECT Elenco_Archi.codice_fermata,
Elenco_Fermate.Denominazione,
Elenco_Fermate.Ubicazione, Elenco_Fermate.Comune,
Elenco_Fermate.Latitudine, Elenco_Fermate.Longitudine,
Elenco_Fermate.Zona, Elenco_Archi.codice AS codice_arco
FROM Elenco_Fermate INNER JOIN Elenco_Archi ON
Elenco_Fermate.codice = Elenco_Archi.codice_fermata;
```

2) Query_Archi_Fermata_Linea

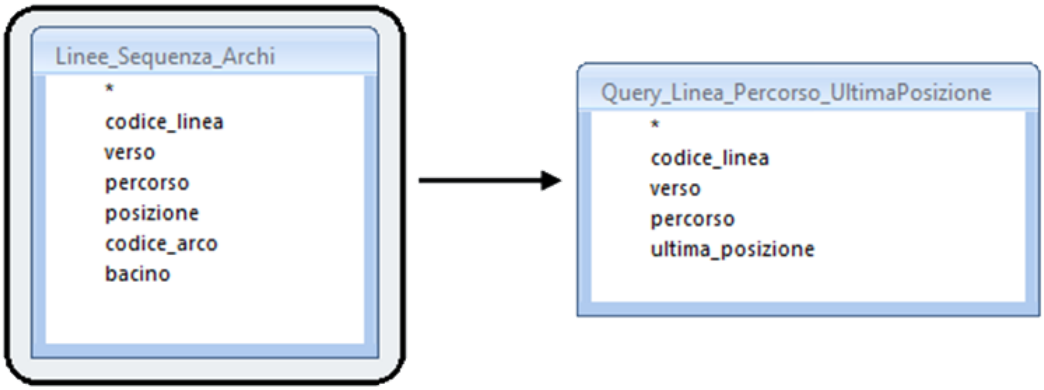
Seleziona per ogni linea tutti gli archi incidenti su ciascuna delle sue fermate.




```
CREATE VIEW Query_Archi_Fermata_Linea AS
SELECT Query_Archi_Fermata.codice_fermata,
Query_Archi_Fermata.Denominazione,
Query_Archi_Fermata.Ubicazione, Query_Archi_Fermata.Comune,
Query_Archi_Fermata.Latitudine,
Query_Archi_Fermata.Longitudine, Query_Archi_Fermata.Zona,
Linee_Sequenza_Archi.codice_linea, Linee_Sequenza_Archi.verso,
Linee_Sequenza_Archi.percorso,
Linee_Sequenza_Archi.posizione,
Linee_Sequenza_Archi.codice_arco
FROM Linee_Sequenza_Archi INNER JOIN Query_Archi_Fermata ON
Linee_Sequenza_Archi.codice_arco =
Query_Archi_Fermata.codice_arco;
```

3) Query_Linea_Percorso_UltimaPosizione

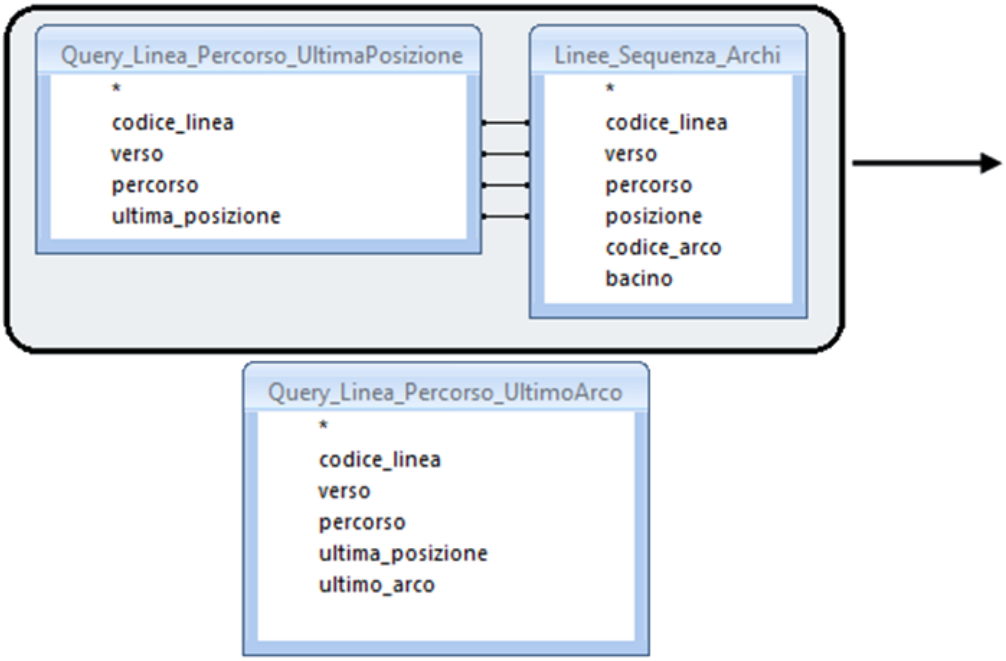
Seleziona l'ultima posizione di ogni percorso effettuato da ciascuna linea.




```
CREATE VIEW Query_Linea_Percorso_UltimaPosizione AS
SELECT codice_linea, verso, percorso,
MAX(posizione) AS ultima_posizione
FROM Linee_Sequenza_Archi
GROUP BY codice_linea, verso, percorso;
```

4) Query_Linea_Percorso_UltimoArco

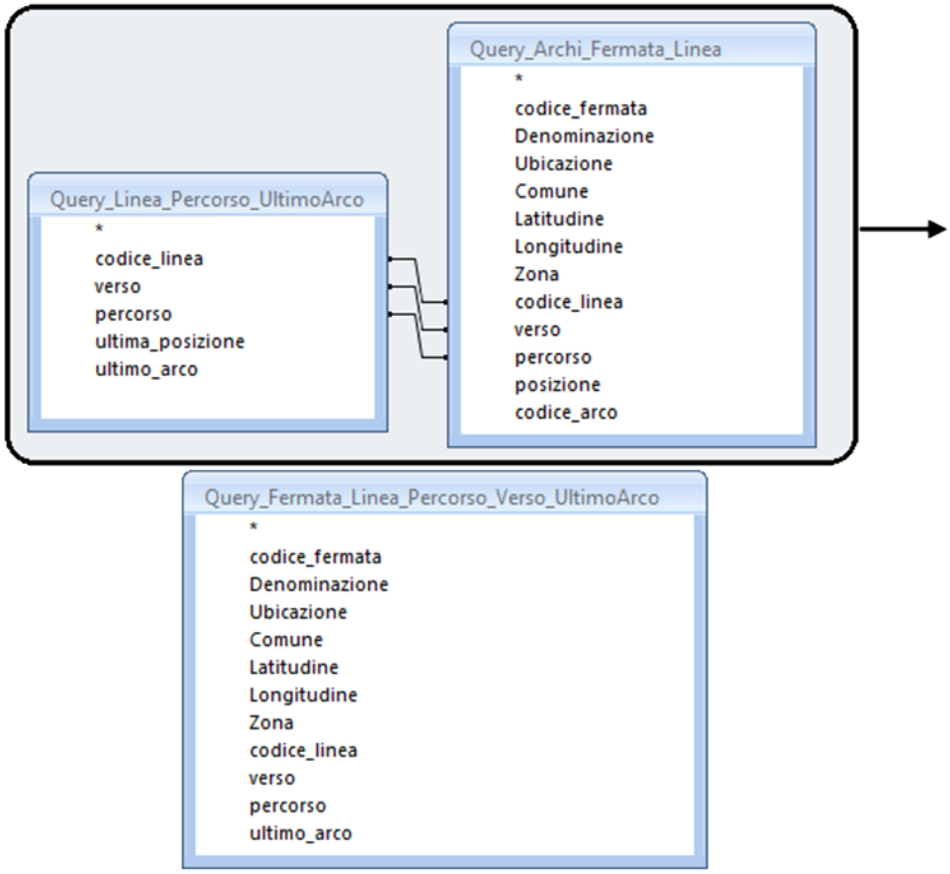
Seleziona l'ultimo arco di ogni percorso effettuato da ciascuna linea.




```
CREATE VIEW Query_Linea_Percorso_UltimoArco AS
SELECT Linee_Sequenza_Archi.codice_linea,
Linee_Sequenza_Archi.verso, Linee_Sequenza_Archi.percorso,
Query_Linea_Percorso_UltimaPosizione.ultima_posizione,
Linee_Sequenza_Archi.codice_arco AS ultimo_arco
FROM Linee_Sequenza_Archi
INNER JOIN Query_Linea_Percorso_UltimaPosizione ON
(Linee_Sequenza_Archi.posizione =
Query_Linea_Percorso_UltimaPosizione.ultima_posizione) AND
(Linee_Sequenza_Archi.verso =
Query_Linea_Percorso_UltimaPosizione.verso) AND
(Linee_Sequenza_Archi.percorso =
Query_Linea_Percorso_UltimaPosizione.percorso) AND
(Linee_Sequenza_Archi.codice_linea =
Query_Linea_Percorso_UltimaPosizione.codice_linea);
```

5) Query_Fermata_Linea_Percorso_Verso_UltimoArco

Seleziona, per ogni fermata, ogni possibile arco finale dei percorsi seguiti da ciascuna linea che transita su di essa.




```
CREATE VIEW Query_Fermata_Linea_Percorso_Verso_UltimoArco AS
SELECT DISTINCT Query_Archi_Fermata_Linea.codice_fermata,
Query_Archi_Fermata_Linea.Denominazione,
Query_Archi_Fermata_Linea.Ubicazione,
Query_Archi_Fermata_Linea.Comune,
Query_Archi_Fermata_Linea.Latitudine,
Query_Archi_Fermata_Linea.Longitudine,
Query_Archi_Fermata_Linea.Zona,
Query_Archi_Fermata_Linea.codice_linea,
Query_Archi_Fermata_Linea.verso,
Query_Archi_Fermata_Linea.percorso,
Query_Linea_Percorso_UltimoArco.ultimo_arco
```

```
FROM Query_Linea_Percorso_UltimoArco
INNER JOIN Query_Archi_Fermata_Linea ON
(Query_Linea_Percorso_UltimoArco.percorso =
Query_Archi_Fermata_Linea.percorso) AND
(Query_Linea_Percorso_UltimoArco.verso =
Query_Archi_Fermata_Linea.verso) AND
(Query_Linea_Percorso_UltimoArco.codice_linea =
Query_Archi_Fermata_Linea.codice_linea);
```

6) Query_Archi_OffsetMax

Seleziona l'`offset_posizione` massimo per ogni arco.

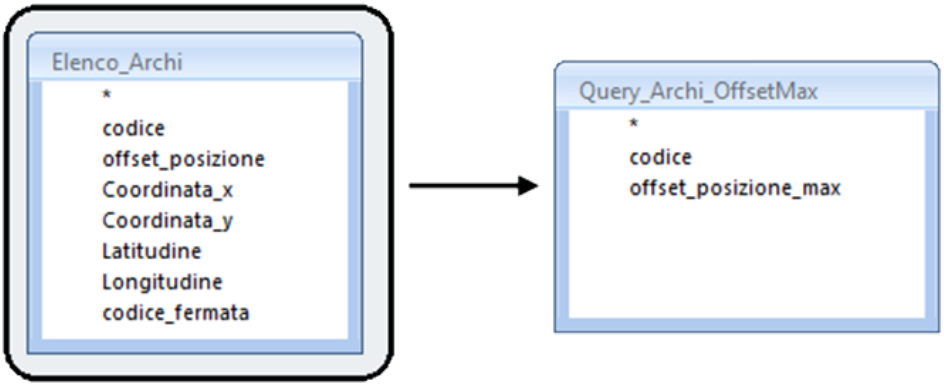
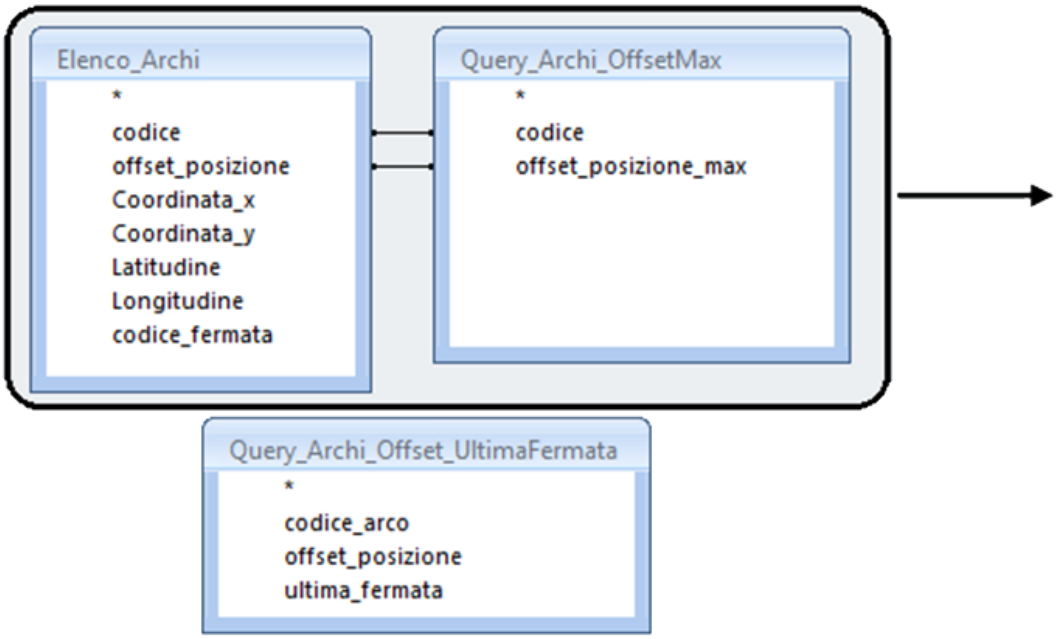


Figura 2.6: *Query_6*

```
CREATE VIEW Query_Archi_OffsetMax AS
SELECT Elenco_Archi.codice,
Max(Elenco_Archi.offset_posizione) AS offset_posizione_max
FROM Elenco_Archi
GROUP BY Elenco_Archi.codice;
```

7) Query_Archi_Offset_UltimaFermata

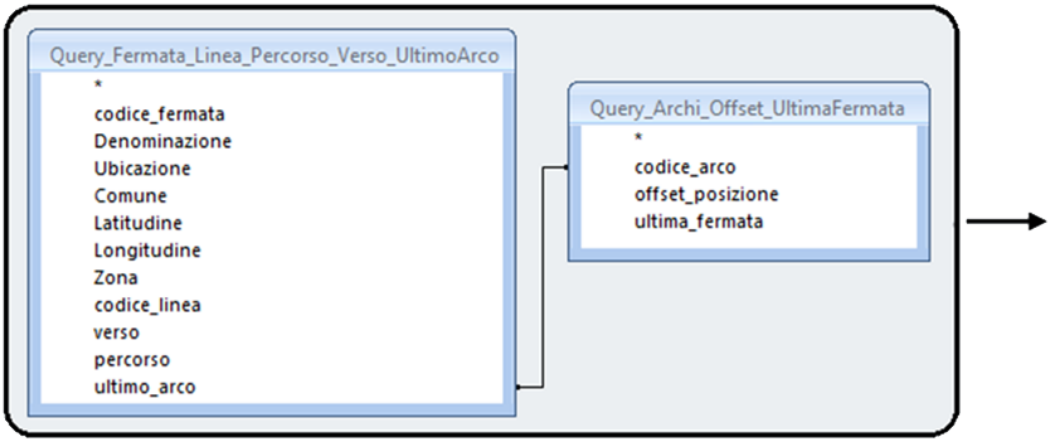
Seleziona l'ultima fermata di ogni arco, ovvero quella con l'`offset_posizione` massimo.

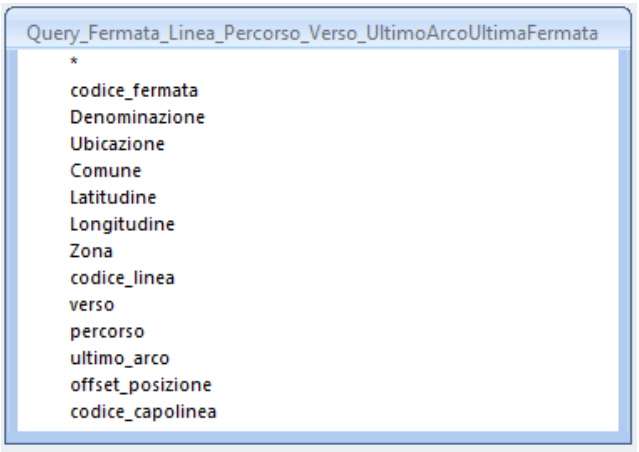



```
CREATE VIEW Query_Archi_Offset_UltimaFermata AS
SELECT Elenco_Archi.codice AS codice_arco,
Elenco_Archi.offset_posizione,
Elenco_Archi.codice_fermata AS ultima_fermata
FROM Elenco_Archi INNER JOIN Query_Archi_OffsetMax
ON (Elenco_Archi.codice=Query_Archi_OffsetMax.codice)
AND (Elenco_Archi.offset_posizione =
Query_Archi_OffsetMax.offset_posizione_max);
```

8) Query_Fermata_Linea_Percorso_Verso_ UltimoArcoUltimaFermata

Presa una determinata fermata, per ogni percorso delle linee che la prevedono, viene selezionato il codice della fermata capolinea. Si noti che più percorsi di una linea possono avere lo stesso capolinea.



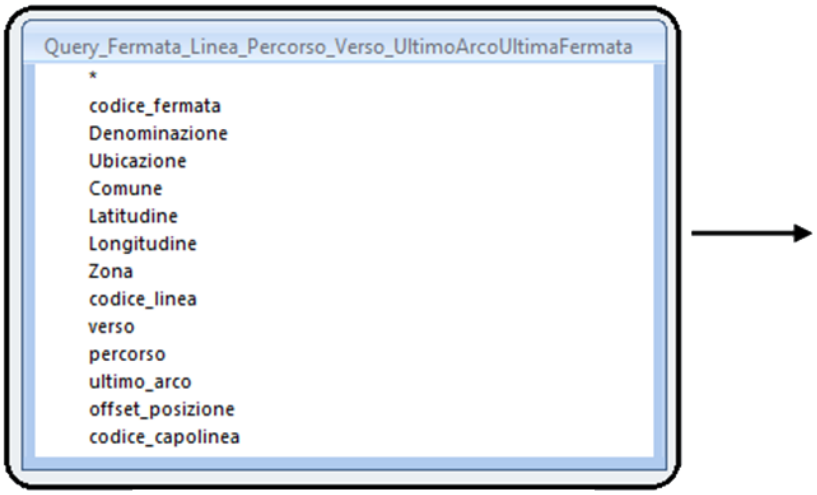


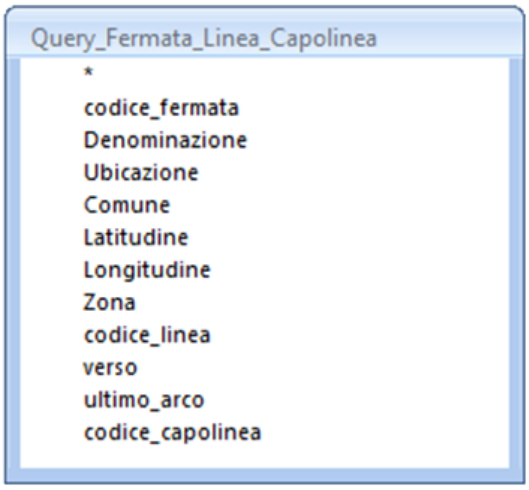

```
CREATE VIEW
Query_Fermata_Linea_Percorso_Verso_UltimoArcoUltimaFermata
AS
```

