HOTMapper: Historical Open Data Table Mapper

Henrique Varella Ehrenfried
C3SL Labs, UFPR - Brazil
hvehrenfried@inf.ufpr.br

Rudolf Copi Eckelberg
C3SL Labs, UFPR - Brazil
rc16@inf.ufpr.br

Hamer Iboshi
C3SL Labs, UFPR - Brazil
hi15@inf.ufpr.br

Eduardo Todt
C3SL Labs, UFPR - Brazil
todt@inf.ufpr.br

Daniel Weingaertner
C3SL Labs, UFPR - Brazil
danielw@inf.ufpr.br

Marcos Didonet Del Fabro
C3SL Labs, UFPR - Brazil
marcos.ddf@inf.ufpr.br

ABSTRACT

We present HOTMapper, a tool that maps tables of Open Data with historical information into unified data sources. The tool couples data exchange and integration techniques implemented into two main components: 1) a CLI script with commands to create and update tables, and to insert and update the data, using 2) a simple mapping definition file, interpreted by the CLI script, to store the schema and data mappings throughout different years. The tool is implemented using Python and MonetDb. This demo will show the creation of the mapping definition and the execution flow of the CLI script for creating a unified data source from scratch and then updating an existing one. It will unify real world data sources, with millions of records, containing information about the Brazilian educational system.

1 INTRODUCTION

The availability of large Open Data sources raises several opportunities for researchers from different domains to extract and process the data. Governments from different countries are releasing vast amounts of governmental data, i.e. regarding public services and expenditures. However, in order to effectively use the data, some difficult problems must be addressed: open data sources are often heterogeneous, with different representation formats, data models, schemas (if any), data quality, and others. Such problems can be categorized into Data Exchange, Data Integration and Table Stitching issues. Data Exchange is the problem of transforming data from one source, which has a source schema, into a target schema [10]. Data Integration is defined as the problem of uniformly accessing different source schemes through an integrated source [10]. Finally, Table Stitching is defined as the problem of unifying many tables with identical schemes into a single table identifying extra attributes for it [7].

Open Data is often de-normalized (e.g. in large CSV (Comma Separated Values) files) and represents a small period of time (i.e. a semester or a year). New data is periodically released, with several files that need to be integrated. In addition, from one release to another, data and schema changes often occur. While each periodic instance could be handled separately, having a unified view of the data allows to do historical analysis and comparisons. Thus, it is important to develop methods and tools that are able to create a unified view of the data, to translate each periodic source into the unified view, to perform transformations on the data sources and to update them periodically. This means it is necessary to provide simple ways to maintain schema and data mappings from several input schemes and data sources into a unified one, keeping data compatibility.

There are different solutions/tools partially covering these issues. The Clio tool [5] is one of the best known, being developed before the advent of Open Data and a precursor to many others. More recently the *Polystore*-like approaches, such as *BigDawg*[3], ESTOCADA [1] or MISO[6] focus on the data integration issues, as does the Data Civilizer system[2]. *Ling at al.* [7] handled the problem of table stitching: new columns in the data source need to be identified and stitched into unified tables. The approach from [8] presents a brief history on data integration solutions and the current issues when having multiple Open Data sources. Their solution for creating union-able tables [9] could be valuable for a mapping framework, though they do not focus on maintenance and updates of the data. As a drawback, the solutions completeness and adaptability to several kinds of queries and data sources makes it hard to be used in specialized scenarios, such as the historical Open Data mapping presented in this paper.

The **HOTMapper tool** [1] (Historical Open Data Table Mapper) was developed as a domain specific data mapping/integration solution, targeted to de-normalized Open data sources, spread into several input files, where the mappings are simple and need to be stored and modified periodically. The implementation couples different aspects of data integration, data exchange and table stitching. It consists of a Command Line Interface (CLI) tool for historical data and schema mapping, translated from CSVs into the *MonetDb*[2] column store.

The tool receives as input the CSV files and a mapping definition file with information on how to unify the data. The mappings are defined in CSV as well, relating all desired input data with the corresponding target columns. The mapping definition file has information to guide the following actions: 1) creation of the target unified schema; 2) source-to-target data transformations, which are maintained for each given period; 3) creation of new derived data 4) update of an existing source with new columns and mappings; and 5) full reconstruction of the unified source based on the input sources and mappings. More actions could be added if needed.

This demonstration will present the execution flow of the tool:

- development of the periodical data and schema mappings;
- creation of a new data source;
- insertion of the corresponding data into the unified model;
- update with a new data source;
- update in the columns and data transformations.

