# Data Integration Patterns for Data Warehouse Automation

Kalle Tomingas[1], Margus Kliimask[2], and Tanel Tammet[1]

[1] Tallinn University of Technology, Ehitajate tee 5, Tallinn 19086 Estonia
[2] Eliko Competence Center, Teaduspargi 6/2, Tallinn 12618 Estonia

**Abstract.** The paper presents a mapping-based and metadata-driven modular data transformation framework designed to solve extract-transform-load (ETL) automation, impact analysis, data quality and integration problems in data warehouse environments. We introduce a declarative mapping formalization technique, an abstract expression pattern concept and a related template engine technology for flexible ETL code generation and execution. The feasibility and efficiency of the approach is demonstrated on the pattern detection and data lineage analysis case studies using large real life SQL corpuses.

**Keywords:** data warehouse, etl, data mappings, template based sql generation, abstract syntax patterns, metadata management.

## 1 Introduction

The delivery of a successful Data Warehouse (DW) project in a heterogeneous landscape of various data sources, limited resources, and lack of requirements, an unstable focus and budgeting constraints is always challenging and risky. Many long-term DW project failures are related to the requirements and reality mismatch between available data, defined needs and information requirements for decision making [5]. Extract, transform and load (ETL) is a database usage process widely used in the data warehouse field. ETL involves extracting data from outside sources, transforming it to fit operational needs and loading it into the end target: a database or a Data Warehouse.

Mappings between source and target data structures or schemas are the basic specifications of data transformations. Mappings can be viewed as metadata capturing the relationships between information sources and targets. Mappings document the decisions for information structuring and modeling [12]. They are used for several different goals in DW processes: writing a specification for ETL programmers, generating a transformation query or program that uses the semantics of the mapping specification (e.g., a SQL query that populates target tables from source tables), providing metadata about relationships between structures or schemas, providing metadata about data flows and origin sources [4].

Programming mappings in the ETL environment involves writing special database loading scripts (e.g. Oracle Sql*Loader, MSSQL Bulkload, Teradata Fastload,



N. Bassiliades et al. (eds.), *New Trends in Database and Information Systems II*,
Advances in Intelligent Systems and Computing 312, DOI: 10.1007/978-3-319-10518-5_4

Postgresql Copy etc.) and SQL queries (i.e. select, insert, update and delete statements) which are incremental, iterative, time consuming and routine activities. The manual programming of the data loadings is test-and-error based and not too efficient in case there is no support from the environment and no methodology. Manual scripting and coding of SQL gives high flexibility but backfires in terms of efficiency, complexity, reusability and maintenance of data loadings [7]. The execution and optimization of existing loading programs can be a very complex and challenging task without access to the full dependencies and intelligent machinery to generate optimized workflows [3], [1].

The processes of creation, integration, management, change, reuse, and discovery of data integration programs are not especially efficient without the dependencies and semantics of data structures, mappings and data flows. The creation and management of human- and machine readable documentation, impact analysis (IA) and data lineage (DL) capabilities has become critical for maintaining complex sequences of data transformations. We can control the risks and reduce the costs of dynamic DW processes by making the data flows and dependencies available to developers, managers and end-users.

The paper describes a methodology for formalizing data transformations to an extent that allows us to decouple unique mapping instances from reusable transformation patterns. We demonstrate handling and storing declarative column mappings join and filter predicates in a reusable expression pattern form. We use the Apache Velocity template engine and predefined scenario templates to handle reusable procedural parts of data transformations. We show how the combination of those techniques allows us to effectively construct executable SQL queries and generate utility loading scripts. We also take a look at what has been previously done in the ETL and DW automation field.

In the third chapter we present our open-source architecture of knowledge and metadata repository (MMX[1]) with the related data transformation language and runtime environment (XDTL[2]) which is used as the technology stack for our metadata-driven Data Warehousing process. In the fourth chapter we describe the decoupling of procedural and declarative parts of data mappings and the template-based SQL construction technique. We introduce a case study of Abstract Syntax Pattern (ASP) discovery from a real life DW environment in the fifth chapter and two data lineage analysis case studies in the sixth chapter.

## 2 Related Work

The roles and functions of general programming and ETL tools as well as the relations between manual scripting and script generation are discussed in [7]. The first generation ETL tools were similar to procedural programming or scripting tools, allowing a user to program specific data transformations. The concept of mapping was used for initial specification purposes only.