```
SELECT
Query_Fermata_Linea_Percorso_Verso_UltimoArco.codice_fermata,
Query_Fermata_Linea_Percorso_Verso_UltimoArco.Denominazione,
Query_Fermata_Linea_Percorso_Verso_UltimoArco.Ubicazione,
Query_Fermata_Linea_Percorso_Verso_UltimoArco.Comune,
Query_Fermata_Linea_Percorso_Verso_UltimoArco.Latitudine,
Query_Fermata_Linea_Percorso_Verso_UltimoArco.Longitudine,
Query_Fermata_Linea_Percorso_Verso_UltimoArco.Zona,
Query_Fermata_Linea_Percorso_Verso_UltimoArco.codice_linea,
Query_Fermata_Linea_Percorso_Verso_UltimoArco.verso,
Query_Fermata_Linea_Percorso_Verso_UltimoArco.percorso,
Query_Fermata_Linea_Percorso_Verso_UltimoArco.ultimo_arco,
Query_Archi_Offset_UltimaFermata.offset_posizione,
Query_Archi_Offset_UltimaFermata.ultima_fermata
AS codice_capolinea
FROM Query_Fermata_Linea_Percorso_Verso_UltimoArco
INNER JOIN Query_Archi_Offset_UltimaFermata
ON Query_Fermata_Linea_Percorso_Verso_UltimoArco.ultimo_arco =
Query_Archi_Offset_UltimaFermata.codice_arco;
```

9) Query_Fermata_Linea_Capolinea

Partendo dalla vista precedente, ignorando il concetto di percorso, seleziona solo i capolinea distinti delle linee.

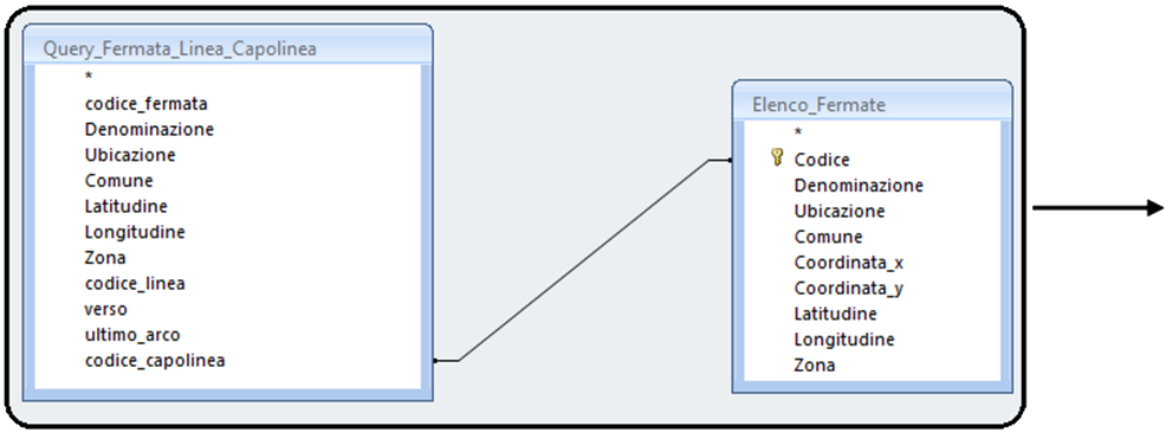


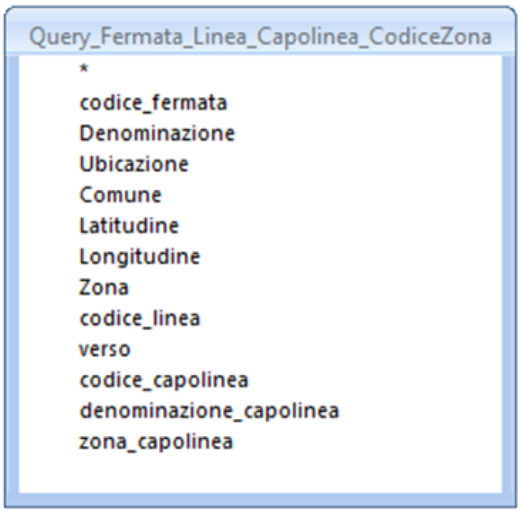



```
CREATE VIEW Query_Fermata_Linea_Capolinea AS
SELECT codice_fermata, Denominazione, Ubicazione, Comune,
Latitudine, Longitudine, Zona, codice_linea, verso,
ultimo_arco, codice_capolinea   FROM
Query_Fermata_Linea_Percorso_Verso_UltimoArcoUltimaFermata
GROUP BY codice_fermata, Denominazione, Ubicazione, Comune,
Latitudine, Longitudine, Zona, codice_linea, verso,
ultimo_arco, codice_capolinea;
```

10) Query_Fermata_Linea_Capolinea_CodiceZona

Associa ad ogni `codice_capolinea` la sua `Denominazione` e il codice della `zona`.



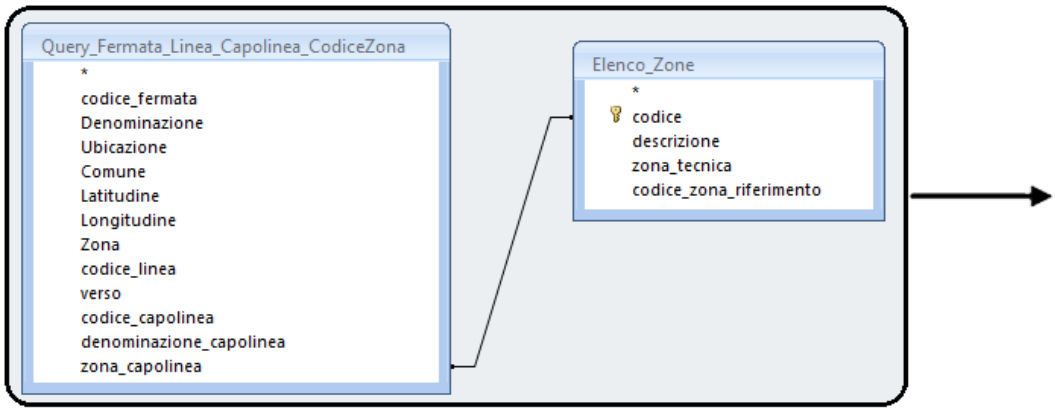


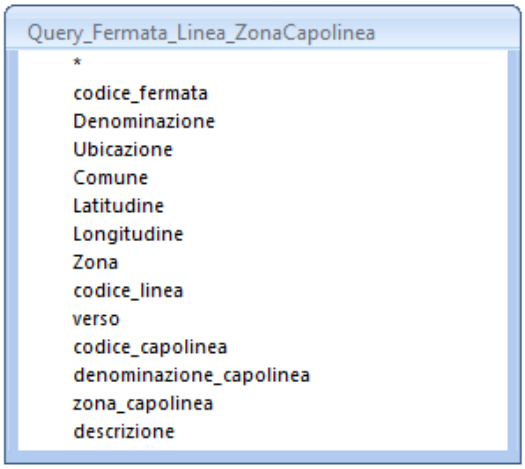

```
CREATE VIEW Query_Fermata_Linea_Capolinea AS
SELECT Query_Fermata_Linea_Capolinea.codice_fermata,
Query_Fermata_Linea_Capolinea.Denominazione,
Query_Fermata_Linea_Capolinea.Ubicazione,
Query_Fermata_Linea_Capolinea.Comune,
Query_Fermata_Linea_Capolinea.Latitudine,
Query_Fermata_Linea_Capolinea.Longitudine,
Query_Fermata_Linea_Capolinea.Zona,
Query_Fermata_Linea_Capolinea.codice_linea,
Query_Fermata_Linea_Capolinea.verso,
Query_Fermata_Linea_Capolinea.codice_capolinea,
Elenco_Fermate.Denominazione AS denominazione_capolinea,
Elenco_Fermate.Zona AS zona_capolinea
FROM Query_Fermata_Linea_Capolinea
INNER JOIN Elenco_Fermate ON
Query_Fermata_Linea_Capolinea.codice_capolinea =
Elenco_Fermate.codice;
```

11) Query_Fermata_Linea_ZonaCapolinea

Osservando la vista precedente si può notare che prendendo come fermata un capolinea, questo ha come capolinea in una certa direzione se stesso. Poichè lo scopo della query è indicare la destinazione delle linee, tutti i record che presentano fermata e capolinea uguali devono essere eliminati.

Pertanto la vista presente associa ad ogni capolinea il nome della zona ed elimina tutti i record nei quali la fermata e il capolinea coincidono.






```
CREATE VIEW Query_Fermata_Linea_ZonaCapolinea AS
SELECT Query_Fermata_Linea_Capolinea_CodiceZona.codice_fermata,
Query_Fermata_Linea_Capolinea_CodiceZona.Denominazione,
Query_Fermata_Linea_Capolinea_CodiceZona.Ubicazione,
Query_Fermata_Linea_Capolinea_CodiceZona.Comune,
Query_Fermata_Linea_Capolinea_CodiceZona.Latitudine,
Query_Fermata_Linea_Capolinea_CodiceZona.Longitudine,
Query_Fermata_Linea_Capolinea_CodiceZona.Zona,
Query_Fermata_Linea_Capolinea_CodiceZona.codice_linea,
Query_Fermata_Linea_Capolinea_CodiceZona.verso,
```

```
Query_Fermata_Linea_Capolinea_CodiceZona.codice_capolinea,
Query_Fermata_Linea_Capolinea_CodiceZona.
denominazione_capolinea,
Query_Fermata_Linea_Capolinea_CodiceZona.zona_capolinea,
Elenco_Zone.descrizione
FROM Query_Fermata_Linea_Capolinea_CodiceZona
INNER JOIN Elenco_Zone ON
Query_Fermata_Linea_Capolinea_CodiceZona.zona_capolinea =
Elenco_Zone.codice
WHERE
(Query_Fermata_Linea_Capolinea_CodiceZona.codice_fermata <>
Query_Fermata_Linea_Capolinea_CodiceZona.codice_capolinea);
```

12)Query_Fermate_Linee_Capolinea

Concretizza la vista precedente in una tabella fisica sul database. Tale tabella verrà utilizzata per la ricerca delle fermate interne all'area di interesse.

```
CREATE TABLE Fermate_Linee_Capolinea(
        id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
        codice_fermata VARCHAR(10),
        Denominazione VARCHAR(30),
        Ubicazione VARCHAR(30),
        Comune VARCHAR(30),
        Latitudine DOUBLE(8,6),
        Longitudine DOUBLE(8,6),
        Zona VARCHAR(10),
        codice_linea VARCHAR(10),
        verso VARCHAR(4),
        codice_capolinea VARCHAR(10),
        denominazione_capolinea VARCHAR(30),
        zona_capolinea VARCHAR(10),
        INDEX latitudineIDX(Latitudine)
)
SELECT codice_fermata, Denominazione, Ubicazione,
       Comune, Latitudine, Longitudine, Zona,
       codice_linea, codice_capolinea,
       denominazione_capolinea, zona_capolinea
FROM Query_Fermata_Linea_ZonaCapolinea;
```

Mediante la query

```
SELECT codice_fermata,Denominazione,Ubicazione,
codice_linea,verso,codice_capolinea,denominazione_capolinea
FROM Fermate_Linee_Capolinea
WHERE (codice_linea = 14) AND
(Denominazione = Certosa);
```

si ottiene la vista seguente, che riporta i capolinea corretti come si può verificare con le fig. 2.3 e 2.4.

codice fermata	Denominazione	Ubicazione	codice linea	verso	codice capolinea	denominazione capolinea
7120	CERTOSA	VIA COSTA A.226/D	14	asc	1399	DUE MADONNE
7120	CERTOSA	VIA COSTA A.226/D	14	asc	2476	ROTONDA NEGRONI
7120	CERTOSA	VIA COSTA A.226/D	14	asc	3056	PILASTRO
7119	CERTOSA	VIA COSTA A.P.L.99	14	disc	7146	PIAZZA GIOVANNI XXIII

Al fine di migliorare le prestazioni delle queries sono stati aggiunti alcuni indici sui campi interessati dai join, in particolare alla tabella *Linee_Sequenza_Archi* sono stati aggiunti l'indice multiplo *percorsoLineaIDX* sui campi *codice_linea*, *verso*, *percorso*, *posizione* e l'indice *codiceArcoIDX* sul campo *codice_arco*.

```
CREATE INDEX percorsoLineaIDX ON Linee_Sequenza_Archi
(codice_linea, verso, percorso, posizione);

CREATE INDEX codiceArcoIDX ON Linee_Sequenza_Archi
(codice_arco);
```

Mentre nella tabella *Elenco_Archi* è stato creato l'indice *codiceFermataIDX*.

```
CREATE INDEX codiceFermataIDX ON Elenco_Archi(codice_fermata);
```

Infine allo scopo di migliorare la ricerca delle fermate all'interno dell'area di interesse, che viene fatta sulla base delle coordinate geografiche sono stati inseriti gli indici *latitudineIDX* e *longitudineIDX*.

```
CREATE INDEX longitudineIDX
ON Fermate_Linee_Capolinea(Longitudine);

CREATE INDEX latitudineIDX
ON Fermate_Linee_Capolinea(Latitudine);
```

Capitolo 3

Progetto dell'Applicazione

3.1 Geocoding

L'utente interagisce con l'applicazione, come descritto nella sezione 2.1 inserendo inizialmente la sua posizione e il massimo tempo di percorrenza che vuole impiegare per raggiungere una fermata. La posizione può essere espressa in formato di indirizzo postale oppure di coordinate geografiche (latitudine e longitudine). Nel primo caso è necessaria una traduzione, in letteratura nota con il nome di Geocoding, da indirizzo postale a coordinate geografiche, ovvero latitudine e longitudine. A tale scopo viene utilizzato il servizio di Geocoding di *open.mapquestapi.com*[21], il quale utilizza gli OpenData di *OpenStreetMap*[7].
Si può interagire con esso attraverso il protocollo `HTTP` inviando messaggi nei quali si specifica l'indirizzo, e il formato dell'output. Di seguito viene riportato l'header del messaggio `HTTP` per richiedere la traduzione dell'indirizzo di Piazza Maggiore di Bologna in coordinate geografiche con output in formato xml. Si noti che per poter usufruire del servizio è necessario specificare una chiave, che viene fornita previa registrazione sul sito.

```
http://open.mapquestapi.com/geocoding/v1/address?
key=YOUR\_KEY\_HERE&callback=renderGeocode&
outFormat=xml&location=piazza maggiore bologna
```

Segue un frammento del file xml in output dove vengono riportati l'indirizzo postale della richiesta effettuata, e le coordinate latitudine e longitudine corrispondenti.

```
<locations>
  <location>
    <street>Piazza Maggiore</street>
    <adminArea5 type="City">Bologna</adminArea5>
    <adminArea3 type="State">Emilia-Romagna</adminArea3>
    <adminArea4 type="County">BO</adminArea4>
    <postalCode>40124</postalCode>
    <adminArea1 type="Country">IT</adminArea1>
    <geocodeQuality>ADDRESS</geocodeQuality>
    <geocodeQualityCode>L1CAX</geocodeQualityCode>
    <dragPoint>false</dragPoint>
    <sideOfStreet>N</sideOfStreet>
    <displayLatLng>...</displayLatLng>
    <linkId>0</linkId>
    <type>s</type>

    <latLng>
      <lat>44.493664</lat>
      <lng>11.343051</lng>
    </latLng>

  ...
  </location>
</locations>
```

In seguito le coordinate vengono estrapolate e trasformate in float mediante l'utilizzo di un parser. Nel caso in cui l'indirizzo inserito presenta più corrispondenze il servizio di Geocoding restituisce all'interno del file la traduzione di ognuna di esse. Un esempio è la richiesta di traduzione dell'indirizzo di viale Carlo Pepoli 3 di Bologna con il seguente messaggio.

```
http://open.mapquestapi.com/geocoding/v1/address?
key=YOUR\_KEY\_HERE&callback=renderGeocode&
outFormat=xml&location=Viale Carlo Pepoli 3 Bologna
```

Segue un frammento del file xml in output dove vengono riportate le coordinate latitudine e longitudine degli indirizzi postali che corrispondono alla richiesta effettuata:

- viale Carlo Pepoli 3 (BO);
- viale Carlo Pepoli 3/5 (BO);
- viale Carlo Pepoli 3/2 (BO).

In questi casi il parser estrapolerà sempre le coordinate relative alla prima corrispondenza della lista.

```
<locations>
    <location>
        <street>Viale Carlo Pepoli 3</street>
        <adminArea5 type="City">Bologna</adminArea5>
        <adminArea3 type="State">Emilia-Romagna</adminArea3>
        <adminArea4 type="County">BO</adminArea4>
        <postalCode>40123</postalCode>
        <adminArea1 type="Country">IT</adminArea1>
        ...
        <latLng>
            <lat>44.491765</lat>
            <lng>11.32959</lng>
        </latLng>
    </location>
    <location>
        <street>Viale Carlo Pepoli 3/5</street>
        <adminArea5 type="City">Bologna</adminArea5>
        <adminArea3 type="State">Emilia-Romagna</adminArea3>
        <adminArea4 type="County">BO</adminArea4>
        <postalCode>40123</postalCode>
        <adminArea1 type="Country">IT</adminArea1>
        ...
        <latLng>
            <lat>44.492971</lat>
            <lng>11.330936</lng>
        </latLng>
    </location>
    <location>
        <street>Viale Carlo Pepoli 3/2</street>
        <adminArea5 type="City">Bologna</adminArea5>
        <adminArea3 type="State">Emilia-Romagna</adminArea3>
        <adminArea4 type="County">BO</adminArea4>
        <postalCode>40123</postalCode>
        <adminArea1 type="Country">IT</adminArea1>
        ...
        <latLng>
            <lat>44.492508</lat>
            <lng>11.329483</lng>
        </latLng>
    </location>
</locations>
```

3.2 Selezione fermate

Una volta note le coordinate geografiche del punto, il server deve stabilire quali fermate si trovano ad una distanza percorribile in un tempo massimo pari a quello specificato dall'utente. La distanza viene calcolata assumendo come velocità media di un uomo che sta camminando 1.12 m/s.
All'interno del sito Tper è disponibile un elenco di tutte le fermate con le relative coordinate geografiche, quindi si può procedere in due modi:

- si prende un punto alla volta, si calcola la distanza da questo al punto in cui è localizzato l'utente, e infine si verifica se essa è inferiore alla soglia specificata;

- si calcolano gli intervalli di coordinate dentro ai quali si trovano i punti con distanza dall'utente inferiore alla soglia, e infine si verifica per ogni fermata se le sue coordinate appartengono o meno agli intervalli.

Poichè il calcolo per ottenere una buona approssimazione della distanza tra due punti sulla superficie della Terra mediante le loro coordinate geografiche è complesso, risulta molto più efficiente il secondo approccio, nel quale il calcolo viene fatto una volta sola e vale per tutti i punti, diversamente dal primo, nel quale il calcolo deve essere fatto tante volte quanti sono i punti.
Per semplificare il calcolo degli intervalli delle coordinate si può utilizzare un'area di forma quadrata con lato di dimensione uguale al doppio del distanza di interesse e poi si verifica dai dati di Tper quali fermate sono situate all'interno dell'area.
Date le coordinate geografiche del punto O, come si può notare nella figura seguente, per conoscere l'intervallo delle coordinate dell'area quadrata ABCD di interesse è necessario ricavare le coordinate dei punti A e D.

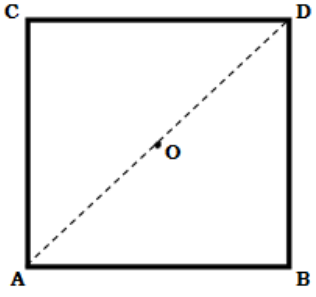
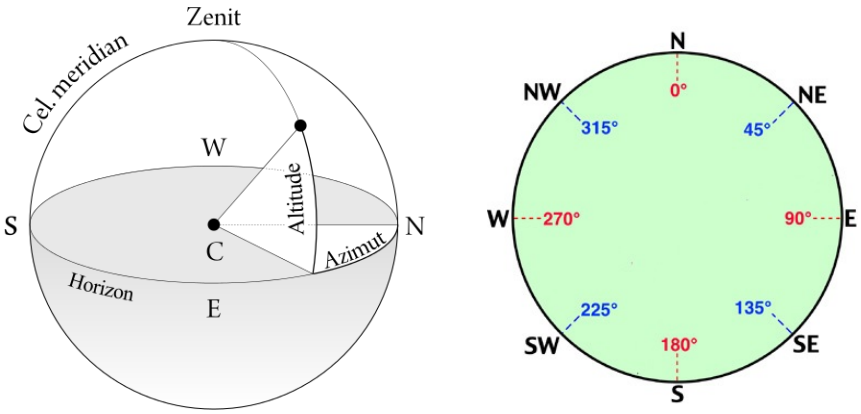


Figura 3.1

Approssimando la Terra ad una sfera di raggio R = 6372,795477598 Km le coordinate geografiche (latitudine ϕ, longitudine λ) dei punti A e D si possono ottenere attraverso le formule seguenti:

$$\phi_B = \arcsin\left(\sin(\phi_A)\cdot\cos\left(\frac{d}{R}\right) + \cos(\phi_A)\cdot\sin\left(\frac{d}{R}\right)\cdot\cos(\theta)\right)$$
$$\lambda_B = \lambda_A + \arctan 2\left(\sin(\theta)\cdot\sin\left(\frac{d}{R}\right)\cdot\cos(\phi_A), \cos\left(\frac{d}{R}\right) - \sin(\phi_A)\cdot\sin(\phi_B)\right)$$

dove A è il punto di partenza, B il punto di destinazione, d è la distanza tra A e B e θ è l'angolo di Azimut.
L'angolo di Azimut è la distanza angolare di un punto dalla direzione del Nord, alla direzione in cui cade la proiezione di un punto sull'orizzonte, calcolata muovendosi in senso orario.
Per convenzione, il Nord ha Azimut pari a 0 gradi, l'Est ha Azimut pari a 90 gradi, il Sud a 180 gradi e l'Ovest a 270 gradi.



(a) *Angolo di Azimuth* (b) *Valori angolo di Azimuth*

Figura 3.2

La funzione $\arctan 2(y, x)$ è un'estensione della funzione arcotangente $\arctan(y/x)$ che tiene conto anche del segno degli argomenti, ed è anche definita nel caso $x = 0$.

$$\arctan 2(y,x) = \begin{cases} \arccos\left(\frac{x}{\sqrt{x^2+y^2}}\right) & y \geq 0 \\ \arccos\left(\frac{x}{\sqrt{x^2+y^2}}\right) & y < 0 \end{cases}$$

Per ottenere gli intervalli di coordinate dei punti appartenenti all'area quadrata di interesse è sufficiente conoscere le coordinate dei punti A e D :

$$\phi \in [\phi_A, \phi_D] \qquad \lambda \in [\lambda_A, \lambda_D]$$

Il valore delle coordinate del punto dell'area quadrata di Fig.3.1 si può trovare assumendo come punto di partenza O, distanza $d = raggio\_di\_interesse \cdot \sqrt{2}$, e angolo di Azimut: $\theta = 225°$ per A e $\theta = 45°$ per D.
Segue la query che seleziona le fermate appartenenti all'area di interesse. Si noti che viene creata la vista e gli estremi degli intervalli sono passati all'interno delle variabili php *$latA*, *$longA*, *$latD*, *$longD*.

