Automating Component Dependency Analysis for Enterprise Business Intelligence

Completed Research Paper

Kalle Tomingas
Tallinn Uni of Technology
Ehitajate tee 5
Tallinn, Estonia
kalle.tomingas@gmail.com

Tanel Tammet
Tallinn Uni of Technology
Ehitajate tee 5
Tallinn, Estonia
tanel.tammet@ttu.ee

Margus Kliimask
Eliko Competence Center
Teaduspargi 6/2
Tallinn, Estonia
margus.kliimask@gmail.com

Priit Järv
Eliko Competence Center
Teaduspargi 6/2
Tallinn, Estonia
priit.jarv@gmail.com

Abstract

We address common problems in the field of Business Intelligence, Data Warehousing and Decision Support Systems: the complexity to manage, track and understand data lineage and system component dependencies in long series of data transformation chains. The paper presents practical methods to calculate meaningful data transformation and component dependency paths, based on program parsing, heuristic impact analysis, probabilistic rules and semantic technologies. Case studies are employed to explain further data aggregation and visualization of the results to address different planning and decision support problems for various groups of technical and business users.

Keywords: Component dependency analysis, impact analysis, data lineage, data warehouse, rule-based reasoning.

Introduction

Developers and managers are facing similar Data Lineage (DL) and Impact Analysis (IA) problems in complex data integration (DI), business intelligence (BI) and Data Warehouse (DW) environments where the chains of data transformations are long and the complexity of structural changes is high. The management of data integration processes becomes unpredictable and the costs of changes can be very high due to the lack of information about data flows and internal relations of system components. The amount of different data flows and system component dependencies in a traditional data warehouse environment is large. Important contextual relations are coded into data transformation queries and programs (e.g. SQL queries, data loading scripts, open or closed DI system components etc.). Data lineage dependencies are spread between different systems and frequently exist only in program code or SQL queries. This leads to unmanageable complexity, lack of knowledge and a large amount of technical work with uncomfortable consequences like unpredictable results, wrong estimations, rigid administrative and development processes, high cost, lack of flexibility and lack of trust.
We point out some of the most important and common questions for large DW environments (see Figure 1) which usually become a topic of research for system analysts and administrators:

- Where does the data come or go to in/from a specific column, table, view or report?
- When was the data loaded, updated or calculated in a specific column, table, view or report?
- Which components (reports, queries, loadings and structures) are impacted when other components are changed?
- Which data, structure or report is used by whom and when?
- What is the cost of making changes?
- What will break when we change something?

![Figure 1. General scheme of DW/BI data flows.](image)

The ability to find ad-hoc answers to many day-to-day questions determines not only the management capabilities and the cost of the system, but also the price and flexibility of making changes. The dynamics in business, environment and requirements ensure that regular changes are a necessity for every living organization. Due to its reflective nature, the business intelligence is often the most fluid and unsteady part of enterprise information systems.

Obviously, the most promising way to tackle the challenges in such a rapidly growing, changing and labor-intensive field is automation. We claim that efficient automatization in this particular field requires the use of semantic and probabilistic technologies. Our goal is to aid the analysts with tools which can reduce several hard tasks from weeks to minutes, with better precision and smaller costs.

We will draw a short overview of the previous body of research and related problems in the next chapter. We will continue by describing the components of the presented system in the chapter 3. Chapter 4 will describe query parsing, analysis, resolving and semantics extraction techniques. These and the probabilistic rule-based reasoning techniques described in the chapters 5 and 6 forms the backbone of the presented automated solution. We will present illustrative examples in the chapter 7 and chapter 8 presents real life case studies with a functional analytical application. Chapter 9 concludes by presenting the current status of the work and describing further research.

**Related Work**

Impact analysis, traceability and data lineage issues are not new. A good overview of the research activities of the last decade is presented in an article by (Priebe et al. 2011). We can find various research approaches and published papers from the early 1990’s with methodologies for software traceability (Ramesh and Jarke 2001). The problem of data lineage tracing in data warehousing environments has been formally founded by Cui and Widom (2000, 2003). Our recent papers build background to the theory by introducing the Abstract Mapping representation of data transformations and rule-based impact analysis (Tomingas et.al. 1014, 2015).

