

BUSINESS KNOWLEDGE EXTRACTION USING PROGRAM UNDERSTANDING AND DATA ANALYSIS TECHNIQUES*

Bronius Paradauskas, Aurimas Laurikaitis

Kaunas University of Technology, Department of Information Systems,
Studentų St. 50, LT-51368 Kaunas, Lithuania,
parad@soften.ktu.lt, aurimaras@yahoo.com

Abstract. This article discusses the process of enterprise knowledge extraction from relational database data dictionary and data and source code of legacy information systems. Problems of legacy systems and main solutions for them are briefly described here. The use of database reverse engineering and program understanding techniques to automatically infer as much as possible the schema and semantics of a legacy information system is analyzed. Six step database reverse engineering algorithm for knowledge extraction from legacy systems is provided. Hypothetical example of knowledge extraction from legacy information system is presented.

Keywords: business knowledge extraction, relational database, database reverse engineering, legacy information systems.

1 The Problems of Legacy Information Systems

Legacy information system – any information system that significantly resists modification and evolution to meet new and constantly changing business requirements [2],[27]. Legacy systems typically contain incredible detailed business rules and form the backbone of the information flow of organization that consolidates information about its business [4]. A failure in one of these systems may have a serious business impact. Legacy information systems are currently posing numerous and important problems to their host organizations. The most serious of these problems are:

- systems usually run on obsolete hardware which is slow and expensive to maintain;
- maintenance of software is generally expensive; tracing faults is costly and time consuming due to the lack of documentation and a general lack of understanding of the internal workings of the system;
- integration efforts are greatly hampered by the absence of clean interfaces;
- legacy systems are very difficult, if not impossible, to expand.

In response to these problems, several approaches to change or replace legacy systems have been proposed. They are classified into the following three categories [25]: redevelopment, wrapping, and migration. Redevelopment involves process of developing system from scratch, using a new hardware platform, architecture, tools and databases. Wrapping involves developing a software component called wrapper that allows an existing software component to be accessed by other components. Migration allows legacy systems to be moved to new environments that allow information systems to be easily maintained and adapted to new business requirements, while retaining functionality and data of the original legacy systems without having to completely redevelop them.

Usually, in the process of replacing legacy systems the above three categories are combined in varying degrees. First thing needed when solving the problems of legacy systems is the exact picture of information system is provided. Hypothetical example of knowledge extraction from legacy information system is presented.

2 Database Reverse Engineering

Database reverse engineering (DBRE) is defined as the application of analytical techniques to one or more legacy data sources to elicit structural information (e.g. term definitions, schema definitions) from the legacy information sources in order to improve the database design or produce missing schema documentation [15],[34].

Formally DBRE can be described as follows: Given a legacy database \(DB_L\) defined as \((\{R_1, R_2, ..., R_n\}, D)\), where \(R_i\) denotes the schema of the \(i\)-th relation with attributes \(A_1, A_2, ..., A_m\), keys \(K_1, K_2, ..., K_m\) and data \(D=\{r_1(R_1), r_2(R_2), ..., r_n(R_n)\}\), such that \(r_i(R_i)\) denotes the data for schema \(R_i\) at time \(t\). Furthermore, \(DB_L\) has functional dependencies \(F=\{F_1, F_2, ..., F_k\}\) and reference and equality constraints \(I=\{I_1, I_2, ..., I_l\}\) expressing relationships among the relations in \(DB_L\). The goal of DBRE is to first extract \(\{R_1, R_2, ..., R_n\}, I, F\) and then use \(I, F, D, C_L\) (program code) to produce a semantically enhanced description of \(\{R_1, R_2, ..., R_n\}\) that includes, all relationships among the relations in \(DB_L\) (incl. those that are implicit), semantic descriptions of the relations as well as the business knowledge that is encoded in \(DB_L\) and \(C_L\).

* The work is supported by Lithuanian State Science and Studies Foundation according to High Technology Development Program Project "VeTIS" (Reg.No. B-07042).

- 337 -
Since 1980, a wide range of DBRE methods have been published. All of them consist of extracting the conceptual schema from an operational legacy system. The conceptual schema can be expressed in some variant of the entity-relationship model or of the object-oriented model (ODMG, UML, OMT).

The approaches of Hainaut [13], Dumpala [10], Navathe [24], Casanova [6], Markowitz [23], Davis [9], Johannesson [22], Ramanathan [30] and Alhaij [1] focuses on transforming physical or logical schema to a conceptual schema. These methods require physical or logical schema to be supplied with all primary and foreign keys, some have heavy prerequisites on schema (3NF, meaningful names, etc.) so they can be applied only on ideally designed databases. These approaches use schema as the only input and extracted conceptual schema is not enriched with implicit constraints.

