**Università degli studi di Salerno**

**Facoltà di Informatica**



**Dipartimento di Informatica**

**Corso di laurea magistrale in Informatica**

**Data science e machine learning**

**Studio di usabilità degli operatori iconici di CoDIT**

**Docenti Candidati**

Francesca Tassatone 0522500568

Roberta Gesumaria 0522500569

Daniele Lupo

0522500579

Prof. Giuseppe Polese

Dott.ssa Loredana Caruccio

Dott. Bernardo Breve

**Anno accademico 2018-2019**

Sommario

[Introduzione 3](#_Toc19804797)

[IBench 4](#_Toc19804798)

[Installazione 4](#_Toc19804799)

[Problemi riscontrati 4](#_Toc19804800)

[Utilizzo 4](#_Toc19804801)

[Problemi riscontrati 7](#_Toc19804802)

[CoDIT 8](#_Toc19804803)

[Installazione 8](#_Toc19804804)

[Problemi riscontrati 9](#_Toc19804805)

[Struttura XMI riconosciuta da CoDIT 10](#_Toc19804806)

[Sezione iniziale 10](#_Toc19804807)

[Sezione entità e attributi 10](#_Toc19804808)

[Sezione relazioni 11](#_Toc19804809)

[Sezione finale 11](#_Toc19804810)

[Coordinate elementi 12](#_Toc19804811)

[Connessione a MySQL e creazione database 13](#_Toc19804812)

[creaConnessione 13](#_Toc19804813)

[creaDb 14](#_Toc19804814)

[caricaDati 14](#_Toc19804815)

[Parser 15](#_Toc19804816)

[Sezione iniziale 15](#_Toc19804817)

[Sezione entità e attributi 15](#_Toc19804818)

[Sezione relazioni 16](#_Toc19804819)

[Sezione finale 16](#_Toc19804820)

[Statistiche di utilizzo 17](#_Toc19804821)

[Studio di usabilità 19](#_Toc19804822)

# Introduzione

Il progetto svolto riguarda lo studio di usabilità sull’espressività e la complessità percepita degli operatori iconici di CoDIT utilizzando scenari che coinvolgano schemi di grosse dimensioni. Per generare tali schemi ci è stato indicato di utilizzare il tool iBench.

Vediamo nel dettaglio il lavoro da svolgere.

Utilizzo di CoDIT, un tool per l’integrazione di schemi. Gli schemi su cui lavora sono schemi concettuali sui quali è possibile applicare operatori iconici per l’integrazione. I formati richiesti da CoDIT sono XMI o SVG.

Per lo studio richiesto, abbiamo però bisogno di schemi più grandi di quelli che si hanno a disposizione, per ottenere ciò è stato utilizzato IBench.

IBench genera schemi logici in formato XML basandosi su un file di configurazione.

Le prime differenze da notare sono le tipologie di file e di schemi differenti che si ottengono da un tool e che vengono elaborati dall’altro. Inoltre, i file XMI riconosciuti da CoDIT hanno una struttura non generica, quindi è stato necessario convertire lo schema, da logico a concettuale, e ottenere una strutturara adeguata in maniera da rendere il file comprensibile al tool.

Successivamente, è stato effettuato uno studio di usabilità sull’espressività e la complessità percepita degli operatori iconici di CoDIT, sia dai membri del gruppo che da altri utenti, sempre con competenze informatiche.

# IBench

iBench è un generatore di metadati per la creazione di mappature, schemi e vincoli di schemi arbitrariamente grandi e complessi. Può essere utilizzato con un generatore di dati per generare efficacemente scenari realistici di integrazione dei dati con diversi gradi di dimensione e complessità. Può essere utilizzato per creare benchmark per diverse attività di integrazione dei dati (virtuali), integrazione dei dati, scambio di dati, evoluzione degli schemi, mappatura degli operatori come composizione e inversione e corrispondenza degli schemi.

L'utente fornisce un file di configurazione che specifica quali tipi di metadati e dati generare (es. schemi, dati, vincoli, mappature, ...) e proprietà dello scenario generato. iBench produce un file XML che memorizza i metadati generati in base alla configurazione dell'utente. Se richiesto, genera anche i dati di istanza per gli schemi generati.

## Installazione

IBench è scritto in Java, per cui c’è bisogno di utilizzare ant per il building e ivy per scaricare le dipendenze.