Other theoretical works for data lineage tracing can be found in (Fan and Poulavassilis 2003) and (Giorgini et al. 2005). Fan and Poulavassilis developed algorithms for deriving affected data items along the transformation pathway (Fan and Poulavassilis 2003). These approaches formalize a way to trace tuples (resp. attribute values) through rather complex transformations, given that the transformations are known on a schema level. This assumption does not often hold in practice. Transformations may be documented in source-to-target matrices (specification lineage) and implemented in ETL tools (implementation lineage). Woodruff and Stonebraker create solid base for the data-level and the operators processing based the fine-grained lineage in contrast to the metadata based lineage calculation in their research paper (Woodruff and Stonebraker 1997).
Other practical works that based on conceptual models, ontologies and graphs for data quality and data lineage tracking can be found in (Vassiliadis et.al. 2002), (Skoutas and Simitsis 2007) and (Widom 2004). De Santana proposes the integrated metadata and the CWM metamodel based data lineage documentation approach (de Santana and de Carvalho Moura 2004). The workflows and the manual annotations based solution proposed by Missier et al. (2008).

Priebe et al. (2011) concentrates on proper handling of specification lineage, a huge problem in large-scale DWH projects, especially in case different sources have to be consistently mapped to the same target. They propose a business information model (or conceptual business glossary) as the solution and a central mapping point to overcome those issues.

Several commercial ETL products are addressing the impact analysis and data lineage problems to some extent (e.g. Oracle Data Integrator, Informatica PowerCenter, IBM DataStage or Microsoft SQL Server Integration Services), but those tools and the dependency analysis performed is often limited to the basic functions of the particular system. Another group of commercial tools is formed by the specialized metadata integration products not related to a particular ETL tool, offering a more sophisticated suite of dependency analysis functionality. The examples are ASG Rochade, Adaptive Metadata Manager, Troux Enterprise Architecture Solution or Teradata Metadata Services (MDS): all of those have their own limitations in terms of available functionality and adapters to other products (Priebe et al. 2011).

Our approach to Impact Analysis and Data Lineage differs from previous work in several aspects. Our aim is to merge technical data lineage (Cui and Widom 2003) with semantic integration approaches (Priebe et al. 2011, Reisser and Priebe 2009), using grammar based methods for metadata extraction from program texts and a probabilistic rule-based inference engine for weight calculations and reasoning approaches (Tammet et al. 2010). We also use the novel and powerful web based data flow and the graph visualization techniques with the multiple view approach (Wang Baldonado et.al. 2000) to deliver the extraction and the calculation of the result to the end-users.

**System Architecture**

We present a working Impact Analysis solution which can be adopted and implemented in an enterprise environment or provided as a service (SaaS) to manage organization information assets, analyse data flows and system component dependencies. The solution is modular, scalable and extendable. The core functions of our system architecture are built upon the following components presented in the Figure 2:

1) Scanners collect metadata from different systems that are part of DW data flows (DI/ETL processes, data structures, queries, reports etc.). We build scanners using our xml-based data transformation language and runtime engine XDTL\(^1\) written in Java.

2) The SQL parser is based on customized grammars, GoldParser\(^2\) parsing engine and the Java-based XDTL engine.

3) The rule-based parse tree mapper extracts and collects meaningful expressions from the parsed text, using declared combinations of grammar rules and parsed text tokens.

4) The query resolver applies additional rules to expand and resolve all the variables, aliases, sub-query expressions and other SQL syntax structures which encode crucial information for data flow construction.

5) The expression weight calculator applies rules to calculate the meaning of data transformation, join and filter expressions for impact analysis and data flow construction.

6) The probabilistic rule-based reasoning engine propagates and aggregates weighted dependencies.

7) The directed and weighted sub-graph calculations, visualization and web based UI for data lineage and impact analysis applications.

8) The MMX\(^3\) open-schema relational database using PostgreSQL or Oracle for storing and sharing scanned, calculated and derived metadata.