We will execute the tool in a real world scenario, processing information about the Brazilian educational system. The data includes the enrollments and related information (schools, courses or teachers) in schools and Universities, with hundreds of columns. The mappings and commands can be modified on

[1] https://gitlab.c3sl.ufpr.br/tools/hotmapper
[2] https://www.monetdb.org/

the fly following the audience interactions, showing how fast and simple the tool is.

The unified data produced by the tool is currently being used and can be visualized in an open web portal [4] [3]. The amount of data already processed by the tool is summarized in Table 1. It was extracted from Open Data sources produced by INEP (Instituto Nacional de Pesquisa Educacionais Anísio Teixeira)[4].

Table 1: Summary of data from the INEP Open Data Source processed by HOTMapper

Year	Tables	Records
2017	13	102.176.661
2016	20	116.009.013
2015	18	116.946.948
2014	17	121.115.913
2013	18	112.645.020
2012	11	36.029.271
2011	7	12.025.035
2010	7	8.768.490

The paper is organized as follows: section 2 presents a motivating example of HOTMapper; section 3 explains HOTMapper's design and working principle and Section 4 describe the steps of the tool demonstration.

2 MOTIVATING EXAMPLE

Consider the necessity of extracting a metric (or indicator) regarding the number of enrolled students in all Brazilian schools from 2013 to 2017. The information about enrollments is available at an Open Data source produced by INEP. While this is a particular metric, the idea can be generalized as "extracting indicators from public Open Data sources". The input data is de-normalized in a set of large CSV files, with approx. 100 different fields each. The choice for publishing de-normalized data is common in open data sources, because it decreases the number of files that need to be managed in the long run.

The metric/indicator must be correctly extracted from this source and can be aggregated in different dimensions, i.e, location dimensions such as country, state or city; enrollments on public vs. private schools, enrollments on scientific courses, among others. The metric also needs to support yearly updates, because a new set of files is released every year, raising three main issues:

(1) **Integrated source creation**: a first set of tables needs to be created to be able to execute queries considering the historical data. In this case, the input data has 87 columns for the 2013 and 2014 years, and 96 columns in 2015-2017.
(2) **Schema evolution**: every year, the data sources definition changes, so it is necessary to provide schema mappings:
   (a) direct mappings:
   ```
   NATIONALITY <- [2013-2017] NATIONALITY
   SPECIAL_NECESSITY <- [2013-2014] HAS_NECESSITY
   SPECIAL_NECESSITY <- [2015-2017] SPECIAL_NECESSITY
   ```
   (b) new mapping (when a column is added):
   ```
   REGION <- [2013-2015] not-available
   REGION <- [2016-2017] REGION
   ```
(3) **Data evolution**: data has to be transformed and kept compatible along all years, requiring instance mappings:
   (a) mapping creation: a new dimension is generated from existing data
   ```
   PROFESSIONAL_STUDIES<-[2013-2014]
   WHEN STUDIES_KIND between 30 and 40 THEN 1
   WHEN STUDIES_KIND between 41 and 50 THEN 2
   PROFESSIONAL_STUDIES<-PROFESSIONAL_STUDIES[2015-2017]
   ```
   (b) mapping update: the data mapping from the data sources may change (dimension or metric), which means the previous mappings, for all previous years, has to be updated
   ```
   GENDER <- [2013-2014]
             WHEN M THEN 1
             WHEN F THEN 2
   GENDER <- [2015-2017] GENDER
   ```

This motivating example considered just one indicator and illustrative examples, with intersections of data exchange, integration and table stitching. This kind of mapping is commonly applied to more or less 90 columns of annual data. In addition, INEP publishes at least four main raw data sets: Students, Teachers, Schools and Sessions, containing more than one hundred columns. There are other sources released on a similar basis, replicating this scenario for every new indicator produced, presenting a challenging setting for the extraction and maintenance of Open Data sources.

3 HOTMAPPER

In this section we present the HOTMapper tool, addressing the maintenance of data and mappings of Open Data sets.

3.1 Tool description

HOTMapper's architecture is shown in Figure 1. It consists of a CLI (Command Line Interface) manager interacting with three kinds of input and/or output, on which six predefined actions can be performed. With a flexible implementation using Python and SQL programming languages, the tool allows for an easy extension by adding more actions.