[1] www.mmxframework.org
[2] www.xdtl.org

Modern ETL tools - e.g. Informatica PowerCentre, IBM WebSphere DataStage or Oracle Data Integrator - exploit the internal mapping structure for transformation design and script or query generation purposes. In addition to specific ETL technologies there exist the general purpose schema mapping tools that allow discovery and support documenting the transformations or generating transformation scripts (XSLT transformation between XML-schemas, XQuery or SQL DML statements etc.). The meaning and purpose of mappings and general application areas, tools and technologies is discussed by Roth et al. with the generic usage scenarios in different enterprise architecture environments [12]. The declarative mappings designed for ETL program generation and vice versa are discussed in [4] and [6].

In addition to mappings with program generation instructions and data transformation semantics, there exist relations and dependencies between mappings and source or target schema objects. The declarative representation of the dependencies allows us to generate, optimize and execute data transformations workflows effectively. We can find different optimization approaches described in papers [1],[2] and [3]. An extensive study of common models for ETL and ETL job optimization is published by Simitsis et al. [13], [14] and Patil et al. [9]. The dynamic changes of data structures, connections between mapping and jobs and a rule-based ETL graph optimization approach is discussed in [8].

An effective ETL job optimization is related to data mappings, dependencies between mappings, dynamics and changes of source structures and the data quality (DQ). All of those aspects can be formalized and taken into account by ETL job automation where timing, the right order and data volumes are always important issues. Estimation and evaluation of data structures, quality of data and discovery of rules can be automated and integrated into ETL processes [11]. By adding rule-based DQ into ETL process, we can automate the mapping generation and improve the success rate of data loadings. Rodiç et al. demonstrated that most of the integration rules can be generated automatically using the source and target schema descriptions [11].

## 3 System Architecture

The modern enterprise data transformation systems are built according to the model-driven architecture principles, including internal metadata about the source and target models, mappings, transformations and dependencies between models. We introduce a new architectural concept, based on open source java and xml technologies that can be used in lightweight scripting configuration, mixed configuration with partial mapping formalization and full model- and metadata driven knowledge base implementations. When the first lightweight configuration gives you a quick start, low-cost and small technology track and metadata-driven approach gives a knowledge base with different new possibilities (e.g. mapping generation and management, dynamic dependency management and job automation, impact analysis, data quality integration etc.), then both exploit the template-based code construction and automation principles. Our design goals of the new ETL architecture were an open

and flexible environment, extensible and reusable programming techniques with moderate formalization and decoupling of declarative knowledge from procedural parts of executable code.

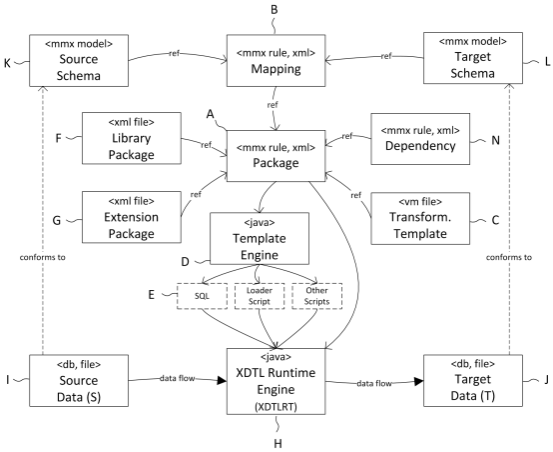


**Fig. 1.** System architecture components

The general system architecture with main building blocks is drawn in Figure 1. The main system components are ETL Package (A) that can be written in XDTL language or represented as Tasks and Rules and Dependencies (N) in MMX repository. The mapping (B) is a formalized representation of source schema objects (K), target (L), column transformations and patterns, join and filter conditions that can be again be a part of an XDTL package or stored in MMX repository. The transformation Template (C) is a reusable and repeating part of SQL query patterns or some other scripting executable language scenarios that are written in the Apache Velocity macro language (or any other language of template engine). The template Engine (D) is a configurable java code that is responsible for runtime code construction using Mappings (B) and Templates (C). Examples of mappings and templates are discussed in more detail level in Chapter 4. The tasks in Package (A) can be created using previously prepared Library Packages (F) or managed modularly, reused and published as Extension (G) modules. The XDTL Runtime Engine (H) is a preconfigured environment and java package, able to interpret packages written in the XDTL language, execute those and deliver actual data transformations from the source (I) to the target (J).