---

\(^1\) [http://xdtl.org](http://xdtl.org)

\(^2\) [http://goldparser.org](http://goldparser.org)
Query Parsing and Metadata Extraction

In order to construct the data flows from the very beginning of the data sources (e.g. the accounting system) to the end points (e.g. the reporting system) we have to be able to find and connect both the identical and the related objects in different systems. In order to connect the objects we have to understand and extract the relations from the SQL queries (e.g. ETL tasks, database views, database procedures), scripts (e.g. loader utility scripts) and expressions (e.g. report structure) collected and stored by scanners. In order to understand the data transformation semantics encoded in the query language statements (e.g. insert, update, select and delete queries) and expressions, we have to involve external knowledge about the syntax and grammatical structure of the query language. We use a general purpose Java-based GoldParser engine (Cook 2010) and we have developed a custom SQL grammar written in the Extended Backus-Naur Form (EBNF). Our grammar is based on the ANSI/SQL syntax, but it also contains a large set of dialect-specific notations, syntactical constructions and functions that are developed and trained using large real life SQL query corpuses from the field of data warehousing. The current version of our grammar supports also Teradata, Oracle, Greenplum, Vertica, Postgres and MsSQL dialects.

Grammar-based parsing functionality is built into the scanner technology. A configurable “parse” command brings semi-structured text parsing and information extraction into the XDTL data integration environment. As the result of SQL parsing step (No2 in Figure 2) we get a large parse tree with every SQL query token assigned a special disambiguated meaning by the grammar.

In order to convert different texts into the tree structure, to reduce tokens and to convert the tree back to the meaningful expressions (depending on search goals), we use a declarative rule set presented in the Json format, combining token and grammar rules. A configurable grammar and a synchronized reduction

---

3 http://mmxframework.org
rule set makes the XDTL parse command suitable for general purpose information extraction and captures the resource hungry computation steps into one single parse-and-map step with the flat table outcome. The Parse Tree Mapper (No3 in Figure 2) uses three different rule sets with more than eighty rules to map the parse tree to data transformation expressions:

- Stopword list and grammar rules are used to flush the buffer and start the token collection to construct a new expression;
- Mapword list and grammar rules are used to map the collected expressions to meaningful items (e.g. sources, targets, data transformations, joins and filters);
- Tagword list and grammar rules are used to tag special meaningful tokens in expressions to identify all the database object references (e.g. tables, views, columns, functions and constants).

After extraction and mapping of each SQL query statement into a series of expressions we execute the SQL Query Resolver (No4 in Figure 2) which contains a series of functions to resolve the SQL query structure:

- Solve source and target object aliases to full qualified (schema name + object name) object names;
- Solve sub-query aliases to context specific source and target object names;
- Solve sub-query expressions and identifiers to expand all the query level expressions and identifiers with fully qualified and functional ones;
- Solve syntactic dissymmetry in different data transformation expressions (e.g. insert statement column lists, select statement column lists and update statement assign list etc.).

The following list describes the fields and measures of the parse results, which are sources for the following calculation steps. We also use the defined field names in rules and metrics definitions in the next chapters:

- **StrList** - String constant list used in each expression;
- **NbrList** - Number constant list used in each expression;
- **FncList** - Function list used in each expression;
- **IdCount** - Column identifiers count in each expression;
- **StrCount** - String constant count in each expression;
- **NbrCount** - Number constant count in each expression;
- **FncCount** - Functions count in each expression;
- **PrdCount** - Predicate operators count in each expression.

### Data Transformation Weight Calculation

Data structure transformations are parsed, extracted from queries and stored as formalized, declarative mappings in the system. To add additional quantitative measures to each column transformation or column usage in the join and filter conditions we evaluate each expression and calculate transformation and filter weights for those.