```
"CREATE VIEW FermateInteresse AS SELECT *
FROM Fermate_Linee_Capolinea
WHERE (Latitudine >= '$latA') AND (Latitudine <= '$latD')
AND (Longitudine >= '$longA') AND (Longitudine <= '$longD');"
```

La distanza utilizzata per calcolare gli intervalli delle coordinate, come descritto precedentemente, è in linea d'aria; perciò, dopo aver fatto la selezione delle fermate sulla base di questa distanza, è opportuno effettuare una seconda selezione sulla base del percorso reale a piedi necessario per raggiungere le varie fermate dal punto nel quale è localizzato l'utente.
Per farlo occorrerà calcolare per ogni fermata precedentemente selezionata la distanza reale a piedi dalla posizione in cui si trova l'utente, e verificare che non sia superiore alla distanza di interesse. Questi valori si possono ottenere utilizzando il servizio di *Directions* di *open.mapquestapi.com*[22], in particolare la funzione `RouteMatrix` che permette di calcolare tempi e distanze dei percorsi tra più posizioni tenendo conto del mezzo di trasporto.
Si può interagire con esso attraverso il protocollo `HTTP` inviando messaggi nel quale si specificano le coordinate dei punti di partenza e di destinazione, il formato di input, il formato dell'output, l'unità di misura della distanza (di default è miglia) e il mezzo di trasporto.
Di seguito viene riportato l'header del messaggio `HTTP` per richiedere la distanza a piedi da *via Carlo Pepoli 3 (BO)* (44.491837, 11.329501) alla fermata di *via Saragozza 132 (BO)* (44.490166, 11.324943).

```
http://open.mapquestapi.com/directions/v1/routematrix?
key=YOUR\_KEY\_HERE&outFormat=xml&from=44.491837,11.329501&
to=44.490166,11.324943&unit=k&
routeType=pedestrian&doReverseGeocode=false
```

Segue un frammento del file xml in output dove vengono riportate le coordinate dei due punti, il tempo in secondi e la distanza in km del primo punto da se stesso, cioè 0, e del secondo punto dal primo.

```
<response>
    <info>...</info>
    <allToAll>false</allToAll>
    <manyToOne>false</manyToOne>
    <locations>
        <location>
           ...
           <latLng>
              <lat>44.491837</lat> <lng>11.329501</lng>
           </latLng>
        </location>
        <location>
           <latLng>
              <lat>44.490166</lat> <lng>11.324943</lng>
        </latLng>
        </location>
    </locations>

    <time>0,478</time>
    <distance>0.0,0.535</distance>
</response>
```

Il Web Service Directions in generale viene utilizzato per richiedere la distanza tra più punti. Quindi inserendo la lista delle coordinate dei punti all'interno un file xml o json, si ottiene la matrice NxN delle distanze di ogni punto da tutti gli altri. Poichè gli elementi sulla prima colonna di questa matrice sono le distanze di tutti i punti dal primo della lista, inserendo per primo il punto nel quale è localizzato l'utente e poi di seguito le fermate che hanno superato la prima selezione, si ottengono le distanze reali di tutte queste fermate dall'utente.
Ad esempio se l'utente è posizionato di fronte al Liceo Righi di Bologna che ha coordinate 44.491837, 11.329501 e le fermate selezionate hanno coordinate {(44.492155, 11.329336), (44.491510, 11.329185), (44.490492, 11.328339), (44.490332, 11.328257), (44.490604, 11.330783), (44.493288, 11.329004)} il file xml in input alla query, contenente la lista delle cooordinate, sarà come quello che segue.

```
<route>
    <locations>
       <location>
           <latLng>
               <lat> 44.491837 </lat> <lng> 11.329501 </lng>
           </latLng>
           </location>
           <location>
           <latLng>
               <lat> 44.492155 </lat> <lng> 11.329336 </lng>
           </latLng>
           </location>
           <location>
           <latLng>
               <lat> 44.491510 </lat> <lng> 11.329185 </lng>
           </latLng>
           </location>
           <location>
           <latLng>
               <lat> 44.490492 </lat> <lng> 11.328339 </lng>
           </latLng>
           </location>
           <location>
           <latLng>
               <lat> 44.490332 </lat> <lng> 11.328257 </lng>
           </latLng>
           </location>
           <location>
           <latLng>
               <lat> 44.490604 </lat> <lng> 11.330783 </lng>
           </latLng>
           </location>
           <location>
           <latLng>
               <lat> 44.493288 </lat> <lng> 11.329004 </lng>
           </latLng>
           </location>
        </locations>
    <options> <allToAll> true </allToAll> <unit> k </unit>
      <routeType> pedestrian </routeType>
      <doReverseGeocode> false <doReverseGeocode>
      <ambiguities> ignore </ambiguities> </options>
<route>
```

Segue un frammento del file xml in output dove si possono notare la matrice degli intervalli di tempo (in secondi) necessari per raggiungere ogni punto partendo da ogni altro, e la matrice delle distanze (in km).

```
<response>
 ...
<time>
<time0>0,34,158,230,340,203,299</time0>
<time1>34,0,192,260,370,237,114</time1>
<time2>83,81,0,178,289,192,303</time2>
<time3>262,260,178,0,110,206,482</time3>
<time4>278,265,194,178,0,207,497</time4>
<time5>275,237,192,206,317,0,495</time5>
<time6>149,114,196,375,380,352,0</time6>
</time>

<distance>
<distance0>0.0,0.038,0.177,0.257,0.381,0.226,0.334</distance0>
<distance1>0.038,0.0,0.215,0.291,0.415,0.265,0.128</distance1>
<distance2>0.093,0.091,0.0,0.199,0.323,0.215,0.339</distance2>
<distance3>0.294,0.291,0.199,0.0,0.123,0.231,0.54</distance3>
<distance4>0.312,0.296,0.217,0.199,0.0,0.231,0.558</distance4>
<distance5>0.308,0.265,0.215,0.231,0.355,0.0,0.555</distance5>
<distance6>0.167,0.128,0.22,0.42,0.424,0.394,0.0</distance6>
</distance>

</response>
```

Una volta ricevuto il file, viene estrapolata la la prima colonna della matrice delle distanze che, come descritto precedentemente, rappresenta la lista delle distanze dai punti in corrispondenza di una fermata dal punto in cui è localizzato l'utente. Sulla documentazione del sito di *open.mapquestapi.com*[22] viene specificato che il servizio può funzionare con liste contenenti al massimo 100 elementi ma è preferibile che liste lunghe siano suddivise in più richieste separate al fine di ottenere performance migliori. Pertanto nell'applicazione nel caso in cui vi sia una lista con un numero elevato di elementi viene suddivisa in liste di massimo 20 elementi. Si noti che il primo elemento di ogni lista sarà sempre il punto corrispondente alla posizione dell'utente.
A questo punto l'applicazione, partendo dalle fermate ottenute nella prima selezione, effettua una seconda selezione lasciando solo le fermate la cui distanza del percorso reale a piedi dall'utente è inferiore o uguale alla soglia di interesse.

Nella Tab.3.1 vengono riportati il numero di fermate scelte nella prima selezione, per diverse distanze, nel caso in cui l'utente sia localizzato in corrispondenza del *Liceo Righi di Bologna (viale Carlo Pepoli 3)* ed in un altro caso in cui l'utente sia localizzato in *via Indipendenza accanto a piazza XX Settembre(Bologna)*.
Dato il numero elevato di fermate selezionate all'aumentare della distanza inserita dall'utente, come si evince dalla Tab.3.1, e poichè il WebServer HelloBus di Tper restituisce gli orari in tempo reale solo per autobus che arrivano entro 15 min dal momento in cui viene effettuata la query, mentre per intervalli di tempo maggiore restituisce gli orari programmati, il servizio dovrà trattare un'area limitata intorno alla posizione dell'utente la cui distanza di interesse è dell'ordine di 1 km.

Raggio di interesse	*via Carlo Pepoli 3 (BO)* fermate selezionate:	*via Indipendenza (BO)* fermate selezionate:
50 m	2	1
100 m	2	6
150 m	4	11
200 m	11	17
250 m	14	22
300 m	18	27
350 m	23	31
400 m	28	39
450 m	29	43
500 m	31	48
1 km	100	135
1.5 km	249	293
2 km	353	430

Tabella 3.1

La seconda selezione può avere tre diversi esiti, ognuno dei quali porta alla creazione di una vista diversa:
1) Tutte le fermate si trovano ad una distanza inferiore o uguale alla soglia, quindi non ne deve essere scartata nessuna

```
CREATE VIEW FermateFinalSelection AS
SELECT * FROM FermateInteresse;
```

2) Deve essere scartata una sola fermata, che sarà quella con codice uguale all'unico elemento contenuto nell'array *$fermateSecondSelection*.
Si noti che la stringa *$whereString* contiene la condizione da inserire nella clausola WHERE della query.

```
mysql_query("CREATE VIEW FermateFinalSelection AS
                  SELECT *
                  FROM FermateInteresse
                  WHERE ".$whereString.";");
dove

$whereString =
"(codice_fermata <> ".$fermateSecondSelection[0].")";
```

3) Deve essere scartata più di una fermata, che saranno quelle con codici uguali agli elementi dell'array *$fermateSecondSelection*.
Si noti che la stringa *$whereString* viene composta ricorsivamente e contiene gli i codici separati da una virgola delle fermate che devono essere scartate.

```
mysql_query("CREATE VIEW FermateFinalSelection AS SELECT *
FROM FermateInteresse
WHERE codice_fermata NOT IN ".$whereString.";");

dove

$whereString = "(";

for($i=0;$i<sizeof($fermateSecondSelection);$i++) {
 if ($i=== sizeof($fermateSecondSelection)-1) {
  $whereString = $whereString.$fermateSecondSelection[$i].")";
 } else {
  $whereString = $whereString.$fermateSecondSelection[$i].",";
 }
}
```

Al termine della seconda selezione avviene l'interazione con il Web Service HelloBus (descritta dettagliatamente nel capitolo **??**), il quale restituirà gli orari di arrivo in tempo reale dei prossimi due autobus. Dopo viene mostrata all'utente la lista di fermate che hanno superato entrambe le selezioni con relative linee, e gli viene chiesto di esprimere il suo interesse verso tutte o parte di esse. Infine vengono mostrate all'utente solo le informazioni relative alle fermate di cui egli ha precedentemente espresso interesse. I possibili scenari di funzionamento dell'applicazione sono due: il primo nel quale l'utente

inserisce la sua posizione in formato di indirizzo postale, mentre il secondo nel quale la posizione viene inserita in formato di coordinate geografiche. Le Fig.3.3 e 3.4 mostra l'ordine delle interazioni tra il server e i Web Service esterni *open.mapquestapi.com* e *HelloBus* di Tper descritte precedentemente.

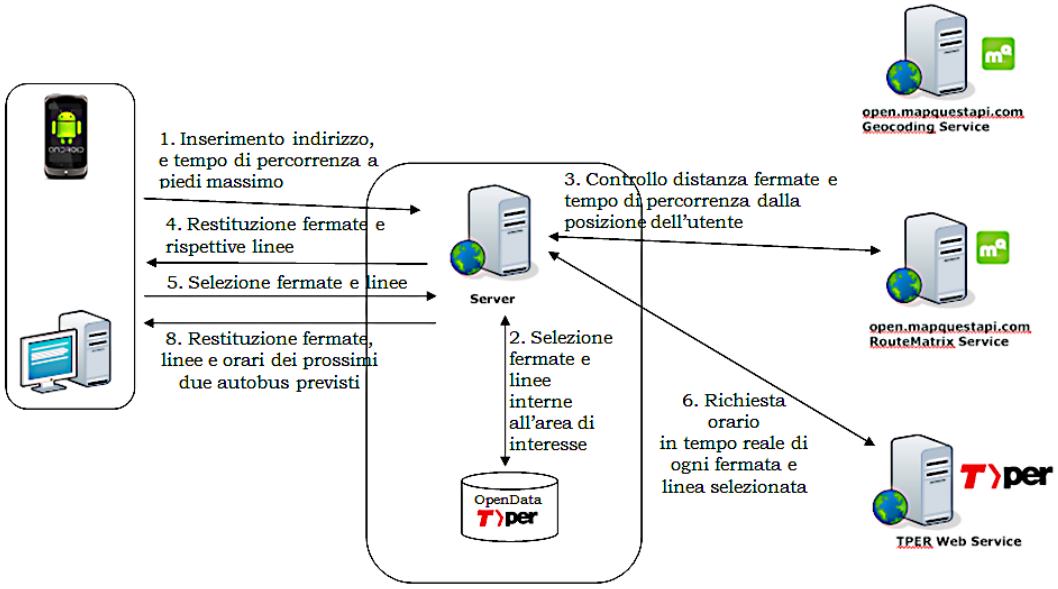


Figura 3.3: *Scenario con posizione in formato di coordinate geografiche.*

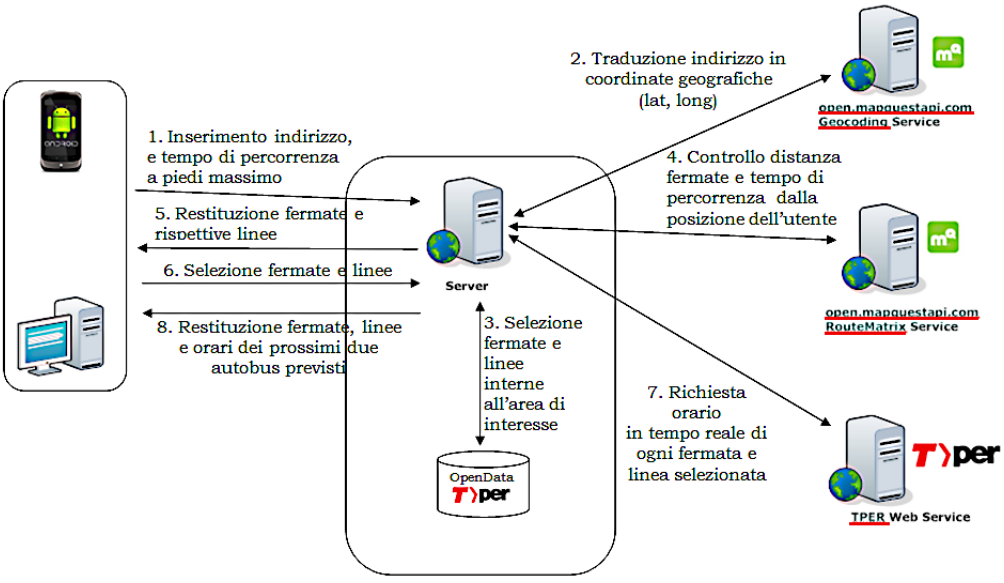


Figura 3.4: *Scenario con posizione in formato di indirizzo postale.*

L'applicazione è stata implementata in php e il database in mysql, come schematizzato in fig. 3.5, il codice è stato organizzato nei seguenti script:

index.php
Pagina web iniziale con la form che permette all'utente di inserire la sua posizione, il tempo di percorrenza massimo e richiedere al sistema di mostrargli le fermate e relative linee che rispettano tali vincoli.

requestHandler.php
Script che si occupa di gestire la richiesta dell'utente e di visualizzare i risultati.

HTTPPostFunctions.php
Script nel quale sono raggruppate le funzioni per fare le query con i Web Service esterni di open.mapquestapi e HelloBus. Ogni funzione crea il messaggio HTTP con le informazioni opportune per la specifica Query, poi alla ricezione della risposta estrapola i dati e li restituisce.

ParserFunctions.php
Script nel quale sono raggruppate le funzioni specifiche che effettuano il parsing delle risposte dei Web Service. Per quanto riguarda il Web Service HelloBus non è necessario effettuare il parsing della risposta in quanto verrà mostrata all'utente la stringa esattamente come viene ricevuta.

SqlQueryFunctions.php
Script nel quale sono raggruppate le funzioni per effettuare le queries al database. La funzione *queryCapolinea* realizza la query per creare la tabella che associa il capolinea ad ogni fermata e relativa linea, come discusso nella sezione 2.2.
Mentre la funzione *querySelectionFermate* realizza le due fasi della selezione delle fermate all'interno dell'area di interesse con le relative queries.

Utilities.php
Script che contiene alcune funzioni di utilità per l'applicazione:

- *calculateMaxDistance* che calcola la distanza percorribile mediamente da un uomo che cammina in un dato tempo;

- *getIntervalliCoordinateInteresse* che prende in ingresso le coordinate del punto nel quale è localizzato l'utente e calcola gli intervalli di coordinate ai quali appartengono i punti che si trovano all'interno dell'area di interesse.
- *changeCommaWithPoint* che trasforma i separatori dei numeri decimali all'interno dei file csv da virgola a punto.

UpdateDatabase.php

Script che si occupa di aggiornare il database locale contenente gli OpenData di Tper. Procede eliminando il database esistente, poi ricrea la struttura, in seguito popola tutte le tabelle prendendo i dati dai file csv che possono essere scaricati dal sito di Tper e infine effettua la query che crea la tabella delle fermate con il capolinea.

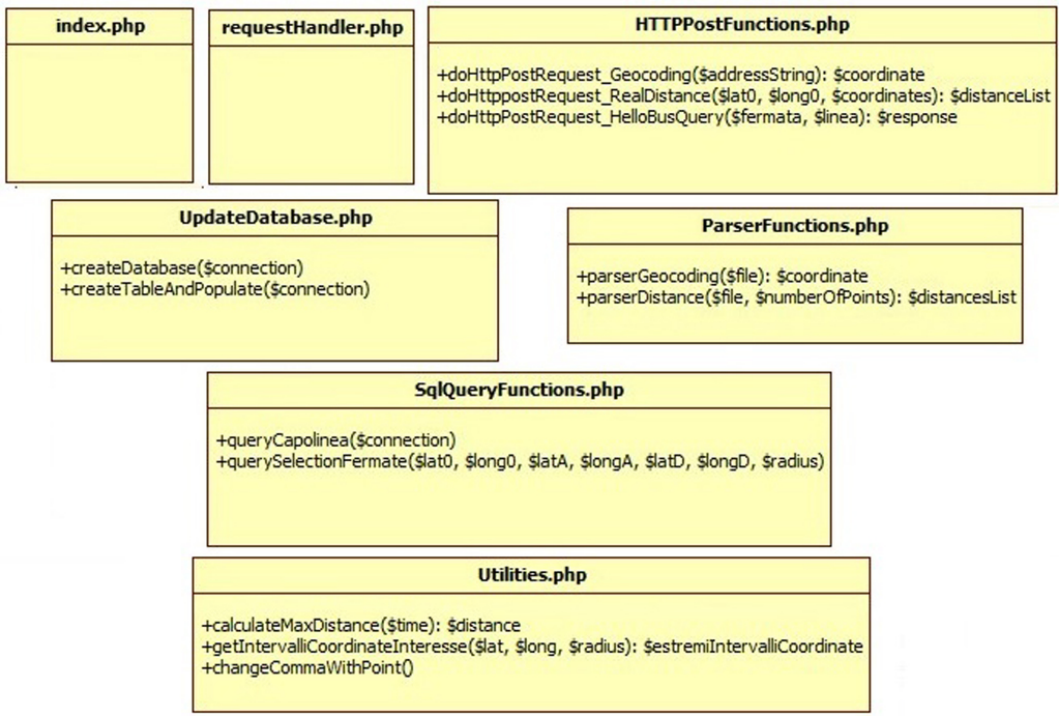


Figura 3.5: *Script implementati nell'applicazione.*

Le interazioni tra gli script, schematizzate nella figura 3.7, seguono i passi descritti nel capitolo e hanno inizio quando l'utente inserisce i dati nella form principale e preme il pulsante *RichiediInformazioni*.
I dati vengono inviati allo script principale dell'applicazione *requestHandler* il quale calcola la distanza massima percorribile da un uomo che cammina nel tempo dato chiamando la funzione *calculateMaxDistance*.
In seguito nel caso in cui l'utente ha inserito la posizione in formato di indirizzo postale deve essere tradotto in coordinate geografiche, altrimenti si può procedere subito al calcolo degli intervalli di coordinate di interesse.
La traduzione viene fatta chiamando la funzione *doHttpRequestPost_Geocoding*. Essa una volta ricevuta la risposta dal Web Service *open.mapquestapi* ne effettua il parsing mediante la funzione *parserGeocoding* e infine restituisce le coordinate estrapolate. Si noti che *$coordinate* è un array con due elementi uno con chiave *lat* e l'altro con chiave *long*, come riporta il codice seguente.

```
$coordinate = array (
      'lat' => value,
      'long' => value
);
```

Gli estremi degli intervalli delle coordinate di interesse vengono calcolati mediante la funzione *getIntervalliCoordinateInteresse* che restituisce l'array *$estremiIntervalliInteresse* con le coordinate degli estrami A e D, come riporta il codice seguente.