### 3.1 Data Transformation Language (XDTL)

The Extensible Data Transformation Language (XDTL) is an XML based descriptive language designed for specifying data transformations between different data formats, locations and storage mechanisms. XDTL is created as a Domain Specific Language (DSL) for the ETL domain and is designed by focusing on the following principles: modular and extensible, re-usable, decoupled declarative (unique) and procedural

(repeated) patterns. The XDTL syntax is defined in an XML Schema document. The wildcard elements of an XML Schema enable extending the syntax of core language with a new functionality implemented in other programming languages or in XDTL itself. The XDTL scripts are built as reusable components with the clearly defined interfaces via parameter sets. The components can be serialized and deserilized between the XML and database representations, thus making XDTL scripts suitable for storing and managing in a data repository. XDTL provides the functionality to use data mappings stored independently of the scripts, being efficiently decoupled from the scripts. Therefore the mappings stored in a repository can exist as objects independent from the transformation process and be reused by several different processes. XDTL acts as a container for a process that often has to use facilities not present in XDTL itself (e.g. SQL, SAS language etc.).

### 3.2 Knowledge Repository Structure (MMX)

The MMX metadata framework is a general purpose integrative metadata repository built on the relational database technology for different knowledge management (KM) and rule-based analytical applications. The MMX repository is designed according to the OMG Metadata Object Facility (MOF) idea with separate abstraction and modeling layers (M0-M3). The MMX physical data model (schema) is based on principles and guidelines of EAV (Entity-Attribute-Value) or EAV/CR (Entity-Attribute-Value with Classes and Relationships) modeling technique suitable for modeling highly heterogeneous data with very dynamic nature. The metadata model and schema definition in EAV is separated from physical storage and therefore it is easy to modifications to schema on 'data' without changing the DB structures: by just modifying the corresponding metadata. The approach chosen is suitable for open-schema implementations (similar to key-value stores) where the model is dynamic and semantics is applied in query time, as well as model-driven implementations with a formal, well defined schema, structure and semantics.

The MMX physical schema (Figure 2) provides a storage mechanism for various knowledge- or meta-models (M2) and corresponding data or metadata (M1). Three physical tables - object, property and relation - follow the subject-predicate-object or object-property-value representation schemes, where object_type, property_type and relation_type tables are like advanced coding or dictionary tables for object, property and value types. The separate dictionary tables give us an advanced schema representation functionality using special attributes and relational database foreign keys (FK) mechanism. The formalized schema description and relations between different schemas make our metadata understandable and exchangeable between the other system components or external agents. The URI reference mechanism used and the resource storage schema makes an MMX repository a semantic data store, comparable to Resource Description Framework (RDF), serializable in different semantic formats or notations (e.g. RDF/XML , N3 , N-Triples , XMI etc.) using XML or RDF APIs.

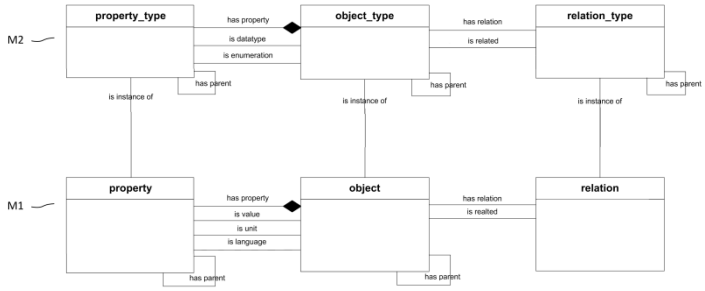


**Fig. 2.** MMX physical schema design

The MMX physical schema can be seen as a general-purpose, multi-level and hierarchical storage mechanism for different knowledge models, but also as communication medium or information integration and exchange platform for different software agents or applications (e.g. metadata scanners, metadata consumers etc.). Built in limited reasoning capability based on recursive SQL technique and it is captured to data and metadata APIs to implement inheritance and model validation functions. Semantic representation of data allows extend functionality with predicate calculus or apply other external rule-based reasoners (e.g. Jena) for more complicated reasoning tasks, like deduction of new knowledge.

Repository contains integrated object level security mechanism and different data access APIs (e.g. data API, metadata API, XML API, RDF API etc.) that are implemented as relational database procedures or functions. We have live implementations on PostgeSql, Oracle and MsSql platforms and differences in database SQL dialects and functionality are hidden and captured into API packages. Using common and documented API-s or an Object Relational Mapper (ORM) technology (e.g. Hibernate) we can choose and change repository DB technology without touching related applications.

An arbitrary number of different data models can exist inside MMX Metadata Model simultaneously with relationships between them. Each of these data models constitutes a hierarchy of classes where the hierarchy might denote an instance relationship, a whole-part relationship or some other form of generic relationship between hierarchy members. We have several predefined metadata models in MMX repository, e.g.

- terminology (ontology) and classification (based on ISO/IEC11179 [18]);
- relational database (based on Eclipse SQL Model [19]);
- abstract mappings and general ETL models;
- role-based access control model (based on NIST RBAC [20]).