Expression Weight Calculation (No5 in Figure 2) is based on the idea that we can evaluate column data “transformation rate” and column data “filtering rate” using data structure and structure transformation information captured from SQL queries. Such a heuristic evaluation enables us to distinguish columns and structures used in the transformation expressions or in filtering conditions or both, and gives probabilistic weights to expressions without the need to understand the full semantics of each expression. We have defined two measures that we further use in our probabilistic rule system for deriving new facts:

**Definition 1.** A primitive data transformation operation \(O(X,Y,M_1,F_1,W)\) is a data transformation between a source column \(X\) and a target column \(Y\) in a transformation set \(M\) (mapping or query) having expression similarity weight \(W\) and having conditions set \(F_1\).

**Definition 2.** Column transformation weight \(W\) is based on the similarity of each source column and column transformation expression: the calculated weight expresses the source column transfer rate or strength. The weight is calculated on scale \([0,1]\) where 0 means that the data is not transformed form...
source (e.g. constant assignment in a query) and 1 means that the source is directly copied to the target
(no additional column transformations).

Definition 3. Column filter weight $W_f$ is based on the similarity of each filter column in the filter
expression and the calculated weight expresses the column filtering rate or strength. The weight is
calculated on scale $[0,1]$ where 0 means that the column is not used in the filter and 1 means that the
column is directly used in the filter predicate (no additional expressions).

The general column weight $W$ algorithm in each expression for $W_t$ and $W_f$ components is calculated as a
column count ratio over all the expression component counts (e.g. column count, constant count, function
count, predicate count).

$$W = \frac{IdCount}{IdCount + FncCount + StrCount + NbrCount + PrdCount}$$

All the used expression column weight calculation figures are listed and defined in the previous chapter.
The counts are normalized using the $FncList$ evaluation over a positive function list (e.g. CAST, ROUND,
COALESCE, TRIM etc.). If $FncList$ member is in a positive function list then the normalization function
reduces the according component count by 1 to “pay a smaller price” in case the function used does not
have a significant impact to column data.

Example 1. Column transformation weight calculation using expression measures:

- When a column data is copied directly from the source column to the target column in the SQL DML
  statement (e.g. insert-select, update) then the data transformation weight is 1. The following simple
  query q1: "INSERT INTO T2 (c1) SELECT c1 FROM T1" interpreted as the data transformation
  operation $O$ with the weight $1$: $O(T1.c1,T2.c1,q1,null,1)$.
- When a source column is not defined and a data (e.g. constant or function) is assigned to the target
  column in the SQL DML statement then the data transformation weight is 0 and the direct relation
  between the source and the target columns does not exist. The following simple query q2: "INSERT
  INTO T2 (c1) SELECT '10'" interpreted as the data transformation operation $O$ with the weight
  $0$: $O(null,T2.c1,q2,null,0)$.
- When a column data is be mapped from the source column to the target column in the SQL DML
  statement column expression then the data transformation weight depends on complexity of
  expression and the weight is between 0 and 1.
- The following expression samples and the calculated weights for each source-target column pair
  illustrates the variation of the data transformations:

  q3: CAST(T1.LogDate AS DATE) as Request_Date $\Rightarrow 1.0$
  q4: T1.First_Name||' '||T1.Last_Name as Full_Name $\Rightarrow 0.67$
  q5: MIN(T1.Balance_Amt) as Min_Balance_Amt $\Rightarrow 0.5$
  q6: SUM(ZEROIFNULL(T1.Payment_Amt)) as Sales_Amt $\Rightarrow 0.33$
  q7: CASE WHEN T1.Feature_Id is not null THEN 'Y' ELSE 'N' END as Dynamic_Ind
      $\Rightarrow 0.17$

- The previous expression q7 contains parts and measures like $IdCount$: 1 (T1.Feature_Id), $FncCount$: 2
  (Case,WhenThen) and $StrCount$: 3 (null,Y,N). When using those values and the weight $W$ definition
  then we can calculate the column pair operation $O(T1.Feature_Id,T2. Dynamic_Ind,q7,null,0.17)$
  weight in the expression q8 like this:

  $$W = \frac{1}{1+2+3+0+0} = \frac{1}{6} = 0.16667 \cong 0.17$$
Rule System and Dependency Calculation

Rule System

The defined figures, operations and weights are used with combinations of declarative inference rules with probabilistic reasoning to calculate the possible relations and dependencies between data structures and software components. Applying the rule system to extracted query graphs we calculate and produce a full dependency graph that is used for data lineage or impact analysis.