La prima operazione svolta è stata quella di scaricare e installare ant.

Eseguendo il comando ant nella cartella principale, è stato scaricato ivy ed utilizzato automaticamente per scaricare il jar delle dipendenze. Successivamente, è stata generata una cartella build in cui sono presenti vari script di shell (.sh per Linux e Mac OS e .bat per Windows).

Il tutto è riportato in modo semplice nel file README messo a disposizione.

### Problemi riscontrati

Nell’utilizzo del file .bat per l’esecuzione di iBench è stata riscontrata la mancanza, nel codice, dell’indicazione dell’acquisizione dell’argomento dato da linea di comando. Successivamente all’aggiunta di tale sezione di codice, l’esecuzione è andata senza problemi.

Codice iBench.bat non funzionante:

*java -Xmx4096m -jar ibench-fat.jar*

Codice iBench.bat funzionante:

*java.exe -jar ibench-fat.jar -c %2*

Inoltre, il parametro -*Xmx4096m* limitava l’utilizzo della RAM a 4GB.

## Utilizzo

IBench permette di generare source schema e target schema.

Per poter generare questi schemi viene messo a disposizione un file di configurazione. Attraverso quest’ultimo è possibile settare le dimensioni e la struttura che si vuole ottenere per gli schemi risultanti.

I nomi dell’intera struttura e i dati vengono generati in modo casuale.

Vediamo nel dettaglio le varie sezioni del file di configurazione che abbiamo utilizzato per la generazione degli schemi.

La prima parte è relativa alla definizione della *path* in cui devono essere salvati gli schemi e i dati generati, poi viene indicato il nome del file che conterrà i metadati relativi alla struttura dello schema generato.



Successivamente c’è la parte per la configurazione sul numero di istanze, ossia entità per ogni schema, il numero di attributi per ogni entità, ed altre configurazioni relative ai dati generati.



Poiché i dati sono generati in modo casuale è possibile indicare il seme per il generatore casuale. Inoltre, per introdurre variabilità nella costruzione delle relazioni, è possibile indicare la deviazione relativa al numero di attributi per relazione.



La definizione delle dipendenze viene stabilita tramite percentuali. La percentuale delle dipendenze viene applicata al totale degli schemi generati, considerando sia source che target; quella delle chiavi esterne viene applicata sul numero di dipendenze generate.

Per esempio, utilizzando una configurazione con i seguenti parametri: 100 entità, 50% di dipendenze e 50% di chiavi esterne avremo 50 dipendenze con 25 chiavi esterne.



Per la generazione dei dati viene messa a disposizione una sezione in cui è possibile definire tipologie specifiche (dati UDT - User Definition Data Type), come e-mail, numero di telefono ed altri che possono essere scelti tra le classi che si trovano al seguente link <https://www.cs.toronto.edu/tox/toxgene/javadoc/allclasses-noframe.html>. In questa sezione bisogna indicare il numero di UDT che saranno definiti, poi per ognuno di essi il nome univoco, la classPath relativa alla classe della tipologia del dato che si vuole generare, la percentuale di generazione ed il tipo in cui verrà codificato il campo in un database SQL.



Inoltre, è possibile scegliere i dati, sempre in modo casuale, ma da un set fornito tramite file csv. Per configurare questa parte deve essere indicato il file csv, il nome della colonna all’interno del file, la percentuale di generazione ed infine il tipo in cui verrà convertito il campo in un database SQL.



Tramite i seguenti parametri è possibile gestire alcune differenze tra lo schema source e target.  
Utilizzandoli verrà utilizzato il partizionamento (orizzontale, verticale) o verranno aggiunti o rimossi alcuni attributi.



Infine, sono presenti due sezioni relative ad opzioni sui dati di output che verranno generati.





### Problemi riscontrati

I problemi riscontrati si possono dividere in base all’obiettivo della configurazione, cioè due:

* Generazione dati per lo schema target;
* Utilizzo di csv per la generazione dei dati.

#### Generazione dati per lo schema target

Avendo bisogno dei dati relativi agli schemi generati, abbiamo configurato iBench in modo da abilitare la creazione dei file csv relativi alla generazione di tali dati. Per la creazione dei dati relativi allo schema target abbiamo avuto dei problemi in caso di particolari configurazioni.