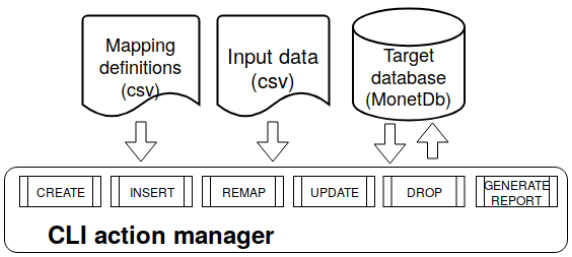

Figure 1: HOTMapper overview

The **mapping definition** files are in CSV format. There is one file for each table in the target RDBMS. It describes the columns that are created and the mappings, per year, of each input column.

The **input data** is the set of CSV files that are included in the target database, one file per metric (or set of metrics), and per year. For instance, the *enrollments* metric has five input files, from 2013 to 2017. The files are de-normalized, which means they have a high number of columns.

The **target database** has the set of tables that contain the integrated information produced by the tool. We use the *MonetDb* column store and keep the included data as similar as possible to the input data, i.e., we do not normalize the target data. This has some advantages: the definition and maintenance of mappings remains relatively simple; the data model does not need to be

[3] http://dadoseducacionais.c3sl.ufpr.br
[4] http://portal.inep.gov.br/microdados

constantly adapted for every new release; the queries are fast, because joins are minimized; and bulk insertion operations are fast. It is important to emphasize that the target database is only updated by new releases of the input data, without OLTP operations.

The actions currently handled by HOTMapper are: **table maintenance** actions (1) `CREATE` and (2) `DROP`; **data and table manipulation** actions (3) `INSERT`, (4) `REMAP` and (5) `UPDATE`; and a **reporting** action (6) `GENERATE REPORT`:

(1) `CREATE` action takes as input the table definition and executes the DML (Data Manipulation Language) commands;
(2) `DROP` action deletes the table passed as parameter and any related data. The tool executes standard DML commands, so that it can be used in different RDBMS's;
(3) `INSERT` action executes a bulk insert of the input CSV files into a temporary table in the target database. Then, it reads the mapping definition to transfer the data into the target table. The creation of a temporary table is important to facilitate the manipulation of the input data, avoiding direct operations on the CSV file;
(4) `REMAP` action modifies the initial table definition;
(5) `UPDATE` action updates a table after a change in the mappings; and
(6) `GENERATE REPORT` action produces a report with column equivalences between the input table and the current database. It is not necessary for inserting or updating data, but eases the creation of the mapping definition file.

Figure 2 illustrates a mapping definition and two input CSV files, with information about the universities in Brazil.

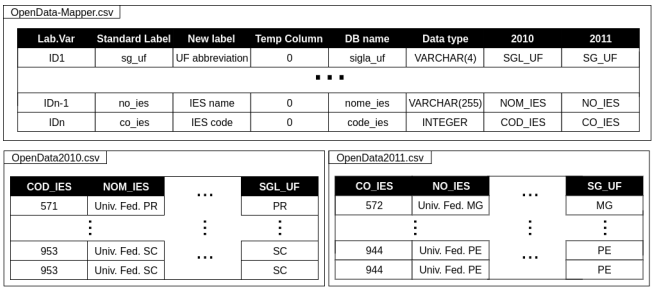

Figure 2: Illustration of a mapping definition and two input CSV data files

File `OpenData-Mapper.csv` contains the mapping definition, composed of eight columns: `[Lab.Var]` is an identifier for each mapping; `[Standard Label]` has the name of the mapped columns; `[New Label]` has the description of the column; `[Temp Column]` defines whether the column is temporary (used when pre-calculations are necessary); `[DB name]` is the name of the column in the target database; `[Data type]` has the data type of the column defined by `[DB name]`; and columns `[2010],[2011]` have the yearly mappings of each input column into the corresponding target column. Every time a new release is available, a new year column is added, with its corresponding mappings. The periodicity can be different, depending of the characteristics of the input data.

HOTMapper also supports source-to-target mappings. For a given line in the CSV file, the `[DB name]` column points to the target column in the database. Then, for each year, it is possible to define two kinds of mappings: 1) simple equivalence mappings, by indicating the name of the source column; 2) data transformation mappings, defined by `CASE` statements in SQL. The `CASE` statements are injected in the code responsible for the transformation. The context of each statement is the current cursor of the SQL execution.

