SAPIENZA
UNIVERSITÀ DI ROMA

Constraint-Aware Query Processing for Geo-Distributed Data

Department of Computer, Control, and Management Engineering (DIAG)

Corso di Laurea Magistrale in Engineering in Computer Science

Candidate

Alessandro Benedetto Melchiorre
ID number 1772850

Thesis Advisors

Prof. Ioannis Chatzigiannakis
Dr. Kaustubh Beedkar (TU Berlin)

Co-Advisor

Prof. Volker Markl (TU Berlin)

Academic Year 2018/2019

Thesis defended on 28 March 2019
in front of a Board of Examiners composed by:

Prof. Alessandro De Luca (chairman)
Prof. Roberto Berladi
Prof. Silvia Bonomi
Prof. Ioannis Chatzigiannakis
Prof. Fabrizio D'Amore
Prof. Luca Iocchi
Prof. Roberto Navigli

Constraint-Aware Query Processing for Geo-Distributed Data
Master's thesis. Sapienza – University of Rome

© 2019 Alessandro Benedetto Melchiorre. All rights reserved

This thesis has been typeset by LaTeX and the Sapthesis class.

Version: 28 March 2019

Author's email: alessandro.b.mel@gmail.com

Dedicated to
my family and friends

Abstract

Nowadays, many large organizations operate data centers that produce huge amounts of data at different locations around the globe. Analyzing such geographically distributed data as a whole is essential to derive valuable insights. Typically, Geo-distributed data analysis is carried out either by first communicating all data to a central location where processing is performed or by a distributed execution strategy that minimizes data communication. However, a whole new dimension of constraints arising from regulations pertaining to data sovereignty and data movement is increasingly receiving more attention and posing serious limitations to the above mentioned ways of processing Geo-distributed data.

In this work, we provide a new formalism for data movement constraints definition and look at how to enable current data analysis tools and frameworks to deal with restrained data movements. More specifically, we aim our attention at SQL analytics on Geo-distributed data stored in relational databases which are part of federated database systems. Given an user query issued on the federated systems and the formalized data constraints, the result of our work generates compliant execution plans by specifying when and where data shipments are allowed to take place between the data centers.

Contents

Chapter 1

Introduction

Nowadays, more and more data is generated by the numerous automated systems around us and is typically collected and stored in sophisticated distributed systems whose autonomous nodes are scattered around the globe, ensuring low communication latency and high availability. For example, Amazon Web Service Cloud spans an area which covers America, Asia, Europe, and Oceania [1] and Google's data centers are present in multiple countries in Europe and North America [3]. Moreover, the latest trends in Big Data involve the use of several popular tools and frameworks, such as Spark [19], Flink [7], Hive [15], Hadoop [2], and many more, to derive valuable insights from the analysis of such Geo-distribute data which benefits many diverse applications. A few examples are: cancer prediction and prognosis in predictive medicine using Machine Learning methods [8], intrusions detection in cybersecurity discovering correlations between security events across heterogeneous sources [20], tailored and personalized recommendations for multimedia entertainment such as Youtube [10].

Analysis of Geo-distributed data is typically accomplished by either (i) transferring all data to a central location or (ii) distributing the execution task over the multiple data engines and moving data around the sources in such a way that the least amount of data transfer occurs across the sites. However, following the recent events on data protection, a whole new dimension of constraints is increasingly receiving more attention and posing serious limitations to the above mentioned ways of processing Geo-distribute data. Legal restrictions pertaining to data sovereignty and data movements control the flows of data from one location to another in order to impede any information disclosure which might damage individuals, companies, or, in general, any kind of organization. Consider the case of the General Data Protection Regulation (GDPR) [5] whose jurisdiction especially concerns companies residing in the EU and which forbids personal data transfers to third parties without the owner's consent. Depending on the disclosure risks associated to hypothetical leaks of information, data conveyances are either completely cut off, obliging organizations to carry out solely local data analysis, or they undergo several transformations so that the result is an exposure-friendly version of the input having sensitive data concealed.

In this work, we explore the new dimension of constraints and look at how to enable current data analysis tools and frameworks to deal with restrained data

movements. More specifically, we focus on Geo-distributed data stored in relational databases, as they are widely adopted, and that are part of federated database systems which provide a coherent and comprehensive view of the whole regardless of the sources heterogeneity. Furthermore, we aim our attention at SQL analytics, based on relational algebra augmented with communication (shipping) operators, allowing us to leverage the extensive literature work on query optimization, data integration, and query rewriting. Given a user query posed on the federated systems and the formalized data constraints, the result of our work looks for execution strategies that are compliant to these restrictions, specifying when and where data shipments are allowed to take place, and are inherently efficient from both local resources usage and network transmission point of view.

In summary, we make the following contributions in this thesis:

- We provide a formalization for data movement constraints using extended relational views. An extended view definition includes a legal set, which is a set of locations where the data can be legitimately shipped to. Specifying that some data D from location L_1 is allowed to be transferred to location L_2 implies the existence of a view on D with L_2 in its legal set.

- We show how to enumerate compliant execution strategies based on optimization rules that enforce constraints by rewriting parts of the user query using views. If a (sub) query matches or subsumes a view, then the query results can be legitimately be shipped to the locations in the legal set. Using this mechanism we specify where operations may take place and when to ship data without information leaks.

- We implement a prototype of our system using Apache Calcite, an extensible query parser and optimizer framework. Since Calcite is internally employed by many other well-known data frameworks, as for example Flink, Storm, and Drill, stacking our implementation on top of it easily enables these tools to deal with data movement constraints.

In this way we show that constraint-aware query processing for Geo-distributed data is a feasible approach and can be easily achieved by our proposed solution, supporting the current data protection demands in the era of Big Data analysis.

The thesis is organized as follows. In Chapter 2, we explore the background work we leveraged for the thesis, namely query processing, data integration, and query rewriting. Chapter 3 discusses the formalism for the data movements and describes its implementation using extended relational views. In Chapter 4, core of our work, we explain the constraint enforcer mechanism based on query rewriting using views and show how to generate compliant plans using a set of optimization rules. Chapter 5 describes the details of the system implementation and the Geo-distributed environment we simulated to test our solution, for then providing some example use cases. Lastly, Chapter 6 contains our final remarks and future objectives.

Chapter 2

Background

As stated previously, we will focus on query processing in distributed relational database systems, where user queries are issued on the federated schema and then executed on the underlying heterogeneous sources. As we will see later in Chapters 3 and 4, our solution specifies data movements constraints by means of relational views and extends a top-down optimizer that explores only compliant execution plans as a result of constraint-tailored rules set. In particular, one type of such rules enforces constraints by using an inference mechanism that attempts to rewrite parts of the user query in terms of the given constraint-views.

Given this premise, in the following chapter, we briefly describe the aforementioned concepts by touching the topics of traditional and distributed query processing, data integration, and query rewriting using views. The chapter is structured as follows. Section 2.1 provides a high level description of a general query processing architecture and a concise description of its components, for then focusing on query optimization. For additional information of the topic, we suggest the reader to refer to [9] and [18]. Section 2.2 reviews query processing in distributed systems highlighting the differences from the centralized case. An interesting reading on the general techniques in this context is [14]. Section 2.3 provides a summary on federated database systems and explains how they provide a coherent view of the data in presence of heterogeneous sources. Lastly, Section 2.4 shortly describes the major concepts in query rewriting using views. We suggest [13] for a survey on the topic.

2.1 Traditional Query Processing

All types of database systems (centralized, distributed, parallel, etc.) normally agree on the general query processing architecture depicted in figure 2.1. As shown, it is composed by three macro-steps; the first two translate a high-level query into a sequence of low-level operations on data sources, while the last one actually carries out the previous specified instructions to return the resulting tuples to the user.

The input of the process is a SQL query that is converted by the Parser into an abstract syntax tree, an internal representation more suitable for machine manipulation. The Parser also checks the query syntax for errors and ensures that the attributes and relations referenced in the query are actually in the catalog. If

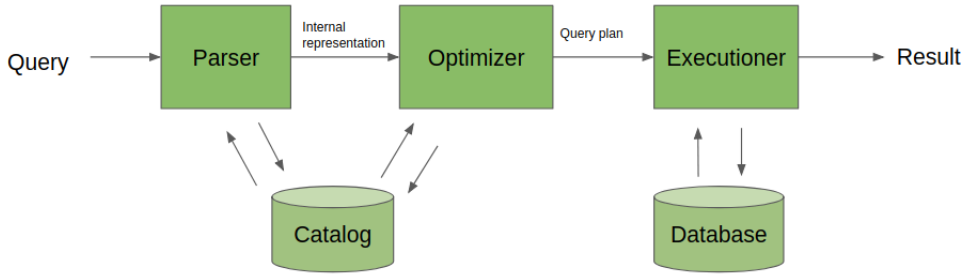

Figure 2.1. General architecture for Query Processing.

the query is acceptable, the abstract syntax tree is casted into another tree whose nodes are relational algebra operators, which, in turn, represent the operations to carry out on data. The Optimizer, then, accepts the algebra tree in input, generates multiple feasible execution plans, and selects the best one according to a function based on cost metrics, like CPU, I/O, etc., and statics about the data. Lastly, the selected query plan is evaluated on the underlying database system and the answer is returned to the user.

The plan generation phase, in particular, consists of two steps which might be interleaved: logical plans generation and physical plans generation. The first one concerns only the logical transformations of the data, i.e. the types of relational operators in the plan and their order. To do so, the optimizer applies algebraic equivalences rules which transform the original query tree into another semantically equivalent relational algebra tree. Relational algebra equivalences such as join associativity and join commutativity are an example of these algebraic rules. Physical plans generation, instead, defines the physical details of both operators and data involved in the query. In this step, the optimizer decides which algorithms should be used (merge-join, hash-join, etc.), how intermediate results should be treated (materialized or pipelined), and which properties the output data should have (sorted on an attribute, distributed on nodes, etc.).

Both previous steps are usually based on optimization rules that perform the logical and physical transformations described above; the optimizer begins with an initial query plan and sequentially applies rules until a feasible and satisfactory solution is found. Moreover, there are two popular approaches for searching the best execution plan: top-down and bottom-up. The bottom-up approach computes the best plans for all expressions on k relations before considering expressions involving more than k relations. On the other hand, the top-down approach computes the best plans for only those expressions on k relations which are included in some expression on more than k relations being expanded. In other words, the top-down approach will never consider solutions on k relations whose cost is higher than solutions on more than k relations.