The basic operations used in the rules for the dependency graph calculations are following:

- The primitive data transformation is the elementary operation between the source column X and the target column Y in the query mapping id set M1 with the filter condition set F1 and the transformation weight Wt (see Definition 1) - O(X,Y,M1,F1,Wt);
- The function member(X,F1,Wt) used in the filter impact calculation rule to detect that if column X is the member of the filter id set F1 with the filter weight Wt;
- The function disjoint(M1,M2) used in the impact aggregation rule to detect that two query mapping id sets M1 and M2 are the disjoint sets. The disjoint function is needed to aggregate the data transformation relations and the weights when more than one path from the different queries connects the same column pairs;
- The function parent(X0,X1) used in the parent aggregation rule to detect that table X0 is the owner or parent object for column X1. The parent function is needed to aggregate all the column level relations and the weights between two tables to the table level impact relation and the weight;
- The function union(M1,M2) used to calculate the impact relations over two disjoint query mapping id sets M1 and M2, and the function returns the distinct id lists of two sets (M1 and M2);
- The function sum(W1,W2) used to calculate the aggregated impact relations weight when the basic operations are the disjoint sets and the part of the independent queries. The weight calculation based on non-mutually exclusive event probabilities (two independent queries means possible overlap between two events) and calculated as probability sum of W1 and W2: sum(W1,W2)=(W1+W2)-(W1*W2);
- The function avg(W1,W2) used to calculate the parent impact weight when the basic operations having same the same parent structures. The weight calculation based on the average sum and calculated as the arithmetic mean of W1 and W2: avg(W1,W2)=(W1+W2)/2;

The main inference rules with the basic operations and the weighs for the dependency graph calculations are following:

The inference rules with the basic operations and the weighs for the dependency graph calculations are the following:

- The basic impact calculation rule for the operation O with no additional filters produces the impact predicate I (R1):
  \[ O(X,Y,M1,F1,Wt) \Rightarrow I(X,Y,M1,F1,Wt); \]
- The basic impact calculation rule for the operation O with a related filter condition produces the impact predicate I with multiplied weights (R2):
  \[ O(X,Y,M1,F1,Wt) \& member(X,F1,Wf) \Rightarrow I(X,Y,M1,F1,Wt*Wf); \]
- The transitivity rule is used to calculate the sequences of the consecutive impact relations (R3):
  \[ I(X,Y,M1,F1,W1) \& I(Y,Z,M2,F2,W2) \& disjoint(M1,M2) \Rightarrow I(X,Z,union(M1,M2),union(F1,F2),W1*W2); \]
- The column aggregation rule is used when multiple different paths from the different queries connect the same columns: calculate the impact relations with aggregated query id-s and the aggregated weights (R4):
  \[ I(X,Z,M1,F1,W1) \& I(X,Z,M2,F2,W2) \& disjoint(M1,M2) \Rightarrow I(X,Z,union(M1,M2),union(F1,F2),sum(W1,W2)); \]
The parent aggregation rule is used when multiple different impact relations connect the column pairs of the same tables: calculate the table level impact relations with aggregated query id-s and aggregated weights (R5):

\[
I(X_1,Y_1,M_1,F_1,W_1) \land I(X_2,Y_2,M_2,F_2,W_2) \land \text{parent}(X_0,X_1) \land \text{parent}(X_0,X_2) \land \text{parent}(Y_0,Y_1) \\
\land \text{parent}(Y_0,Y_2) \implies I(X_0,Y_0,\text{union}(M_1,M_2),\text{union}(F_1,F_2),\text{avg}(W_1,W_2)).
\]

**Fact Inference**

Building the knowledge base of object relations is done by applying the inference rules to the existing relations iteratively. Rules R1 and R2 apply to the facts that are prior knowledge and remain unchanged during the inference step, therefore R1 and R2 need to be evaluated only once to perform a conversion from 'O()' facts to 'I()' facts. Rules R3, R4 and R5 apply to the predicates created during the inference process and therefore need to be iterated over, until no new predicates are created.