Figura 1: proprietà per l'abilitazione della generazione dei csv del target schema e il numero di tuple da creare.

Le configurazioni che hanno dato problemi sono state:

* Abilitazione delle proprietà relative alle percentuali sulle dipendenze e chiavi esterne;
* Partizionamento orizzontale e verticale.

Vediamo nel dettaglio gli errori per entrambi i casi.

##### Abilitazione delle proprietà relative alle percentuali sulle dipendenze e chiavi esterne

La generazione dei file csv è stata possibile solo se le proprietà sulle percentuali relative alle dipendenze e chiavi esterne non era abilitate. Infatti, nel caso di abilitazione di tali proprietà non venivano generati i dati relativi allo schema target restituendo il seguente errore:



Seguendo lo stack di eccezioni giungiamo al seguente snipet di codice:



Figura 2: Snipet di codice presente nella classe MappingScenario.

Sì, probabilmente è questo il problema!

##### Partizionamento orizzontale e verticale

Nel generare schemi con l’abilitazione delle proprietà per la partizione orizzontale o verticale i nomi delle entità che vengono generate nel target vengono creati con una sezione in maiuscolo, ad esempio “HP0FR0”. Con l’abilitazione delle proprietà per la generazione dei dati per il target restituisce il seguente errore:



Analizzando il codice, presumiamo che l’errore sia dovuto ad un’acquisizione con una forzatura al minuscolo del nome delle entità relative agli schemi.



#### Utilizzo di csv per la generazione dei dati

Nella sezione per UDT si possono specificare i file da cui prendere i valori per generare i dati.  
Provando con un solo file csv venivano generate righe contenenti più tuple. Inoltre, le percentuali delle chiavi esterne e delle dipendenze non erano coerenti alla configurazione specificata.

# CoDIT

CoDIT (Conceptual Data Integration Tool), supporta il processo di integrazione dei dati a livello concettuale, permette la costruzione di uno schema globale correlando sottoschemi delle sorgenti, il DBA potrà specificare in maniera visuale come combinare le sorgenti di dati. Lo schema globale potrà essere specificato allineando i sottoschemi delle sorgenti oppure specificando nuovi costrutti e mappando i sottoschemi delle sorgenti ad essi.

A livello metodologico CoDIT si basa su un linguaggio visuale, CoDIL (Conceptual Data Integration Language).

CoDIL permette di gestire l’eterogeneità degli schemi, per far fronte alla necessità di risolvere i conflitti dovuti all’utilizzo di diversi costrutti e/o nomi per modellare la stessa realtà.

Attraverso CoDIL il DBA potrà selezionare costrutti degli schemi sorgenti (uno per ogni schema) in corrispondenza ed associare un operatore iconico per specificare come risolvere il conflitto.

Il linguaggio CoDIL fornisce un insieme di operatori iconici che possono essere applicati a coppie di sottoschemi concettuali. Attraverso l’utilizzo di tali operatori è possibile specificare in che modo i sottoschemi selezionati sono correlati e scegliere come questi devono essere mappati nello schema riconciliato.

Le funzionalità offerte da CoDIT sono:

1. Disegna schema;
2. Caricamento dinamico;
3. Create target scheme.



Figura 3: Home tool CoDIT.

L’operazione sulla quale ci siamo focalizzati è il caricamento dinamico. I formati riconosciuti da CoDIT sono XMI e SVG. Il file XMI, però, ha una struttura specifica che riporteremo di seguito.

## Installazione

Per l’installazione del tool è stato utilizzato il file README messo a disposizione. Il file è molto dettagliato sia per l’istallazione su Mac OS che per Windows. Inoltre, la guida per il sistema Mac OS può essere eseguita anche per Windows.

### Problemi riscontrati

Nel file README viene menzionata una guida per modificare il file *web.xml* della configurazione di Catalina. Purtroppo, all’interno della guida sono presenti alcuni errori, come ad esempio dei tag di apertura o chiusura mancanti. Una volta risolti i vari errori, si ottiene il file *web.xml* allegato.

## Struttura XMI riconosciuta da CoDIT

La struttura del file XMI può essere suddivisa in quattro parti:

* Una sezione iniziale;
* Una sezione in cui vengono indicate tutte le entità con i relativi attributi;
* Una sezione in cui vengono indicate tutte le relazioni;
* Una sezione finale.