```
$estremiIntervalliInteresse = array (
                  'latA' => value,
                  'longA' => value,
                  'latD' => value,
                  'longD' => value
);
```

Dopo aver calcolato gli estremi degli intervalli si passa alla selezione delle fermate, la quale viene fatta mediante la funzione *querySelectionFermate*.
Essa esegue entrambe le fasi della selezione: la prima sulla base degli intervalli di coordinate precedentemente calcolati; mentre la seconda sulla base della lunghezza del percorso a piedi reale dall'utente alle diverse fermate.
La lunghezza del percorso reale la ottiene con la funzione *httpPostRequest_RealDistance* che fa la query al Web Service *Directions* di *open.mapquestapi*, ed estrapola i dati ricevuti con il parser *parserDistance*.

In seguito ogni fermata selezionata e relative linee vengono inserite all'interno della tabella *InfoRealTime_FermateInteresse*, poi per ogni coppia (fermata,linea) viene effettuata la query al Web Service *HelloBus* e inserito il risultato nel campo *informazione_realtime* della tabella.
Successivamente le fermate e relative linee vengono mostrate all'utente che dovrà esprimere il suo interesse verso tutte o parte di esse e infine il sistema esporrà gli orari dei prossimi due autobus per ogni (fermata,linea) scelta dall'utente.

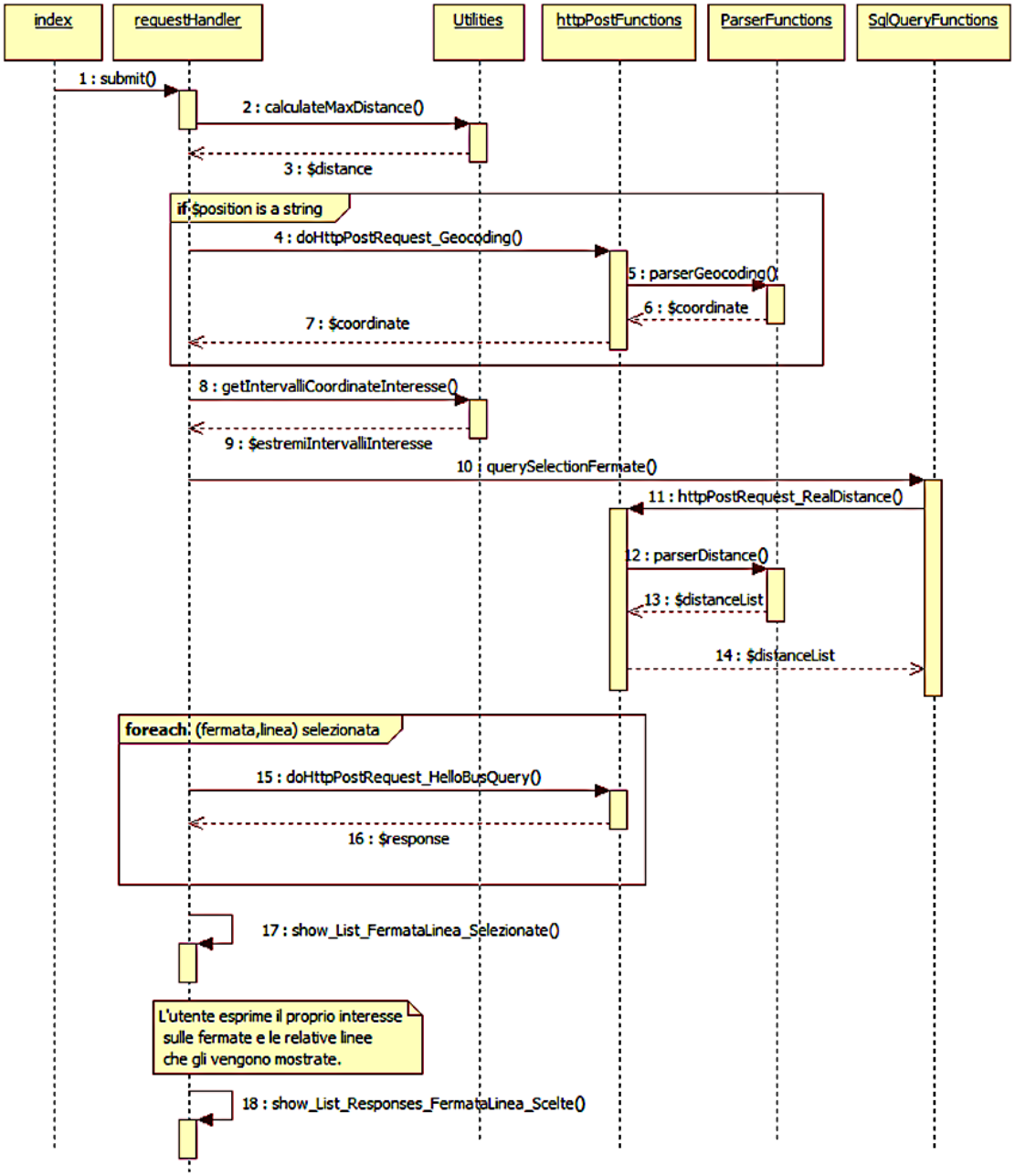


Figura 3.6: *Interazioni tra gli script negli scenari principali.*

Per l'aggiornamento del database ogni 6 mesi si dovranno scaricare i file aggiornati dal sito di Tper in formato csv, e ricreare interamente il database locale con i dati di Tper aggiornati. Tale operazione dovrà risultare il più trasparente possibile per gli utenti del servizio.
Al fine di testare il funzionamento dello script *UpdateDatabase*, che si occupa dell'aggiornamento del database, è stato inserito un link nella pagina web principale.
Una volta avviato *UpdateDatabase*, questo per prima cosa si occupa di modificare i file csv che contengono numeri decimali, trasformando il separatore degli stessi da virgola a punto, poi si connette a mysql e crea il database assegnandogli i privilegi opportuni. In seguito crea la struttura delle tabelle e le popola con i dati contenuti nei file csv, quindi effettua l'insieme di query che porta alla creazione della tabella *Fermate_Linee_Capolinea* che contiene per ogni coppia (fermata, linea) tutti i capolinea. Infine effettua la disconnessione da mysql.

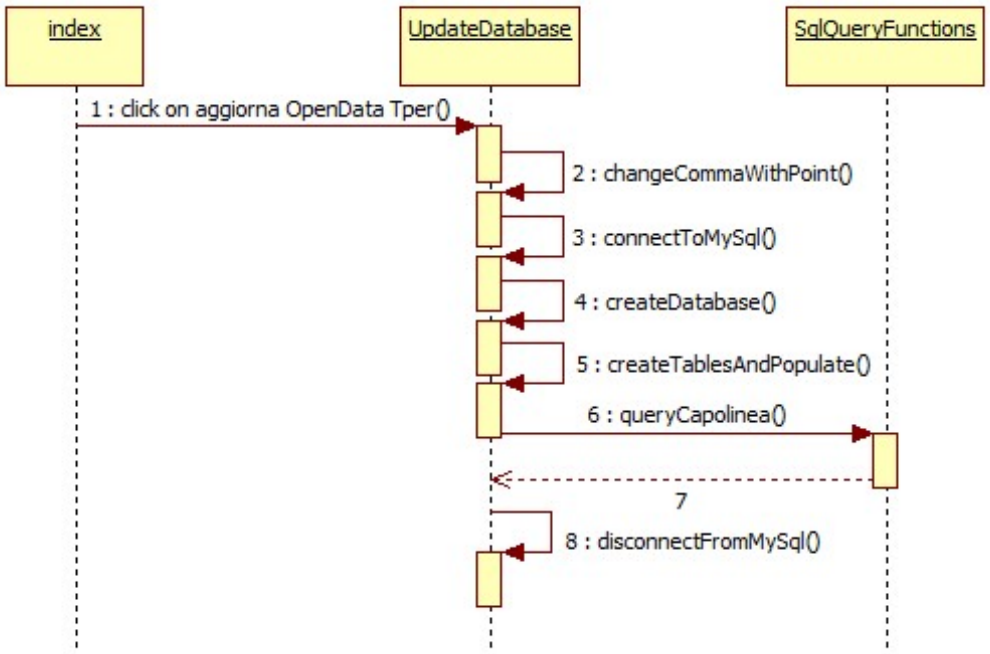


Figura 3.7: *Interazioni tra gli script nello scenario di aggiornamento del database.*

3.3 Interazione con il Web Service HelloBus

La query che consente di richiedere l'orario in tempo reale degli autobus al servizio HelloBus, come si evince dalla documentazione presente nei siti di Tper [23] e [24], in ingresso vuole i seguenti parametri:

- *codice fermata*
- *codice linea*
- *ora*

Il Web Service HelloBus alla ricezione di una query può inviare tre tipi di risposte:

1. $''$OGGI NESSUNA ALTRA CORSA DI $< codice\_linea >$ PER FERMATA $< codice\_fermata >''$;
2. $'' < codice\_linea >$ Da Satellite $< orario >''$ che indica l'orario **reale** di arrivo di quell'autobus su quella fermata ;
3. $'' < codice\_linea >$ Previsto $< orario >''$ che indica l'orario **programmato** di quell'autobus su quella fermata .

In generale specificando una *fermata* e una *linea* il Web Service può restituire la risposta di tipo 1, oppure l'orario reale (tipo 2) e/o programmato (tipo 3) dei prossimi due autobus della *linea* in arrivo sulla *fermata*.
E' possibile inoltre inviare query senza specificare necessariamente tutti i parametri:

- se il parametro mancante è l'*ora*, il Web Service utilizzerà l'orario in cui avrà ricevuto la query.
- se il parametro mancante è la *linea*, il Web Service restituirà i prossimi due autobus in arrivo nella fermata specificata tra tutte le linee che transitano in quella fermata.

Di seguito nelle Fig. 3.8-3.11 vengono riportati alcuni esempi che mostrano tutti i possibili output delle query al servizio HelloBus.

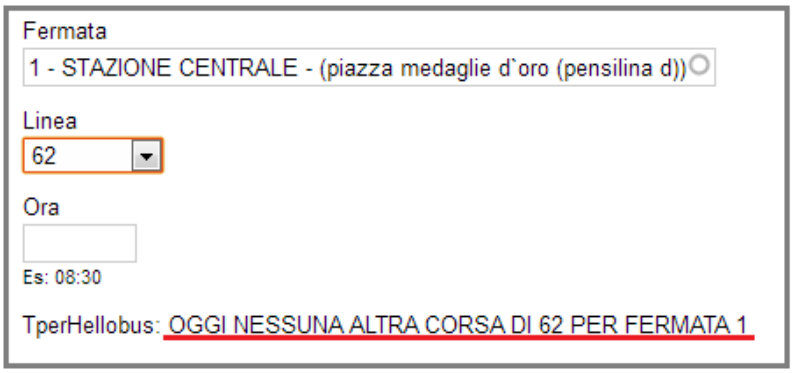


Figura 3.8: Caso in cui non sono presenti ulteriori corse programmate per la linea

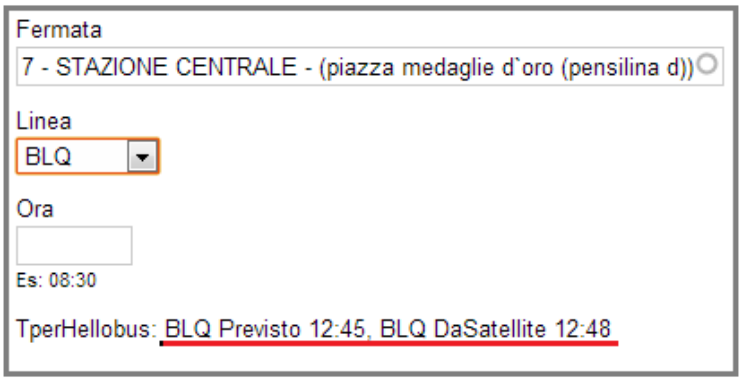


Figura 3.9: Caso in cui l'autobus è in ritardo di 3 min

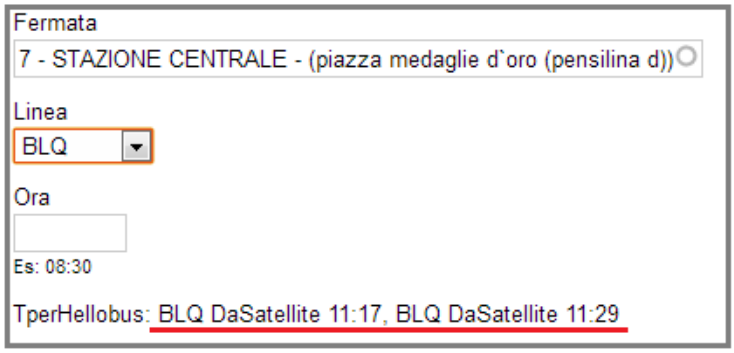


Figura 3.10: Caso in cui l'autobus arriverà alle 11:17 e alle 11:29

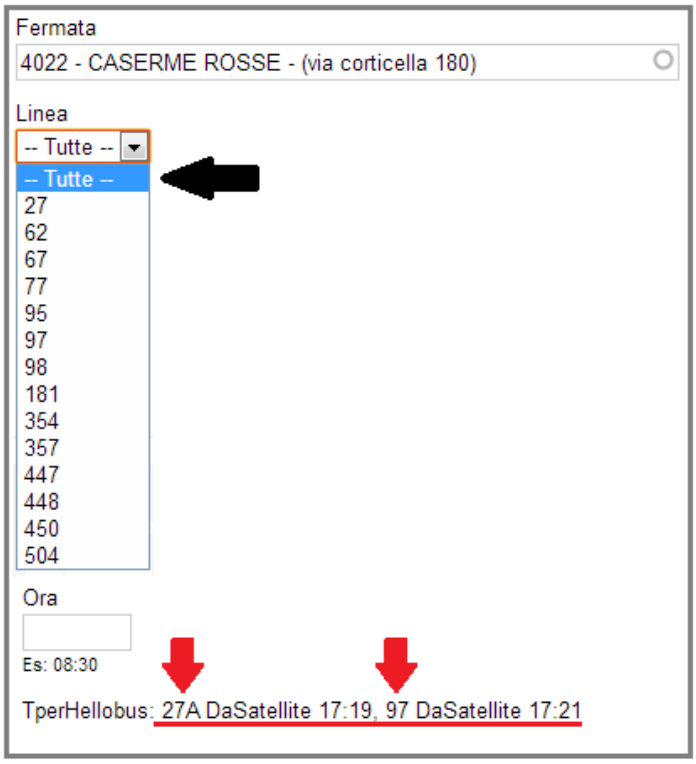


Figura 3.11: Caso in cui vengono selezionate tutte le linee di una certa fermata e vengono restituiti gli orari delle prossime due linee che transiteranno in quella fermata

Si può interagire con il Web Service HelloBus via HTTP, in particolare la richiesta deve avere la seguente sintassi:

```
POST /tperit/webservices/hellobus.asmx/QueryHellobus HTTP/1.1
Host: solweb.tper.it
Content-Type: application/x-www-form-urlencoded
Content-Length: length

fermata=string&linea=string&oraHHMM=string
```

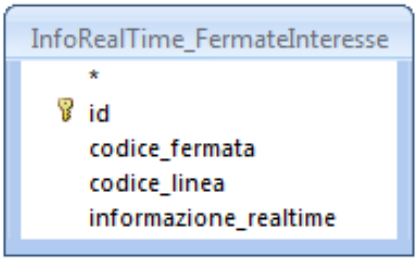
Mentre la risposta sarà un messaggio HTTP come il seguente:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<string xmlns="https://solweb.tper.it/tperit/webservices/
               hellobus.asmx">
string
</string>
```

Ad esempio per richiedere i prossimi autobus, considerando che la query viene effettuata alle ore 16:30, nella fermata *4022 Corticella di via Corticella 180* i messaggi HTTP scambiati con il WebServer HelloBus sono i seguenti:

```
POST /tperit/webservices/hellobus.asmx/QueryHellobus HTTP/1.1
Host: solweb.tper.it
Content-Type: application/x-www-form-urlencoded
Content-Length: length

fermata=4022&linea=&oraHHMM=
-----------------------------------------------------------------
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<string xmlns="https://solweb.tper.it/tperit/webservices/
               hellobus.asmx">
TperHellobus: 27B DaSatellite 16:35, 27A DaSatellite 16:39
</string>
```

L'applicazione seleziona le fermate (e relative linee) in base ai requisiti inseriti inizialmente dall'utente, come descritto nel capitolo 3, poi le inserisce nella tabella *InfoRealTime_FermateInteresse*.



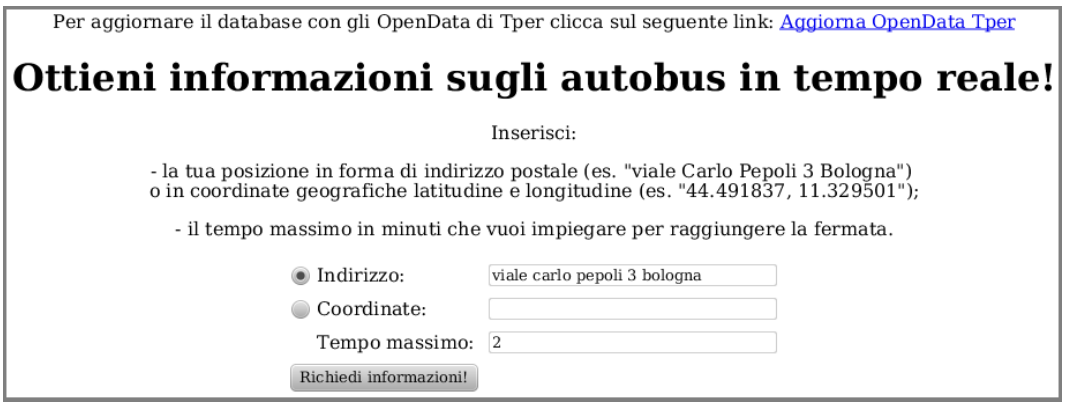
L'idea alla base di questa tabella è la possibilità di avere una sorta di cache per gestire efficientemente gruppi di utenti, caratterizzati dal fatto che le loro richieste hanno gli stessi requisiti o molto simili. Un esempio di gruppo può essere l'insieme degli studenti di una scuola che richiedono gli orari degli autobus che transitano nelle fermate adiacenti all'edificio scolastico.
In casi come questo risulta molto efficiente effettuare la selezione e l'inserimento dei dati nella tabella una volta per tutte, e mantenere gli orari delle fermate e linee selezionate sempre aggiornati. Così ogni volta che uno studente effettua una richiesta non è si deve eseguire nuovamente la selezione delle fermate, ma si possono trovare i dati richiesti direttamente all'interno della suddetta tabella.
Allo stato attuale l'applicazione, come si può notare in fig. 3.7, procede inserendo le fermate (e relative linee) all'interno della tabella *InfoRealTime_FermateInteresse*, poi per ognuna di esse esegue la query al Web Service HelloBus ed aggiorna il relativo campo *informazione_realtime*, infine mostra all'utente tutte le coppie (fermata,linea) con il corrispondente risultato della query.

Capitolo 4

Casi d'uso dell'applicazione

Questa sezione contiene diversi esempi che descrivono dettagliatamente tutti i possibili casi d'uso dell'applicazione.

1) Inserimento posizione ″via Carlo Pepoli 3 Bologna″ e tempo ″2 min″



L'utente compila la form inserendo la sua posizione in formato di indirizzo postale, selezionando il relativo radio button, ed inserendo il tempo massimo di percorrenza a piedi che vuole impiegare per raggiungere le fermate e poi preme il pulsante che invia i dati e avvia la gestione della richiesta.