2.2 Query Processing in Distributed Databases

The general practice of query processing described in the previous section is valid even for the cases of distributed database systems, however, some steps must be

adjusted to the new underlying infrastructure. While queries in centralized systems involve only the data stored and maintained in a single location, queries in distributed systems refer to relational tables which are fragmented and replicated across multiple autonomous physical locations. The query optimization process itself is distributed; The practice typically consists of a global optimization step involving all the sources, followed by local optimization steps at each affected site.

Besides deciding the type and the order of the operations, the global optimizer also has to establish at which site each operation is to be executed and when intermediate results have to be relocated between sources. In the case of table replication, there is the further choice of which local copy it should access among the ones available. The preceding aspects are handled in the physical plan generation phase through location properties and shipping operators. In particular, each operator is labeled with an execution site, where the operation takes place, and the data movement processes are captured by specifying where the output of a previous operation should be moved to, using shipping operators. Hence, a plan in the global optimization step is deemed to be completely specified when all the operations are labeled with an execution site and shipping operators ensure the data lies at right location. An example of an enhanced query tree in the context of distributed sources is depicted in fig. 2.2.

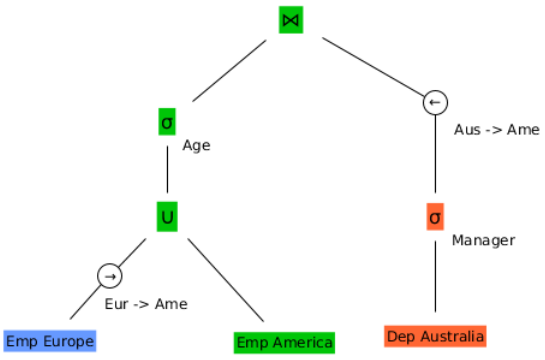

Figure 2.2. Query tree in a Geo-distribute environment.

However, since data movements are costly, not all execution plans are explored by the optimizer search strategy. For this reason, the network traffic is taken into account in the cost function and, in order to minimize it, the optimizer attempts to reduce the amount of data transfers occurring among the sources.

Finally, the global optimizer decomposes the original query plan in multiple subplans, one for each involved source, and then the local optimizers independently operate on them to seek for the best local solution.

2.3 Federated Databases

Generally, data nodes in a distributed system might present different logical schemata and users can issue queries only if they are aware of the underlying distribution of the data sources. Through data abstraction, federated database systems provide

a uniform user interface, enabling users to store and retrieve data from multiple non-contiguous databases with a single query - even if the component databases are heterogeneous and the users does know about the system infrastructure. Queries posed on the so-called *global schema* are then transparently answered using data contained in the sources, regardless of the *sources schemata*.

The relationship between the global schema and sources schemata is expressed in terms of *mappings* which essentially establish what data populates the global tables and how retrieving it. For relational databases, mappings are equivalent to views. They can be basically specified in two manners: Global As View (GAV) and Local As View (LAV). The first kind of mappings impose that the global schema must be defined in terms of the sources, namely a table in the global schema is a query over the sources. The user query, which references global relations, can be easily translated into a query over the sources - global tables are replaced by the query defined in the mappings. LAV mappings are quite the opposite; they require the sources to be defined in terms of the global schema, that is to say, for each relation in the sources there is a query over the tables in the global schema. For this reason, user queries must undergo a more complicated query rewriting procedure to generate a query over the sources. Nevertheless, LAV mappings are superior to GAV in describing the data contained in the sources and result more meaningful in query answering. Another type of approach, GLAV mappings, is a hybrid obtained by mixing the previous two.

Following the choice of mappings, data in the global schema might either be materialized, creating an actual database, or not, keeping mappings virtual. In terms of materialized relations, user queries are straightforwardly answered with the tuples in the global tables while virtual mappings require one or more queries to be issued on the sources to retrieve the necessary data.

Whatever type of mappings one decides to employ, user queries always have to be rewritten in terms of sources in a way or another. This raises the issue of source completeness, or rather, do sources have all the appropriate information to answer the query? Under the Closed World Assumption (CWA), the set of all tuples fetched by mappings is enough to provide a 'certain' answer to the query. On the other hand, under the Open World Assumption (OWA), sources are acknowledged to be incomplete and some tuples may not appear in the query answer.

2.4 Query rewriting using views

As we have seen in the last section, query rewriting using views holds a central role in federated database systems since user queries always have to be rewritten in terms of the sources using mappings (views). Even in the context of query optimization its application is relevant, as materialized views can greatly speed up the query answering process with precomputed results. Furthermore, as we mentioned before, constraints enforcement in our work is carried out by means of an inference mechanism based on query rewriting in the optimization context; the optimizer attempts to rewrite parts of the user queries using views (constraints).

Regardless of the application, query rewriting techniques lean on the fundamental concept of *query equivalence*; Two queries are said to be equivalent if for each database

state, they compute the same set of tuples. In the context of optimization, we are interested in finding an *equivalent rewriting* for the whole query or for part of it with a set $\mathcal{V}$ of views $V_1, V_2, ..., V_n$. A query Q' is said to be an equivalent rewriting of the (sub) query Q if:

1. Q' refers only to the views in $\mathcal{V}$
2. Q' is equivalent to Q.

Trivially, database relations might be involved in the rewriting procedure using views that mimic their definition. Nevertheless, since the query equivalence problem in full relational algebra is undecidable, all the practical approaches on finding an equivalent rewriting of Q are restricted to smaller subsets of solutions. They mostly differ in the type of queries and views they consider (select-project-join queries, aggregate queries,etc.) and they type of rewriting they produce (single query rewriting, union rewriting, etc.). Furthermore, not all views in $\mathcal{V}$ are usable to answer a specific query Q even if they involve the same table and each rewriting approach formally define their condition for usability. Informally, a view can be used in the rewriting if either is equal to the query (trivial case) or the query is subsumed by the view, that is, the view results to be more 'general' than the query itself, and the results can still be computed from its output. The second case implies that additional operations should be carried out on the view to obtain the same result of the query such as filtering with additional predicates, removing some columns, etc.

Chapter 3

Constraints Specification

As we know from Section 2.2, answering a query in geographically decentralized systems means to distribute the execution task across the involved data sources and to relocate intermediate data results when needed. However, in Chapter 1, we introduced the emerging dimension of data movement constraints and how it affects data analysis in such a context. Data shipments between Geo-distributed sites are strictly regulated by data sovereignty measures and must adhere to a set of data protection rules in order to prevent any type of information disclosure. The severity of these 'disclosure constraints' varies according to the needs of the issuing entities. Strict restrictions might impose a forced localization by banning data transfers, while more flexible rules may allow shipments under some conditions (e.g. data is masked before sending).

In the following Chapter, we provide a new formalism for data movement constraints which enables both entities to easily define customized shipment restrictions and data analysis frameworks to deal with these recent processing limitations. Section 3.1 holds a fully and clear description on the context in which constraints fit in and provides basic observations to their dynamics. In Section 3.2, we discuss the definition of data movement constraints. Finally, in Section 3.3, we describe a simple implementation for relational databases using extended relational views which includes location disclosure policies in their definition. Chapter 4, then, will explain how we leveraged this definition to achieve a constraint-aware query processing.

3.1 Constraints Context and Dynamics

Before starting to discuss about data movement constraints and their definition, we firstly introduce their context with the respect of federated relational databases and provide few basic rules to their enforcement. As already mentioned, data in a Geo-distributed system is collected and stored at multiple autonomous relational sources and it is subjected to different data sovereignty measures that depend on the location of the sites. In particular, we assume data sources are organized in groups of sites, each one belonging to a jurisdictional area where the same data movement constraints apply. We denote the areas with $A_1, A_2, ..., A_n$. Furthermore, we consider that areas do not overlap and locations where jurisdictions intersect are handled as new areas subjected to the constraints of both intersecting parts. The shape of these

areas may either be defined by international and national regulations, matching borders with actual boundaries between countries or international organizations, or by company policies and individual needs, creating custom areas tailored to more specific data protection needs. We use the term 'constraint administrators' to refer to the generic entities (international organization, companies, and so on) that hold the responsibility to set the data movement constraints for the relational tables in their area. A single table might be involved in many constraints while a single constraint may refer to the data in multiple tables as long as they lie in the same area. The reason for this choice is to avoid to create unnecessary dependencies between constraints leading to flawed definitions. When a new data source is attached to the federated system, constraint administrators supply the new rules that guide the data interactions between the new site and the other locations. We denote with $A_x \to A_y$ the set of data movements rules from A_x to A_y.

Basically, there are two main observations on these interactions:

1. Data lying at one site in A_x can freely move to other sites within the location boundaries. No need to enforce constraints since data is still in a 'safe zone'.

2. Data leaving their origin site in A_x and crossing the area boundary of A_y must be compliant to the disclosure constraints of A_x relative to A_y. In other words, rules regarding the pair $A_x \to A_y$ must apply. Data might be masked to ensure compliance.

Let us notice that the second point also concerns data transits in third locations A_z; if data originally collected in A_x moves to A_z and then to A_y, it is subjected to both constraints rules of $A_x \to A_z$ and $A_x \to A_y$.

3.2 Data movement constraints definition

As we said in the previous section, data is stored in relational databases which, in turn, belong to some jurisdictional areas where the same rules apply. For this reason, in this section, we will define data movement constraints referring to areas the data belongs to, rather than the specific residing sites. We will use the term area and location interchangeably.

Essentially, we define constraints as triples:

$$(\mathcal{D}, \mathcal{M}, \mathcal{L})$$

$\mathcal{D}$ delineates the portion of the data affected by the constraint and determines its extent, $\mathcal{M}$ is the masking operation which yields to an exposure-friendly version of the data, and $\mathcal{L}$, called *legal set*, is the collection of locations (areas) where the concealed data, consequent of masking, is entitled to go. Hence, a constraint can be interpreted as follows; the data specified by $\mathcal{D}$, coming from tables within the same location, must undergo the transformations $\mathcal{M}$ before being moved to locations in $\mathcal{L}$. Let us notice that, as default, if no constraint refer to some data D, then all transfers of D across area boundaries are forbidden in any case.

The nature of $\mathcal{M}$ is defined by the constraint administrators and depends on the strictness of their data concealment requirements. It might be that the same

data is transferred unaltered to some locations (no masking involved) while it has to be heavily transformed when sent to 'less secure' areas. In particular, according to the interpretation given above, masking operations may be carried out even in sites and areas distinct from the one where the data has been originally collected. Consider the case in which data from A_x travels to A_z after having been masked according to the rules $A_x \rightarrow A_z$. If data has to be further sent to A_y, then the masking operations for $A_x \rightarrow A_y$ can be carried out either in A_x or A_z. We embrace various types of masking operations in $\mathcal{M}$, which some examples are: identity (data can be shipped as it is), suppression, aggregation, shuffling, as well as user defined functions for customized masking.