### Sezione iniziale

La seguente sezione risulta essere identica in tutti i file forniti come esempi per un primo utilizzo di CoDIT.



Figura 4: Struttura XML CoDIT, sezione iniziale.

### Sezione entità e attributi

Per indicare l’entità viene utilizzato il tag "packagedElement", con i seguenti attributi:

* xmi:type = uml:Class, per indicare che si tratta di un’entità;
* xmi:id, per indicare l’id;
* name, per indicare il nome dell’entità;
* type, per indicare il tipo di entità. Non sempre viene specificato, di solito è usato per indicare le entità deboli (WeakEntity).



Figura 5: Struttura XML CoDIT, entità.

Per ogni entità, come figlio dell’elemento "packagedElement", viene indicato l’attributo “xmi:Extension” per specificare che si tratta di un’entità.



Figura 6: Struttura XML CoDIT, estensione identificativa dell'entità.

Per indicare gli attributi relativi all’entità viene utilizzato il tag "ownedAttribute” come figlio dell’elemento "packagedElement".

Per ogni attributo vengono definiti:

* xmi:type=”uml: Property, per indicare che è un attributo;
* xmi:id, per indicare l’id;
* name, per indicare il nome dell’attributo;
* visibility, per indicare la visibilità dell’attributo.



Figura 7: Struttura XML CoDIT, attributo.

Nel caso in cui l’attributo sia chiave primaria, viene utilizzato il tag “xmi:Extension”, come figlio di “ownedAttribute”, per indicare che si tratta di una chiave primaria.



Figura 8: Struttura XML CoDIT, attibuto chiave primaria.

### Sezione relazioni

In questa sezione vengono definite le relazioni tra le entità sopra definite.

Viene usato sempre il tag "packagedElement", questa volta però è utilizzato con lo scopo di definire le associazioni, ciò è specificato nell’attributo “xmi: type= uml:Association”.

La relazione tra le entità è indicata attraverso l’attributo “xmi:id” in cui è specificato il nome delle due entità relazionate, usando il seguente formato E1\_\_E2\_\_id.

Inoltre, se si deve definire una specializzazione, oltre ad indicare un id in cui ci sono i nomi delle entità, viene inserito anche l’attributo “type=ISA”.



Figura 9: Struttura XML CoDIT, sezione relazioni tra le entità.

### Sezione finale

La seguente sezione risulta essere identica in tutti i file forniti come esempi per un primo utilizzo di CoDIT.



Figura 10: Struttura XML CoDIT, sezione finale.

### Coordinate elementi

Un’ulteriore funzionalità offerta da CoDIT è quella di salvare la posizione degli elementi, memorizzando le relative coordinate all’interno del file XMI. Quindi, per ogni elemento identificativo di una entità, attributo o relazione, sono presenti gli attributi x e y.



Figura 11: Struttura XML CoDIT, entità e attributi con coordinate.

# Connessione a MySQL e creazione database

Per alcune funzionalità messe a disposizione da CoDIT è necessario che, oltre ad essere in possesso di uno schema concettuale, ci sia anche un database popolato. Abbiamo, quindi, creato uno script, in Python, che permettesse di connettersi a MySQL e ci permettesse di creare e popolare nuovi database.



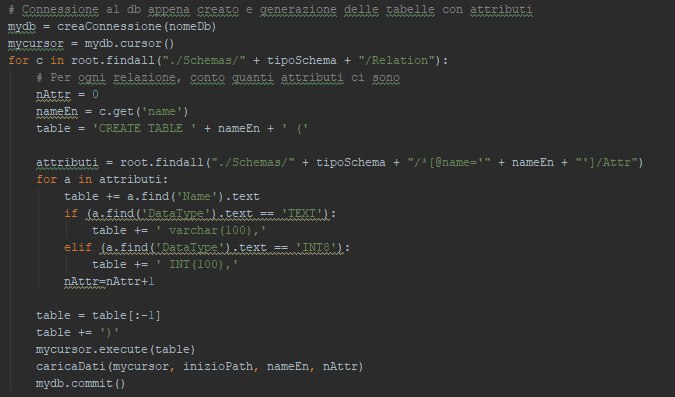
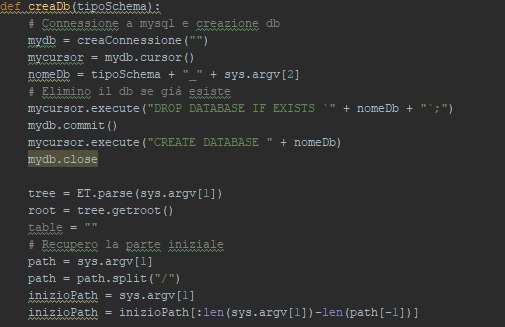
Di seguito verranno illustrati nel dettaglio i vari metodi implementati.