Poichè la posizione è in formato di indirizzo postale, come si può vedere dalla figura che segue, la prima operazione eseguita dal sistema è la traduzione in coordinate geografiche che restituisce le coordinate (44.491765, 11.32959). Successivamente calcola la distanza percorribile mediamente da un uomo in 2 minuti ottenendo 134.4 m, e gli estremi A e D degli intervalli di coordinate di interesse.
A questo punto può eseguire la selezione delle fermate che rispettano i vincoli specificati dall'utente. Si può notare dalla figura che vengono selezionate nel primo step 3 fermate {46, 51, 711}, mentre nel secondo viene scartata la fermata 711.

```
inserito indirizzo
viale carlo pepoli 3 bologna
latitudine = 44.491765
longitudine = 11.32959
estrapolazione dati
tempo = 2 min
distanza = 134.4 metri


coordinateInteresse :
puntoA = 44.4905566,11.3282819
puntoD = 44.4929733,11.3309011

Query_Fermate OK
Query_FermateSelezionate OK

 num fermate selezionate = 3

*** TABELLA FermateInteresse ***
codice_fermata | Denominazione | Ubicazione | codice_linea |
codice_capolinea | denominazione_capolinea | zona_capolinea

51 | LICEO RIGHI | VIALE PEPOLI 20 | 20 | 7163 | ROTONDA MALAGUTI | 500
51 | LICEO RIGHI | VIALE PEPOLI 20 | 20 | 6265 | STAZIONE CASALECCHIO GARIBALDI | 507
51 | LICEO RIGHI | VIALE PEPOLI 20 | 20 | 72421 | SAN BIAGIO RESISTENZA | 581
51 | LICEO RIGHI | VIALE PEPOLI 20 | 20 | 7046 | CASALECCHIO CENTRO | 507
51 | LICEO RIGHI | VIALE PEPOLI 20 | 33 | 43 | PORTA SAN MAMOLO | 500
51 | LICEO RIGHI | VIALE PEPOLI 20 | 61 | 3 | STAZIONE CENTRALE | 500
51 | LICEO RIGHI | VIALE PEPOLI 20 | 61 | 2010 | TRIACHINI | 500
46 | LICEO RIGHI | VIALE PEPOLI FR 56 | 20 | 3052 | SALGARI | 500
46 | LICEO RIGHI | VIALE PEPOLI FR 56 | 32 | 6 | STAZIONE CENTRALE | 500
46 | LICEO RIGHI | VIALE PEPOLI FR 56 | 32 | 4 | STAZIONE CENTRALE | 500
46 | LICEO RIGHI | VIALE PEPOLI FR 56 | D | 8 | STAZIONE CENTRALE | 500
711 | FRASSINAGO | PIAZZA DI PORTA SARAGOZZA PL 1 | D | 7236 | RAVONE | 500
*** fine TABELLA FermateInteresse ***

711 eliminata
```

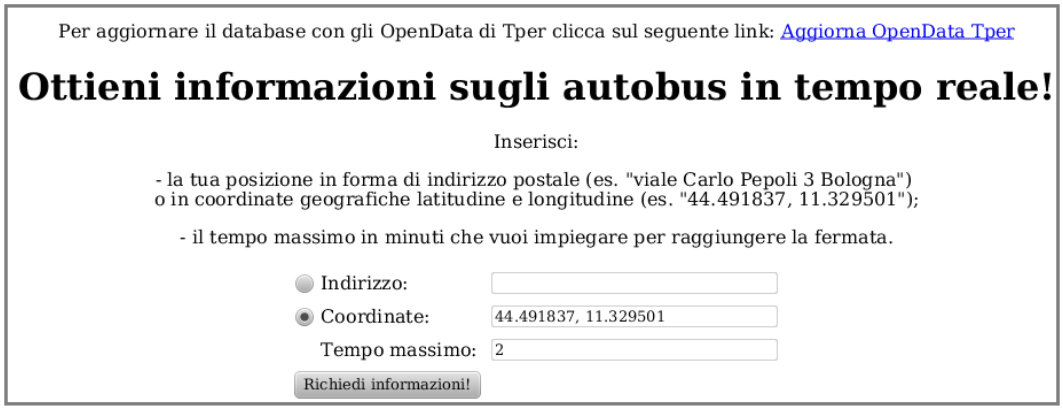
```
*** TABELLA FermateFinalSelection ***
codice_fermata | Denominazione | Ubicazione | codice_linea |
codice_capolinea | denominazione_capolinea | zona_capolinea

51 | LICEO RIGHI | VIALE PEPOLI 20 | 20 | 7163 | ROTONDA MALAGUTI | 500
51 | LICEO RIGHI | VIALE PEPOLI 20 | 20 | 6265 | STAZIONE CASALECCHIO GARIBALDI | 507
51 | LICEO RIGHI | VIALE PEPOLI 20 | 20 | 72421 | SAN BIAGIO RESISTENZA | 581
51 | LICEO RIGHI | VIALE PEPOLI 20 | 20 | 7046 | CASALECCHIO CENTRO | 507
51 | LICEO RIGHI | VIALE PEPOLI 20 | 33 | 43 | PORTA SAN MAMOLO | 500
51 | LICEO RIGHI | VIALE PEPOLI 20 | 61 | 3 | STAZIONE CENTRALE | 500
51 | LICEO RIGHI | VIALE PEPOLI 20 | 61 | 2010 | TRIACHINI | 500
46 | LICEO RIGHI | VIALE PEPOLI FR 56 | 20 | 3052 | SALGARI | 500
46 | LICEO RIGHI | VIALE PEPOLI FR 56 | 32 | 6 | STAZIONE CENTRALE | 500
46 | LICEO RIGHI | VIALE PEPOLI FR 56 | 32 | 4 | STAZIONE CENTRALE | 500
46 | LICEO RIGHI | VIALE PEPOLI FR 56 | D | 8 | STAZIONE CENTRALE | 500
*** fine TABELLA FermateFinalSelection ***
```

Infine viene creta la tabella *InfoRealTime_FermateInteresse* e riempita con le informazioni in tempo reale sulle coppie (fermata, linea) che hanno superato entrambi gli step della selezione.
La figura seguente mostra il contenuto della tabella *InfoRealTime_FermateInteresse* così interpretabile:

- la linea 20 non effettua più nessuna corsa sia per la fermata 46 che per la fermata 51, allo stesso modo della linea 61 per la fermata 51
- la linea 33 per la fermata 51 è in ritardo di 2 min poichè ha come orario programmato 12:06 e orario reale 12:08
- la linea 32 per la fermata 46 arriverà alle 12:02 e alle 12:17
- la linea D per la fermata 46 ha orari programmati 12:24 e 12:49

```
*** TABELLA InfoRealTime_FermateInteresse ***
codice_fermata | codice_linea | informazione_realtime
51 | 20 | OGGI NESSUNA ALTRA CORSA DI 20 PER FERMATA 51

51 | 33 | 33 DaSatellite 12:06, 33 DaSatellite 12:08

51 | 61 | OGGI NESSUNA ALTRA CORSA DI 61 PER FERMATA 51

46 | 20 | OGGI NESSUNA ALTRA CORSA DI 20 PER FERMATA 46

46 | 32 | 32 DaSatellite 12:02, 32 DaSatellite 12:17

46 | D | D Previsto 12:24, D Previsto 12:49


*** fine TABELLA InfoRealTime_FermateInteresse ***
```

2) Inserimento ″44.491837, 11.329501″ e tempo ″2 min″



In questo caso l'utente specifica la sua posizione, che è identica a quella del caso precedente, inserendo le coordinate geografiche ottenute da *googleMaps* in corrispondenza dell'indirizzo *viale Carlo Pepoli 3 Bologna*.

```
inserite coordinate
44.491837, 11.329501
estrapolazione dati
tempo = 2 min
distanza = 134.4 metri


coordinateInteresse :
puntoA = 44.4906286,11.3281928
puntoD = 44.4930453,11.3308121

Query_Fermate OK
Query_FermateSelezionate OK

num fermate selezionate = 2

*** TABELLA FermateInteresse ***
codice_fermata | Denominazione | Ubicazione | codice_linea |
codice_capolinea | denominazione_capolinea | zona_capolinea

51 | LICEO RIGHI | VIALE PEPOLI 20 | 20 | 7163 | ROTONDA MALAGUTI | 500
51 | LICEO RIGHI | VIALE PEPOLI 20 | 20 | 6265 | STAZIONE CASALECCHIO GARIBALDI | 507
51 | LICEO RIGHI | VIALE PEPOLI 20 | 20 | 72421 | SAN BIAGIO RESISTENZA | 581
51 | LICEO RIGHI | VIALE PEPOLI 20 | 20 | 7046 | CASALECCHIO CENTRO | 507
51 | LICEO RIGHI | VIALE PEPOLI 20 | 33 | 43 | PORTA SAN MAMOLO | 500
51 | LICEO RIGHI | VIALE PEPOLI 20 | 61 | 3 | STAZIONE CENTRALE | 500
51 | LICEO RIGHI | VIALE PEPOLI 20 | 61 | 2010 | TRIACHINI | 500
46 | LICEO RIGHI | VIALE PEPOLI FR 56 | 20 | 3052 | SALGARI | 500
46 | LICEO RIGHI | VIALE PEPOLI FR 56 | 32 | 6 | STAZIONE CENTRALE | 500
46 | LICEO RIGHI | VIALE PEPOLI FR 56 | 32 | 4 | STAZIONE CENTRALE | 500
46 | LICEO RIGHI | VIALE PEPOLI FR 56 | D | 8 | STAZIONE CENTRALE | 500
*** fine TABELLA FermateInteresse ***

fermate scartate = 0

Query_FermateSecondSelectionFinal OK
```

Poichè la posizione e il tempo sono gli stessi ci si aspetta che vengano selezionate anche le medesime fermate, e come viene documentato nelle figure seguenti vengono scelte proprio la fermata 46 e la 51.

*** TABELLA FermateFinalSelection ***
codice_fermata | Denominazione | Ubicazione | codice_linea |
codice_capolinea | denominazione_capolinea | zona_capolinea

51 | LICEO RIGHI | VIALE PEPOLI 20 | 20 | 7163 | ROTONDA MALAGUTI | 500
51 | LICEO RIGHI | VIALE PEPOLI 20 | 20 | 6265 | STAZIONE CASALECCHIO GARIBALDI | 507
51 | LICEO RIGHI | VIALE PEPOLI 20 | 20 | 72421 | SAN BIAGIO RESISTENZA | 581
51 | LICEO RIGHI | VIALE PEPOLI 20 | 20 | 7046 | CASALECCHIO CENTRO | 507
51 | LICEO RIGHI | VIALE PEPOLI 20 | 33 | 43 | PORTA SAN MAMOLO | 500
51 | LICEO RIGHI | VIALE PEPOLI 20 | 61 | 3 | STAZIONE CENTRALE | 500
51 | LICEO RIGHI | VIALE PEPOLI 20 | 61 | 2010 | TRIACHINI | 500
46 | LICEO RIGHI | VIALE PEPOLI FR 56 | 20 | 3052 | SALGARI | 500
46 | LICEO RIGHI | VIALE PEPOLI FR 56 | 32 | 6 | STAZIONE CENTRALE | 500
46 | LICEO RIGHI | VIALE PEPOLI FR 56 | 32 | 4 | STAZIONE CENTRALE | 500
46 | LICEO RIGHI | VIALE PEPOLI FR 56 | D | 8 | STAZIONE CENTRALE | 500
*** fine TABELLA FermateFinalSelection ***

*** TABELLA InfoRealTime_FermateInteresse ***
codice_fermata | codice_linea | informazione_realtime
51 | 20 | OGGI NESSUNA ALTRA CORSA DI 20 PER FERMATA 51

51 | 33 | 33 DaSatellite 12:42, 33 DaSatellite 12:49

51 | 61 | OGGI NESSUNA ALTRA CORSA DI 61 PER FERMATA 51

46 | 20 | OGGI NESSUNA ALTRA CORSA DI 20 PER FERMATA 46

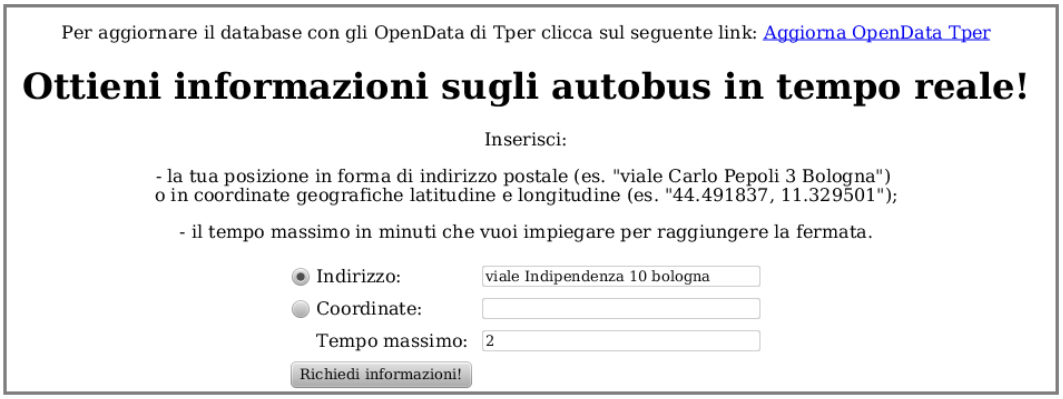
46 | 32 | 32 DaSatellite 12:38, 32 DaSatellite 12:53

46 | D | D Previsto 12:49, D Previsto 13:14

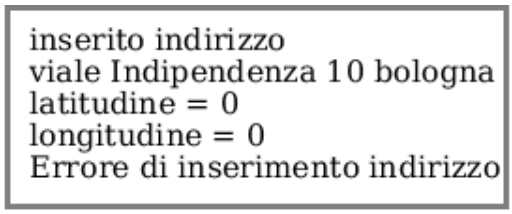
*** fine TABELLA InfoRealTime_FermateInteresse ***

3) Inserimento indirizzo non corretto ″viale Indipendenza 10 bologna″ e tempo ″2 min″

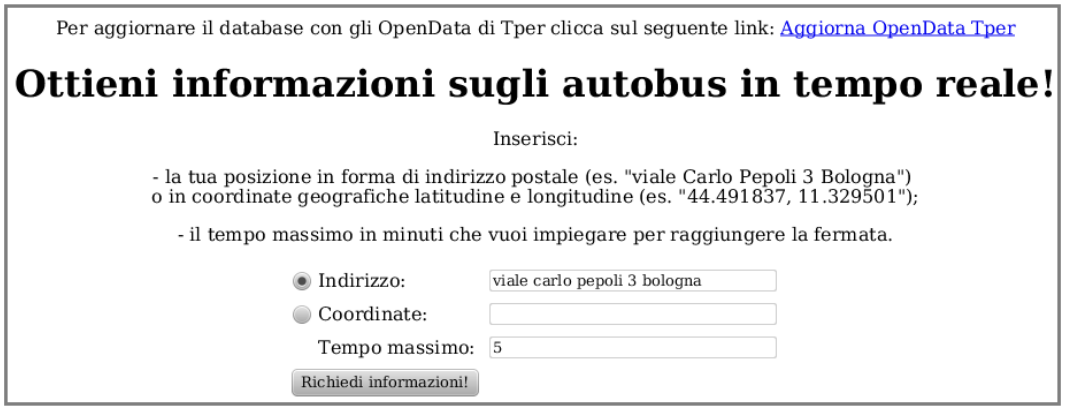
In questo scenario l'utente vorrebbe inserire ″viale dell'Indipendenza 10 bologna″ ma si sbaglia e inserisce ″viale Indipendenza 10 bologna″ il quale non esiste.



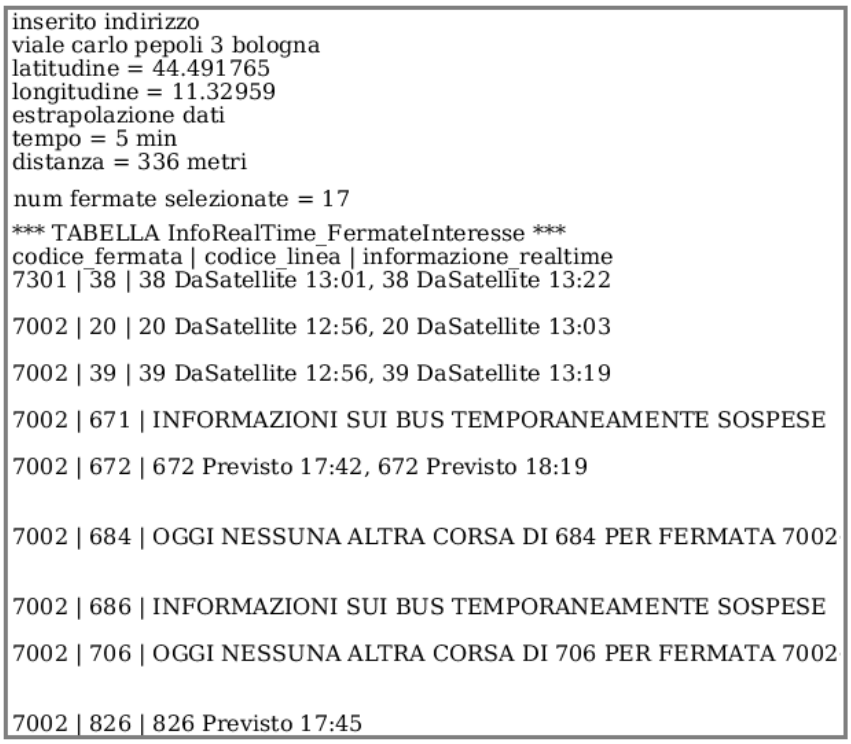
La traduzione dell'indirizzo in coordinate geografiche restituisce, come si può osservare nella figura seguente, le coordinate (0,0). Questi valori vengono utilizzati per indicare che la query al Web Service Geocoding di *open.mapquestapi* non è andata a buon fine. In casi come questo l'esecuzione dell'applicazione termina segnalando l'errore con un opportuno messaggio.



4) Inserimento posizione ″via Carlo Pepoli 3 Bologna″ e tempo ″5 min″



Questo scenario e i due successivi mostrano come aumenta il numero di fermate selezionate al crescere del tempo di percorrenza specificato dall'utente.



7002 | 94 | 94 DaSatellite 13:19, 94 DaSatellite 13:55

7002 | 963 | OGGI NESSUNA ALTRA CORSA DI 963 PER FERMATA 7002

7002 | 964 | OGGI NESSUNA ALTRA CORSA DI 964 PER FERMATA 7002

7002 | D | D Previsto 13:12, D Previsto 13:37

7001 | 20 | 20A DaSatellite 13:01, 20 DaSatellite 13:08

7001 | 38 | 38 DaSatellite 12:59, 38 DaSatellite 13:21

7001 | 671 | 671 DaSatellite 13:05, 671 DaSatellite 13:51

7001 | 672 | OGGI NESSUNA ALTRA CORSA DI 672 PER FERMATA 7001

7001 | 684 | OGGI NESSUNA ALTRA CORSA DI 684 PER FERMATA 7001

7001 | 686 | 686 Previsto 14:29, 686 Previsto 17:15

7001 | 706 | OGGI NESSUNA ALTRA CORSA DI 706 PER FERMATA 7001

7001 | 826 | 826 Previsto 16:59, 826 Previsto 18:24

7001 | 850 | OGGI NESSUNA ALTRA CORSA DI 850 PER FERMATA 7001

7001 | 856 | 856 Previsto 13:29, 856 Previsto 17:39

7001 | 94 | 94 DaSatellite 13:18, 94 DaSatellite 14:07

7001 | 963 | OGGI NESSUNA ALTRA CORSA DI 963 PER FERMATA 7001

7001 | 964 | OGGI NESSUNA ALTRA CORSA DI 964 PER FERMATA 7001

7001 | D | D DaSatellite 13:14, D Previsto 13:41

53 | 20 | OGGI NESSUNA ALTRA CORSA DI 20 PER FERMATA 53

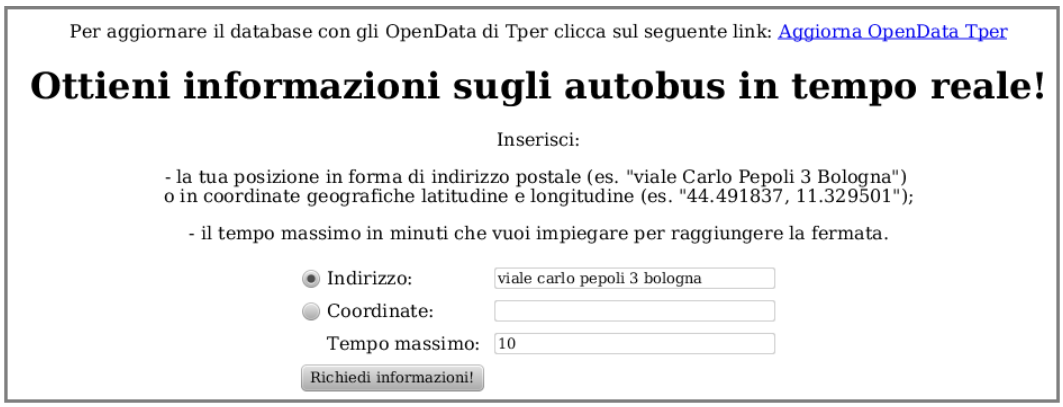
53 | 33 | 33 DaSatellite 12:58, 33 Previsto 13:05

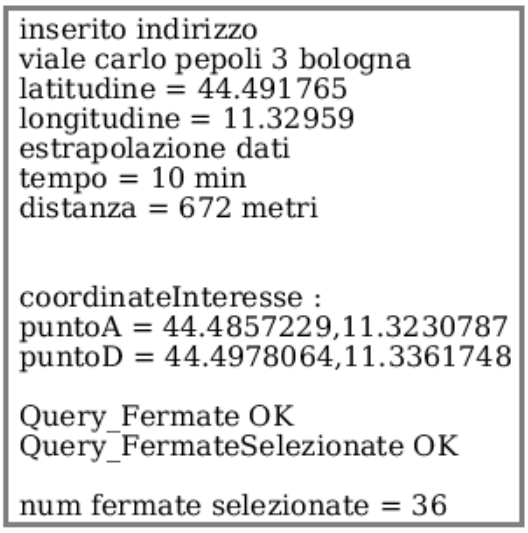
53 | 61 | OGGI NESSUNA ALTRA CORSA DI 61 PER FERMATA 53

53 | 970 | OGGI NESSUNA ALTRA CORSA DI 970 PER FERMATA 53

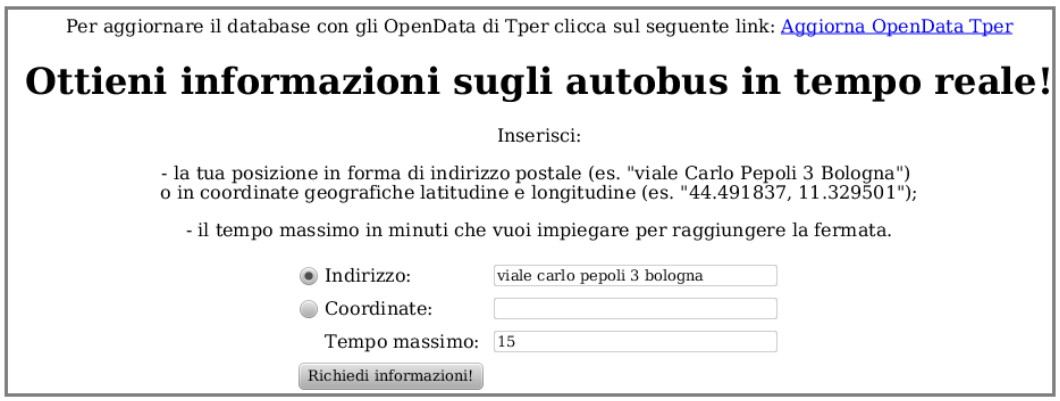
```
99028 | 970 | OGGI NESSUNA ALTRA CORSA DI 970 PER FERMATA 99028

51 | 20 | OGGI NESSUNA ALTRA CORSA DI 20 PER FERMATA 51
51 | 33 | 33 DaSatellite 12:58, 33 Previsto 13:06

51 | 61 | OGGI NESSUNA ALTRA CORSA DI 61 PER FERMATA 51
46 | 20 | OGGI NESSUNA ALTRA CORSA DI 20 PER FERMATA 46
46 | 32 | 32 DaSatellite 13:07, 32 DaSatellite 13:15
46 | D | D Previsto 13:14, D Previsto 13:39

47 | 20 | OGGI NESSUNA ALTRA CORSA DI 20 PER FERMATA 47
47 | 33 | 33 DaSatellite 12:59, 33 DaSatellite 13:06
44 | 32 | 32 DaSatellite 13:06, 32 Previsto 13:16

711 | D | D DaSatellite 13:13, D Previsto 13:41

*** fine TABELLA InfoRealTime_FermateInteresse ***
```

5) Inserimento posizione ″via Carlo Pepoli 3 Bologna″ e tempo ″10 min″





6) Inserimento posizione ″via Carlo Pepoli 3 Bologna″ e tempo ″15 min″



Capitolo 5

Conclusioni

L'obiettivo principale del lavoro svolto nell'elaborato consisteva nella realizzazione di un'applicazione in grado di fornire informazioni in tempo reale sulle linee di autobus, e relative fermate, in una determinata area di interesse utilizzando i dati messi a disposizione di Tper in formato OpenData. L'enorme mole di dati da gestire e dato che l'applicazione deve poter essere utilizzata sia da singoli utenti ma anche da microcommunity, come ad esempio l'insieme degli studenti di una scuola che effettuano un numero elevato di richieste con i medesimi requisiti di collocazione geografica delle fermate, giustificano la presenza di un server che si incarichi dell'elaborazione.