We build an in-memory data structure for efficient application of iterative inference from the 'I()'-facts. Each fact is kept in a record, with several tree indexes for record lookup by object identifier. The relations contain sets of data flow mappings that the fact is based on, which are stored in variable length arrays ('M-set' records). We also store the set of siblings for the objects in the relation to efficiently evaluate expressions like parent(Xo, XI) & parent(Xo, X2) ('C-set' records).

An 'M-set' record is a variable sized array containing all the mapping identifiers originating from 'O()' facts. The inference procedure is as follows:

derive I() facts using rules R1 and R2 (generation 0)

\[
i := 0
\]
repeat
    infer I() facts using R3, R4 and R5 (generation i+1)
\[
i := i + 1
\]
until no more new facts were generated

Each iteration step takes the set \(\{x|x.generation = i}\) \(\times\) \(\{y.generation \leq i\}\) as input and produces the set \(\{z|z.generation = i + 1\}\), where \(x,y,z\) denote 'I()' facts.

As an example that makes use of most of the features of the in-memory knowledge base representation, consider the algorithm for the parent aggregation rule (R5):

\[
i := \text{current generation}
\]
\[
\text{Seen} := \emptyset
\]
for each I() predicate where I.generation = i and I \(\notin\) Seen
\[
M := I.M
\]
\[
W := I.W
\]
\[
c := 1
\]
for each I’() predicate where I’.X is a sibling of I.X
    if I’X \preceq I.X and I’.Y is a sibling of I.Y
    \[
    M := M \cup I’.M \\
    W := W + I.W \\
    \text{Seen} := \text{Seen} \cup I’
    \]
    c := c + 1
    if c > 1
    create the aggregate \(I_p\) \((X.parent, Y.parent, M, W/c)\)
    \(I_p\).generation := i+1
    Seen := Seen \cup I’

Note that the algorithm skips pairwise examination of predicates and immediately groups all facts where the source and target objects respectively have common parents to produce the aggregate fact. The tree index is used to retrieve predicates with given object identifiers, while the sibling sets are directly linked to the predicate records. The amortized time complexity of this algorithm is in \(O(n^2)\) if we consider set membership tests to be in \(O(1)\) and union operation in \(O(n)\); here \(n = |\{x|x.generation \leq i\}|\).

The prototype implementation inferred 32900 'I()'-'facts from 24671 data flow mappings and 7484 filter mappings in 96 seconds on a desktop-class computer. This problem size corresponds to a real-life data-warehouse setting, making the chosen approach applicable in practice.
Dependency Score Calculation

We can use the derived dependency graph to solve different business tasks by calculating the selected component(s) lineage or impact over available layers and details chosen details. Business questions like: “What reports are using my data?”, “Which components should be changed or tested?” or “What is the time and cost of change?” are converted to directed sub-graph navigation and calculation tasks. The following definitions add new quantitative measures to each component or node in the calculation. We use those measures in the UI to sort and select the right components for specific tasks.

Definition 4. Local Lineage Dependency % (LLD) is calculated as the ratio over the sum of the local source and target Lineage weights \( W_t \).

\[
LLD = \sum \frac{source(W_t)}{source(W_t) + target(W_t)}
\]

Local Lineage Dependency 0 % means that there are no data sources detected for the object. Local Lineage Dependency 100 % means that there are no data consumers (targets) detected for the object. Local Lineage Dependency about 50 % means that there are equal numbers of weighted sources and consumers (targets) detected for the object.

Definition 5. Local Impact Dependency % (LID) is calculated as the ratio over the sum of local source and target impact weights \( W(W_t, W_i) \).

\[
LID = \sum \frac{source(W)}{source(W) + target(W)}
\]

Local Impact Dependency 0 % means that there are no dependent data sources detected for the object. Local Dependency 100 % means that there are no dependent data consumers (targets) detected for the object. Local Impact Dependency about 50 % means that there are equal numbers of weighted dependent sources and consumers (targets) detected for the object.