## creaConnessione



La libreria mysql.connector mette a disposizione un’oggetto dove vengono salvate tutte le informazioni necessarie per la connessione. Dato che serve una connessione per ogni database, è stato implementato un metodo che si occupa di configurare la connessione al database, specificato come parametro.

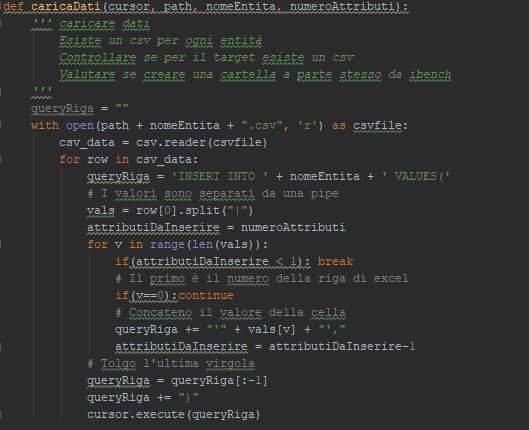
## creaDb



Questo metodo di occupa di leggere le informazioni dal file *xml* contenente la struttura degli schemi.  
Una volta eliminati i database se già esistenti, crea i nuovi database che conterranno le relazioni degli schemi *Source* e *Target*.

Leggendo la struttura delle varie relazioni, si occuperà di effettuare le *query* di creazione delle varie tabelle con i relativi attributi.

## caricaDati



Una volta creata la singola tabella, verranno caricati i dati inerenti letti dal csv denominato come la tabella.  
Leggendo il csv, verrà costruita la query di inserimento utilizzando i dati divisi per riga.  
Dato che iBench, con alcune configurazioni, genera più valori rispetto al numero di campi, è stato introdotto un controllo sul numero di valori utilizzati.

# Parser

Come specificato nel paragrafo introduttivo, iBench genera uno schema logico su file XML, mentre CoDIT lavora su schemi concettuali su file XMI con una struttura mostrata precedentemente. Per risolvere questo problema è stato creato un parser, che letto il file XML genera un nuovo file XMI con la struttura richiesta.

Per implementare il parser è stato utilizzato il linguaggio di programmazione Python, il quale offre una soluzione finalizzata proprio all’interfacciamento con i file XML. Infatti, grazie all’utilizzo del modulo xml.etree.ElementTree è stato possibile leggere il contenuto del file e creare una struttura per il nuovo file XMI da generare.

Di seguito mostreremo le varie parti del parser in relazione alla struttura del file XML mostrata nel capitolo “Struttura XMI riconosciuta da CoDIT”.

## Sezione iniziale



## Sezione entità e attributi





## Sezione relazioni



## Sezione finale



# Statistiche di utilizzo

Prima di svolgere lo studio di usabilità, sono state effettuate le seguenti fasi:

* Generazione schema iBench, fase in cui vengono generati gli schemi logici (source e target) in formato xml e i dati relativi, solo per lo schema source, attraverso le configurazioni suddette;
* Traduzione xml to xmi con parser, fase in cui viene tradotta la struttura dello schema logico generato da iBench in xml in schema concettuale compreso da CoDIT in xmi;
* Creazione database e caricamento dati su MySQL, fase in cui, basandoci sulla struttura dello schema generato da iBench, creiamo un database per ogni schema (sourceSchema e targetSchema) e lo popoliamo.