A flexible data scope $\mathcal{D}$ is a crucial requirement in constraints definition since constraint administrators might design rules regarding different data granularity levels (rows, columns, etc.). On its basis, there are three types of constraint specifications: attribute-based, tuple-based, and value-based. In order to familiarize with these concepts, we will consider examples of constraints involving mock relations "Employee" (Table 3.1) and "Department" (Table 3.2) which contain information about employees and departments in a company located in Europe. We will momentarily focus only on the $\mathcal{D}$ and $\mathcal{M}$ while $\mathcal{L}$ will be considered in the next section.

Table 3.1. Employee table.

Name	Surname	Age	Salary	Department
Mario	Rossi	36	50k	HR
Tim	Fischer	54	45k	IT
Adrienne	Canard	43	40k	BB
Freja	Nilsson	55	60k	HR
Carmen	Serrano	22	20k	IT
Dimitrios	Pilos	59	62k	BB

Table 3.2. Department table

Name	FullName	Manager
HR	Human Resources	Robert
IT	Information Technologies	Timothy
BS	Business	Robert

Attribute-based specification

Attribute-based, or column-based, constraint specifications impose restrictions on sets of columns in a relational table. Assume for example that attributes Salary and Age cannot be disclosed outside Europe and must be suppressed before crossing any boundary. With attribute-based constraints we are able to express this requirement and produce the table in 3.3 for foreign locations. If an user query involves only the remaining attributes, query execution can be distributed to locations outside

Europe, otherwise if the query refers to Salary and Age, then, since they cannot trespass, intermediate results are rigidly localized in their origin area.

Table 3.3. Employee table without the Salary and Age attributes

Name	Surname	Department
Mario	Rossi	HR
Tim	Fischer	IT
Adrienne	Canard	BB
Freja	Nilsson	HR
Carmen	Serrano	IT
Dimitrios	Pilos	BB

Another policy could allow us to disclose the attribute Salary only if the values are shuffled among people belonging to the same Department as in Table 3.4. As we can see, employee "Dimitrios Pilos" has a wrong salary of 40k instead of the true value of 62k. Therefore, querying for Person-Salary would yield to incorrect results outside Europe while querying for the sum of salaries grouped by department would lead to the same results everywhere.

Table 3.4. Employee table with Salary values shuffled within the Department.

Name	Surname	Age	Salary	Department
Mario	Rossi	36	60k	HR
Tim	Fischer	54	20k	IT
Adrienne	Canard	43	62k	BB
Freja	Nilsson	55	50k	HR
Carmen	Serrano	22	45k	IT
Dimitrios	Pilos	59	40k	BB

Tuple-based specification

Tuple-based, or row-based, constraint specifications target tuples instead of attributes. They select the rows from tables which should undergo masking operations before shipping. Assume, for instance, we cannot disclose employees that earn more than 48k as in Table 3.5. It is reasonable to query the Employee table for people earning

Table 3.5. Employees earning less or equal than 48k.

Name	Surname	Age	Salary	Department
Tim	Fischer	54	45k	IT
Adrienne	Canard	43	40k	BB
Carmen	Serrano	22	20k	IT

less than 48k since the result is a subset of tuples fully contained the previous table.

However, if the selection predicate includes salaries higher than that, some tuples would be missing from the result when exported.

As an example of a constraint involving more tables, consider the case where we disclose only the employees that work in the departments managed by Robert and are older than 40 as in Table 3.6. As we mentioned, the tables involved in

Table 3.6. Employees working in departments managed by Robert and older than 40.

Name	Surname	Age	Salary	Department	FullName	Manager
Adrienne	Canard	43	40k	BB	Business	Robert
Freja	Nilsson	55	60k	HR	Human Resources	Robert
Dimitrios	Pilos	59	62k	BB	Business	Robert

a constraints must lie in the same area. If the Department relation was located outside Europe then the previous constraint would be useless since we cannot match employees to departments without disclosing values. A weak solution would be to provide further constraints which allow the movement of data for the matching purposes. However, the previous constraint would be depended on these and, in general, we may possibly create unnecessary circular dependencies.

Value-based specification

Lastly, value-based, or cell-based, constraint specifications have a more finer area of action, namely cells of tables. Consider the case where employees choose if they want to disclose their information as in the table 3.7. Y means we have consent to reveal the information while N forbid us to do so. After the masking, the table will

Table 3.7. Employee table showing people consent to reveal information.

Name	Surname	Age	Salary	Department
Mario (Y)	Rossi (Y)	36 (Y)	50k (N)	HR
Tim (Y)	Fischer (N)	54 (Y)	45k (N)	IT
Adrienne (Y)	Canard (Y)	43 (N)	40k (Y)	BB
Freja (Y)	Nilsson (N)	55 (N)	60k (N)	HR
Carmen (Y)	Serrano (Y)	22 (Y)	20k (Y)	IT
Dimitrios (Y)	Pilos (N)	59 (Y)	62k (N)	BB

look like the one portrayed in fig. 3.8. In this case, it is less clear which classes of queries may return the right results outside Europe

3.3 Expressing constraints through extended views

Referring to the previous section, data movement constraints are defined as triple of targeted data, masking operations, and a set of locations (the legal set). While the first two aspects involve the manipulation of data by retrieving it from relational tables and transforming it, the third establish where the masking outcome can

Table 3.8. Employee table with suppressed values.

Name	Surname	Age	Salary	Department
Mario	Rossi	36		HR
Tim		54		IT
Adrienne	Canard		40k	BB
Freja				HR
Carmen	Serrano	22	20k	IT
Dimitrios		59		BB

legitimately be shipped. Since in our work we focus on SQL analytics in relational databases, we decided to implement the constraint data manipulation using relational views. Thereby, it allows us to reuse the data definition language of the SQL framework without specifying a separate rule formalism. Views can project out specific columns and select subset of rows allowing a fine-grained data access compatible with the previously mentioned constraint specifications. For example, we could obtain the table 3.3 with no salary and age attributes using the following view.

```
CREATE VIEW EmpNoSalaryNoAge AS
SELECT Name, Surname, Department FROM Employee;
```

Or, for the case of the tuple-based constraint in table 3.5, we could select employee with a salary lower or equal than 48k using the following view.

```
CREATE VIEW EmpLowSalaries AS
SELECT * FROM Employee WHERE Salary <= 48k;
```

In general, we may define more complex queries involving multiple tables. Consider the case where we mask the salary values computing the sum of salaries per department.

```
CREATE VIEW SumSalariesPerDepartment AS
SELECT Dep.FullName, Sum(Salary)
FROM Employee AS Emp JOIN Department
AS Dep ON Emp.Deparment = Dep.Name
GROUP BY Dep.FullName;
```

Furthermore, since most of relational databases allow the definition of stored procedures, it may be possible to employ user defined functions to customize the masking processes. For each person in the Employee table, the following view computes a hash function on the pair Name-Surname, a binary value depending on if the age is above 30, and a range of values where their salary should be in.

```
CREATE VIEW MaskedData AS
SELECT Hash(Name,Surname),IsOlderThan30(Age),Bucketize(Salary)
FROM Employee;
```

Views alone, though, are not able to express where the data can be legally shipped to, since they just manipulate it. To address the previous problem, we extend view definition with a list of areas as below.

```
CREATE VIEW <name> WITH LEGAL SET <list of areas> AS
<sql>
```

Trivially, the data origin location is always implicitly included in the legal set of the view referring to it. One example of usage in the case the filtered table Employee is allowed to be shipped from Europe to America and China is shown below.

```
CREATE VIEW EmpNoHighSalaries WITH LEGAL SET America, China AS
SELECT * FROM Employee WHERE Salary >= 48k;
```

In general, a single table may be involved in many constrains depending on the SQL query and the list of locations in the view. In particular, it may happen that the same table appears in different constraints for the same location as in the following example.

```
CREATE VIEW C1 WITH LEGAL SET America, China AS
SELECT * FROM Employee WHERE Age < 50;

CREATE VIEW C2 WITH LEGAL SET America, Australia AS
SELECT Name, Surname, Age FROM Employee WHERE Salary BETWEEN(30k,40k);
```

Observe that, both constraints include America in their legal set, despite applying different type of masking on different part of the same table. Depending on the query issued on the Employee table, one or both constraints might be used to safely distribute the data outside the Europe location.

Even if the constraint definition allows any type of transformation for $\mathcal{M}$, in the following work we only focus on masking operations in extended relational algebra (projection, selection, aggregation, etc.), leaving other types of masking for future goals. The next Chapter explains how extended relational views enable us to convert the issue of enforcing disclosure rules in a distributed environment to the well-studied problem of query rewriting using views. The constraints, in particular, will be used to generate compliant plans where intermediate results are moved to locations specified by the legal sets.

Chapter 4

Constraint-Aware Query Processing

As we saw in Section 2.2, the physical plan generation phase in Geo-distributed query processing enriches the logical query trees with implementation details by labeling the data operations with execution sites and injecting shipping operators to ensure that intermediate results are processed at the right source. Optimization rules are normally used by optimizers to provide these physical properties in query plans, as well as to evaluate other semantically equivalent execution strategies. In Section 2.4, we briefly reviewed the notion of equivalent rewritings and mentioned that different practical approaches of query rewriting using views have different conditions to evaluate the views usability. In Chapter 3, we introduced the notion of jurisdictional areas, locations where all data sites conform to the same data movement policies, defined data movement constraints as triples ($\mathcal{D}$, $\mathcal{M}$, $\mathcal{L}$), and provided an implementation for relational databases using extended view definition.

In this chapter, we explain how the three previous aspects are blended together to achieve a constraint-aware query processing in a Geo-dispersed environment. Essentially, we provide a set of Constraint-tailored optimization rules to a global optimizer and bias the optimizer search strategy towards compliant execution plans using a constraint enforcer mechanism based on query rewriting using views. The chapter is structured as follows. First, in section 4.1, we detail the distributed query optimization process in presence of locations. There, we introduce location-aware physical properties for operators, called location conventions, and adapt the definition of shipping operators. In section 4.2, we explain the constraint enforcer mechanism based on compliance-inference using views. Lastly, in section 4.3, we describe the compliant plan enumeration process in a top-down optimizer with the use of a special set of optimization rules concerning the location. The implementation details of our work are deferred to Chapter 5.