Definition 6. Global Dependency Count (GDC) is the sum of all source and target Lineage and Impact relations counts (GDC=GSC+GTC).

The Global Dependency Count is a good differential metric that allows us to see clear distinctions in the dependencies of each object. We can take the GDC metric as a sort of “gravity” of the object that can be used to develop new rules, to infer the time and cost of changes of object(s) (e.g. database table, view, data loading programs or report).

A Motivating Example

The table 1 below presents four SQL DML queries as an example. The queries are parsed to abstract mappings (M1..M4) with all the available source and target tables (T1..T9). Each mapping has data transformation elements (m1..m4,2), joins (j1.1..j4.1) and filter conditions (f1.1..f4.1) according to the query structure and expressions. All the source and target tables have columns (t1.1..t9.2) according the usage in query expressions. Additional transformation key-value constraints and conditions (c1, c2, c3) are extracted from the query expressions when possible. The result of the parsed and processed query text is the directed query graph.

<table>
<thead>
<tr>
<th>SQL query parsed to mapping M1</th>
<th>SQL query parsed to mapping M2</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT INTO T4(t4.1, t4.2, t4.3) SELECT T1.t1.1, coalesce(T1.t1.2, '10'), T1.t1.3 FROM T1 JOIN T2 ON T2.t2.1 = T1.t1.2 WHERE T2.t2.2 = '10' AND T1.t1.2 = 'A'</td>
<td>INSERT INTO T5 (t5.1, t5.2) SELECT T4.t4.1, coalesce(T1.t4.2, '10') FROM T4 JOIN T3 ON T4.t4.1 = T3.t3.1 WHERE T3.t3.2 = '10' AND (T1.t4.2 = '10' OR T1.t4.2 is null)</td>
</tr>
</tbody>
</table>
### Table 1. SQL query examples that parsed to mapping M3..M4.

The following Figure 3 presents the query transformation graph with all the source and target data structures at column and table level, obtained by parsing and resolving the queries above:

![Parsed query graph](image)

**Figure. 3.** Parsed query graph.

The expression weight calculation step (No 5 in Figure 2) produces probabilistic weights on scale of \([0,1]\) to all the graph relations that can be derived from the formalized mapping structures.

From the result of rule-based reasoning (No 6 in Figure 2) we calculate the full dependency graph with all the possible inferred relations and their weights. The following component impact analysis and data lineage tasks will be handled as sub-graph navigation and calculation problems. Finally the constructed dependency graph is stored in an open schema metadata database MMX (No 8 in Figure 2) for different applications.

The component impact graph (Figure 4) solves several kinds of component and data structure dependency problems like „What happens when changed?” or „How many reports will be broken when changed?“.

The Data Lineage Graph (Figure 5) solves other types of data lineage and data flow management problems like „Where does the data come from?“, „What reports are using the data?“ or „How are the report column values combined and calculated?“.

---

```sql
<table>
<thead>
<tr>
<th>SQL query parsed to mapping M3</th>
<th>SQL query parsed to mapping M4</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT INTO T4(t4.1, t4.2, t4.3)</td>
<td></td>
</tr>
<tr>
<td>SELECT T6.t6.1, ‘20’,</td>
<td></td>
</tr>
<tr>
<td>case when T6.t6.2 = ‘B’</td>
<td></td>
</tr>
<tr>
<td>then 20 else 0 end</td>
<td></td>
</tr>
<tr>
<td>FROM T6</td>
<td></td>
</tr>
<tr>
<td>JOIN T7 ON T6.t6.3 = T7.t7.1</td>
<td></td>
</tr>
<tr>
<td>WHERE T6.t6.2 = ‘B’</td>
<td></td>
</tr>
<tr>
<td>AND T7.t7.2 = ‘B’</td>
<td></td>
</tr>
<tr>
<td>INSERT INTO T9 (t9.1, t9.2)</td>
<td></td>
</tr>
<tr>
<td>SELECT T4.t4.2,</td>
<td></td>
</tr>
<tr>
<td>coalesce(T4.t4.3*100, 100)</td>
<td></td>
</tr>
<tr>
<td>FROM T4</td>
<td></td>
</tr>
<tr>
<td>JOIN T8 ON T8.t8.1 = T4.t4.1</td>
<td></td>
</tr>
<tr>
<td>WHERE T4.t4.2 = ‘20’</td>
<td></td>
</tr>
</tbody>
</table>
```
The component impact graph (Figure 4) illustrates the impact dependencies, directions and weights calculated by our formulas and rules for the example queries in the Table 1. The solid lines stem from the O relations, i.e. those derived directly from the query expressions. The dashed lines are based on the parent aggregation rules with the weights calculated as averages over the column level weights. In order to maintain the readability of the diagram we do not show the results of the transitivity rule applications.