Consider the first equivalence mapping from the motivating example, i.e., for `NATIONALITY`. Five columns, from 2013 to 2017, have the value of the input column `NATIONALITY`. The same is valid for `SPECIAL_NECESSITY`, though it will have different names depending on the year. Non-existent mappings are just left blank. All other data transformations are written using `CASE`, one per year, independently from each other.

Considering the `PROFESSIONAL_STUDIES` column, following code is written to process years 2013 and 2014:

```
CASE
  WHEN (STUDIES_KIND >= 30 AND STUDIES_KIND <= 40) THEN 1
  WHEN (STUDIES_KIND > 40 AND STUDIES_KIND <= 50) THEN 2
END
```

Simple join operations are also supported. For instance, the following expression (`~ SCHOOL.REGION_ID`) written in the mappings of the `ENROLLMENTS` table causes a join with the `SCHOOL` table and returns the `REGION_ID` column.

The mapping definition is kept as simple as possible, yet rich enough to express data translations. The choice of a sub-set of SQL enables fast SELECT + INSERT executions, which would demand handcrafted loops over millions of records with an imperative programming language. The complexity of the translation depends on the `CASE` expression written. They support 1-to-1 and N:1 mappings, and simple joins with other tables. This choice for simplicity is crucial for the continuous maintenance of the mappings.

4 HOTMAPPER DEMONSTRATION

The demonstration will show how to create a mapping definition file, how to use it to create and update tables and its corresponding data. The process will be interactive, so the mapping can be modified if necessary and the tool can be re-executed, to show its efficiency and ease of use.

4.1 Requirements

HOTMapper has following software requirements:

- A Python environment
- The MonetDB[5] database
- The Open Data files in CSV[6]
- A connection configuration file[6]
- The HOTMapper code[6]

After the installation, it is necessary to set up the configurations in the file `settings.py`. It contains the database configurations (login, host, database name) and the path to the CSV files. The utilization of the CLI interface is straightforward:

```
./manage.py create|insert|remap|drop|remap|generate_report
            INPUT_TABLE_NAME PARAMETERS
```

All the options process a mapping definition file with the same name of the input table name, except for the period indication. The development of this file is a central part of the tool usage.

4.2 Using the HOTMapper

This demonstration will show two scenarios, illustrating two workflows: 1) the creation and insertion of an open data source from scratch, and 2) the update of mappings and data of an existing table. In both scenarios the audience will be able to

[5]https://www.monetdb.org/
[6]https://gitlab.c3sl.ufpr.br/tools/hotmapper

propose modifications in the mapping definitions and check the results on the fly.

In the **first scenario**, we will map a table containing information about the undergraduate institutions in Brazil, from 2010 to 2016 (`localoferta_ens_superior`). It is a relatively small table, with approximately 200K records per year, and for its creation the mapping definition file must contain the necessary columns. We execute the command:

```
./manage.py create localoferta_ens_superior
```

In addition, HOTMapper creates an auxiliary table to store the mapping definitions, called `mapping_localoferta_ens_superior`. These mappings can be used for any subsequent modifications, enabling a faster processing and generation of SQL commands. A final commit is done after the tables are created. Then, we execute the command below to insert the data into the target table.

```
./manage.py insert /FILEPATH/DM_LOCAL_OFERTA_2010.CSV
            localoferta_ens_superior 2010 --sep="|"
```

The process is repeated for data files from each year, i.e., from 2010 to 2016, using the corresponding mapping definitions for each year. An excerpt of the mapping is shown in the three items below: (1) the labels of the mapping file; (2) a complete 1-to-1 mapping with the institution code; (3) a mapping with the institution start date. In this case, we do not have data for every year, leaving the corresponding column blank.

(1) `Label,Std.Label,New Label,DB Name,Type,`
`2010,2011,2012,2013,2014,2015,2016`

(2) `LOCAL-OFERTA,CO_IES,Institution code,cod_ies,INTEGER,`
`CO_IES,CO_IES,CO_IES,CO_IES,CO_IES,CO_IES,CO_IES`