DB-MAIN [11], Premerlani [28], Signore [32], Petit [29], Chang [7], MeRCI [8], Valet [20], [21] methods uses data and program and DML code as input and extract explicit and implicit constructs. All of these approaches explore DDL code. The approaches of Premerlani and Chang examine data, Signore method analyze program code. DB-MAIN, Petit, MeRCI and Varlet methods analyze data and application code. Petit method is the only to extract functional dependencies from DDL code and program code, MeRCI method is the only to elicit redundancy from DDL code and program code. DB-MAIN method allows supplementing extracted specification with knowledge of analyst.

All DBRE approaches concentrate their research on some area: extracting some constraints and/or analyzing some information sources. No one from above mentioned methods proposes tools to automatically extract secondary identifiers, enumerated value domains, constraints on value domains, existence constraints and semantics of attributes.

This article presents VeŽI (Lithuanian: Veiklos žinių išgavimas – Business Knowledge Extraction) approach that semi-automatically discovers and extracts knowledge from legacy information systems [26], i.e. generates conceptual specification of legacy information system, including entities, attributes, relations, constraints, business rules, etc. VeŽI approach generates conceptual schema in Extended Entity-Relationship model. This model represents conceptual model constructs in graphical form; it is widely used in CAD tools. VeŽI approach is applied to relational databases only.

The basic schema information is extracted (Figure 1) from legacy information system database management system (DBMS). Then this schema information can be semantically enhanced using details from application code and data. The final conceptual schema is generated using the set of conceptual transformations.

![Figure 1. The Conceptual Diagram for Business Knowledge Extraction.](image-url)
3 The Approach for Extracting Business Knowledge

VeŽI approach includes two main database reverse engineering processes: data structure extraction and data structure conceptualization. An overview of VeŽI approach which is comprised of six main steps is shown in Figure 2.

The data structure extraction is the most crucial and difficult part of DBRE. Data structure extraction analyzes the existing legacy system to recover complete logical schema that includes implicit and explicit structures and constructs. Various program understanding and data analysis techniques are used to recover implicit structures and constraints. Data structure extraction consists of three main steps: data dictionary extraction, schema refinement and schema cleaning.

The data dictionary extraction is the simplest process of data structure extraction. It produces raw physical schema, which consists of declared data structures and constraints.

The schema refinement is the most important process of data structure extraction. The main problem of the data structure extraction is that most of structures and constraints were not explicitly defined, but instead implemented in procedural part of application. Many implicit constraints exist: primary and secondary identifier, reference and equality constraints, functional dependencies, meaningful names, etc. Schema refinement process enriches physical schema with implicit constructs and produces complete physical schema.

The schema cleaning removes or replaces physical constructs into logical ones and transforms complete physical schema into complete logical schema.

Figure 2. Conceptual View of the VeŽI Algorithm.

The data structure conceptualization interprets logical schema in a conceptual view and recovers conceptual schema. It detects and transforms or discards non-conceptual structures, redundancies, technical
optimizations and DMS-dependent constructs. Data structure conceptualization consists of three main steps: preparation, basic conceptualization and normalization.

The preparation discards dead data structures and technical data structures, i.e. prepares the schema, that it only contains structures and constraints that are necessary to understand the semantics of the schema.

The basic conceptualization extracts a basic conceptual schema without worrying about esthetical aspects of the result. Two kinds of transformation are used: untranslation and de-optimization. Though distinguishing between them may be arbitrary in some situations (some transformations pertain to both).

The conceptual normalization transforms various constructs and gives expressiveness, simplicity, minimality and extensibility of conceptual schema. It tries to make higher level semantic constructs explicit (e.g. is-a relation).

The Business Knowledge Encoding is a technical step that extracts knowledge in the form of an XML document.