In conclusione è stato progettato e implementato il suddetto server, il quale nota la posizione dell'utente e il tempo che questi è disposto a impiegare nel caso peggiore per raggiungere le fermate degli autobus restituisce gli orari reali dei prossimi due autobus in arrivo, a partire dal momento nel quale viene effettuata la richiesta, per ogni linea di autobus che transita nelle fermate collocate intorno all'utente ad una distanza percorribile nell'intervallo di tempo specificato.
E' stata realizzata anche una procedura che permette l'aggiornamento in automatico della copia locale del database degli OpenData di Tper.
Durante l'elaborazione di una richiesta le informazioni riguardanti le fermate selezionate e gli orari di arrivo delle relative linee non solo vengono restituite all'utente ma vengono salvate all'interno del database, questo approccio fornisce una solida base sulla quale possono essere sviluppate diverse funzionalità aggiuntive, ad esempio un meccanismo di cache che permetta la

gestione efficiente di un numero elevato di richieste di utenti appartenenti ad una microcommunity, oppure introdurre la possibilità per utenti non esperti della città di richiedere indicazioni stradali per raggiungere una determinata fermata.
Ulteriori estensioni possono riguardano lo sviluppo della parte frontend dell'applicazione tramite un'app per smart phone e tablet, per accrescere l'interattività con l'utente, ad esempio possibilità di salvare una lista di fermate preferite, oppure l'introduzione di un'icona che negli orari nei quali si è soliti prendere l'autobus tenendo conto della posizione dell'utente indichi se questo deve correre alla fermata o può camminare sulla base delle preferenze da questo impostate.
Infine è stata ipotizzata la possibilità di inserire un tablet all'interno di alcune strutture pubbliche (come scuole e musei) che fornisca informazioni sui trasporti in maniera automatica.

Appendice A

Codice implementato

index.php

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>Bus RealTime</title>
</head>
<body>
 <?php
   error_reporting(-1);
   ini_set('display_errors', true);
 ?>
 <hr>
 <!--<iframe style="display:none;" name="target"></iframe>-->
 <p align="center">Per aggiornare il database con gli OpenData
                   di Tper clicca sul seguente link:
 <a href="UpdateDatabase.php" target="target">
 Aggiorna OpenData Tper</a></p>
 <h1 align="center">Ottieni informazioni sugli autobus
                    in tempo reale!</h1>
 <p align="center">Inserisci: </p>
 <p align="center">
 - la tua posizione in forma di indirizzo postale
   (es. "viale Carlo Pepoli 3 Bologna") <br>
   o in coordinate geografiche latitudine e longitudine
   (es. "44.491837, 11.329501");
 </p>
 <p align="center">
```

```
 - il tempo massimo in minuti che vuoi impiegare
                      per raggiungere la fermata.
 </p>
 <form action="requestHandler.php" method="post">
 <!--<form action="coordinateHandler.php" method="post"> -->
 <table align="center">
 <tr>
 <td>
 <input name="InputType" type="radio" value="indirizzo">
 </td>
 <td> Indirizzo:</td>
 <td>
 <input type="text" name="stringaIndirizzo" value=""
 size="30">
 </td>
 </tr>
 <tr>
 <td>
 <input name="InputType" type="radio" value="coordinate">
 </td>
 <td> Coordinate: </td>
 <td>
 <input type="text" name="stringaCoordinate" value=""
 size="30">
 </td>
 </tr>
 <tr>
 <td></td>
 <td>Tempo massimo: </td>
 <td><input type="text" name="time" value="" size="30"></td>
 </tr>
 <tr>
 <td colspan="2" align=center><input type="submit"
 value="Richiedi informazioni!">
 </td>
 </tr>
 </table>
 </form>
 </body>
</html>
```

requestHandler.php

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>Bus RealTime</title>
</head>
<body>
<?php
 error_reporting(-1);
 ini_set('display_errors', true);
 include './HTTPPostFunctions.php';
 include './Utilities.php';
 include './SqlQueryFunctions.php';

 $stringaCoordinate = $_POST['stringaCoordinate'];
 $stringaIndirizzo = $_POST['stringaIndirizzo'];
 $time = $_POST['time'];
 $radius = calculateMaxDistance($time);

 if($_POST['InputType']=="indirizzo"){
  echo "scelto indirizzo </br>";
  echo $stringaIndirizzo;
  echo "</br>";
  $coordinate =
  doHttpPostRequest_Geocoding($stringaIndirizzo);

 }else{
  echo "scelto coordinate </br>";
  echo $stringaCoordinate;
  echo "</br>";
  $stringaCoordinatePieces=explode(",",$stringaCoordinate);
  $coordinate=array('lat' => $stringaCoordinatePieces[0],
                    'long' => $stringaCoordinatePieces[1]
                   );
  $lat = $coordinate['lat'];
  $long = $coordinate['long'];

  if ($lat == 0){
   echo "Errore di inserimento indirizzo";
  }else{
   echo "estrapolazione dati";
   echo "<br/>";
   /*echo "coordinate lat = ";
```

```
echo $coordinate['lat'];
echo "</br>";
echo "coordinate long = ";
echo $coordinate['long'];
echo "</br>";  */
echo "tempo = ".$time."</br>";
echo "distanza = ".$radius."</br>";
echo "</br>";

$estremiIntervalliCoordinate =
getIntervalliCoordinateInteresse($lat,$long,$radius);

$latA = $estremiIntervalliCoordinate['latA'];
$longA = $estremiIntervalliCoordinate['longA'];
$latD = $estremiIntervalliCoordinate['latD'];
$longD = $estremiIntervalliCoordinate['longD'];

echo "</br>";
echo "coordinateInteresse : </br>";
echo "puntoA = ".$latA.",".$longA."</br>";
echo "puntoD = ".$latD.",".$longD."</br>";

//seleziona le fermate all'interno dell'area di interesse:
//-la prima fase sulla base della distanza in linea d'aria
//-la seconda fase sulla base della lunghezza del percorso
   reale a piedi
querySelectionFermate($lat, $long, $latA, $longA, $latD,
                      $longD,$radius);

$connection=mysql_connect('localhost', 'root', 'root',
                          false,128);
if(!$connection){
 die('Not connected : ' . mysql_error());
}else{
 echo "connesso!!! </br>";
}
$db_selected = mysql_select_db('tper', $connection);
if($db_selected === TRUE){
 echo "db selezionato </br>";
}

mysql_query("DROP TABLE InfoRealTime_FermateInteresse;");

$tab_infoRealTime =
mysql_query("CREATE TABLE InfoRealTime_FermateInteresse(
```

```
            id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
            codice_fermata VARCHAR(10),
            codice_linea VARCHAR(10),
            informazione_realtime VARCHAR(50)
          ) SELECT DISTINCT codice_fermata,codice_linea
            FROM FermateFinalSelection;");

// debug query

if($tab_infoRealTime === TRUE){
 echo "Tabella InfoRealTime_FermateInteresse OK <br>";
}else{
 echo "Errore creazione Tabella
 InfoRealTime_FermateInteresse:".
 mysql_error($connection)."<br>";
}

$query_infoRealTime_Test =
mysql_query("SELECT *
             FROM InfoRealTime_FermateInteresse;");


//echo "inizio TABELLA FermateFinalSelection: </br>";
$tempoInizioQuery = date("d/m/y : H:i:s", time());
echo "tempoInizioQuery = ".$tempoInizioQuery."<br>";
// DEBUG
while($row_query_final =
      mysql_fetch_array($query_infoRealTime_Test)){
 $fermata=$row_query_final['codice_fermata'];
 $linea=$row_query_final['codice_linea'];
 $response =
   doHttpPostRequest_HelloBusQuery($fermata,$linea);
 $updateQuery =
   mysql_query("UPDATE InfoRealTime_FermateInteresse
                SET informazione_realtime='$response'
                WHERE (codice_fermata = '$fermata') AND
                (codice_linea = '$linea');");
 if($updateQuery===TRUE){
  //echo "Tabella InfoRealTime_FermateInteresse
          aggiornata OK </br>";
 }else{
  echo "Errore aggiornamento tabella
  InfoRealTime_FermateInteresse:".
  mysql_error($connection)."</br>";
 }
```

```
    }

    $query_infoRealTime_Update_Test =
      mysql_query("SELECT *
                   FROM InfoRealTime_FermateInteresse;");

    while($row_query_update_final =
          mysql_fetch_array($query_infoRealTime_Update_Test)){
     $fermata=$row_query_update_final['codice_fermata'];
     $linea=$row_query_update_final['codice_linea'];
     $info=$row_query_update_final['informazione_realtime'];
     echo $fermata." | ";
     echo $linea." <br>";
     echo $info;
     echo "<br>";
     echo "<br>";

    }

 }

 $tempoFinale = date("d/m/y : H:i:s", time());
 echo "tempoFinale = ".$tempoFinale."</br>";
?>
</body>
</html>
```

Utilities.php

```
<?php

// stampa gli errori
error_reporting(-1);
ini_set('display_errors', true);

// calcola la distanza in metri percorribile mediamente
// da un uomo in un dato tempo(espresso in minuti)

function calculateMaxDistance($time){
 $vel = 1.12; //m/s
 $distance = $vel * ($time * 60);
 return $distance;
}
```

```
function getIntervalliCoordinateInteresse($lat,$long,$radius){
 // raggio in  metri
 echo "lat = ".$lat."</br>";
 echo "long = ".$long."</br>";*/
 $R = 6372.795477598 ; // Raggio Terra
 $angleA = 225.0;
 $angleD = 45.0;
 $azimutA = ($angleA*pi())/180;
 $azimutD = ($angleD*pi())/180;
 $lat0 = ($lat*pi())/180;
 $long0 = ($long*pi())/180;

 $diagonal = round(($radius/1000) * sqrt(2),9);

 // Calcolo latA, longA
 $latA_rad =
  asin(sin($lat0) * cos($diagonal / $R) +
       cos($lat0) * sin($diagonal / $R) * cos($azimutA));
 $latA = round(($latA_rad*180)/pi(),7);

 $longA_rad = $long0 +
   atan2(sin($azimutA) * sin($diagonal/$R) * cos($lat0),
         cos($diagonal/$R) - sin($lat0) * sin($latA));
 $longA = round(($longA_rad*180)/pi(),7);

 // Calcola latD, longD
 $latD_rad =
   asin(sin($lat0) * cos($diagonal / $R) +
        cos($lat0) * sin($diagonal / $R) * cos($azimutD));
 $latD = round(($latD_rad*180)/pi(),7);

 $longD_rad = $long0 +
   atan2(sin($azimutD) * sin($diagonal/$R ) * cos($lat0),
         cos($diagonal / $R) - sin($lat0) * sin($latD));
 $longD = round(($longD_rad*180)/pi(),7);

 $estremiIntervalliCoordinate = array(
        'latA' =>  $latA,
        'longA' => $longA,
        'latD' => $latD,
        'longD' => $longD
 );

 return $estremiIntervalliCoordinate;
}
```

```
function changeCommaWithPoint(){
 // modifica Elenco_Fermate
 $fileEF = fopen("elenco_fermate.csv", "r");
 $fileNewEF = fopen("elenco_fermatePunto.csv", "w");
 while (!feof($fileEF)) {
  $line_of_text = fgets($fileEF);
  //$members = explode('\n', $line_of_text);
  // replace , with . for decimal spaces
  $membersN = preg_replace( '/\,/', '.', $line_of_text);

  fwrite($fileNewEF, $membersN);
 }
 fclose($fileEF);

 // modifica Linea_sequenza_fermate
 $fileLSF = fopen("linee_sequenza_fermate.csv", "r");
 $fileNewLSF = fopen("linee_sequenza_fermatePunto.csv", "w");
 while (!feof($fileLSF)) {
  $line_of_text = fgets($fileLSF);
  //$members = explode('\n', $line_of_text);
  // replace , with . for decimal spaces
  $membersN = preg_replace( '/\,/', '.', $line_of_text);
  fwrite($fileNewLSF, $membersN);
 }
 fclose($fileLSF);

 // modifica Elenco_Archi
 $fileEA = fopen("elenco_archi.csv", "r");
 $fileNewEA = fopen("elenco_archiPunto.csv", "w");
 while (!feof($fileEA)) {
  $line_of_text = fgets($fileEA);
  //$members = explode('\n', $line_of_text);
  // replace , with . for decimal spaces
  $membersN = preg_replace( '/\,/', '.', $line_of_text);
  fwrite($fileNewEA, $membersN);
 }
 fclose($fileEA);
}
?>
```

HTTPPostFunctions.php

```
<?php

include './ParserFunctions.php';


// Effettua la traduzione da indirizzo postale a
// coordinate geografiche mediante query
// al servizio di geocoding

function doHttpPostRequest_Geocoding($positionString){
 $parameter =
  array('key' => 'Fmjtd|luub2dub20%2Crw%3Do5-9u2ldz',
        'callback' => 'renderGeocode',
        'inFormat' => 'kvp',
        'outFormat' => 'xml',
        'location' => $positionString);
 $url = 'http://open.mapquestapi.com/geocoding/v1/address';
 $options =
  array('http' => array('header' =>
  "Content-type: application/x-www-form-urlencoded\r\n",
  'method' => 'POST',
  'content' => http_build_query($parameter),),
  );
 $context  = stream_context_create($options);
 $xml = file_get_contents($url, false, $context);
 $nameFile = 'testFile1.xml';
 $byte = file_put_contents($nameFile, $xml);
 /* echo "numero byte scritti  ";
 echo $byte;
 echo "<br/>";*/
 //fa il parsing del file ed estrapola le coordinate del punto
 $coordinate = parserGeocoding($nameFile);
 echo "latitudine = ";
 echo $coordinate['lat'];
 echo "<br/>";
 echo "longitudine = ";
 echo $coordinate['long'];
 echo "<br/>";
 return $coordinate;
}
```

```
// Calcola le distanze reali per una persona a piedi
// dalla sua posizione alle diverse fermate

function doHttpPostRequest_RealDistance($lat0, $long0,
$coordinates){
 // creazione stringa xml che permette di
 // includere piu' richieste in una
 $xmlStringStart = "<route><locations>";
 $xmlStringLeft = "<location><latLng>";
 $xmlStringRight = "</latLng></location>";
 $xmlStringOptions = "<options><allToAll>true</allToAll>
 <unit>k</unit><routeType>pedestrian</routeType>
 <doReverseGeocode>false</doReverseGeocode>
 <ambiguities>ignore</ambiguities></options>";
 $xmlStringEnd = "</locations>".$xmlStringOptions."</route>";

 $lengCoordinates = sizeof($coordinates);
 echo "size coordinates = ".$lengCoordinates."</br>";
 $i=0;
 $distanceList = array();
 $xmlStringContent0 =
     $xmlStringLeft."<lat>".$lat0."</lat><lng>".
     $long0. "</lng>".$xmlStringRight;

 while($lengCoordinates>0){
  $xmlStringContent = "";
  /* echo "inizio WHILE </br>";
  echo "lengCoordinates = ".$lengCoordinates."</br>";
  echo "i = ".$i."</br>";*/

  if ($lengCoordinates>20){
   $max = $i+20;
   for ($i; $i<$max; $i++){
    $xmlStringContent=
    $xmlStringContent.$xmlStringLeft."<lat>".
    $coordinates[$i]['lat']."</lat><lng>".
    $coordinates[$i]['long']. "</lng>".$xmlStringRight;
   }

   $xmlString =
     $xmlStringStart.$xmlStringContent0.
     $xmlStringContent.$xmlStringEnd;

   file_put_contents('string.xml', $xmlString);
```

```
  $parameter =
   array('key' => 'Fmjtd|luub2dub20%2Crw%3Do5-9u2ldz',
         'inFormat' => 'xml',
         'outFormat' => 'xml',
         'xml' => $xmlString
        );
  $url='http://open.mapquestapi.com/directions/
        v1/routematrix';
  $options = array( 'http' => array('header'=>
    "Content-type:application/x-www-form-urlencoded\r\n",
   'method' =>'POST',
   'content' =>http_build_query($parameter),),);
  $context  = stream_context_create($options);
  $xml = file_get_contents($url, false, $context);
  $fileName = 'realDistance.xml';
  file_put_contents($fileName, $xml);

  $resList = parserDistance($fileName, 21);
  $distanceList = array_merge($distanceList,$resList);
  $lengCoordinates = $lengCoordinates - 20;
 }else{
  $max = $i+$lengCoordinates;
  /* echo "max = ".$max."</br>";
  echo "lengCoordinates = ".$lengCoordinates."</br>";*/
  for ($i; $i<$max; $i++){
   $xmlStringContent =
    $xmlStringContent.$xmlStringLeft."<lat>".
    $coordinates[$i]['lat']."</lat><lng>".
    $coordinates[$i]['long']. "</lng>".$xmlStringRight;
  }

  $xmlString=$xmlStringStart.$xmlStringContent0.
             $xmlStringContent.$xmlStringEnd;

  file_put_contents('string.xml', $xmlString);
  $parameter =
    array('key'=>'Fmjtd|luub2dub20%2Crw%3Do5-9u2ldz',
          'inFormat' => 'xml',
          'outFormat' => 'xml',
          'xml' => $xmlString
         );
  $url='http://open.mapquestapi.com/directions/v1/
        routematrix';
```

```
   $options = array('http' => array(
    'header'=>
    "Content-type:application/x-www-form-urlencoded\r\n",
    'method'=>'POST',
    'content'=>http_build_query($parameter),),);
   $context  = stream_context_create($options);
   $xml = file_get_contents($url, false, $context);
   $fileName = 'realDistance.xml';
   file_put_contents($fileName, $xml);
   $resList = parserDistance($fileName, $lengCoordinates+1);
   $distanceList = array_merge($distanceList,$resList);
   $lengCoordinates = 0;
  }
 }
 return $distanceList;
}

// effettua la richiesta al Web Service HelloBus
function doHttpPostRequest_HelloBusQuery($fermata,$linea){
 $parameter = array('fermata'=> $fermata,
                    'linea' => $linea ,
                    'oraHHMM' => '',
                    );
 $url='https://solweb.tper.it/tperit/webservices/
       hellobus.asmx/QueryHellobus';
 $options =
   array('http' => array('header' =>
   "Content-type:application/x-www-form-urlencoded\r\n",
   'method'  => 'POST',
   'content' => http_build_query($parameter),),
   );
 $context  = stream_context_create($options);
 $response = file_get_contents($url, false, $context);
 $responsePieces = explode("TperHellobus:", $response);
 /*echo "numero pezzi = ";
 echo  sizeof($responsePieces);
 echo "</br>";
 echo "answer = ".$responsePieces[1]." </br>";
 echo "answer = ".$responsePieces[0]." </br>";*/
 return $responsePieces[1];
}

?>
```