In addition to existing models we can implement any other type of data mode for specific needs, like business process management (business rules, mappings, transformations, computational methods); data processing events (schedule, batch and task); data demographics, statistics and quality measures, etc.

The purpose of MMX repository depends on system configuration and desired functionality. In current paper we handle MMX repository as persistent storage mechanism for ETL metadata and we discuss about relational database and abstracted mapping knowledge models (KM) to store required data and relations. In addition to repository storage and access technologies MMX Framework has web-based navigation and administration tools, semantic-wiki like content management application and different scanner agents written in XDTL (e.g. DB dictionary scanner) to feel and detect surrounding environment and context. Due to space limitation we do not discuss all those topics in this paper.

# 4 Template Based SQL Construction

SQL is and probably remains the main workforce behind any ETL (and especially ELT flavor of ETL) tool. Automating SQL generation has arguably always been the biggest obstacle in building an ideal ETL tool (i.e. completely metadata-driven), with small foot-print, multiple platform support on single code base. While SQL stands for Structured Query Language, ironically the language itself is not too well 'structured', and the abundance of vendor dialects and extensions does not help either. Attempts to build an SQL generator supporting full feature list of SQL language have generally fallen into one of the two camps: one of them trying to create a graphical click-and-pick interface that would encompass the syntax of every single SQL construct, another one designing an even more high-level language or model to describe SQL itself, a kind of meta-SQL. The first approach would usually be limited to simple SQL statements, be appropriate mostly for SELECT statements only and struggle with UPDATEs and INSERTs, and be limited to a single vendor dialect.

## 4.1 Mappings, Patterns and Templates

Based on our experience we have extracted a set of SQL 'patterns' common to practical ETL (ELT) tasks. The patterns are converted into templates for processing by a template engine (e.g. Apache Velocity), each one realizing a separate SQL fragment, a full SQL statement or a complete sequence of commands implementing a complex process. Template engine merges patterns and mappings into executable SQL statements so instead of going as deep as full decomposition we only separate and extract mappings (structure) and template (process) parts of SQL. This limits us to only a set of predefined templates, but anyone can add new or customize the existing ones. Templates are generic and can be used with multiple different mappings/data structures. The mappings are generic as well and can be used in multiple different patterns/templates. Template engine instantiates mappings and templates to create executable SQL code which brings us closer to OO mind-set. The number of tables joined, the number of columns selected, the number of WHERE conditions etc. is arbitrary and is affected by and driven by the contents of the

mappings only, i.e. well-designed templates are transparent to the level of complexity of the mappings. The same template would produce quite different SQL statements driven by minor changes in mappings. We have built a series of template libraries to capture the syntax of basic SQL constructs that are used to build complex statements. XDTL Basic SQL Template Library is a set of Apache Velocity templates that implements 'atomic' SQL constructs (INSERT, SELECT, UPDATE, FROM, WHERE etc.) as a series of Velocity macros. Each macro is built to expand into a single SQL construct utilizing the mappings in the form of predefined collections (targets, sources, columns, conditions). On top of Basic SQL library one or more higher-level layers can be built to realize more specific or more complex concepts, e.g. loading patterns, scenarios or process flows, as well as specifics of various SQL dialects.

It appears that, by use of Abstract Syntax Patterns, the same principle of reducing a disparate and seemingly diffuse set of all possible transformations in SQL statements to a limited set of patterns applies here as well. Abstract Syntax Pattern (ASP) is a reappearing code fragment that, similarly to Abstract Syntax Tree, has all the references to concrete data items removed. Thus, mappings between different data domains can be reduced to ASPs to be later processed synchronously with the process template by the same template engine turning them into executable code. Identifying and building a library of common syntax patterns enables creation of a user interface to generate a focused (limited) set of SQL statements without coding or even automatic SQL generation, validation of existing SQL statements [10]. More detailed ASP discovery case study can be found on chapter 5.

Various ETL metamodels discussed in previous works of [15],[16] and [17], but we decided to use pragmatic approach with ASP idea instead of complex and expressive modeling. We had modeling goals like: minimum footprint and complexity, effective code generation for different languages (e.g. SQL, SAS, R etc.), efficient storage and serialization, decomposition to the level where interesting parts would be identified and exposed with clear semantics (i.e. database objects, vendor specific terms and keywords, generic and reusable expressions etc.). In Figure 3 we have implemented mappings knowledge model with four basic classes which are designed as derivation type of rules in MMX repository. Mapping Group (B) is collection of Mappings (C) which used on multiple mapping cascade definition that produces single SQL statement with sub-queries or temporary table implementation. Each Transformation class (D) instance represents one column transformation in SQL select, insert-select or update statement with required source, target and pattern attribute definitions. Condition class (E) represents join and filter predicate conditions that required constructing data set from defined source variables (tables).