4.1 Location Conventions

When a new user query is asked on a distributed database system, the query optimizer eventually finds an execution plan where each operation is labeled with an execution site. In other words, for each operation, it selects a node which will (1) receive the

intermediate results produced by previous operations from the network, if they are not already present in the node, (2) carry out the operation on the gathered data and possibly ship the results to another node. Rather than designating a specific site for an operation, in our work we localize operations on data to sets of sites, each set representing a location, and defer the specific execution node choice to subsequent optimization steps. To implement this labeling procedure and associate operations to jurisdictional areas, we use a type of physical properties called *location conventions*. When an operation is labeled with a location convention L_x, any site belonging to the location A_x can carry it out. In this context, shipping operators assume a new meaning: they represent data transfer between locations rather than between specific sites. In figure 4.1 we show an illustrative compliant plan with location conventions and shipping operators. There are two location conventions in the plan encoded with different colors: Europe convention in blue and American convention in green. As we can tell from the picture, the projection on Employee table is carried out in Europe and the results are shipped to a generic site in America. There, the last two operations, a join and a selection, are executed. With the adoption of the

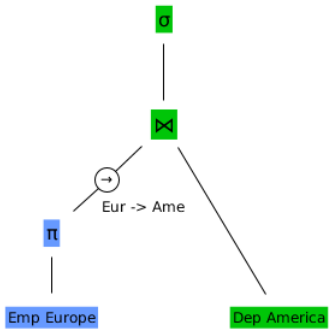

Figure 4.1. Example of plan with location conventions

above notation, we will able to fully characterize the compliant execution strategies generated by our optimizer.

4.2 Constraint Compliance Inference

In Section 2.4, we mentioned that since the problem of query rewriting using views is unfeasible in full relational algebra, the different practical approaches are restrained to some classes of solutions which consider only certain types of queries, views, and rewritings. Consequently, there are different types of conditions to view usability as they depend on the approach followed. However, these requirement always rely on subsumption or equivalence between the view and the query. Furthermore, as we remember from the previous Chapter, a data movement constraint is formalized by a relational view and a set of locations where the results can be shipped to. Specifying constraints in this way allows us to convert the issue of data restriction enforcement to the well-studied query rewriting problem. Essentially, views offer a safe window on data manipulation by specifying which transformations should be applied on data before safely shipping it, and if the query can be rewritten in terms of them,

we infer the location execution properties of the operations in the query plan. For such purpose, we will consider only single-view equivalent rewritings in which we try to rewrite the (sub) query solely using one view. Hence, if a (sub) query is rewritten in terms of a view then we infer that the query results can be legitimately shipped to the locations specified in the legal set.

Let us consider a simple query in which we retrieve the names and ages from all employees

```
SELECT Name, Age FROM Employee;
```

and suppose that Employee is horizontally partitioned into two tables, one in Europe and one in America. An equivalent way to write the user query would be

```
SELECT Name, Age FROM Employee.Europe
UNION ALL
SELECT Name, Age FROM Employee.America;
```

Assume furthermore we have the following constraint on the Employee partition in Europe

```
CREATE VIEW EmpEuropeToAmerica WITH LEGAL SET America AS:
SELECT Name, Surname, Age FROM Employee.Europe;
```

which allows to ship to America only the Name, Surname, and Age attributes. As we can see, the query fragment that selects Name and Age from Employee.Europe is subsumed by the previous constraint and an additional projection on top of the view would generate an equivalent rewriting. It follows that the results of the query fragment can be safely moved to the locations in the legal set, namely America. Given the following, a compliant plan is shown in Figure 4.2.

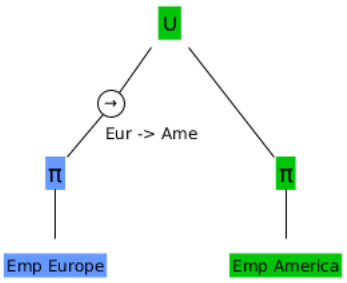

Figure 4.2. A compliant plan

In general, a single table may be involved in many constrains depending on the SQL query and the list of locations in the view, hence, multiple viable compliant plans can be feasible. Consider the following query where the department table Department is located in Australia.

```
SELECT * FROM Employee AS Emp
JOIN Department AS Dep ON Emp.Department = Dep.Name
WHERE Emp.Age BETWEEN(20,40) AND Dep.Manager = "Timothy";
```

Assume we have the following constraints:

1. **CREATE VIEW** EmpEuropeToAustralia WITH LEGAL **SET** Australia **AS**:
   SELECT ∗ **FROM** Employee.Europe **AS** Emp **WHERE** Emp.Age < 40;

2. **CREATE VIEW** DepToAmerica WITH LEGAL **SET** America **AS**:
   SELECT ∗ **FROM** Department.Australia **AS** Dep
   WHERE Dep.Manager **LIKE** "T∗";

3. Employee.Europe can be freely shipped to America

4. Employee.America can be freely shipped to Australia

Following the same reasoning of before, two compliant query trees are depicted in Figures 4.3 and 4.4. In the plan shown in Fig 4.3, the final location is America since

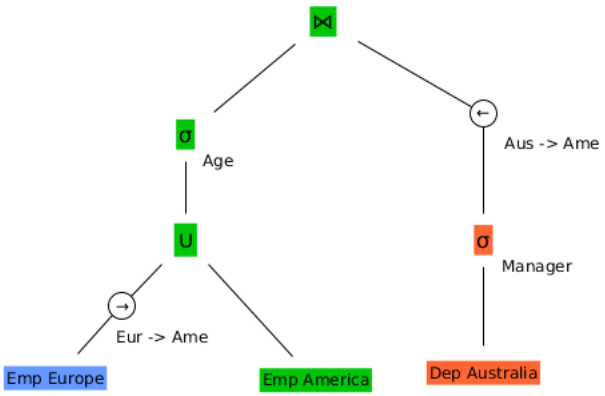

Figure 4.3. First compliant execution plan

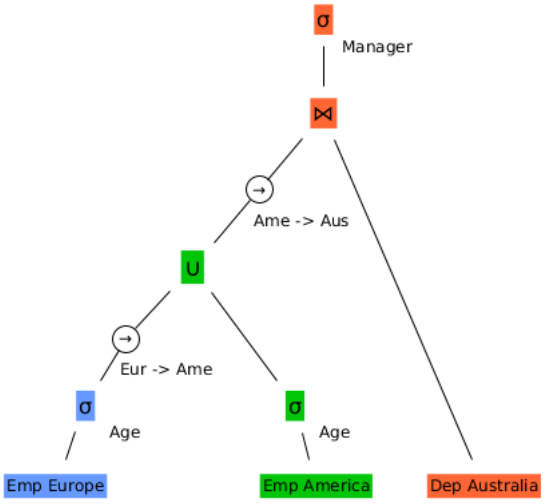

Figure 4.4. Second compliant execution plan

both the data from Employee Europe and Department Australia are shipped there. The third constraint allows the shipping of the Employee.Europe table to America while the shipping operator Aus→Ame is inferred from the second constraint. As for the latter, it is easy to see that the condition Manager = "Timothy" in the query is more specific than the predicate Manager LIKE "T*", hence, it allows us to ship the filtered table to America. In the plan in Fig. 4.4, instead, the data from Europe is firstly shipped to America and then, together with the American data, is shipped to Australia. Let us notice that, although the European data lies in America after the first shipping, the first rule clearly specify that tuples from Employee Europe can be disclosed to Australia if and only if filtered on age. For this reason, the shipping operator Ame→Aus is inferred from the first and third constraints for the European data and from the fourth for the American data.

In the next section we will show how this compliance inference mechanism is used by a global optimizer to seek for feasible execution strategies that adhere to data movement restrictions. In particular, the plan exploration phase is steered by optimization rules which specify location conventions for operators and inject shipping operations according to the given constraints.

4.3 Constraint-Aware Plan Enumeration

The previous section showed the main mechanism in data movement constraints enforcement based on the generation of single-view equivalent rewritings of the user (sub) query. The compliant query plans we saw in the last example were only two alternatives among of many potential execution strategies that could be attained for a user query and data movements constraints. We know that an optimizer normally explores various feasible solutions by the way of optimization rules before choosing one on cost-basis and our global optimizer is no exception. In this section, we describe the compliant plan enumeration process that happens within our system, momentarily ignoring the cost dimension associated to the execution strategies. Essentially, we decided to employ a top-down optimizer, widely used in practice, and provide a set of optimization rules concerning the location properties of data and operations. Before delving into the details of plan enumeration, we will firstly take a look at the AND-OR DAG representation [12] which compactly represent all feasible plans for a given query.

A Logical AND-OR graph is a directed acyclic graph (DAG) whose nodes can be partitioned into AND nodes, also called *operation nodes*, and into OR nodes, also named *equivalence nodes*. AND nodes have only OR nodes as children and, in turn, OR nodes have only AND nodes as children. An AND node coincides with an algebraic operation, such as a selection or an union, and it is characterized by the logical operation and its inputs. Operation nodes are called AND nodes because all their children muFt be evaluated in order to evaluate the AND node themselves. An equivalence node, instead, represents the equivalence class of logical expressions that generate the same result or, in other words, an OR node encloses the equivalent rewritGngs for a given subquery. The name OR node originates from the fact that any child node can be evaluated in order to evaluate the OR node parent. In figure 4.5 we show the AND-OR DAG for the following query

SELECT Name, Age **FROM** Employee **AS** Emp;

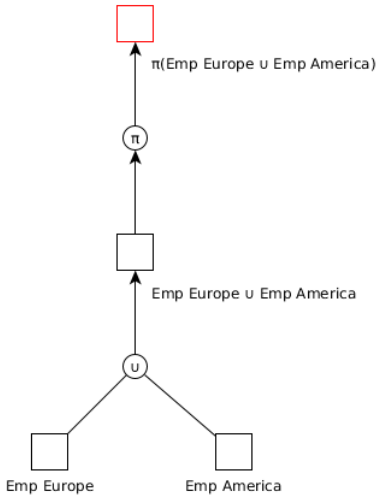

Figure 4.5. Initial Logical AND-OR DAG

The circular nodes enclosing logical operations are the AND nodes, while the rectangular nodes represents the OR nodes. In order to distinguish equivalence nodes from one another, we label them with an expression picked from their relative equivalence classes. The root of the DAG, which represent the final result, is highlighted in red and is always an equivalence node.
Assume now, the optimizer generates an equivalent plan where the projection is pushed down the union. The modified AND-OR graph is shown in Figure 4.6. Since the two expressions π(Emp Europe $\cup$ Emp America) and π(Emp Europe) $\cup$ π(Emp America) are equivalent, they are encoded by the same equivalence node.