Trivial example of legacy information system is provided in order to highlight each of the six steps and related activities outlined in Figure 2. Assume that the underlying legacy database $DB_L$ is managed by a relational database management system. For simplicity, we assume without lack of generality or specificity that only the following relations exist in $DB_L$, whose schema will be discovered using DBRE:

**CLIENT** [CID, NAME, ADDR_CITY, ADDR_STREET, ADDR_HOME_NUM, ADDR_ROOM_NUM]

**PHONE** [CID, PHONE]

**CUSTOMER** [CID, CATEGORY]

**SUPPLIER** [CID, ACCOUNT]

**PRODUCT** [PRODNUM, NAME, PRICE, SUPPLIER]

**ORDER** [CID, ONUM, DOC_DATE, OP_DATE]

**DETAIL** [CID, ORDER, PRODUCT, PROD_NAME, PROD_PRICE, QUANTITY]

In order to illustrate the code analysis and how it enhances the schema extraction, the following C code fragment is used representing a simple, hypothetical interaction with a legacy database:

```c
char *aValue, *cValue;
int bValue = 100;
........
/* more code */
........
EXEC SQL SELECT DOC_DATE, OP_DATE INTO :aValue,:cValue
FROM ORDER WHERE ONUM = :bValue;
........
/* more code */
........
if (*cValue < *aValue)
{ cValue = aValue; }
........
/* more code */
........
printf("Document Date %s ", aValue);
printf("Operation Date %s ", cValue);
```

### 3.1 Data Dictionary Extraction

The goal of data dictionary extraction is to obtain the relational specifications from the legacy source. It is the simplest process of data structure extraction. Almost all CASE tools propose some kind of DDL code analysis for the most popular DMS. Some of them are able to extract relational specifications from the system data dictionary as well. VeŽI approach uses Java Database Connectivity (JDBC) that is an API for the Java programming language that extracts database structure and properties: collections, entities, attributes, primary identifiers, foreign keys and indexes.

Result. Extracted raw physical schema is presented in Figure 3.
3.2 Schema Refinement

The goal of most important process of data structure extraction is to identify and extract structures and constraints that were implemented implicitly or were discarded during application development. Many implicit constraints exist: primary and secondary identifier, reference and equality constraints, functional dependencies, meaningful names, etc.

3.2.1 Data Analysis

The goal of data analysis is to augment raw physical schema with constraints that were not discovered during data dictionary extraction:

- Finding optional attributes. Optional constraints of attributes were obtained during data dictionary extraction process, but there still exist optional attributes that were declared as mandatory attributes.
  - There are some values of attribute „CLIENT.ADDR_ROOM_NUM“ with space symbol, so this attribute is appended to the list of possible optional attributes.
- Finding enumerated value domains. Many attributes must draw their values from a limited set of predefined values.
  - There are very few distinct values of attribute „CUSTOMER.CATEGORY“, so possible values of this attribute are noted.
- Finding constraints on value domains. In most DMS, declared data structures are very poor as far as their value domain is concerned. Quite often, though, strong restriction is enforced on the allowed values.
  - The minimal value of attribute „PRODUCT.PRICE“ is 0 and the maximum value is more than 0, so valid value domain is positive numbers and zero (>=0).

Following data analysis, if possible optional attributes were identified; list of these attributes with possible nullable values must be presented to the user for a final decision.

Result. Physical schema with additional attribute constraints is presented in Figure 4.
3.2.2 Abstract Syntax Tree Generation

The goal of generation of abstract syntax tree (AST) for the legacy application code is to support various techniques of program understanding for discovery of implicit structures and constraints. Lexical analyzers and parsers are used for AST generation (AST is described in [5], [33]). The AST will be used by few steps of data structure extraction.

Result. Abstract syntax tree of trivial example is presented in Figure 5.

3.2.3 Identifier Extraction

The goal of identifier extraction is to obtain primary identifiers, if primary key information was not retrieved from data dictionary, and discover secondary identifiers.

The algorithm of primary identifier extraction proceeds as follows: for each entity, it first identifies the set of candidate attributes, which are mandatory and unique attributes. If there is only one candidate attribute per entity, then that attribute is the primary identifier. Otherwise, AST is analyzed for rule-out patterns and those attributes are eliminated from the candidate set, which occur in the rule-out pattern. The rule-out patterns, which are expressed as SQL queries, occur in the application code whenever programmer expects to select a SET of tuples. By definition of primary key, this rules out the possibility that the attributes $a_1 \ldots a_n$ form a primary key. Three sample rule-out patterns are:

1. $\text{SELECT DISTINCT <selection> FROM <table>}$
WHERE a1=<expr1> AND a2=<expr2> AND ... AND an=<exprn>

2. SELECT <selection> FROM <table>
WHERE a1=<expr1> AND a2=<expr2> AND ... AND an=<exprn>
GROUP BY ...

3. SELECT <selection> FROM <table>
WHERE a1=<expr1> AND a2=<expr2> AND ... AND an=<exprn>
ORDER BY ...

Following AST, if a primary identifier cannot be identified, the reduced set of candidate attributes must be presented to the user for a final primary identifier selection.