Mapping model (Figure 3) implementation in MMX physical schema (Figure 2) is straightforward transformation where each class implemented as one row in object_type table, each class attribute implemented as one row in property_type table, and each association implemented as one row in relation_type table. Instances of mapping model stored as corresponding rows in object, property and relation tables.

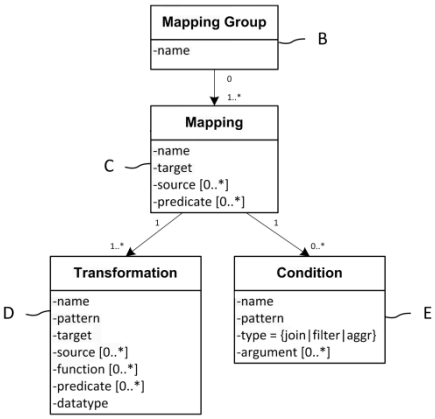


**Fig. 3.** Knowledge model (schema) for mappings representation and storage

Described method and model for constructing SQL statements complies with the following criteria:

- construction of all significant DML statements (INSERT, UPDATE, DELETE) based on a single mapping;
- construction of SQL statements from a single mapping for several different SQL dialects;
- construction of SQL statements covering different loading scenarios and performance considerations;
- construction of SQL statements based on multiple mappings (mapping groups);
- minimum footprint and complexity of the processing environment.

The next chapter example gives the basic idea of one mapping implementation and usage scenarios for SQL code generation in ETL process.

## Mapping Example

Following simple and generic insert-select SQL statement represents one very basic transformation code for everyday data transformation inside DW from multiple staging tables to target table.

*Example 1*. Insert-select SQL DML statement:

```
INSERT INTO person t0 (id, name, sex, age)
SELECT t1.cust_id,
       t1.firstname || ' ' || t1.lastname,
       DECODE(t1.sex, 'M', 1, 'N', 2),
       ssn_to_age(t2.ssn)
FROM customer t1
JOIN document t2 ON t2.cust_id = t1.cust_id
WHERE t1.cust_id IS NOT NULL;
```

Same query contains declarative mapping part formalized and represented by following objects and collections in MMX repository tables:

**Table 1.** Mapping objects source and target properties

| Target | isVirtual | Source | isQuery |
|---|---|---|---|
| t0:person | f (false) | t1:customer | f (false) |
| | | t2:document | f (false) |

**Table 2.** Column Transformation object(s) properties and values

| Pattern | Target | Source | Function | Key | Upd |
|---|---|---|---|---|---|
| %c1 | %c0:t0.id | %c1:t1.cust_id | null | t | f |
| %c1\|\| ' '\|\|%c2 | %c0:t0.name | %c1:t1.firstname; %c2:t1.lastname | null | f | t |
| decode(%c1,'M',1,'F',2) | %c0:t0.sex | %c1:t1.sex | null | f | f |
| %f1(%c1) | %c0:t0.age | %c1:t2.ssn | %f1:ssn_to_age | f | t |

**Table 3.** Condition objects properties(s) and their values

| Pattern | Source | Function | Condition Type | Join Type |
|---|---|---|---|---|
| %a2 = %a1 | %a2:t2.cust_id; %a1:t1.cust_id | null | join | inner |
| %a1 IS NOT NULL | %a1:t1.cust_id | null | filter | null |

Simple insert-select template produces initial SQL statement from given mapping (Table 4). When using the same mapping with the different template(s) we can generate update statement or series of different statements.

**Table 4.** Insert-Select template and result query

| Template | SQL Statement |
|---|---|
| `#foreach($tgt in $Targets)` | `INSERT INTO person (id, name, sex, age)` |
| `#if("$tgtmap" == "0")` | `SELECT` |
| `#INSERT($tgt $Columns)` | `  t1.cust_id` |
| `#SELECT($Columns)` | `, t1.firstname \|\| ' ' \|\| t1.lastname` |
| `#FROM($Sources $Conditions)` | `, DECODE(t1.sex, 'M', 1, 'F', 2)` |
| `#WHERE($Conditions)` | `, ssn_to_age(t2.ssn)` |
| `#GROUPBY($Columns $Conditions)` | `FROM customer t1` |
| `#HAVING($Conditions)` | `INNER JOIN document t2` |
| `#end` | `ON t2.cust_id = t1.cust_id` |
| `#end;` | `WHERE t1.cust_id IS NOT NULL;` |

The described example gives very basic idea and method how to formalize column mappings with reusable patterns and how to construct source and target data sets applying join and filter conditions that described again with reusable expression patterns. Using one single mapping together with limited number of scenario templates (e.g. full load, initial load, incremental load, insert only, 'upsert' (with or without deletion), versioned insert (history tables), slowly changing dimensions etc.) we can generate long SQL statement batches, that are adapted for specific dialect (when needed), validated, robust and working with expected performance. The main idea here is to formulate and describe as less as possible and reuse and generate as much as possible.