(3) `LOCAL-OFERTA,DT_INICIO_FUNCIONAMENTO,Start date,`
`data_incio_funcionamento,VARCHAR(255),,,,`
`DT_INICIO_FUNCIONAMENTO,DT_INICIO_FUNCIONAMENTO,`
`DT_INICIO_FUNCIONAMENTO,DT_INICIO_FUNCIONAMENTO`

```
(env) c3sl@ironman HOTMapper (master)$ ./manage.py insert /home/c3sl/HOTMapper/open_data/DM_LOCAL_OFERTA_2010.CSV localoferta_ens_superior 2010 --sep="|"
INFO - sqlalchemy.engine.base.Engine: {'table_id': 8118}
INFO - database.database_table: Acquiring mapping table for localoferta_ens_superior
INFO - sqlalchemy.engine.base.Engine: BEGIN (implicit)
INFO - database.database_table: Acquiring temporary table with name 't_localoferta_ens_superior'
INFO - sqlalchemy.engine.base.Engine:
CREATE TEMPORARY TABLE tmp.t_localoferta_ens_superior (
        "CO_LOCAL_OFERTA_IES" INTEGER NOT NULL,
        "CO_IES" INTEGER,
        "CO_MUNICIPIO_LOCAL_OFERTA" INTEGER,
        "NO_MUNICIPIO_LOCAL_OFERTA" VARCHAR(255),
        "CO_UF_LOCAL_OFERTA" INTEGER,
        "SGL_UF_LOCAL_OFERTA" VARCHAR(2),
        "IN_SEDE" INTEGER,
        "CO_CURSO_POLO" INTEGER,
        "CO_CURSO" INTEGER NOT NULL,
        "ANO_CENSO" VARCHAR(255) NOT NULL,
        PRIMARY KEY ("ANO_CENSO", "CO_LOCAL_OFERTA_IES", "CO_CURSO")
)


INFO - sqlalchemy.engine.base.Engine: {}
INFO - database.database_table: Copying /home/c3sl/HOTMapper/open_data/DM_LOCAL_OFERTA_2010.CSV into temporary table t_localoferta_ens_superior
INFO - sqlalchemy.engine.base.Engine: COPY OFFSET 2 INTO "t_localoferta_ens_superior" FROM '/home/c3sl/HOTMapper/open_data/DM_LOCAL_OFERTA_2010.CSV' USING DELIMITERS '|','\n' NULL AS ''
INFO - sqlalchemy.engine.base.Engine: {}
INFO - database.database_table: Inserting data into localoferta_ens_superior from temporary table t_localoferta_ens_superior
INFO - sqlalchemy.engine.base.Engine: INSERT INTO localoferta_ens_superior (cod_local_oferta, cod_ies, cod_municipio, nome_municipio, cod_uf, sigla_uf, sede, cod_curso_polo, cod_curso, ano_censo) SELECT t_localoferta_ens_superior."CO_LOCAL_OFERTA_IES", t_localoferta_ens_superior."CO_IES", t_localoferta_ens_superior."CO_MUNICIPIO_LOCAL_OFERTA", t_localoferta_ens_superior."NO_MUNICIPIO_LOCAL_OFERTA", t_localoferta_ens_superior."CO_UF_LOCAL_OFERTA", t_localoferta_ens_superior."SGL_UF_LOCAL_OFERTA", t_localoferta_ens_superior."IN_SEDE", t_localoferta_ens_superior."CO_CURSO_POLO", t_localoferta_ens_superior."CO_CURSO", t_localoferta_ens_superior."ANO_CENSO"
FROM t_localoferta_ens_superior
INFO - sqlalchemy.engine.base.Engine: {}
INFO - sqlalchemy.engine.base.Engine: COMMIT
(env) c3sl@ironman HOTMapper (master)$
```

Figure 3: Screenshot of the insertion execution flow

The screen shot of the execution log is shown in Figure 3. The command first inserts the table into a temporary table, with the same structure as the input CSV, in a bulk insert action. Then, it inserts the data into a final table applying the mapping definitions. If this insertion is successful, the tool commits all changes to the MonetDb database. Once a first insertion is done, we will execute the drop command (`./manage.py drop localoferta_ens_superior`), and the audience can propose alterations to the mapping definition file to see different outcomes.

In the **second scenario**, we start from an already existing table, with more complex mappings and records. We use the table from the motivating example in Section 2, with the enrollments of all students from 2013 to 2017. As already stated, it has about 90 columns and millions of records per year. We execute the *remap* and *update* actions for 2013 as follows:

```
./manage.py remap matricula
./manage.py update_from_file /FILEPATH/MATRICULA_2013.csv
   matricula 2013 --columns="profissionalizante" --sep="|"
```

