Open Data Analytic Querying using a Relation-free API

Lucas F. de Oliveira `lfoliveira@inf.ufpr.br`,
Alessandro Elias, Fabiola Santore, Diego Pasqualin, Luis C. E. Bona, Marcos Sunye and Marcos Didonet Del Fabro

23 de abril de 2020

Centro de Computação Científica e Software Livre - C3SL

Table of contents

Our Research Group

Our Research Group

Centro de Computação Científica e Software Livre (C3SL).
`http://www.c3sl.ufpr.br`

Free Software and Scientific Computing Center.

- Developing end-to-end solutions that requires more than a single domain of knowledge.
    - Artificial Intelligence, databases, networks
- Several project with social-inclusion initiatives
    - OER - Open Educational Resources portal
    - Digital inclusion projects monitoring
    - Brazilian Educational Data indicators

UFPR
UNIVERSIDADE FEDERAL DO PARANÁ

C3 SL

Introduction

Open Data Today

- The amount of data available in Open Data sources has not stopped growing in recent years.
- Semi-structured (CSV files) and unstructured (JSON - Free Text) data is available.
- Data integration of Open Data allows querying over several relations and attributes (Miller, 2018).
  But querying over the sources is still difficult.

SPJG - Select-Project-Join-GroupBy

Structured data can be stored in Relational Databases.

A typical query has four elements:

1. The attributes to be retrieved.
2. The restrictions to be satisfied.
3. The relations that are used.
4. How to combine these relations.

These queries are often called as SPJ (Select-Project-Join) queries.

When adding grouping, we can call them SPJG queries.

The relations to use and how to combine then are dependent of the database.

Our approach

Problems:

- To perform simple analytic queries (*stupid analytics* (Abadi and Stonebraker, 2015)) with SPJG format.
- Data accessible though an API.
    - Could be consumed by application developers.
    - Broadening the access of available public data.

Our Approach:

- Relation Free Queries (RFQ).
- Creating SPJG queries using an API.
- Do not requires the definition of relations nor joins (elements 3 and 4).

How we do it

Approach summary:

- RESTFul API as the top layer to write queries (Open Data Micro Service).
- API calls translated in SQL into a **virtual database schema**.
- Queries in the virtual schema are rewritten into the real schema.
    - Joins found based on views names
- The API returns the result of the query in the real schema.

Case of Study:

- Brazilian public data about education and digital inclusion.
- 24 relations,   1000 attributes and   2.5 billion records.

RFQ

RFQ - Relation Free Query

- Theoretical model of our approach.
- Simplified representation of SPJG queries.
- A Relation Free Query Q is a triple (M, D, C):
  - A set of metric M (aggregated attributes).
  - A set of dimensions D (attributes used in grouping).
  - A set of clauses C (restrictions to be satisfied).

$$Q(m_0, m_1, \ldots, m_n)(d_0, d_1, \ldots, d_p)(c_0, c_1, \ldots, c_q)$$

Example

Database schema (simplified version of the case study):

- *student(st_name; st_id; sc_id;grade;age)*
- *school(sc_name; sc_id; city_id;category)*
- *city(city_id; city_name; state; region)*

Metrics:

- $n\_student = (COUNT; st\ id)$
- $mean\_age = (AVG;age)$

Question: What is the mean age and how many students of the forth grade exist by region?

Example

RFQ:

$$Q(n\_student; mean\_age)(region)(\{grade = 4\})$$

SQL:

```
SELECT
    COUNT(st.st_id) AS n_student,
    AVG(st.age) AS mean_age,
    c.region
FROM student st
INNER JOIN school sc ON sc.sc_id = st.sc_id
INNER JOIN city c ON sc.city_id = c.city_id
WHERE st.grade = 4
GROUP BY c.region
```

Similarities and differences

- All RFQ metrics contains aggregation functions in SQL.
- All RFQ dimensions are in the GROUP BY statement in SQL.
- All RFQ restrictions (clauses) are in WHERE statement in SQL.

- `n_student` and `mean_age` are not explicit defined in RFQ.
- Which relations to use are not explicit defined in RFQ.

RFQ to SQL

Virtual Schema

To use RFQ in Relational Databases, it must be translated to SQL. This is done using a virtual schema.

- The virtual schema is a set of views.
- These views are created over the real database.
- These views can be used to define "new" attributes (n_student and mean_age).
- These views respect some restrictions, allowing fast, reliable and predictable translation.

Virtual Schema restrictions

- There is one relation for each metric, called A_m.
- The metric m exists only in A_m.
- A_m also contains all dimensions which m can be grouped by.

- All queries that use m must use A_m.
- A query with a single metric m only requires A_m.

Query in two steps

1. $Q'_m = \gamma_{D,f(m)}(\sigma_C(A_m))$
2. $Q' = Q'_{m_1} \bowtie Q'_{m_2} \bowtie ... \bowtie Q'_{m_n}$

1. Gets only the required information of a single view A_m.
2. Join all partial results using INNER JOIN.

- Relational Algebra expressions can be directly translated to SQL (Virtual Schema).
- The virtual schema is a set of views over the real schema.
- Expanding the view definitions result in a valid query in real schema.

Implementation

BlenDB

- BlenDB tool is the implementation of RFQ theoretical model.
- It is a RESTful API (NodeJs).
- API calls has a 1-to-1 correspondence with RFQ queries.
    - A BlenDB request parameters are: metrics, dimensions, clauses, format.
- Returns data in CSV or JSON format.