By using described set of methods we have effectively decomposed SQL statement into mappings and patterns. The same method can be applied to any SQL data

manipulation statement (e.g. insert, update and delete) of reasonable complexity and we have used same approach, same mappings and different templates to generate code for different execution engines (e.g. SAS script). More expressive examples can be presented when to take transformations with more than 5-10 source tables and targets with 20-100 columns (common in analytical DW environment) then propositions of generated, defined and reused code parts change dramatically.

To conclude the mapping example we can say that presented methodical approach allows us to:

- migrate and translate vendor-specific (SQL) pattern dialects between different platforms or use same mapping code with different transformation scenarios or generate code for different execution engines;
- construct data transformation flows from sources to targets with column level transformation semantics for Impact Analysis and Audit Trail applications;
- construct system component dependency graphs for better management and automation of development and operation processes;
- automate change management and deployment of new functionality between different environments (e.g. development, test, production).

## 5 Experimental Abstract Syntax Pattern Case Study

Abstract Syntax Pattern (ASP) is the practical idea to narrow down expressiveness of SQL Data Manipulation Language (DML) to allow formalized descriptions of reusable patterns, decoupled data structures and functions. Decomposition of patterns and data structure instances are the central idea of the XDTL environment and the code construction capability, which gives additional flexibility and ergonomics in data transformation design and allows impact analysis capability for maintenance of complex Data Warehouse environment (described in chapter 4). We used existing SQL statements corpus (used for real life data transformations) containing about 26 thousand SQL statements to find hard evidences for existing patterns and we narrowed down the used corpus to 12 thousand DML statement to find specific column, join and where expressions.

We used open source GoldParser[3] library and developed our own custom SQL grammar in EBNF format for SQL text corpus parsing. We developed custom parser program for ASP pattern extraction from SQL corpus, we imported all parsed patterns to database and evaluated and analyzed SQL patterns and "life forms" writing new SQL queries. Implemented parsing program is tuned to recognize column construction patterns, join, where and having condition predicate patterns from SQL DML statements, replacing specific database structure identifiers and constants with `%a` and functions with `%f` pattern.

[3] `www.goldparser.org`

*Example 3.* The parsing program detects pattern `%f(%a,%a)` from the original column expression `COALESCE(Table1.Column1,0)` and assigns operator and operand values to replaced variables: `%f = {'COALESCE'}` and `%a = {'Table1.Column1','0'}`

Very general metrics about SQL parsing work can be described with total figures of 8,380 input SQL DML statements (select, insert, update) and 92,347 parsed expressions that group to 2,671 abstract patterns. It gives us 97/3 percentage division between expressions and patterns. Those figures can be improved by hand tuning of pattern detection technique and SQL grammar that is not currently covering all the aspects of used SQL dialect.

**Table 5.** Insert-Select template and result query

| Statement Type | Patterns Count | Statements Count | Expression Count | Expressions in Top Pattern | Top Pattern Coverage % |
|---|---|---|---|---|---|
| Insert | 1 890 | 4 965 | 61 899 | 37 924 | 61 |
| Update | 836 | 2 240 | 25 361 | 12 997 | 51 |
| Select | 224 | 1 175 | 5 087 | 3 175 | 62 |
| All | 2 950 | 8 380 | 92 347 | 54 096 | 59 |

**Table 6.** Discovered patterns by pattern types

| Pattern Type | Pattern Count | Statement Count | Expression Count | Expressions in Top Pattern | Top Patten Coverage % |
|---|---|---|---|---|---|
| Column | 1 897 | 10 631 | 78 264 | 55 921 | 71 |
| Join | 526 | 4 172 | 11 684 | 2 273 | 19 |
| Filter | 323 | 1 505 | 2 399 | 359 | 14 |

Based on current results we can conclude that top 10 patterns will cover 83% and top 100 patterns will cover 93% of all expressions that used in SQL DML corpus. Those metrics does not count the fact that most of the patterns that are not in top list are constructed from patterns that are in patterns top list.