The actions check if there is a difference between the current table specification (`matricula`) and the new mapping definition provided. It updates the table and the data. The mappings have, in addition to simple definitions as the ones from the *scenario 1*, expressions requiring data conversions using `CASE` statements. We will edit the `PROFESSIONAL_STUDIES` mapping, whose excerpt is shown below:

```
IN_PROFISSIONALIZANTE,Educação Profissional,0,
                                        profissionalizante,BOOLEAN,
~CASE
WHEN ("FK_COD_MOD_ENSINO"=1 OR "FK_COD_MOD_ENSINO"=2
OR "FK_COD_MOD_ENSINO"=3) THEN CASE WHEN null THEN null
WHEN ("FK_COD_ETAPA_ENSINO">=30 AND "FK_COD_ETAPA\_ENSINO"<=40)
OR ("FK_COD_ETAPA_ENSINO">=59 AND "FK_COD_ETAPA_ENSINO"<=65)
OR ("FK_COD_ETAPA_ENSINO">=67 AND "FK_COD_ETAPA_ENSINO"<=68)
OR ("FK_COD_ETAPA_ENSINO">=73 AND "FK_COD_ETAPA_ENSINO"<=74)
OR "FK_COD_ETAPA_ENSINO"=57
THEN 1 ELSE 0 END
END
```

The last column is the mapping for year 2013, also repeated for 2014. The remaining years already have the desired information. During the demonstration, this and other mappings can be changed following audience interactions.

To summarize, the HOTMapper demo will show how to write, store and manage mappings and data, from Open Data sources with historical information into an integrated database. The simple definition format and rapid bulk insertions enables efficient management of several Open Data sources over time.

Acknowledgments: this work was partially funded by CAPES, Sistema de Monitoramento de Políticas de Promoção da Igualdade Racial (SNPPIR), Simulador de Custo-Aluno-Qualidade (SASE/MEC).

REFERENCES

[1] Francesca Bugiotti, Damian Bursztyn, Alin Deutsch, Ioana Ileana, and Ioana Manolescu. 2015. Invisible glue: scalable self-tuning multi-stores. In *Conference on Innovative Data Systems Research (CIDR)*.

[2] Dong Deng, Raul Castro Fernandez, Ziawasch Abedjan, Sibo Wang, Michael Stonebraker, Ahmed K Elmagarmid, Ihab F Ilyas, Samuel Madden, Mourad Ouzzani, and Nan Tang. 2017. The Data Civilizer System.. In *CIDR*.

[3] Jennie Duggan, Aaron J Elmore, Michael Stonebraker, Magda Balazinska, Bill Howe, Jeremy Kepner, S. Madden, David Maier, Tim Mattson, and Stan Zdonik. 2015. The bigdawg polystore system. *ACM Sigmod Record* 44, 2 (2015), 11–16.

[4] Rudolf Eckelberg, Vytor Bezerra Calixto, Marina A. Hoshiba Pimentel, Marcos Didonet Del Fabro, Marcos Sfair Sunyé, Leticia Mara Peres, Eduardo Todt, Thiago Alves, Adriana Dragone, and Gabriela Schneider. 2018. Educational Open Government Data: From Requirements to End Users. In *Web Engineering - 18th ICWE, Cáceres, Spain, June 5-8, 2018.* 463–470.

[5] Ronald Fagin, Laura M. Haas, Mauricio Hernández, Renée J. Miller, Lucian Popa, and Yannis Velegrakis. 2009. *Clio: Schema Mapping Creation and Data Exchange.* Springer Berlin Heidelberg, Berlin, Heidelberg, 198–236.

[6] Jeff LeFevre, Jagan Sankaranarayanan, Hakan Hacigumus, Junichi Tatemura, Neoklis Polyzotis, and Michael J Carey. 2014. MISO: souping up big data query processing with a multistore system. In *Proc. of the 2014 SIGMOD*. 1591–1602.

[7] Xiao Ling, Alon Halevy, Fei Wu, and Cong Yu. 2013. Synthesizing union tables from the Web. In *IJCAI*.

[8] Renée J. Miller. 2018. Open Data Integration. *Proc. VLDB Endow.* 11, 12 (Aug. 2018), 2130–2139. https://doi.org/10.14778/3229863.3240491

[9] Fatemeh Nargesian, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. 2018. Table union search on open data. *Proceedings of the VLDB Endowment* 11, 7 (2018), 813–825. https://doi.org/10.14778/3192965.3192973

[10] Cong Yu and Lucian Popa. 2004. Constraint-based XML query rewriting for data integration. *Proceedings of the 2004 ACM SIGMOD international conference on Management of data - SIGMOD '04* (2004), 371. https://doi.org/10.1145/1007568.1007611