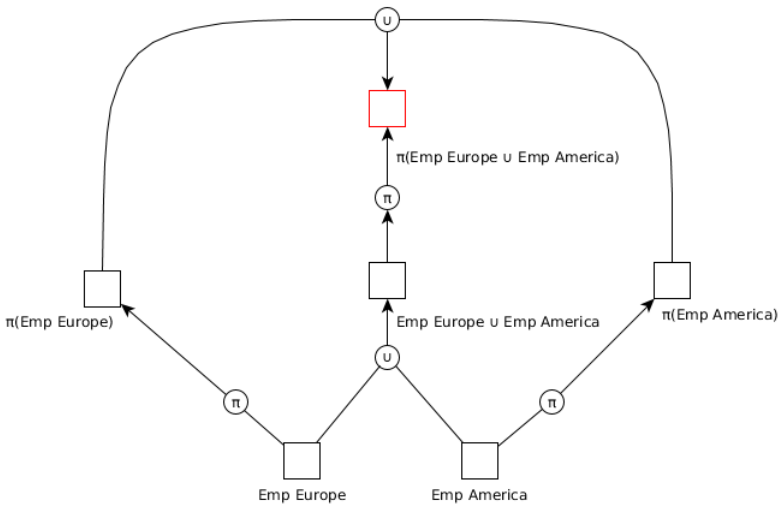

Figure 4.6. Expanded Logical AND-OR DAG

While logical AND-OR DAGs take into account only the logical transformation of

data, physical AND-OR DAGs consider the implementation details of the plans, adapting the definition of AND and OR nodes. Physical equivalence nodes in the new DAG coincide with pairs [*e*,*p*], where *e* is the logical equivalence node and *p* are the physical properties enforced on the result of *e*. In our context, these physical properties correspond to location conventions. There are two types of operation nodes: algorithms, which physically implement a logical operation, and enforcers, which enforce specific physical properties on the data. Since we leave the choice of algorithms selection for operators to subsequent steps, we are only interested in the enforcer category where Shipping Operators falls in. The initial physical AND-OR DAG of the previous query is depicted in Figure 4.7. The dashed-boxes represent

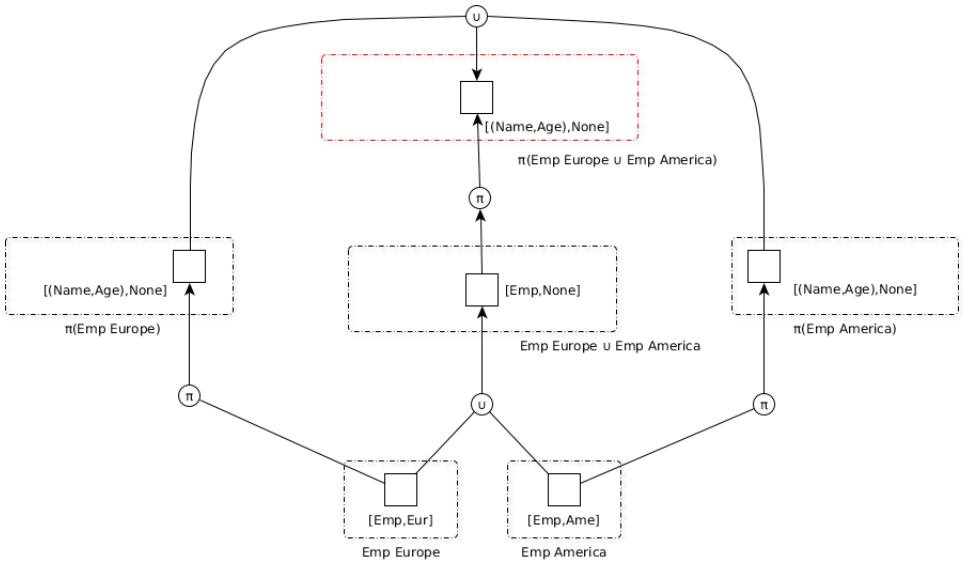

Figure 4.7. Physical AND-OR DAG expanded

the previous logical equivalence nodes while the square inside them are the physical equivalence nodes. There are two location conventions in the above graph: *Eur* for Europe and *Ame* for America. Notice that, initially, only the two sources Emp Europe and Emp America have a location convention, while all the other physical eq. nodes have 'None' as physical property, which basically means that the AND nodes pointing to these OR nodes cannot be currently implemented. The operation nodes that fall outside the dashed-box represent the standard logical operation nodes while the ones that are contained in them are the enforcers (shipping operators).

The goal of optimization in the context of constraint-aware processing is to find an execution plan where all operators are labeled with a location convention. Hence, after having expanded the physical AND-OR DAG, plan selection finally chooses the cheapest plan traversing the graph by starting from the root. Only equivalences nodes with location conventions are taken into account. The process starts selecting the cheapest phyiscal equivalence node in the root and ends when the optimizer reaches the leaves. For each physical equivalence node reached, the optimizer select the cheapest operations (AND nodes) to implement it. For each operation node reached, instead, all its children (OR nodes) are selected.

Given these premises, in the next subsection we describe three type of optimization rules which are used to explore all possible execution strategies.

4.3.1 Location-tailored Optimization Rules

As we already said in the Background Chapter, optimization rules are commonly used in the query optimization process to generate alternative plans through series of logical and physical transformations. The first kind are based on algebraic equivalences between expressions and relational algebra equivalences, such as the join associativity, are an example of such a type. Physical transformations, instead, handle physical operation properties such as the location convention. In this subsection, we provide location-tailored optimization rules, a type of physical rules, whose goal is to expand the AND-OR DAG in two ways: (a) they assign location conventions to operators, creating new physical equivalence nodes, (b) they inject shipping operators, creating new physical operation nodes. These optimization rules are:

View-based rules whose job is to enforce the data movement constraints. Each one is associated to a different constraint;

Propagation rules which propagate location conventions bottom-up the DAG;

Indirect Shipping rules which inject shipping operators where needed.

We will show the application of the above rules on the physical AND-OR DAG of the plan originally shown in Figure 4.5. The physical counterpart is shown in Figure 4.8 .

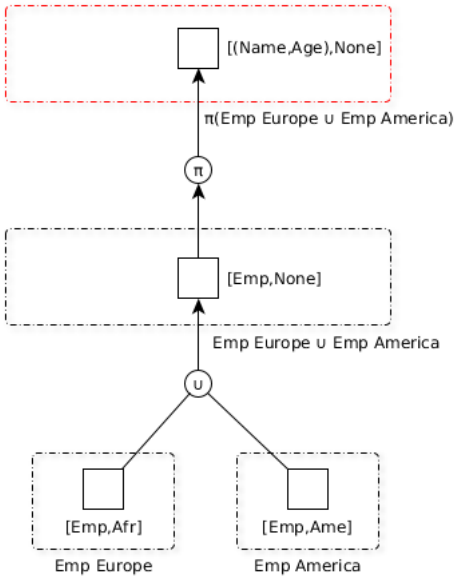

Figure 4.8. Initial Physical AND-OR DAG

View-based Rules

In Section 4.2, we discussed the compliance inference mechanism based on query rewriting using views. If the (sub) query was either equivalent to the view or subsumed by it, then its results could be shipped to the locations specified by the legal set. View-based rules exploit this mechanism in the following way. Given a view and its legal set, a view-based rule detects if it exists a subexpression in the DAG such it 'matches' the view expression by equivalence or subsumption. The matching modalities depends on the underlying query rewriting mechanism one decides to employ. When a match happens, the root of the matched subtree, a physical equivalence node $[e, L_0]$, is replicated for each area $A_1, ..., A_k$ in the legal set, creating the new physical nodes $[e, L_1], ..., [e, L_k]$. Shipping operators are added as many as needed to link the tree root to each of the new nodes. Let us consider the case of where we can ship the whole American partition to Europe

```
CREATE VIEW EmpToEurope WITH LEGAL SET Europe
SELECT * FROM Employee.America;
```

Using the above rules, we will eventually match the physical node [Emp,Ame] and create a new physical equivalence node with the European convention as we see in Figure 4.9.

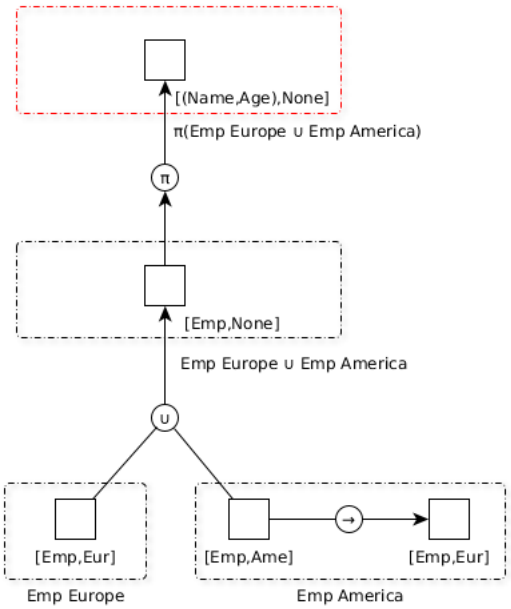

Figure 4.9. Physical DAG after having applied a View-base rule.

Propagation Rules

In order to have a fully specified compliant plan, all the tree nodes should be labeled with a location convention, however, the previous type of rules only matches a limited set of nodes, roots of subtrees associated to views. Propagation rules, as the name suggests, propagate location properties from child to the parent, targeting

a wider group of nodes and, hence, addressing the previous issue. These rules are applied bottom-up since they match nodes which have at least one child labeled with a location convention for then spreading it to the parent. As strict requirement, a parent node can inherit a location property L only if all its children are labeled with L. Trivially, an unary operator always inherit a location convention if its child has it. Back to the plan in fig. 4.9, we see that the union operation has two equivalence nodes in input. The left child has convention Europe while the right one convention America. The union operation cannot take place anywhere since its children have different location properties. However, the right child has a logically equivalent node with the Europe convention, hence, propagation rules can be applied. When this happens, a new union operation whose inputs and output both have Europe as location property is created. The result is depicted in Figure 4.10. As we can notice,

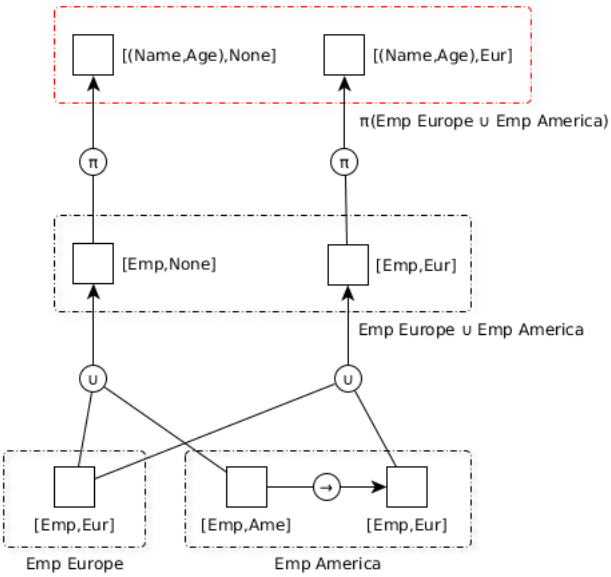

Figure 4.10. Physical DAG after having applied Propagation rules.

the new union equivalence node [Emp,Eur] led to the fire of another propagation rule for the upper projection creating the node [(Name,Address),Eur]. It is easy to notice that now we can generate a compliant plan where the table Emp Europe is shipped to America and all the subsequent operations are carried out there.