To conclude this case-study we can say that actual expressiveness of formalized patterns and mappings will be comparable with expressiveness of real life usage of SQL DML in data transformations. A small set of meaningful patterns (about 100 different patterns) with defined semantics and experimental impact weights will direct us to automated and probabilistic impact analysis calculations that are one of the main applications for SQL formalization technique. We also got the confirmation that SQL parsing technique can be used for data transformation extraction, mapping formalization and future analysis.

## 6 Case Studies for Automating Data Lineage Analysis

The previously described architecture and algorithms form a basis for an integrated data lineage analysis toolset dLineage (http://dlineage.com). dLineage has been tested

in large real-life projects and environments supporting several popular DW database platforms (e.g. Oracle, Greenplum, Teradata, Vertica, PostgreSQL, MsSQL, Sybase) and BI tools (e.g. SAP Business Objects, Microstrategy).

We have conducted two main case studies involving a thorough analysis of large international companies in the financial and the energy sectors. Both case studies involved an automated analysis of 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. The automation tools described in the paper enabled us to set up and conduct the analysis project in a few days by just two developers.

The following example graph from the case study maps DW tables to views and user reports: it is generated automatically from about 5 000 nodes (tables, views, reports) and 20 000 links (data transformations mappings form views and queries).

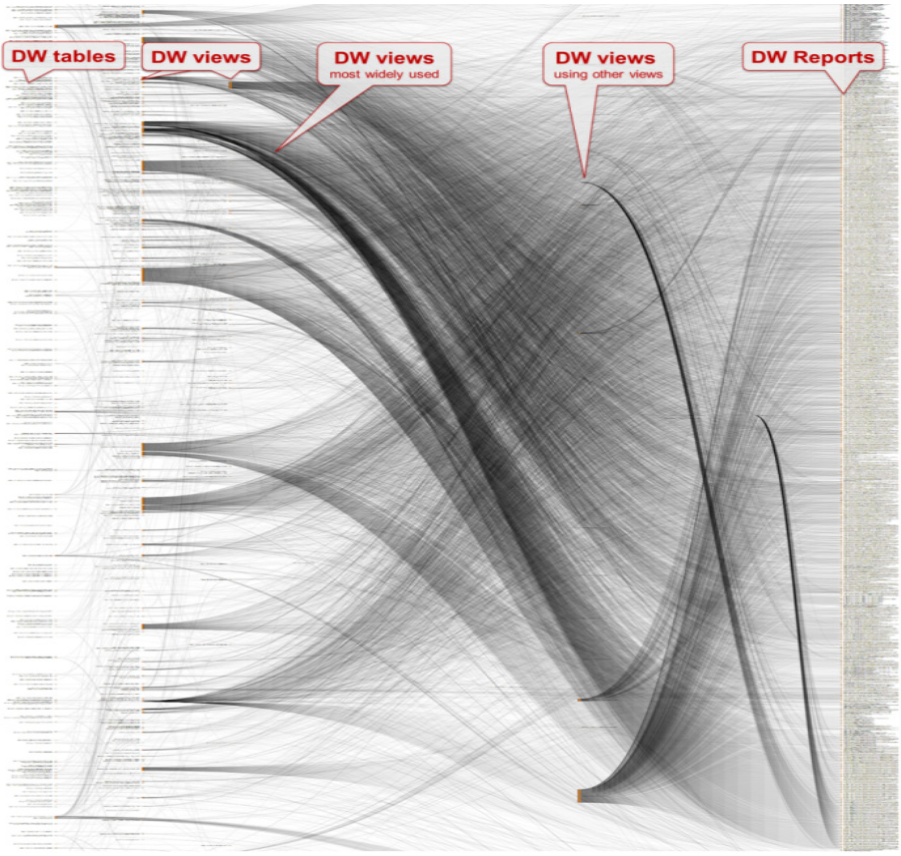


**Fig. 4.** Data lineage graph with dependencies between DW tables, views and reports

## 7 Conclusions and Future Work

We have presented a formalized mapping and abstract pattern methodology supporting template-based program construction. The technique is a development upon the ETL language runtime environment (XDTL) and metadata repository (MMX) designed earlier by the authors. We have introduced the technology and presented working samples motivated by real-life challenges and problems discussed in the first chapter. The described architecture and mapping concept have been used to implement an integrated toolset dLineage (http://dlineage.com) to solve data integration and dataflow visualization problems.