Per testare le tempistiche delle suddette fasi al fine di ottenere materiale atto allo studio di usabilità sono state utilizzate tre macchine con le seguenti specifiche:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PC** | **CPU** | **RAM** | **Tipo memoria** | **Sistema operativo** |
| 1 | I7-5500U 2.40GHz | 8GB | HDD | Windows 10 64bit |
| 2 | Rayzen 5 2500U 2GHz | 8GB | SSD | Windows 10 64bit |
| 3 | I5 2.7GHz | 8GB | SSD | Mac OS Mojave |

Tabella 1: Specifiche pc.

Con il tool iBench sono stati generati diversi schemi. Di seguito verranno descritte le caratteristiche di ognuno di essi. Le descrizioni sottostanti sono quelle relative agli schemi più grandi che siamo riusciti a generare con le macchine a disposizione. I seguenti schemi, successivamente alla conversione in xmi, non sono risultati adatti alla visualizzazione su CoDIT, poiché le dimensioni dei file risultavano superiori alla dimensione massima. Abbiamo, quindi, generato altri tre schemi di dimensioni minori.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** **schema** | #**Entità** | **#Attributi per entità[[1]](#footnote-1)** | **#Tuple** | **Percentuale dipendenze** | **Percentuale chiavi esterne** |
| SC0 | 200 | 5 | 100 | 100 | 50 |
| SC1 | 400 | 7 | 200 | 50 | 50 |
| SC2 | 470 | 9 | 150 | 80 | 50 |
| SC3 | 490 | 10 | 140 | 50 | 50 |

Tabella 2: Schemi che, convertiti in xmi, sono risultati di dimensioni superiori al massimo consentito.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** **schema** | #**Entità** | **#Attributi per entità1** | **#Tuple** | **Percentuale dipendenze** | **Percentuale chiavi esterne** |
| SC4 |  |  |  |  |  |
| SC5 |  |  |  |  |  |
| SC6 |  |  |  |  |  |

Tabella 3: Schemi utilizzati per lo studio di usabilità.

Per ogni macchina utilizzata sarà mostrata una tabella con i tempi di esecuzione per ogni fase. Per i primi quattro schemi sono stati riportati solo i tempi relativi alla generazione non essendo adatti alla visualizzazione su CoDIT. I tempi che sono stati indicati per ogni fase sono il risultato di una media su tre esecuzioni.

Tempi relativi alla generazione di schemi su iBench

|  |  |  |  |
| --- | --- | --- | --- |
| **ID schema** | **PC1** | **PC2** | **PC3** |
| SC0 | 37s | 24s | 57s |
| SC1 | 378s | 319s | 335s |
| SC2 | 546s | 468s | 490s |
| SC3 | 700s | err[[2]](#footnote-2) | 530s |

Tabella 4: Tempi relativi alla generazione di schemi su iBench.

Di seguito sono riportate tre tabelle, una per ogni macchina utilizzata, con i tempi ottenuti per le esecuzioni delle varie fasi.

PC1

|  |  |  |  |
| --- | --- | --- | --- |
| **ID schema** | **Generazione schema iBench** | **Traduzione xml to xmi con parser** | **Creazione database e caricamento dati su MySQL** |
| SC4 |  |  |  |
| SC5 |  |  |  |
| SC6 |  |  |  |

Tabella 5: Tempi relativi alle tre fasi ottenuti con il PC1.

PC2

|  |  |  |  |
| --- | --- | --- | --- |
| **ID schema** | **Generazione schema iBench** | **Traduzione xml to xmi con parser** | **Creazione database e caricamento dati su MySQL** |
| SC4 |  |  |  |
| SC5 |  |  |  |
| SC6 |  |  |  |

Tabella 6: Tempi relativi alle tre fasi ottenuti con il PC2.

PC3

|  |  |  |  |
| --- | --- | --- | --- |
| **ID schema** | **Generazione schema iBench** | **Traduzione xml to xmi con parser** | **Creazione database e caricamento dati su MySQL** |
| SC4 |  |  |  |
| SC5 |  |  |  |
| SC6 |  |  |  |

Tabella 7: Tempi relativi alle tre fasi ottenuti con il PC3.

# Studio di usabilità

1. Per introdurre variabilità nella costruzione delle relazioni, è stata introdotta la variabilità del numero di attributi. Durante i test è stata utilizzata una variabilità di 3 attributi. [↑](#footnote-ref-1)
2. La generazione dello schema SC3 non è stata possibile per il PC2 poiché durante la generazione dei file csv veniva saturata la RAM. [↑](#footnote-ref-2)