Indirect Shipping Rules

The last type of propagation rules is used to ensure that the planner consider different placing of the shipping operator in the tree, not only where a view has been matched. Consider the case of the physical DAG shown in fig. 4.7 where we pushed down the projection past the union and let us focus only of the subtree involving Emp America. Assuming that view-based and propagation rules have already been fired, the result would be the one shown in Figure 4.11. The moment we choose the location for the Emp America table (shipping it or not) all the subsequent operations have to take place there, and, for example, the planner will never explore the possibility

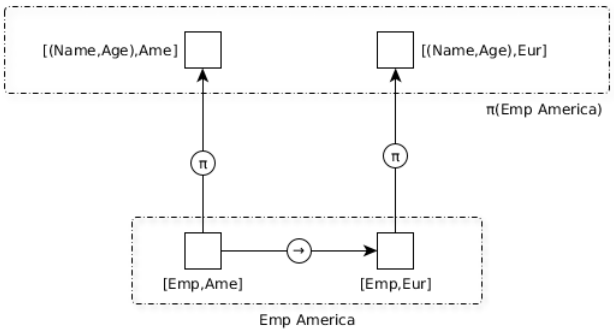

Figure 4.11. Subtree involving Emp America after having applied propagation and view-based rules.

of shipping the result of the projection to another location. Certainly, shipping only projected attributes instead of the whole table is cost-saving but no shipping operator appears in the π(Emp America) node. Indirect propagation rules address this problem creating shipping operators between the nodes; when a new physical eq. node with location convention L_1 is added to the same equivalent set of a physical eq. node with location convention L_2, two shipping operators in opposing direction are created, linking the two nodes. The aftermath of the rule fire is shown in Figure 4.12.

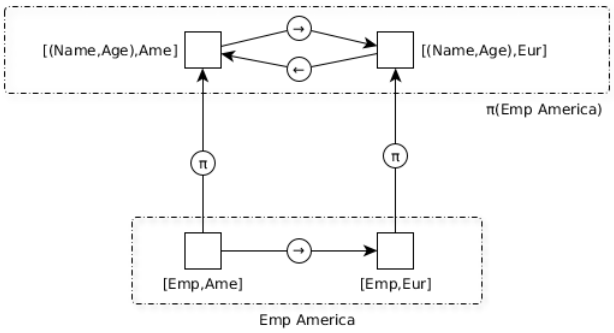

Figure 4.12. Subtree after having fired an Indirect shipping rule.

Chapter 5

Implementation

The previous Chapter 4 addressed the problem of generating compliant execution plans in Geo-Distributed environments in presence of constraints. More specifically, we showed we were able to explore the space of feasible solutions using location-focused optimization rules within a top-down optimizer. In this Chapter, now, we aim our attention at the system implementation of our work, its implementation settings, and some exemplifying cases of user queries and constraints.

The chapter is structured as follow. Section 5.1 describes the data we used and the Geo-distributed environment we set up. In order to carry out multiple experiments, we simulated different data distribution scenarios changing both the data sources involved and the federated schema available to the user. In Section 5.2 we delve into the details of the implementation, describing how we employed the Apache Calcite software, a popular query parser and optimizer framework, to find compliant execution strategies. Lastly, in Section 5.3, we examine some use-cases scenarios and show how our system is able to provide feasible solutions.

5.1 Geo-distributed environment

In Section 2.3, we saw that even if data sources may heavily differ in storing and organizing data, federated database systems address this heterogeneity issue by defining a global schema, a unique logical view of the system which ignores the specific data distributions, and mappings, queries that express which data in the sources will be used to answer the user queries. As for the latter, there are two main approaches to mapping definition: GAV, a table in the global schema is a query over the sources, or LAV, a source table is defined as query over the global schema.

In this section we describe how we set up a simulating Geo-distributed environment comprising of multiple sources administered within a single federated database so that we were able to test our system behavior in different use-cases. More specifically, we decided to change our simulated environment according to: (a) source schemes (b) mappings and global schema. Regardless of the experiment, we preserved the following choices:

- Every data schema (global or local) is based on the logical design of the TPC-H Benchmark [4] introduced shortly

- Along the source dimension, we decided to have a total of five locations, namely Africa, America, Asia, Europe, and Middle East, and have one data source per location.
- In terms of mappings, we opted for the GAV definition since their query rewriting was easy to implement.

With these given premises, we now describe in more details the data employed, the Geo-distributed source settings, and finally the federated schema and mappings.

Data

The TPC-H is a decision support benchmark which includes a business oriented schema and a synthetic data generator algorithm. Entities and relationships in the schema reproduce realistic database settings in the industrial context and they were therefore appropriate for our experimental environment. We designed global and local schemes in our experiments using tables in TPC-H schema as reference and filled them with synthetic data coming from the generator algorithm. TPC-H schema

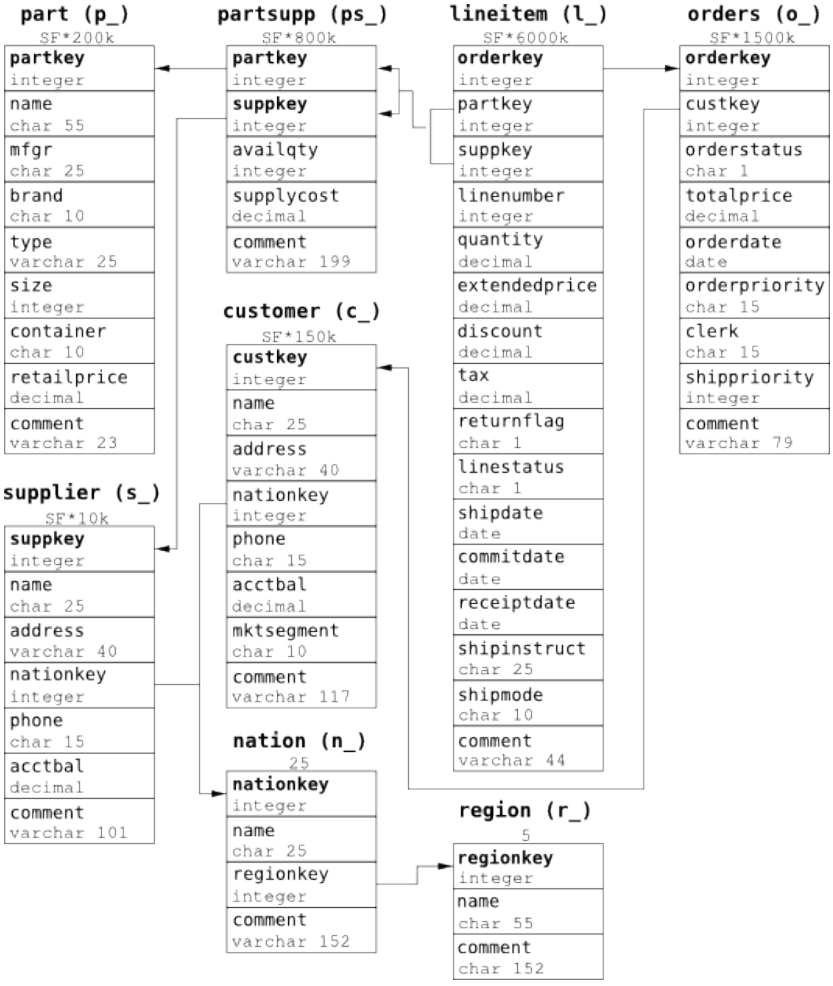

Figure 5.1. TPC-H schema

is depicted in figure 5.1. The table "Region" contains five names in total: Africa, America, Asia, Europe, and Middle East. We chose the locations according to these regions, for this reason we now will use the term region and location interchangeably.

Heterogeneous sources

In order to simulate Geo-distributed data sources, we set up five PostgreSQL servers, one per location, running on the same machine over different ports. An additional server, called the master, was set up to ease the data distribution process. This server stored all the TPC-H tables filled with generated data. Since we wanted to have different schemes for the sources, we wrote a Python script that allowed to distribute data across the five locations in the following way. For a fixed table in the master server, the user could decide to duplicate it at certain location or partition it on region basis. The latter, in particular, is achieved through a series of joins across the TPC-H tables until we reach the Region table. Obviously, not all tables can be partitioned. For example, the table Supplier can be partitioned according to Nation which in turns can be partitioned according to the Region while Part table has no foreign keys chain to table Region and hence cannot be split. When the script runs, it creates the custom schemas for each source and loads the data to the right server. Considering only two locations, an example of Geo-distribution could be the one shown below in Figure 5.2. Both locations have the same copy of the Region table

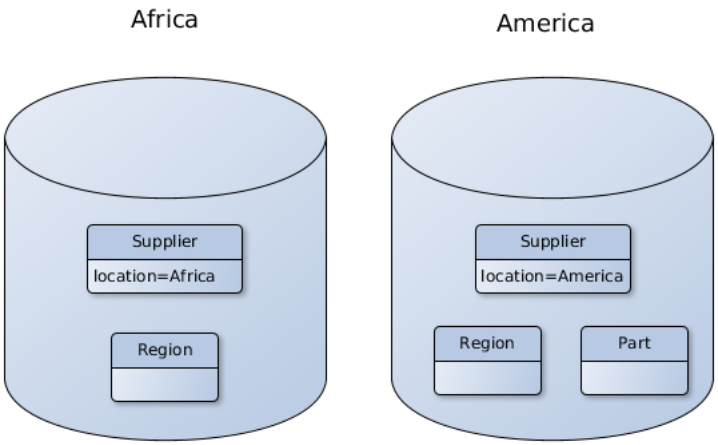

Figure 5.2. An example of Geo-distribution.

while only the America source has the table Part. The table Supplier, instead, has been split on the basis of the location.

Mappings and Global Schema

Following the Geo-distribution of data between the five instances, our system was able to connect to the relational databases using a model file containing the connection credentials. With the connection set, the system read the source schemata and added the source tables to the global schema by default, allowing us to query the single

local tables. In order to define global tables involving more sources, we defined GAV mappings by providing the new table name and the SQL query over the sources. When a new user query is issued on the global schema, the global tables referenced are resolved and substituted by the query definition. Consider the case where we have the following mapping:

$$\text{Customer} \rightarrow \text{Customer.Africa} \cup \text{Customer.America}$$

where the global table Customer is expressed through the union of the American and Africa partition. The user query

```
SELECT Name FROM Customer;
```

is processed and resolved as the following

```
SELECT Name FROM (
        SELECT * FROM Customer.Africa
        UNION ALL
        SELECT * FROM Customer.America
        );
```

Consequently, the initial query tree associated is expanded accordingly before optimization, as we can see in Figure 5.3.