If unique indexes for mandatory attributes exist which are not primary identifiers, then these attributes are marked as secondary identifiers.

Result. Primary and secondary identifiers are as follows (Figure 6):

- Entity „SUPPLIER” has only one candidate attribute (mandatory and unique), so „SUPPLIER.CID” is a primary identifier.
- Entity „PRODUCT” has two candidate attributes „PRODUCT.PRODNUM” and „PRODUCT.NAME”. No one attribute occur in rule-out pattern so the user must select primary identifier. The user chooses attribute „PRODUCT.PRODNUM” as a primary identifier.
- Attribute „PRODUCT.NAME” of entity „PRODUCT” is a secondary identifier, because it is unique and mandatory attribute which is not a primary identifier.

Figure 6. Physical Schema with Identifiers.

3.2.4 Reference and Equality Constraints Extraction

The goal of reference and equality constraints extraction is to identify constraints to help classify the extracted entities, which represent both the real-world entities and the relationships among them. This is done using reference and equality constraints, which indicate the existence of inter-relational constraints including class/subclass relationships.

Reference constraint is described as follows: Let R1 and R2 be 2 entities, and A and B be attributes or set of attributes of R1 and R2 respectively. Reference constraint R1.A<<R2.B denotes that a set of values appearing in R1.A is a subset of R2.B (R2.B must be primary or secondary identifier). Reference constraints are discovered by examining all possible subset relationships between any two entities R1 and R2. Reference constraints can be identified in an exhaustive manner as follows: for each pair of entities R1 and R2 in the legacy source schema, compare the values for each non-key attribute combination A in R1 with the values of each
primary (or secondary) identifier combination \( B \) in \( R2 \) (note that \( A \) and \( B \) may be single attributes). Reference constraint \( R1.A \prec\prec R2.B \) may be present if:

1. \( A \) and \( B \) have same number of attributes.
2. \( A \) and \( B \) must have pair wise domain compatibility (matching data types and matching maximum length of attributes).
3. \( R1.A \subseteq R2.B \).

If there is additional subset dependency \( R2.B \subseteq R1.A \), then equality constraint may exist.

Information for reference and equality constraints extraction is obtained from three sources: physical schema with identifiers provides entities, attributes, primary and secondary identifiers and foreign keys (if available), equi-join query finder explores AST and provides pairs of entities and corresponding attributes, which occur together in equi-join queries in AST (the fact that two entities are used in a join operation is evidence for the existence of an reference or equality constraint between them), and subset dependency explorer run SQL queries against legacy database.

In order to check the subset criteria (3), the following generalized SQL query templates are provided, which are instantiated for each pair of primary (or secondary) identifier of one entity and attribute of the other entity combinations and run against the legacy source:

\[
\begin{align*}
C1 &= \text{SELECT COUNT (*) FROM } R1 \\
\text{WHERE } A \text{ NOT IN (SELECT } B \text{ FROM } R2); \\
C2 &= \text{SELECT COUNT (*) FROM } R2 \\
\text{WHERE } B \text{ NOT IN (SELECT } A \text{ FROM } R1); \\
\end{align*}
\]

If \( C1 \) is zero, we can deduce that a reference constraint \( R1.A \prec\prec R2.B \) may exist, likewise, if \( C2 \) is zero that a reference constraint \( R2.B \prec\prec R1.A \) may exist. Note that it is possible for both \( C1 \) and \( C2 \) to be zero. In that case, we can conclude that the two sets of attributes \( A \) and \( B \) are equal, so equality constraint may exist.

Result. Reference and equality constraints are as follows (Figure 7):

\[
\text{REF=\{} \text{PHONE.CID} \prec \text{CLIENT.CID}, \text{CUSTOMER.CID} \prec \text{CLIENT.CID}, \text{ORDER.CID} \prec \text{CUSTOMER.CID}, \text{SUPPLIER.CID} \prec \text{CLIENT.CID}, \text{PRODUCT.SUPPLIER} \prec \text{SUPPLIER.CID}, \text{DETAIL.PRODUCT} \prec \text{PRODUCT.PRODNUM} \text{\}; EQU=\{} \text{DETAIL.CID}, \text{DETAIL.ORDER} \prec \text{ORDER.CID}, \text{ORDER.ONUM} \text{\}.}
\]

![Figure 7. Physical Schema with Referential Constraints.](image-url)
3.2.5 Other Constraints Extraction

The goal of this step is to obtain implicit constructs, i.e. constraints and dependencies: coex, exact-1, disjoint(R1.A1,R2.A1), R1 U R2.A1->A2, R1.A1 in (R2.A1 U R3.A1), R.BA1==>(BA1=A1) and redundancy constraint rd. In order to find constraints and dependencies, the generalized SQL query templates are used and run against the legacy source.