ParserFunctions.php

```
<?php
function parserGeocoding($file){
 $geocodingResponse=simplexml_load_file($file);

 // controllo inserimento indirizzo
 if(!is_array($geocodingResponse -> results)){
  $val=is_null($geocodingResponse -> results
       -> result -> locations -> location -> latLng);
  if ($val) {
    $coordinate = array(
        'lat' => 0,
        'long' => 0
    );
    return $coordinate;
  }
 }

 //latitudine

 $lat =
  is_array($geocodingResponse->results)?
  (is_array($geocodingResponse->results[0]->result->locations)
  ?
  $geocodingResponse->results[0]->result->locations[0]
  ->location->latLng->lat:
  $geocodingResponse->results[0]->result->locations->location
  ->latLng->lat):
  (is_array($geocodingResponse->results->result->locations)
  ?
  $geocodingResponse->results->result->locations[0]->location
  ->latLng->lat:$geocodingResponse->results->result->locations
  ->location->latLng->lat) ;

  // longitudine

  $long=is_array($geocodingResponse->results)?
   (is_array($geocodingResponse->results[0]->
    result->locations)
   ?
   $geocodingResponse->results[0]->result->locations[0]
   ->location->latLng->lng:$geocodingResponse->results[0]->
   result->locations->location->latLng->lng):
   (is_array($geocodingResponse->results->result->locations)?
```

```
   $geocodingResponse->results->result->locations[0]->
   location->latLng->lng:$geocodingResponse->results->
   result->locations->location->latLng->lng);
   $coordinate = array(
     'lat' => floatval($lat),
     'long' => floatval($long)
   );

 return $coordinate;
 }

 function parserDistance($file, $numberOfPoints){
  $distanceResponse = simplexml_load_file($file);
  $distancesList = array($numberOfPoints-1);
  for($i=1;$i<$numberOfPoints;$i++){
   $distance="distance".$i;
   $stringDistance=$distanceResponse->distance->$distance;
   $stringDistancePieces = explode(",",$stringDistance);
   $distanceFloat = floatval($stringDistancePieces[0]);
   // conversione in metri
   $distancesList[$i-1] = $distanceFloat * 1000;
  }
  return $distancesList;
 }
?>
```

SQLQueryFunctions.php

```
<?php

/* Query per creare tabella Fermate_Linea_Capolinea */
function queryCapolinea($connection){
/*** 1) Query_Archi_Fermata ***/
echo "</br>";
 $query_Archi_Fermata =  mysql_query(
 "CREATE VIEW Query_Archi_Fermata AS
  SELECT Elenco_Archi.codice_fermata,
         Elenco_Fermate.Denominazione,
         Elenco_Fermate.Ubicazione,
         Elenco_Fermate.Comune,
         Elenco_Fermate.Latitudine,
         Elenco_Fermate.Longitudine,
         Elenco_Fermate.Zona,
         Elenco_Archi.codice AS codice_arco
 FROM Elenco_Fermate
```

```
 INNER JOIN Elenco_Archi ON
 Elenco_Fermate.codice=Elenco_Archi.codice_fermata;");

 if($query_Archi_Fermata === TRUE){
  echo "Query_Archi_Fermata OK </br>";
 }else{
  echo "Errore Query_Archi_Fermata: ".
  mysql_error($connection)."</br>";
 }

 // Debug query
 $query_Archi_Fermata_Test = mysql_query(
 "SELECT COUNT(*) AS numero FROM Query_Archi_Fermata;");

 while($rows_Archi_Fermata_Test =
       mysql_fetch_array($query_Archi_Fermata_Test)){
  echo "numero righe Query_Archi_Fermata = ".
       $rows_Archi_Fermata_Test['numero']."</br>";
 }

/*** 2) Query_Archi_Fermata_Linea ***/
$query_Archi_Fermata_Linea = mysql_query(
"CREATE VIEW Query_Archi_Fermata_Linea AS
 SELECT Query_Archi_Fermata.codice_fermata,
        Query_Archi_Fermata.Denominazione,
        Query_Archi_Fermata.Ubicazione,
        Query_Archi_Fermata.Comune,
        Query_Archi_Fermata.Latitudine,
        Query_Archi_Fermata.Longitudine,
        Query_Archi_Fermata.Zona,
        Linee_Sequenza_Archi.codice_linea,
        Linee_Sequenza_Archi.verso,
        Linee_Sequenza_Archi.percorso,
        Linee_Sequenza_Archi.posizione,
        Linee_Sequenza_Archi.codice_arco
 FROM Linee_Sequenza_Archi
 INNER JOIN Query_Archi_Fermata ON
 Linee_Sequenza_Archi.codice_arco =
 Query_Archi_Fermata.codice_arco;");
 if($query_Archi_Fermata_Linea === TRUE){
  echo "Query_Archi_Fermata_Linea OK </br>";
 }else{
  echo "Errore Query_Archi_Fermata_Linea: ".
  mysql_error($connection)."</br>";
 }
```

```
 // Debug query
 $query_Archi_Fermata_Linea_Test = mysql_query(
 "SELECT COUNT(*) AS numero FROM Query_Archi_Fermata_Linea;");
 while ($rows_Archi_Fermata_Linea_Test =
        mysql_fetch_array($query_Archi_Fermata_Linea_Test)){
  echo "numero righe Query_Archi_Fermata_Linea = ".
  $rows_Archi_Fermata_Linea_Test['numero']."</br>";
 }
  echo "</br>";

/*** 3) Query_Linea_Percorso_UltimaPosizione ***/
$query_Linea_Percorso_UltimaPosizione = mysql_query(
"CREATE VIEW Query_Linea_Percorso_UltimaPosizione AS
 SELECT codice_linea, verso, percorso,
 MAX(posizione) AS ultima_posizione
 FROM Linee_Sequenza_Archi
 GROUP BY codice_linea, verso, percorso;");

if ($query_Linea_Percorso_UltimaPosizione === TRUE){
 echo "Query_Linea_Percorso_UltimaPosizione OK </br>";
}else{
 echo "Errore Query_Linea_Percorso_UltimaPosizione: ".
 mysql_error($connection)."</br>";
}
// Debug query
$query_Linea_Percorso_UltimaPosizione_Test = mysql_query(
"SELECT COUNT(*) AS numero
 FROM Query_Linea_Percorso_UltimaPosizione;");

while($rows_Linea_Percorso_UltimaPosizione_Test =
       mysql_fetch_array(
      $query_Linea_Percorso_UltimaPosizione_Test)){

 echo "numero righe Query_Linea_Percorso_UltimaPosizione = ".
 $rows_Linea_Percorso_UltimaPosizione_Test['numero']."</br>";
}
echo "</br>";

/*** 4) Query_Linea_Percorso_UltimoArco ***/
$query_Linea_Percorso_UltimoArco = mysql_query(
"CREATE VIEW Query_Linea_Percorso_UltimoArco AS
 SELECT Linee_Sequenza_Archi.codice_linea,
        Linee_Sequenza_Archi.verso,
        Linee_Sequenza_Archi.percorso,
```

```
        Query_Linea_Percorso_UltimaPosizione.ultima_posizione,
        Linee_Sequenza_Archi.codice_arco AS ultimo_arco
 FROM Linee_Sequenza_Archi
 INNER JOIN Query_Linea_Percorso_UltimaPosizione ON
 (Linee_Sequenza_Archi.posizione =
  Query_Linea_Percorso_UltimaPosizione.ultima_posizione) AND
 (Linee_Sequenza_Archi.verso =
  Query_Linea_Percorso_UltimaPosizione.verso) AND
 (Linee_Sequenza_Archi.percorso =
 Query_Linea_Percorso_UltimaPosizione.percorso) AND
 (Linee_Sequenza_Archi.codice_linea =
 Query_Linea_Percorso_UltimaPosizione.codice_linea)");

// Debug Query
if ($query_Linea_Percorso_UltimoArco === TRUE){
echo "Query_Linea_Percorso_UltimoArco OK </br>";
}else{
echo "Errore Query_Linea_Percorso_UltimoArco: ".
mysql_error($connection)."</br>";
}

// Debug query
$query_Linea_Percorso_UltimoArco_Test = mysql_query(
"SELECT COUNT(*) AS numero
 FROM Query_Linea_Percorso_UltimoArco;");
while ($rows_Linea_Percorso_UltimoArco_Test =
  mysql_fetch_array($query_Linea_Percorso_UltimoArco_Test)){
 echo "numero righe Query_Linea_Percorso_UltimoArco = ".
 $rows_Linea_Percorso_UltimoArco_Test['numero']."</br>";
}

/*** 5) Query_Fermata_Linea_Percorso_Verso_UltimoArco ***/
$query_Fermata_Linea_Percorso_Verso_UltimoArco = mysql_query(
"CREATE VIEW Query_Fermata_Linea_Percorso_Verso_UltimoArco AS
 SELECT DISTINCT Query_Archi_Fermata_Linea.codice_fermata,
                 Query_Archi_Fermata_Linea.Denominazione,
                 Query_Archi_Fermata_Linea.Ubicazione,
                 Query_Archi_Fermata_Linea.Comune,
                 Query_Archi_Fermata_Linea.Latitudine,
                 Query_Archi_Fermata_Linea.Longitudine,
                 Query_Archi_Fermata_Linea.Zona,
                 Query_Archi_Fermata_Linea.codice_linea,
                 Query_Archi_Fermata_Linea.verso,
                 Query_Archi_Fermata_Linea.percorso,
                 Query_Linea_Percorso_UltimoArco.ultimo_arco
```

```
 FROM Query_Linea_Percorso_UltimoArco
 INNER JOIN Query_Archi_Fermata_Linea ON
 (Query_Linea_Percorso_UltimoArco.percorso =
  Query_Archi_Fermata_Linea.percorso) AND
 (Query_Linea_Percorso_UltimoArco.verso =
  Query_Archi_Fermata_Linea.verso) AND
 (Query_Linea_Percorso_UltimoArco.codice_linea =
  Query_Archi_Fermata_Linea.codice_linea);");

// Debug Query
if ($query_Fermata_Linea_Percorso_Verso_UltimoArco === TRUE){
 echo "Query_Fermata_Linea_Percorso_Verso_UltimoArco OK </br>";
}else{
 echo "Errore Query_Fermata_Linea_Percorso_Verso_UltimoArco: ".
 mysql_error($connection)."</br>";
}

// Debug query
$query_Fermata_Linea_Percorso_Verso_UltimoArco_Test =
mysql_query("SELECT COUNT(*) AS numero
 FROM Query_Fermata_Linea_Percorso_Verso_UltimoArco;");

while ($rows_Fermata_Linea_Percorso_Verso_UltimoArco_Test =
 mysql_fetch_array(
   $query_Fermata_Linea_Percorso_Verso_UltimoArco_Test)){
 echo "numero righe
   Query_Fermata_Linea_Percorso_Verso_UltimoArco = ".
 $rows_Fermata_Linea_Percorso_Verso_UltimoArco_Test['numero'].
 "</br>";
}
echo "</br>";

/*** 6) Query_Archi_OffsetMax ***/

$query_Archi_OffsetMax = mysql_query(
"CREATE VIEW Query_Archi_OffsetMax AS
 SELECT codice, MAX(offset_posizione) AS offset_posizione_max
 FROM Elenco_Archi
 GROUP BY codice;");

// Debug Query
if ($query_Archi_OffsetMax === TRUE){
 echo "Query_Archi_OffsetMax OK </br>";
} else {
 echo "Errore Query_Archi_OffsetMax: ".
```

```
 mysql_error($connection)."</br>";
}

$query_Archi_OffsetMax_Test = mysql_query(
"SELECT COUNT(*) AS numero FROM Query_Archi_OffsetMax;");

while($rows_Archi_OffsetMax_Test =
  mysql_fetch_array($query_Archi_OffsetMax_Test)){
 echo "numero righe Query_Archi_OffsetMax = ".
 $rows_Archi_OffsetMax_Test['numero']."</br>";
}

echo "</br>";

/*** 7) Query_Archi_Offset_UltimaFermata ***/

$query_Archi_Offset_UltimaFermata = mysql_query(
"CREATE VIEW Query_Archi_Offset_UltimaFermata AS
 SELECT Elenco_Archi.codice AS
        codice_arco, Elenco_Archi.offset_posizione,
        Elenco_Archi.codice_fermata AS ultima_fermata
 FROM Elenco_Archi
 INNER JOIN Query_Archi_OffsetMax ON
 (Elenco_Archi.codice = Query_Archi_OffsetMax.codice) AND
 (Elenco_Archi.offset_posizione =
  Query_Archi_OffsetMax.offset_posizione_max); ");


// Debug Query
if ($query_Archi_Offset_UltimaFermata === TRUE){
 echo "Query_Archi_Offset_UltimaFermata OK </br>";
} else {
 echo "Errore Query_Archi_Offset_UltimaFermata: ".
 mysql_error($connection)."</br>";
}

$query_Archi_Offset_UltimaFermata_Test = mysql_query(
"SELECT COUNT(*) AS numero
 FROM Query_Archi_Offset_UltimaFermata;");

while($rows_Archi_Offset_UltimaFermata_Test =
 mysql_fetch_array($query_Archi_Offset_UltimaFermata_Test)){
 echo "numero righe Query_Archi_Offset_UltimaFermata = ".
 $rows_Archi_Offset_UltimaFermata_Test['numero']."</br>";
}
```

```
//8)Query_Fermata_Linea_Percorso_Verso_UltimoArcoUltimaFermata
$query_Fermata_Linea_Percorso_Verso_UltimoArcoUltimaFermata =
mysql_query( "CREATE VIEW
 Query_Fermata_Linea_Percorso_Verso_UltimoArcoUltimaFermata
 AS SELECT
  Query_Fermata_Linea_Percorso_Verso_UltimoArco.codice_fermata,
  Query_Fermata_Linea_Percorso_Verso_UltimoArco.Denominazione,
  Query_Fermata_Linea_Percorso_Verso_UltimoArco.Ubicazione,
  Query_Fermata_Linea_Percorso_Verso_UltimoArco.Comune,
  Query_Fermata_Linea_Percorso_Verso_UltimoArco.Latitudine,
  Query_Fermata_Linea_Percorso_Verso_UltimoArco.Longitudine,
  Query_Fermata_Linea_Percorso_Verso_UltimoArco.Zona,
  Query_Fermata_Linea_Percorso_Verso_UltimoArco.codice_linea,
  Query_Fermata_Linea_Percorso_Verso_UltimoArco.verso,
  Query_Fermata_Linea_Percorso_Verso_UltimoArco.percorso,
  Query_Fermata_Linea_Percorso_Verso_UltimoArco.ultimo_arco,
  Query_Archi_Offset_UltimaFermata.offset_posizione,
  Query_Archi_Offset_UltimaFermata.ultima_fermata
  AS codice_capolinea
 FROM
 Query_Fermata_Linea_Percorso_Verso_UltimoArco
 INNER JOIN Query_Archi_Offset_UltimaFermata ON
 Query_Fermata_Linea_Percorso_Verso_UltimoArco.ultimo_arco =
 Query_Archi_Offset_UltimaFermata.codice_arco;");

// Debug Query
if
($query_Fermata_Linea_Percorso_Verso_UltimoArcoUltimaFermata=
   ==TRUE){
 echo
  "Query_Fermata_Linea_Percorso_Verso_UltimoArcoUltimaFermata
   OK </br>";
}else{
 echo "Errore
  Query_Fermata_Linea_Percorso_Verso_UltimoArcoUltimaFermata:
  ".mysql_error($connection)."</br>";
}

$query_Fermata_Linea_Percorso_Verso_UltimoArcoUltimaFermata_
_Test= mysql_query(
"SELECT COUNT(*) AS numero
 FROM
 Query_Fermata_Linea_Percorso_Verso_UltimoArcoUltimaFermata;
");
```

```
while(
$rows_Fermata_Linea_Percorso_Verso_UltimoArcoUltimaFermata_
_Test=
mysql_fetch_array(
$query_Fermata_Linea_Percorso_Verso_UltimoArcoUltimaFermata_
_Test)){

 echo "numero righe
  Query_Fermata_Linea_Percorso_Verso_UltimoArcoUltimaFermata".
  $rows_Fermata_Linea_Percorso_Verso_UltimoArcoUltimaFermata_
  _Test['numero']."</br>";
}

/*** 9) Query_Fermata_Linea_Capolinea ***/
$query_Fermata_Linea_Capolinea= mysql_query(
"CREATE VIEW Query_Fermata_Linea_Capolinea AS
 SELECT codice_fermata, Denominazione, Ubicazione, Comune,
        Latitudine, Longitudine, Zona, codice_linea,
        verso, ultimo_arco, codice_capolinea
 FROM
   Query_Fermata_Linea_Percorso_Verso_UltimoArcoUltimaFermata
 GROUP BY codice_fermata, Denominazione, Ubicazione, Comune,
          Latitudine, Longitudine, Zona, codice_linea, verso,
          ultimo_arco, codice_capolinea;");

// Debug Query
if($query_Fermata_Linea_Capolinea === TRUE){
 echo "Query_Fermata_Linea_Capolinea OK </br>"
}else{
 echo "Errore Query_Fermata_Linea_Capolinea: ".
       mysql_error($connection)."</br>";
}

$query_Fermata_Linea_Capolinea_Test = mysql_query(
 "SELECT COUNT(*) AS numero
  FROM Query_Fermata_Linea_Capolinea;");

while($rows_Fermata_Linea_Capolinea_Test =
      mysql_fetch_array($query_Fermata_Linea_Capolinea_Test)){
  echo "numero righe Query_Fermata_Linea_Capolinea = ".
   $rows_Fermata_Linea_Capolinea_Test['numero']."</br>";
}
```

```
/*** 10) Query_Fermata_Linea_Capolinea_CodiceZona ***/
$query_Fermata_Linea_Capolinea_CodiceZona= mysql_query(
"CREATE VIEW Query_Fermata_Linea_Capolinea_CodiceZona AS
 SELECT Query_Fermata_Linea_Capolinea.codice_fermata,
        Query_Fermata_Linea_Capolinea.Denominazione,
        Query_Fermata_Linea_Capolinea.Ubicazione,
        Query_Fermata_Linea_Capolinea.Comune,
        Query_Fermata_Linea_Capolinea.Latitudine,
        Query_Fermata_Linea_Capolinea.Longitudine,
        Query_Fermata_Linea_Capolinea.Zona,
        Query_Fermata_Linea_Capolinea.codice_linea,
        Query_Fermata_Linea_Capolinea.verso,
        Query_Fermata_Linea_Capolinea.codice_capolinea,
        Elenco_Fermate.Denominazione AS
        denominazione_capolinea,
        Elenco_Fermate.Zona AS zona_capolinea
 FROM Query_Fermata_Linea_Capolinea
 INNER JOIN Elenco_Fermate
 ON Query_Fermata_Linea_Capolinea.codice_capolinea =
 Elenco_Fermate.codice;");

// Debug Query

if ($query_Fermata_Linea_Capolinea_CodiceZona === TRUE){
 echo "Query_Fermata_Linea_Capolinea_CodiceZona OK </br>";
} else {
 echo "Errore Query_Fermata_Linea_Capolinea_CodiceZona:".
      mysql_error($connection)."</br>";
}

$query_Fermata_Linea_Capolinea_CodiceZona_Test =
  mysql_query("SELECT COUNT(*) AS numero
   FROM Query_Fermata_Linea_Capolinea_CodiceZona;");

while($rows_Fermata_Linea_Capolinea_CodiceZona_Test =
   mysql_fetch_array(
   $query_Fermata_Linea_Capolinea_CodiceZona_Test)){
 echo
   "numero righe Query_Fermata_Linea_Capolinea_CodiceZona=".
   $rows_Fermata_Linea_Capolinea_CodiceZona_Test['numero'].
   "</br>";
}
```

```
/*** 11) Query_Fermata_Linea_ZonaCapolinea ***/
$query_Fermata_Linea_ZonaCapolinea = mysql_query(
"CREATE VIEW Query_Fermata_Linea_ZonaCapolinea AS
 SELECT
   Query_Fermata_Linea_Capolinea_CodiceZona.codice_fermata,
   Query_Fermata_Linea_Capolinea_CodiceZona.Denominazione,
   Query_Fermata_Linea_Capolinea_CodiceZona.Ubicazione,
   Query_Fermata_Linea_Capolinea_CodiceZona.Comune,
   Query_Fermata_Linea_Capolinea_CodiceZona.Latitudine,
   Query_Fermata_Linea_Capolinea_CodiceZona.Longitudine,
   Query_Fermata_Linea_Capolinea_CodiceZona.Zona,
   Query_Fermata_Linea_Capolinea_CodiceZona.codice_linea,
   Query_Fermata_Linea_Capolinea_CodiceZona.verso,
   Query_Fermata_Linea_Capolinea_CodiceZona.codice_capolinea,
   Query_Fermata_Linea_Capolinea_CodiceZona.
   denominazione_capolinea,
   Query_Fermata_Linea_Capolinea_CodiceZona.zona_capolinea,
   Elenco_Zone.descrizione
   FROM Query_Fermata_Linea_Capolinea_CodiceZona
   INNER JOIN Elenco_Zone
   ON
   Query_Fermata_Linea_Capolinea_CodiceZona.zona_capolinea=
      Elenco_Zone.codice;");

// Debug Query
if($query_Fermata_Linea_ZonaCapolinea===TRUE){
 echo "Query_Fermata_Linea_ZonaCapolinea OK </br>";
 $query_Fermata_Linea_ZonaCapolinea_Test1 =
    mysql_query("SELECT *
                 FROM Query_Fermata_Linea_ZonaCapolinea
                 ORDER BY codice_fermata DESC LIMIT 10;");

while($rows_Fermata_Linea_ZonaCapolinea_Test1 =
 mysql_fetch_array($query_Fermata_Linea_ZonaCapolinea_Test1)){
 echo
 $rows_Fermata_Linea_ZonaCapolinea_Test1['codice_fermata']."|";
 echo
 $rows_Fermata_Linea_ZonaCapolinea_Test1['Denominazione']."|";
 echo
 $rows_Fermata_Linea_ZonaCapolinea_Test1['Latitudine']."|";
 echo
 $rows_Fermata_Linea_ZonaCapolinea_Test1['Longitudine']."|";
 echo
 $rows_Fermata_Linea_ZonaCapolinea_Test1['codice_linea']."|";
```

```
 echo
 $rows_Fermata_Linea_ZonaCapolinea_Test1['codice_capolinea'].
 "|";
 echo $rows_Fermata_Linea_ZonaCapolinea_Test1['denominazione_
 _capolinea']."|";
 echo
 $rows_Fermata_Linea_ZonaCapolinea_Test1['zona_capolinea'].
 "</br> ";
}

}else{
  echo "Errore Query_Fermata_Linea_ZonaCapolinea: ".
    mysql_error($connection)."</br>";
}

$query_Fermata_Linea_ZonaCapolinea_Test =
   mysql_query("SELECT COUNT(*) AS numero
                FROM Query_Fermata_Linea_ZonaCapolinea;");

while($rows_Fermata_Linea_ZonaCapolinea_Test =
  mysql_fetch_array($query_Fermata_Linea_ZonaCapolinea_Test)){
  echo "numero righe Query_Fermata_Linea_ZonaCapolinea = ".
  $rows_Fermata_Linea_ZonaCapolinea_Test['numero']."</br>";
}

/*** 12) Query_Finale ***/
$tab_finale = mysql_query(
"CREATE TABLE Fermate_Linee_Capolinea(
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  codice_fermata VARCHAR(10),
  Denominazione VARCHAR(30),
  Ubicazione VARCHAR(30),
  Comune VARCHAR(30),
  Latitudine DOUBLE(8,6),
  Longitudine DOUBLE(8,6),
  Zona VARCHAR(10),
  codice_linea VARCHAR(10),
  verso VARCHAR(4),
  codice_capolinea VARCHAR(10),
  denominazione_capolinea VARCHAR(30),
  zona_capolinea VARCHAR(10))
 SELECT codice_fermata,Denominazione,Ubicazione,Comune,
       Latitudine,Longitudine,Zona,codice_linea,
       codice_capolinea,denominazione_capolinea,zona_capolinea
 FROM Query_Fermata_Linea_ZonaCapolinea;");
```

```
// debug query
if($tab_finale===TRUE){
 echo "Tabella Fermate_Linee_Capolinea creata </br>";
}else{
 echo "Errore creazione tabella Fermate_Linee_Capolinea: ".
      mysql_error($connection)."</br>";
}

// creazione indice latitudine
$queryFLC_IDX1 = mysql_query(
"CREATE INDEX latitudineIDX
 ON Fermate_Linee_Capolinea(Latitudine);");

if($queryFLC_IDX1===TRUE){
 echo "Indice latitudineIDX creato </br>";
}else{
 echo "Errore creazione indice latitudineIDX: ".
 mysql_error($connection)."</br>";
}
 echo "</br>";


// creazione indice longitudine
$queryFLC_IDX2 = mysql_query(
"CREATE INDEX longitudineIDX
 ON Fermate_Linee_Capolinea(Longitudine);");

if($queryFLC_IDX2===TRUE){
 echo "Indice longitudineIDX creato </br>";
}else{
 echo "Errore creazione indice longitudineIDX:".
 mysql_error($connection)."</br>";
}
echo "</br>";

$query_Finale_Test = mysql_query(
"SELECT COUNT(*) AS numero FROM Fermate_Linee_Capolinea;");

while($rows_Finale_Test =
      mysql_fetch_array($query_Finale_Test)){
 echo "numero righe Fermate_Linee_Capolinea = ".
 $rows_Finale_Test['numero']."</br>";
}
}
```

```
// Realizza i due step della selezione delle fermate
// all'interno dell'area di interesse
function querySelectionFermate($lat0, $long0, $latA, $longA,
                               $latD, $longD, $radius){


$connection =
  mysql_connect('localhost', 'root', 'root', false,128);
if (!$connection) {
 die('Not connected : ' . mysql_error());
} else {
 echo "connesso!!! </br>";
}
$db_selected = mysql_select_db('tper', $connection);
if($db_selected === TRUE){
 echo "db selezionato </br>";
}

mysql_query("DROP VIEW FermateInteresse;");
/******* First Selection *******/
$query_Fermate = mysql_query(
"CREATE VIEW FermateInteresse AS
 SELECT *
 FROM Fermate_Linee_Capolinea
 WHERE (Latitudine >= '$latA') AND
       (Latitudine <= '$latD') AND
       (Longitudine >= '$longA') AND
       (Longitudine <= '$longD') ;");
// DEBUG
if($query_Fermate===TRUE){
 echo "Query_Fermate OK </br>";
}else{
 echo "Errore Query_Fermate: ".
 mysql_error($connection)."</br>";
}

$query_Fermate_Test1 = mysql_query("SELECT COUNT(*) AS num
                                    FROM FermateInteresse;");
$queryFermateRowRes = mysql_fetch_array($query_Fermate_Test1);


$numRow = $queryFermateRowRes['num'];
echo "numElem = ".$numRow."</br>";
$coordinates = array((int)$numRow+1);
```

```
$query_Fermate_Test2 = mysql_query("SELECT *
                                    FROM FermateInteresse;");

echo "Inizio TABELLA FermateInteresse : </br>";
// DEBUG
while($row_query = mysql_fetch_array($query_Fermate_Test2)){
 echo $row_query['codice_fermata']." | ";
 echo $row_query['Denominazione']." | ";
 echo $row_query['Ubicazione']." | ";
 echo $row_query['Latitudine']." | ";
 echo $row_query['Longitudine']." | ";
 echo $row_query['codice_linea']." | ";
 echo $row_query['codice_capolinea']." | ";
 echo $row_query['denominazione_capolinea']." | ";
 echo $row_query['zona_capolinea']."</br> ";
}

echo "fine TABELLA FermateInteresse </br>";

$query_Fermate_Test3 =
  mysql_query("SELECT *
               FROM FermateInteresse
               GROUP BY codice_fermata;");

$i = 0;
while($row_query = mysql_fetch_array($query_Fermate_Test3)){
 $coordinates[$i] = array(
   'codiceFermata' => $row_query['codice_fermata'],
   'lat' => $row_query['Latitudine'],
   'long' => $row_query['Longitudine']
);
$i++;
}


/******** Second Selection ************/
$distanceList =
doHttpPostRequest_RealDistance($lat0,$long0,$coordinates);

/* echo "size distanceList = ".sizeof($distanceList)."</br>";
echo "radius = ".$radius."</br>";*/

$fermateSecondSelection = array();
```

```
for($i=0;$i<sizeof($coordinates);$i++){
 if($distanceList[$i]>$radius){
  $fermata = $coordinates[$i]['codiceFermata'];
  echo "".$fermata." eliminata </br>";
  if(!in_array($fermata, $fermateSecondSelection)){
   $fermateSecondSelection[]=$fermata;
  }
 }
}

$sizeFermateSecondSelection = sizeof($fermateSecondSelection);
// echo "size ".$sizeFermateSecondSelection."</br>";

mysql_query("DROP VIEW FermateFinalSelection;");

if($sizeFermateSecondSelection===0) {
 echo "fermate scartate = 0 <br>";
 $query_FermateSecondSelectionFinal =
  mysql_query("CREATE VIEW FermateFinalSelection AS
               SELECT *
               FROM FermateInteresse;" );

}else if($sizeFermateSecondSelection===1) {
  // un solo elemento nell'array
 //echo "fermateSecondSelection con 1 elemento </br>";
 $whereString = "codice_fermata <> ".
                $fermateSecondSelection[0];

 echo "whereString = ".$whereString."</br>";
 $query_FermateSecondSelectionFinal =
  mysql_query("CREATE VIEW FermateFinalSelection AS
               SELECT *
               FROM FermateInteresse
               WHERE ".$whereString.";");
}else{
 $whereString = "(";
 for($i=0;$i<sizeof($fermateSecondSelection);$i++) {
  if ($i=== sizeof($fermateSecondSelection)-1) {
   // ultimo elem array
   $whereString=$whereString.$fermateSecondSelection[$i].")";
  } else {
   $whereString=$whereString.$fermateSecondSelection[$i].",";
  }
}
```

```
echo "whereString = ".$whereString."</br>";
$query_FermateSecondSelectionFinal =
mysql_query("CREATE VIEW FermateFinalSelection AS
             SELECT *
             FROM FermateInteresse
             WHERE codice_fermata NOT IN ".$whereString.";");
}

// DEBUG
if($query_FermateSecondSelectionFinal === TRUE){
 echo "Query_FermateSecondSelectionFinal OK </br>";
}else{
 echo "Errore Query_FermateSecondSelectionFinal: ".
      mysql_error($connection)."</br>";
}

 mysql_close($connection);
}
?>
```