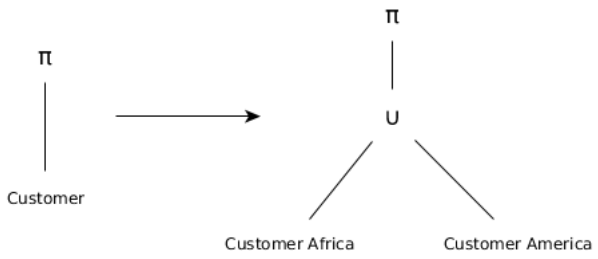

Figure 5.3. Before and after the query with mappings

5.2 System Implementation

In terms of implementation, our system is built in Java on top of Apache Calcite [6], a popular open-source SQL parser and optimizer framework. There are three main reasons on why we decided to employ Calcite in our project. First, Calcite handles data engines heterogeneity providing schema and table abstractions, as well as adapters for connecting to specific database types. Hence, we did not have to worry about differences between database systems implementations. Second, its query processing components, such as cost model, optimization rules and so on, are easily extensible and allowed us to adapt the system to our needs. Last, Calcite is internally used by other Big Data framework such as Flink, Storm, Drill, Hive and many others, and so it makes the adoption of our system much easier. However, location traits and data movement constraints are not concerned in Calcite's optimization process

as it is guided only by the default cost model and logical equivalence rules. We extended the framework in the following three aspects.
Location-awareness for operators: We provided location conventions as physical traits in order to evaluate Geo-distributed plans.
Extended Cost Model: The new cost model considers 'Shipping cost' as fourth dimension. Since standard operations do not move data out of the source, only Shipping Operators increase the shipping cost.
New Optimization Rules: We expanded the optimizer rule set with the new optimization rules described in the previous chapter.

As it regards to the compliance inference mechanism discussed in Section 4.2, we opted for the query rewriting approach discussed in [11] since we wanted to determine in a fast and scalable way if a query could be expressed in terms of views. Furthermore, they handle cases where both the query and the views involve selections, projections, joins, and aggregations which is compatible with our preliminary assumption of SQL-based analytics.

In terms of global optimizer, we employed Volcano [12], a top-down query optimizer which efficiently explores plans thanks to memorization and branch-and-bound pruning. The location-based optimization rules were given to the optimizer together with other rules concerning the logical transformations. Rules were applied starting from the root of the initial query tree until either a satisfactory solution was found or there were no more transformations to apply.

5.3 Example use cases

In this section, we explore some exemplifying use cases of queries and constraints. The cost model we employ is based of four dimensions: # of rows, CPU, I/O, and shipping. They are merged by the following mock cost function

$$\#\text{rowcount} + 2*\text{shipping} + (\text{CPU} + \text{I/O} + 1)/2$$

which gives much more weight to the shipping value. The latter is increased by the shipping operators depending on the bandwidth between locations. Hence, for example, it could be more expensive to send data from Europe to America rather than sending to Africa.

Example #1

As first example, consider the scenario where there are three locations, Africa, America, and Asia, and two tables, Orders and Lineitem. The Order table is partitioned between Africa and America while the Lineitem table lies in Asia. As we can tell from the TPC-H schema, the two tables are linked by the order key attribute. Moreover, assume we have the following constraints:

1. Orders.Africa can freely be shipped to America
2. Orders.Africa can be shipped to Asia only if o_totalprice < 1000
3. Orders.America can freely be shipped to Asia

4. Lineitem.Asia can be shipped to America only if 2016 ≤ l_shipdate ≤ 2019

The extended relational views associated to these data movements constraints are the following:

```
CREATE VIEW C1 WITH LEGAL SET America AS:
SELECT * FROM Orders.Africa;

CREATE VIEW C2 WITH LEGAL SET Asia AS:
SELECT * FROM Orders.Africa WHERE o_totalprice < 1000;

CREATE VIEW C3 WITH LEGAL SET Asia AS:
SELECT * FROM Orders.America;

CREATE VIEW C4 WITH LEGAL SET America AS:
SELECT * FROM Lineitem.Asia WHERE l_shipdate BETWEEN(2016,2019);
```

Lastly, in our user query, we ask for all information on lineitems that have been shipped between the 2017 and 2018 and which are related to orders whose total price is below 500.

```
SELECT * FROM Linteitem
JOIN Orders ON l_orderkey = o_orderkey
WHERE l_shipdate BETWEEN ('01-01-2017','01-01-2018')
AND o_totalprice < 500
```

The query tree of one output of our system is shown in 5.4. It is easy to notice

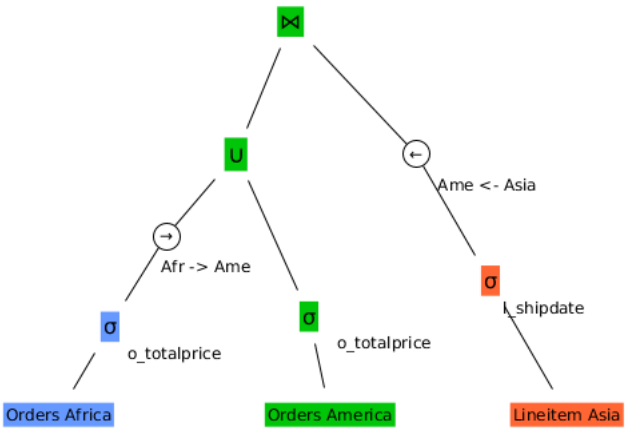

Figure 5.4. First query tree of the first example.

that filters have been pushed down to the sources since it cheaper to carry out the selection locally instead of shipping the whole table and doing the selection later. The shipping operator Afr→Ame is the result of the view-based rule related to constraint C1. Trivially, if the whole table is entitled to be moved from Africa to America, shipping only a subsets of the rows is indeed compliant to C1. The

shipping operator Ame←Asi, instead, comes from the appliance of constraint C4 as its relational view clearly includes the tuples requested by the query.

In the previous example, we consider that the cost of shipping between the locations was equal. Suppose now that it is very expensive to move data outside Asia. In this case the system would output the alternative compliant plan in Figure 5.5. As we could have expected, there are no data movements from Asia to other

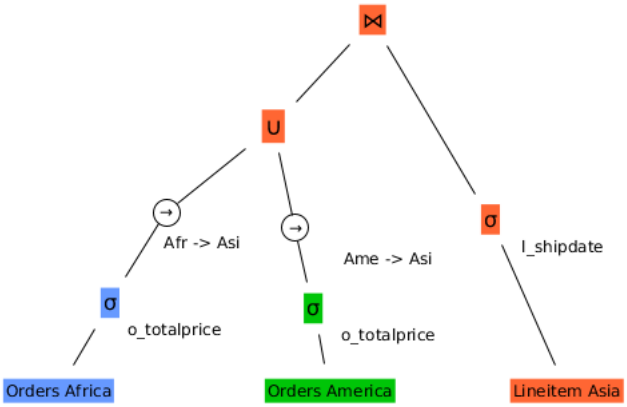

Figure 5.5. Second query tree of the first example.

locations since they are hugely costly and most operations are executed there, even the union. The two shipments from Africa and America to Asia are allowed by the second and third constraints respectively.

Example #2

Suppose now we have the locations Europe and Middle East and two tables Customer and Orders. The latter are partitioned between the locations. We have the following constraints.

```
CREATE VIEW C1 WITH LEGAL SET MiddleEast AS:
SELECT c_name,c_phone,c_address,SUM(o_totalprice)
FROM Customer.Europe JOIN Orders.Europe ON c_custkey=o_custkey
WHERE o_orderpriority < 7
GROUP BY c_name,c_phone,c_address;

CREATE VIEW C2 WITH LEGAL SET Europe AS:
SELECT c_name,c_phone,c_address
FROM Customer.MiddleEast;

CREATE VIEW C3 WITH LEGAL SET Europe AS:
SELECT * FROM Orders.MiddleEast;
```

Let us observe that the first constraint refers to two tables belonging to the same location. Assume the query is the following.

```
SELECT c_name, c_phone, SUM(o_totalprice)
FROM Customer JOIN Orders ON c_custkey=o_custkey
WHERE o_orderpriority = 5
GROUP BY c_name, c_phone
HAVING SUM(o_totalprice) > 500;
```

where we ask for the credentials of customers which spent more than 500 on average priority orders. Two compliant execution strategies are depicted in Figure 5.6. As

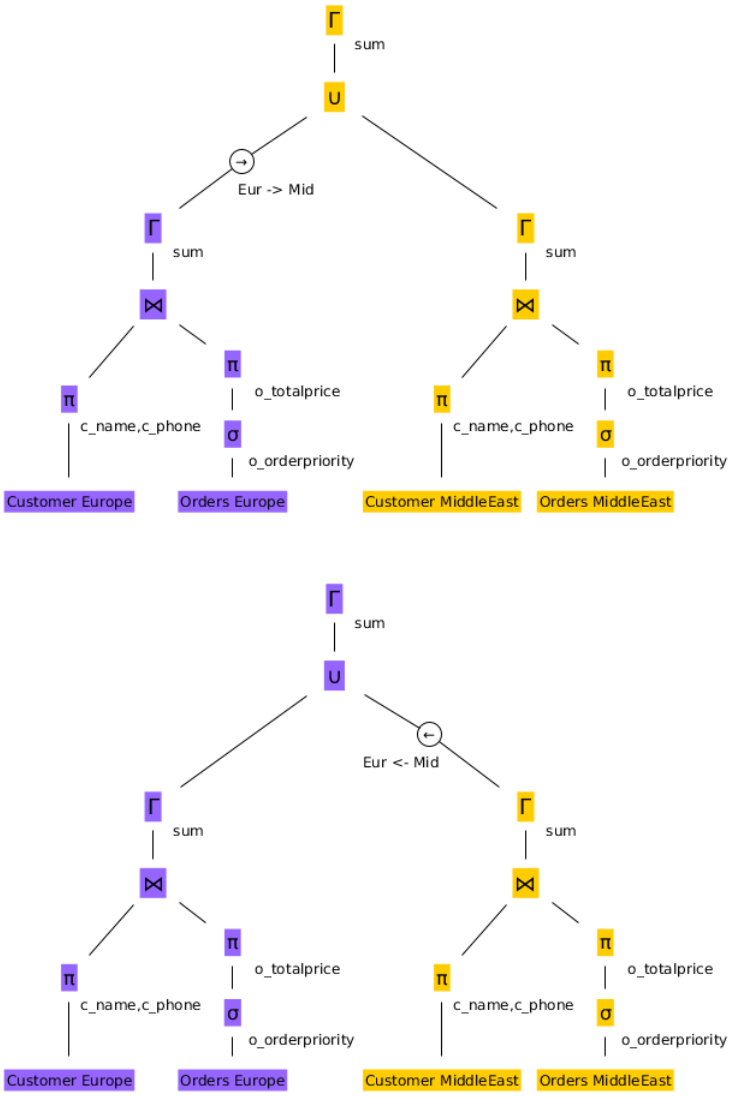

Figure 5.6. Query trees of the second example.

before, most of the operations are pushed down the sources because the cost of shipping heavily impact on the total cost of a plan. The difference between the two lies only on where to execute the last operations. Both the Europe and Middle East are possible final result locations since that constraint C1 allows the generation of

the first plan while constraints C2 and C3 combined allows for the second.