For example, in order to discover coex constraint, the following generalized SQL query template is used, which is instantiated for each pair of the entity optional attribute combinations and run against the legacy source:

\[
C = \text{SELECT COUNT (*) FROM R WHERE NOT((A IS NULL AND B IS NULL) OR (A IS NOT NULL AND B IS NOT NULL)));
\]

If \(C1\) is zero, we can deduce that coex constraint between \(A\) and \(B\) exist, likewise, if \(C\) is greater than zero then coex constraint between \(A\) and \(B\) doesn’t exist.

Result. Other constraints are as follows (Figure 8):

- disjoint(CUSTOMER.CID, SUPPLIER.CID);
- CLIENT.CID in (CUSTOMER.CID U SUPPLIER.CID);
- rd(DETAIL.PROD_NAME << PRODUCT.NAME).

![Figure 8. Complete Physical Schema.](image)

3.2.6 Code Analysis

The objective of code analysis is twofold: (1) augment entities with domain semantics, and (2) identify business rules and constraints not explicitly stored in the database, but which may be important to the process of reverse engineering. This approach to code analysis is based on program understanding, which includes slicing [3], [18], [19] and pattern matching [31].

The mining of semantic information from source code assumes that in the application code there are output statements that support report generation or display of query results. From output message string that usually describes a displayed variable \(v\), semantic information about \(v\) can be obtained. This implies location (tracing) of the statement \(s\) that assigns a value to \(v\). Since \(s\) can be associated with the result set of a query \(q\), we can associate \(v\)'s semantics with a particular attribute of entity.
The first step is the construction of system dependency graph (SDG) from abstract syntax tree. SDG is constructed in four stages [16]: 1) augmented control flow graph construction (ACFG) from the AST; 2) computation of the post dom graph from ACFG; 3) construction of the program dependency graph (PDG) using ACFG and the post dom graph; 4) construction of the SDG using the PDG. The PDG for the AST of Figure 5 is presented in Figure 9.

The next step is the pre-slicing. From the AST all the nodes are identified corresponding to input, output and embedded SQL statements. If an identifier node (which corresponds to the occurrence of a variable in that statement) exists in the subtree of that statement node, then the actual variable name is appended to the list of slicing variable. For example, for the AST in Figure 5, the array contains the following information depicted in Table 1.

Table 1. Information of Slicing Variables.

<table>
<thead>
<tr>
<th>Slicing Variable</th>
<th>Type of Statement</th>
<th>Direction of Slicing</th>
<th>Text String (Only for Print Nodes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>aValue</td>
<td>Output</td>
<td>Backwards</td>
<td>“Document Date”</td>
</tr>
<tr>
<td>cValue</td>
<td>Output</td>
<td>Backwards</td>
<td>“Operation Date”</td>
</tr>
</tbody>
</table>

For each slicing variable identified by the pre-slicing step, code slicing and analysis are performed on the AST. In the above example, the slicing variables that occur in SQL and output statements are aValue and cValue. The direction of slicing is fixed as backwards or forwards depending on whether the variable in question is part of an output (backwards) or input (forwards) statement. The slicing criterion is the exact statement (input or output) vertex that corresponds to the slicing variable.

During code slicing step the flow and control edges of the PDG for the source code are followed for each slicing variable and only those vertices are retained that were reached by traversal. The result of slice consists of the set of vertices and the set of edges induced by this vertex set that are relevant to the slice with respect to the slicing variable. Figure 10 shows backward slice for the SDG in Figure 9 with respect to printf(cValue) vertex. The reduced AST that correspond the SDG in Figure 10 is shown in Figure 11.
During the analysis, information shown in Table 2 is extracted, while traversing the reduced AST.
1. \textit{dcln} node contain information about data type of the identifier.
2. \textit{embSQL} node contain the mapping information of identifier name to corresponding attribute and entity.
3. \textit{print/scanf} nodes contain the mapping information from the text string to the identifier. In other words we can extract the meaning of the identifier from the text string.