UpdateDatabase.php

```
<?php
error_reporting(-1);
ini_set('display_errors', true);
include './Utlities.php';
include './SqlQueryFunctions.php';
// modifica i separatori dei numeri decimali da , a .
changeCommaWithPoint();
// debug tempo esecuzione
//$startTime = microtime(true);
// connessione al database
$connection =
  mysql_connect('localhost', 'root', 'root', false,128);
if (!$connection) {
 die('Not connected : ' . mysql_error());
} else {
 echo "connesso!!! </br>";
}
// Creazione Database
createDatabase($connection);
// selezione database
$db_selected = mysql_select_db('tper', $connection);
```

```
if($db_selected === TRUE){
  echo "db selezionato </br>";
}
// creazione e popolamento tabelle
createTablesAndPopulate($connection);
// creazione tabella con capolinea
queryCapolinea($connection);
$close = mysql_close($connection);
if (!$close) {
 die('Disconnessione fallita : ' . mysql_error());
} else {
echo "Disconnesso!!! </br>";
}
/**** DEBUG TIME
$finishTime = microtime(true);
$deltaTime = $finishTime - $startTime;
echo "</br>";
echo "Tempo esecuzione = ".$deltaTime." ms </br>"; */

// Crea database e gli assegna i privilegi
function createDatabase($connection){
 $db_selected = mysql_select_db('tper', $connection);

 if ($db_selected) {
  // se il database esiste eliminalo
  echo "il database tper esiste </br>";
  mysql_query("drop database tper;");
  echo "database eliminato </br>";
 } else { // il database non esiste
  echo "il database tper non esiste </br>";
 }

  mysql_query("CREATE DATABASE tper;");
  echo "database creato </br>";
  mysql_query("c");
  mysql_query("GRANT FILE ON *.* TO root@localhost ");
  mysql_query("GRANT USAGE ON *.* TO root@localhos'
               WITH MAX_QUERIES_PER_HOUR 0,
               MAX_UPDATES_PER_HOUR 0,
               MAX_CONNECTIONS_PER_HOUR 0,
               MAX_USER_CONNECTIONS 0");
   echo "privilegi assegnati </br>";
}

/* Crea le tabelle e le popola prendendo i dati
```

```
 * dai file.csv, nei quali e' stato precedentemente
 * modificato il separatore dei numeri decimali da
 * virgola a punto.
 * Esempio 44,354647 -> 44.35464
 */
function createTablesAndPopulate($connection){

/**** tabella Elenco_Fermate */

// Creazione tabella
$queryEF = mysql_query(
"CREATE TABLE Elenco_Fermate(
                      codice VARCHAR(10),
                      Denominazione VARCHAR(30),
                      Ubicazione VARCHAR(30),
                      Comune VARCHAR(30),
                      Coordinata_x VARCHAR(10),
                      Coordinata_y VARCHAR(10),
                      Latitudine DOUBLE(8,6),
                      Longitudine DOUBLE(8,6),
                      Zona VARCHAR(10),
                     PRIMARY KEY(codice));");

if ($queryEF === TRUE){
 echo "Tabella Elenco_Fermate creata </br>";
}else{
 echo "Error creazione tabella Elenco_Fermate: ".
 mysql_error($connection)."</br>";
}

// Popolamento tabella
$queryEF2 = mysql_query(
 "LOAD DATA LOCAL INFILE 'elenco_fermatePunto.csv'
  INTO TABLE tper.Elenco_Fermate
  FIELDS TERMINATED BY ';'
  LINES TERMINATED BY '\r\n'
 IGNORE 1 LINES;");

if($queryEF2 === TRUE){
 echo "Tabella Elenco_Fermate popolata </br>";
 $infoEF2 = mysql_info();
 echo "info query = ".$infoEF2."</br>";
}else{
 echo "Errore popolamento tabella Elenco_Fermate:".
       mysql_error($connection)."</br>";
```

```
}

/**** tabella Linee_Sequenza_Archi ****/

// Creazione tabella
$queryLSA = mysql_query(
 "CREATE TABLE Linee_Sequenza_Archi(
     id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
     codice_linea VARCHAR(10),
     verso VARCHAR(4),
     percorso INT,
     posizione INT,
     codice_arco VARCHAR(8),
     bacino VARCHAR(4)
);");

if($queryLSA === TRUE){
 echo "Tabella Linee_Sequenza_Archi creata </br>";
}else{
 echo "Errore creazione tabella Linee_Sequenza_Archi:".
 mysql_error($connection)."</br>";
}
// creazione indice
$queryLSA_IDX1 = mysql_query(
"CREATE INDEX percorsoLineaIDX
 ON Linee_Sequenza_Archi(codice_linea,verso,percorso,
                         posizione);");

if ($queryLSA_IDX1 === TRUE){
 echo "Indice percorsoLineaIDX creato </br>";
}else{
 echo "Errore creazione indice percorsoLIneaIDX:".
      mysql_error($connection)."</br>";
}

// creazione indice
$queryLSA_IDX2 = mysql_query("CREATE INDEX codiceArcoIDX
                 ON Linee_Sequenza_Archi(codice_arco);");

if ($queryLSA_IDX2 === TRUE){
 echo "Indice codiceArcoIDX creato </br>";
}else{
 echo "Errore creazione indice codiceArcoIDX: ".
      mysql_error($connection)."</br>";
}
```

```
// Popolamento tabella
$queryLSA2 = mysql_query(
"LOAD DATA LOCAL INFILE 'linee_sequenza_archi.csv'
 INTO TABLE tper.Linee_Sequenza_Archi
 FIELDS TERMINATED BY ';'
 LINES TERMINATED BY '\r\n'
 IGNORE 1 LINES
 (codice_linea,verso,percorso,posizione,
  codice_arco,bacino);");

if($queryLSA2 === TRUE){
 echo "Tabella Linee_Sequenza_Archi popolata </br>";
 $infoLSA2 = mysql_info();
 echo "info query = ".$infoLSA2."</br>";
}else{
 echo "Errore popolamento tabella Linee_Sequenza_Archi:".
      mysql_error($connection)."</br>";
}

/**** tabella Linee_Sequenza_Fermate ****/

// Creazione tabella

$queryLSF = mysql_query(
"CREATE TABLE Linee_Sequenza_Fermate(
        id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
        codice_linea VARCHAR(10),
        codice_fermata VARCHAR(10),
        Denominazione VARCHAR(20),
        Ubicazione VARCHAR(20),
        Comune VARCHAR(20),
        Coordinata_x VARCHAR(10),
        Coordinata_y VARCHAR(10),
        Latitudine DOUBLE(8,6),
        Longitudine DOUBLE(8,6),
        Zona VARCHAR(5)
);");

if ($queryLSF === TRUE){
 echo "Tabella Linee_Sequenza_Fermate creata </br>";
}else{
 echo "Errore creazione tabella Linee_Sequenza_Fermate:".
 mysql_error($connection)."</br>";
}
```

```
// popolamento tabella

$queryLSF2 = mysql_query(
 "LOAD DATA LOCAL INFILE 'linee_sequenza_fermatePunto.csv'
  INTO TABLE tper.Linee_Sequenza_Fermate
  FIELDS TERMINATED BY ';'
  LINES TERMINATED BY '\r\n'
  IGNORE 1 LINES
  (codice_linea,codice_fermata,Denominazione,Ubicazione,
   Comune,Coordinata_x,Coordinata_y,Latitudine,
   Longitudine,Zona);");

if ($queryLSF2 === TRUE){
 echo "Tabella Linee_Sequenza_Fermate popolata </br>";
 $infoLSF2 = mysql_info();
 echo "info query = ".$infoLSF2."</br>";
}else{
 echo "Errore popolamento tabella Linee_Sequenza_Fermate:".
 mysql_error($connection)."</br>";
}

/**** tabella Elenco_Zone ****/

// Creazione tabella
$queryEZ = mysql_query("CREATE TABLE Elenco_Zone(
                        codice VARCHAR(10),
                        descrizione VARCHAR(30),
                        zona_tecnica VARCHAR(5),
                        codice_zona_riferimento VARCHAR(10),
                         PRIMARY KEY(codice)
);");

if ($queryEZ === TRUE){
 echo "Tabella Elenco_Zone creata </br>";
}else{
 echo "Error creating table: ".
 mysql_error($connection)."</br>";
}

// popolamento tabella
$queryEZ2 = mysql_query(
 "LOAD DATA LOCAL INFILE 'elenco_zone.csv'
  INTO TABLE tper.Elenco_Zone
  FIELDS TERMINATED BY ';'
```

```
  LINES TERMINATED BY '\r\n'
  IGNORE 1 LINES
  (codice,descrizione,zona_tecnica,
  codice_zona_riferimento);");

if ($queryEZ2 === TRUE){
 echo "Tabella Elenco_Zone popolata </br>";
 $infoEZ2 = mysql_info();
 echo "info query = ".$infoEZ2."</br>";
}else{
 echo "Errore popolamento tabella Elenco_Zone: ".
 mysql_error($connection)."</br>";
}

/**** tabella Elenco_Archi ****/

// Creazione tabella
$queryEA = mysql_query(
 "CREATE TABLE Elenco_Archi(
      id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
      codice VARCHAR(10),
      offset_posizione DOUBLE(10,4),
      Coordinata_x VARCHAR(10),
      Coordinata_y VARCHAR(10),
      Latitudine DOUBLE(8,6),
      Longitudine DOUBLE(8,6),
      codice_fermata VARCHAR(10),
      INDEX(codice)
);");

if ($queryEA === TRUE){
 echo "Tabella Elenco_Archi creata </br>";
}else{
 echo "Errore creazione tabella Elenco_Archi:".
 mysql_error($connection)."</br>";
}

// creazione indice
$queryEA_IDX = mysql_query(
"CREATE INDEX codiceFermataIDX
 ON Elenco_Archi(codice_fermata);");

if ($queryEA_IDX === TRUE){
 echo "Indice codiceFermataIDX creato </br>";
}else{
```

```
 echo "Errore creazione indice codiceFermataIDX:".
 mysql_error($connection)."</br>";
}

// popolamento tabella
$queryEA2 = mysql_query(
 "LOAD DATA LOCAL INFILE 'elenco_archiPunto.csv'
  INTO TABLE tper.Elenco_Archi
  FIELDS TERMINATED BY ';'
  LINES TERMINATED BY '\r\n'
  IGNORE 1 LINES
  (codice,offset_posizione,Coordinata_x,
   Coordinata_y,Latitudine,Longitudine,
   codice_fermata);");

if ($queryEA2 === TRUE){
 echo "Tabella Elenco_Archi popolata </br>";
 $infoEA2 = mysql_info();
 echo "info query = ".$infoEA2."</br>";
}else{
 echo "Errore popolamento tabella Elenco_Archi: ".
 mysql_error($connection)."</br>";
}

}

?>
```

Bibliografia

[1] http://www.whitehouse.gov/open/documents/open-government-directive.

[2] http://www.ted.com/talks/tim_berners_lee_on_the_next_web.html.

[3] http://opendefinition.org/.

[4] http://transparency.ge/en/ten-open-data-guidelines.

[5] http://husetsweb.dk/.

[6] http://wheredoesmymoneygo.org/.

[7] http://www.openstreetmap.org/.

[8] http://energy.publicdata.eu/ee/vis.html.

[9] http://epsiplatform.eu/content/european-psi-scoreboard.

[10] http://opendatacommons.org/.

[11] http://okfn.org/.

[12] http://creativecommons.org/.

[13] http://creativecommons.org/publicdomain/zero/1.0/.

[14] http://www.formez.it/iodl/.

[15] http://www.dati.gov.it/iodl/2.0/.

[16] http://folketsting.dk/.

[17] http://bikedistrict.org/.

[18] http://parlamento16.openpolis.it/.

[19] http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData.

[20] http://www.w3.org/TR/2004/REC-rdf-concepts-20040210.

[21] http://open.mapquestapi.com/geocoding/.

[22] http://open.mapquestapi.com/directions/.

[23] http://www.tper.it/hello-bus.

[24]

[25] Open Knowledge Foundation. *Open Data Handbook Documentation*, release 1.0.0 edition, 2012.