We have used our metadata-based ETL technology in the Department of Statistics of Estonian state to implement a system for automated, data-driven statistics production for the whole country. We have also tested our mapping methods and technology for data flow analysis and visualization in large international companies in the financial and the energy sectors. Both case studies contained thousands of database tables and views along with tens of thousands of data loading scripts and BI reports. The analysis of the large SQL data transformation corpus (see chapter 5) gave us taxonomy of reusable transformation patterns and demonstrated the two-way methodology approach from code to mappings and patterns.

The future work involves refining current implementation details, adding semantics to mappings and patterns, constructing dependency graphs of mappings, data structures and data flows and developing aggregation algorithms for different personalized user profiles and their interests (e.g. business user interest in data structures, flows and availability is different from that of a developer or system operator) as well as using those techniques for solving problems described in the first chapter.

**Acknowledgments.** This research has been supported by European Union through European Regional Development Fund.

## References

[1] Behrend, A., Jörg, T.: Optimized Incremental ETL Jobs for Maintaining Data Warehouses (2010)
[2] Boehm, M., Habich, D., Lehner, W., Wloka, U.: GCIP: Exploiting the Generation and Optimization of Integration Processes (2009)
[3] Böhm, M., Habich, D., Lehner, W., Wloka, U.: Model-driven generation and optimization of complex integration processes. In: ICEIS (2008)
[4] Dessloch, S., Hernández, M.A., Wisnesky, R., Radwan, A., Zhou, J.: Orchid: Integrating Schema Mapping and ETL. In: IEEE 24th International Conference on Data Engineering (2008)
[5] Giorgini, P., Rizzi, S., Garzetti, M.: GRAnD: A Goal-Oriented Approach to Requirement Analysis in Data Warehouses. DSS 45(1), 4–21 (2008)
[6] Haas, L.M., Hernández, M.A., Ho, H., Popa, L., Roth, M.: Clio Grows Up: From Research Prototype to Industrial Tool. In: SIGMOD, pp. 805–810 (2005)

[7] Jun, T., Kai, C., Yu, F., Gang, T.: The Research & Application of ETL Tool in Business Intelligence Project, International Forum on Information Technology and Applications. In: FITA 2009, pp. 620–623 (2009)
[8] Papastefanatos, G., Vassiliadis, P., Simitsis, A., Sellis, T., Vassiliou, Y.: Rule-based Management of Schema Changes at ETL sources. In: Grundspenkis, J., Kirikova, M., Manolopoulos, Y., Novickis, L. (eds.) ADBIS 2009. LNCS, vol. 5968, pp. 55–62. Springer, Heidelberg (2010)
[9] Patil, P.S., Rao, S., Patil, S.B.: Data Integration Problem of structural and semantic heterogeneity: Data Warehousing Framework models for the optimization of the ETL processes (2011)
[10] Reiss, S.P.: Finding Unusual Code. In: 2007 IEEE International Conference on Software Maintenance, pp. 34–43 (2007)
[11] Rodiç, J., Baranoviç, M.: Generating Data Quality Rules and Integration into ETL Process (2009)
[12] Roth, M., Hernández, M.A., Coulthard, P., Yan, L., Popa, L., Ho, H.C.T., Salter, C.C.: XML mapping technology: Making connections in an XML-centric world. IBM Systems Journal (2006)
[13] Simitsis, A., Vassiliadis, P., Sellis, T.K.: Optimizing ETL Processes in Data Warehouses. In: ICDE, pp. 564–575 (2005)
[14] Simitsis, A., Wilkinson, K., Dayal, U., Castellanos, M.: Optimizing ETL workflows for fault-tolerance. In: International Conference on Data Engineering (ICDE), pp. 385–396 (2010)
[15] Song, X., Yan, X., Yang, L.: Design ETL Metamodel Based on UML Profile, Knowledge Acquisition and Modeling. In: KAM 2009, pp. 69–72 (2009)
[16] Stöhr, T., Müller, R., Rahm, E.: An Integrative and Uniform Model for Metadata Management in Data Warehousing Environment. In: Workshop on Design and Management of Data Warehouses (DMDW) (1999)
[17] Vassiliadis, P., Simitsis, A., Georgantas, P., Terrovitis, M.: A Framework for the Design of ETL Scenarios. In: Eder, J., Missikoff, M. (eds.) CAiSE 2003. LNCS, vol. 2681, Springer, Heidelberg (2003)
[18] ISO/IEC 11179 Metadata Registry (MDR) standard, http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=35343
[19] Eclipse DB Definition Model, http://www.eclipse.org/webtools/wst/components/rdb/WebPublishedDBDefinitionModel/DBDefinition.htm
[20] NIST Role Based Access Control (RBAC) Standard, http://csrc.nist.gov/groups/SNS/rbac