Example #3

As last example, we want to show a case where the standard view matching approach does not work. Consider the tables Supplier, Partsupp, and Lineitem which reside in Europe, Asia, and America respectively. Assume we have the following constraints:

```
CREATE VIEW C1 WITH LEGAL SET Europe AS:
SELECT l_partkey, l_tax FROM Lineitem.America;

CREATE VIEW C2 WITH LEGAL SET America AS:
SELECT * FROM Partsupp.Asia;

CREATE VIEW C3 WITH LEGAL SET Europe AS:
SELECT ps_partkey, ps_suppkey FROM Partsupp.Asia;
```

Observe that no constraint refer Supplier.Europe and, because it is forbidden to ship data from it outside the location boundaries, it implies that regardless of the query involving the three sources, Europe must be the final location. Assume we want to ask the following query:

```
SELECT partkey, suppkey, tax
FROM Lineitem.America, Partsupp.Asia, Supplier.Europe
WHERE l_partkey=ps_partkey AND l_suppkey=ps_suppkey
AND ps_suppkey=s_suppkey
```

Our system will not be able to find a compliant execution plan since it is limited by the standard notion of equivalence in query rewriting and does not know how to 'combine' constraints. In Figure 5.7, for example, it is not possible to infer the

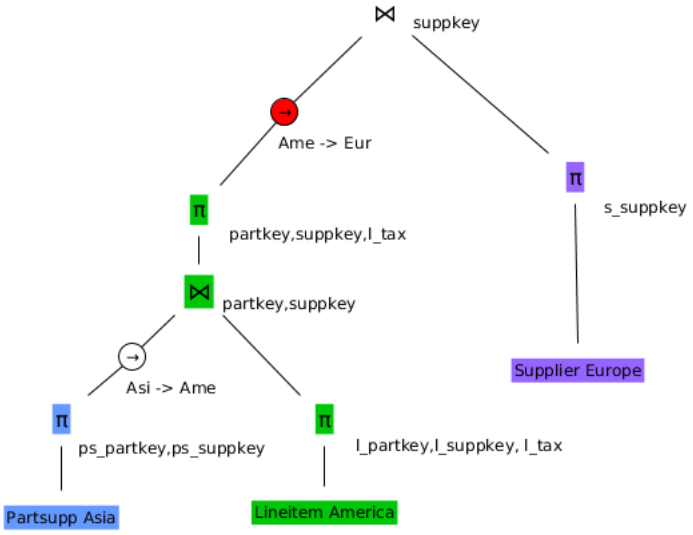

Figure 5.7. Example where query rewriting does not work.

shipping operator highlighted in red even if we are not disclosing more information than the views in the constraints already disclose. The optimizer will try to apply the view-based rules associated to the constraints but it will only match the first local projections in America and Asia. In order to match the projection after the first join, the optimizer should understand that the three constraints can be combined and allow the shipping operation to Europe. Addressing the following problem is one of our future objectives.

Chapter 6

Conclusion & Future work

In this thesis, we studied the problem of analyzing Geo-distributed data that is subjected to data movement constraints. We showed that constraint-aware query processing for Geo-distributed relational data is a feasible approach and the that our solution could support current data analysis frameworks to find compliant execution strategies in presence of strict data movement restrictions. We have provided a formalism for data movements constraints that we believe is intuitive and easy to employ by both constraint administrators and Geo-distributed data frameworks. Through a high level description of constraints based on triples ($\mathcal{D}$, $\mathcal{M}$, $\mathcal{L}$), the constraint administrator of a specific location defines what portion of data should be masked through a series of operations of their choice before being revealed to sites in foreign locations. Mapping these abstract definitions to relational views has the advantage of effortlessly implementing the constraint enforcement mechanism in Geo-distributed data frameworks which already deal with relational data formats. Nevertheless, the concept of "view" itself is present even in non-relational databases such as CouchDB, MongoDB, and Cassandra and using a more specialized query rewriting measure, we may be able to extend the compliance-inference mechanism and deploy it in various data frameworks. Consequently, the set of optimization rules we provided to seek compliant execution strategies can be naturally adapted to these systems. Furthermore, we stacked our implementation on top of Apache Calcite which is already adopted by many data frameworks such as Hive, Flink, Storm, Drill etc. and these software can be easily enabled to work with data flow restrictions between geographically sparsed sources.

To our knowledge, there are no current works on the topic that actually involves the data sovereignty measures with data movements constraints in the optimization process. Vulmiri et al. [16] considers the problems of privacy and data sovereignty in building a data analysis distributed solution which pushes computations to the hedges and optimizes the workflow. However, the authors do not directly address these issues and, apart from avoiding a centralized approach to data processing, they leave it as future objective. Geode [17], an extension of the previous work, includes data sovereignty constraints in the site selection phase for distributed query processing. In particular, they employ integer linear programming for seeking solution that adhere to these constraints which minimize the total bandwidth cost. Nevertheless, as they state in the limitation section, they allow arbitrary queries on

the system and make no attempt to proscribe data movement by queries.

Future directions for our work concern three aspects:

1. extending our implementation to embrace non-relational databases;
2. considering more complex masking operations in constraints;
3. extending the query rewriting mechanism to the respect of constraints.

As we already mentioned, our current implementation in Apache Calcite only works with relational databases and relational views and a reasonable extension would be to augment the types of data engines that can be attached to our system and accept non-relational databases also. In addition, the query rewriting mechanism should be adapted accordingly in order to operate in the new data formats settings.

As regards to the second point, in this work we only focused on masking operations achievable through relational algebra operators. In general, more complex masking transformations are not forced to have a relational algebra counterpart such as shuffling, bucketization, or generalization. The main issue of going beyond these standard operators lies in the compliance-inference mechanism used, as most of the query rewriting approaches are tuned on a limited set of queries in which general user defined functions are ignored. One idea to overcome this issue and generalize our approach to all types of concealment transformations would be to define masking operations in terms of 'masking properties'; these describe the characteristics of the outcome of a masking transformation. As an example, the transformation which shuffles the salaries between people belonging in the same department would still enable the user to query for the sum of salaries grouped by department and hence it is 'sum-preserving' respect to the department attribute. The constraint compliance mechanism should adapted to work only on these properties instead of the definition of the operations.

Finally, the third point would imply to improve the rewriting mechanism and tailoring it to types of constraints. As we saw in an example in Chapter 5, there are cases in which even if there is not an exact match between the views and the query, a compliant plan could be found anyway. However, the system will not be able to find such a compliant solution. The limitation in this approach is that the system does not know how to combine the data movement constraints. Addressing this would mean to understand the effect that more constraints have on the compliance and build a mechanism which combines them accordingly during the plan exploration phase.

In conclusion, we believe we have laid the groundwork for practical use of constraint-aware query processing for Geo-distributed data as we think data sovereignty measures and data movement restrictions will have a huge impact on the future of query processing.

List of Figures

Bibliography

[1] Amazon web service global cloud infrastructure regions and availability zones aws. Available from: `https://aws.amazon.com/about-aws/global-infrastructure/`.

[2] Apache hadoop. Available from: `http://hadoop.apache.org./`.

[3] Google data center locations. Available from: `https://www.google.com/about/datacenters/inside/locations/index.html`.

[4] Tpc-h benchmark. Available from: `http://www.tpc.org/tpch/`.

[5] General data protection regulation (gdpr) (2018). Available from: `https://ec.europa.eu/commission/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules_en`.

[6] Begoli, E., Camacho-Rodríguez, J., Hyde, J., Mior, M. J., and Lemire, D. Apache calcite: a foundational framework for optimized query processing over heterogeneous data sources. In *Proceedings of the 2018 International Conference on Management of Data*, pp. 221–230. ACM (2018).

[7] Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., and Tzoumas, K. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, **36** (2015).

[8] Cruz, J. A. and Wishart, D. S. Applications of machine learning in cancer prediction and prognosis. *Cancer informatics*, **2** (2006), 117693510600200030.

[9] Date, C. J. *An introduction to database systems*. Addison-Wesley (1995).

[10] Davidson, J., et al. The youtube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*, pp. 293–296. ACM (2010).

[11] Goldstein, J. and Larson, P.-Å. Optimizing queries using materialized views: a practical, scalable solution. In *ACM SIGMOD Record*, vol. 30, pp. 331–342. ACM (2001).

[12] Graefe, G. and McKenna, W. J. The volcano optimizer generator: Extensibility and efficient search. In *Proceedings of IEEE 9th International Conference on Data Engineering*, pp. 209–218. IEEE (1993).

[13] Halevy, A. Y. Answering queries using views: A survey. *The VLDB Journal*, **10** (2001), 270.

[14] Kossmann, D. The state of the art in distributed query processing. *ACM Computing Surveys (CSUR)*, **32** (2000), 422.

[15] Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., and Murthy, R. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, **2** (2009), 1626.

[16] Vulimiri, A., Curino, C., Godfrey, B., Karanasos, K., and Varghese, G. Wanalytics: Analytics for a geo-distributed data-intensive world. In *CIDR* (2015).

[17] Vulimiri, A., Curino, C., Godfrey, P. B., Jungblut, T., Padhye, J., and Varghese, G. Global analytics in the face of bandwidth and regulatory constraints. In *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, pp. 323–336 (2015).

[18] Yu, C. T. and Meng, W. *Principles of database query processing for advanced applications*. Morgan Kaufmann Publishers (2004).

[19] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. Spark: Cluster computing with working sets. *HotCloud*, **10** (2010), 95.

[20] Zuech, R., Khoshgoftaar, T. M., and Wald, R. Intrusion detection and big heterogeneous data: a survey. *Journal of Big Data*, **2** (2015), 3.