BlenDB

- The tool in configured with mapping files, that describe the database schema.
- These mapping files provide information not obtainable from raw SQL schema definition.
- Tool is available as Free Software.
- Supports PostgreSQL and MonetDB (so far).
- Source code: `https://gitlab.c3sl.ufpr.br/c3sl/blendb`

BIOD

- Blended Integrated Open Data (BIOD).
- Microservice that distributes Open Data using BlenDB API.
- Brazilian **educational** and **digital inclusion** data.
- 24 tables, 1169 attributes, 2.5 billion records.
- Uses MonetDB as DBMS .
- BlenDB configuration with 682 metrics and 904 dimensions.
- Mapping have been done manually.
- Data avaiable at: `https://biod.c3sl.ufpr.br/index_en.html`

Manual Steps

Creating mapping file (One per database table)

- Define metrics.
- Define dimensions.
- Define how to join with other tables

- Done by a database expert
- Done only once.

Automatic Steps

On start up

- Automatically build the Virtual Schema using mapping files.

On every query

- Parse request into RFQ query
- Validate query
- Search for relations used in query
- Write query in virtual schema
- Parse query to real schema
- Execute and return result of real query

Case Study

Query 1

- Let us suppose that we want to know general descriptions about the city of Curitiba, that can be used to identify development information. This requires attributes about population, economic indicators, internet connection, among others. The query has the following definition:
- **Q** (SumPopulation, CountSchools, CountUniversity, AvgEconomyGDP, AvgEconomyIncomeLevel, CountSimmcPoint) (State, Region, CityName) (CityName = Curitiba, Year = 2017)
-
```
http://tool.domain/v1/data?
    metrics=sumpopulation,countschools,countuniversity,
    sumeconomigdp,avgeconomyincomelevel,countsimmcpoint
    &dimensions=state,region, cityname
    &filters=cityname:Curitiba;year:2017
```

Query 1 - results

```
{
    "region":"Sul"
    ,"state":"Paraná"
    ,"cityname":"CURITIBA"
    ,"sumpopulation":"3728832"
    ,"countschools":448
    ,"countuniversity":59
    ,"sumeconomigdp":78892229400
    ,"avgeconomyincomelevel":5
    ,"countsimmcpoint":19
}
```

Generated SQL

```
SELECT SUM(pop.POPULATION), COUNT(sc.SC_ID) AS N_SCHOOL,
    COUNT(un.UN_ID) AS N_UNIVERSITY, SUM(gdp.GDP),
    AVG(gdp.INCOME_LEVEL), COUNT(poi.POI_ID) AS N_POINT,
    c.STATE, c.REGION
FROM POPULATION pop
INNER JOIN SCHOOL sc ON sc.CITY_ID = pop.CITY_ID
INNER JOIN UNIVERSITY un ON un.CITY_ID = pop.CITY_ID
INNER JOIN GDP gdp ON gdp.CITY_ID = pop.CITY_ID
INNER JOIN POINT poi ON poi.CITY_ID = pop.CITY_ID
INNER JOIN CITY c ON c.CITY_ID = pop.CITY_ID
WHERE c.NAME = Curitiba AND
    sc.YEAR = 2017 AND un.YEAR = 2017 AND
    gdp.YEAR = 2017 AND inc.YEAR = 2017 AND
    poi.YEAR = 2017 AND pop.YEAR = 2017
GROUP BY  c.STATE
```

Query 2

- We consider a specific scenario about schools, group by region of Brazil and administrative dependency of school (federal, state, municipal, private). The query produced is:
- **Q** (CountSchools, AvgSchoolClassroom, AvgSchoolEmployees) (Region, SchoolAdministrativeDependency) (Year = 2017)

-
  ```
  http://tool.domain/v1/data?
      metrics=countschools,
      avgschoolclassroom,avgschoolemployees
      &dimensions=region,schooladministrativedependency
      &filters=year:2017
  ```

Query 2 - results

```
{
    "Region":"Centro-Oeste"
    ,"SchoolAdministrativeDependency":3
    ,"CountSchools":5611
    ,"AvgSchoolClassroom":8.448424091509072
    ,"AvgSchoolEmployees":32.54879051914905
}
```

Query 2 - SQL

```
SELECT COUNT(sc.SC_ID) AS N_SCHOOL, AVG(sc.SC_CLASSROOM),
    AVG(sc.SC_EMPLOYEES),
    sc.SC_ADMINISTRATIVE_DEPENDENCY, c.REGION
FROM SCHOOL sc
INNER JOIN CITY c ON c.CITY_ID = sc.CITY_ID
WHERE sc.YEAR = 2017
GROUP BY c.REGION, sc.SC_ADMINISTRATIVE_DEPENDENCY
```

Conclusions

Key contributions

- Microservice that publishes Open Data (*BIOD*).
  - **Case study:** integrated Billions of records and hundreds of attributes of Brazilian digital inclusion and educational data
- SPJG queries defined through an API.
  - Joins are not explicitly defined, only *dimensions, metrics and filters*
  - *BlenDB* middleware finds the better joins and generates the SQL
- Tool compatible with existing DBMS.

Future Work

- Automated generation of mapping files.
- Extension of RFQ to other DBMS and data lakes.
- Extension of RFQ to support *unionable tables*.