The data lineage graph (Figure 5) illustrates the data lineage and flow dependencies, directions and weights calculated by the previously defined formulas and rules. The solid lines stem from the data transformation operations and weights calculated from the query expressions. The dashed lines represent transitive and aggregate relations on the column and the table level. The weights are calculated as an average (parent aggregation rule) or multiplication (transitivity) over the query expression weights. The condition notes on the figure (c1..c3) are derived from the query filters or the constant assignment list in order to add additional semantics to the transitive dependency calculation. When we connect the condition pairs from different queries that share the same meaning (e.g. c2:t.4.2='10' and c3:t.4.2='10') then we can calculate the conditional transitivity relations (from T1 to T5 and T6 to T9) that reflect the real data flows more precisely.
Real Life Case Studies

The previously described architecture and algorithms have been used to implement an integrated toolset dLineage (http://dlineage.com). Both the scanners and web-based tools of dLineage have been enhanced and tested in real-life projects and environments to support several popular DW database platforms (e.g. Oracle, Greenplum, Teradata, Vertica, PostgreSQL, MsSQL, Sybase), ETL tools (e.g. Pentaho, Oracle Data Integrator, SQL scripts and different data loading utilities) and BI tools (e.g. SAP Business Objects, Microstrategy). The dLineage dynamic visualization and graph navigation tools are implemented in Javascript using the d3.js graphics libraries.

We have tested our solution during two main case studies involving a thorough analysis of large international companies in the financial and the energy sectors. Both case studies analyzed thousands of database tables and views, tens of thousands of data loading scripts and BI reports. Those figures are far over the capacity limits of human analysts not assisted by the special tools and technologies.

![Data lineage graph with dependencies between DW tables, views and reports.](image)

The real-life dependency graph examples (Figure 6 and Figure 7) illustrate automated data collection, parsing and visualization tasks implemented by one-two persons in a few days during the pilot projects. The toolkit requires only the setup and configuration tasks to be performed manually. The rest will be done by the scanners, parsers and the calculation engine. The end result consists of data flows and system component dependencies visualized in the navigable and drillable graph or table form. The result can be...
viewed as a single column, table or report dependency network or the full scale overview graph with all the system dependencies - tens of thousands nodes – visible on one screen.

The Enterprise Dependency Graph example (Figure 7) is an illustration of the complex structure of dependencies between the DW storage scheme, access views and user reports. The example is generated using only 3-4 data lineage layers (sources and ETL layers are not present here) and has details at object level (not at column level). At the column and report level the full data lineage graph would be about ten times bigger and too complex to visualize in a single picture. The following graph from DW tables to views and user reports presents about 5,000 nodes (tables, views, reports) and 20,000 links (data transformations in views and queries) on a single image:

*Figure 7.* Data lineage graph with dependencies between DW tables, views and reports.
Conclusions and Future Work

We have presented several algorithms and techniques for quantitative impact analysis, data lineage and change management. The focus of these methods is on automated analysis of the semantics of data conversion systems followed by employing probabilistic rules for calculating chains and sums of impact estimations. The algorithms and techniques have been successfully employed in several large case studies, leading to practical data lineage and component dependency visualizations.

We are planning to continue this research by considering a more abstract, conceptual/business level in addition to the current physical/technical level of data representation.

References