<table>
<thead>
<tr>
<th>Identifier Name</th>
<th>Meaning</th>
<th>Possible Business Rule</th>
<th>Data type</th>
<th>Column Name</th>
<th>Table Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>aValue</td>
<td>Document date</td>
<td></td>
<td>Char * =&gt; string</td>
<td>DOC_DATE</td>
<td>ORDER</td>
</tr>
<tr>
<td>cValue</td>
<td>Operation date</td>
<td>( \text{if } (*cValue &lt; \ast aValue) { cValue \Rightarrow aValue; } )</td>
<td>Char * =&gt; string</td>
<td>OP_DATE</td>
<td>ORDER</td>
</tr>
</tbody>
</table>

It is important to note, that business knowledge is identified by matching templates against code fragments in the AST. So, patterns for discovering business rules must be developed which are encoded in loop structures and/or conditional statements and mathematical formulae, which are encoded in loop structures and/or assignment statements.

3.3 Schema Cleaning

The goal of schema cleaning is to remove or replace physical constructs into logical ones and transform complete physical schema into complete logical schema. All the physical constructs can be discarded at this point because they do not provide any information about the database logical structure. The main transformations are:

- Removing indexes.
Removing collections. If collection includes only one entity, then the name of collection is denoted as alternative name of entity.

Result. Schema after removal of indexes and collections is presented in Figure 12.

Figure 12. Complete Logical Schema.

3.4 Preparation

The goal of preparation is to prepare the schema that it only contains structures and constraints that are necessary to understand the semantics of the schema. The technical data structures are discarded. The names of the objects are given by the programmers, who have used some naming rules. Now the names can be changed to give more information on the named objects:

- Removing common prefixes. A common naming conversion consists in prefixing each attribute name by the name (or a short name) of the entity type. Those prefixes do not give any information, so they can be removed.

Result. Changes to attribute names are as follows (Figure 13): attribute „PRODUCT.PRODNUM“ name become „PRODUCT.NUM“, „ORDER.ONUM“ – „ORDER.NUM“, „CLIENT.CID“ – „CLIENT.ID“. The name of attribute „PHONE.PHONE“ is not modified, because it completely match the name of entity.
3.5 Basic Conceptualization

The goal of basic conceptualization is to extract a basic conceptual schema without worrying about esthetical aspects of the result. Two kinds of transformations are used: untranslation and de-optimization. Though distinguishing between them may be arbitrary in some situations (some transformations pertain to both). A complete description of transformations can be found in [12], [14], [17]. The main transformations are:

- Removing redundancy.
- Transforming foreign key into relationship type and internal constraint.
- Transforming foreign key into relationship type.
- Transforming list of attributes into a multivalued attribute.
- Aggregating attributes.

Result. Transformations are as follows:

- Redundancy \( \text{rd}(\text{DETAIL.PROD_NAME} \bowtie \text{PRODUCT.NAME}) \) and its attribute \( \text{DETAIL.PROD_NAME} \) are removed.
- Foreign keys \( \text{CUSTOMER.CID} \), \( \text{SUPPLIER.CID} \) and constraints disjoint(CUSTOMER.CID, SUPPLIER.CID), CLIENT.IDin (CUSTOMER.CID \( \cup \) SUPPLIER.CID) are transformed into relationship types \( \text{customer} \) and \( \text{supplier} \) respectively. General entity \( \text{CLIENT} \) includes new constraint \( \text{exact-1} \). Schema after this transformation is presented in Figure 14.
- Foreign keys \( \text{PHONE.CID} \), \( \text{ORDER.CID} \), \( \text{DETAIL.PRODUCT} \), \( \text{PRODUCT.SUPPLIER} \) are transformed into relationship type \( \text{phone} \), \( \text{order} \), \( \text{detail} \), \( \text{product} \) respectively.
- Equality constraint \( \text{DETAIL.CID} \), \( \text{DETAIL.ORDER} \) is transformed into relationship type \( \text{detail1} \).
- Attributes \( \text{CLIENT.ADDR_CITY} \), \( \text{CLIENT.ADDR_STREET} \), \( \text{CLIENT.ADDR_HOME_NUM} \), \( \text{CLIENT.ADDR_ROOM_NUM} \) are aggregated to compound attribute \( \text{ADDR} \). Schema after transformations of basic conceptualization is presented in Figure 15.
3.6 Normalization

The goal of normalization is to improve the expressiveness, the simplicity, the readability and the extensibility of the conceptual schema. It tries to make higher level semantic constructs explicit (e.g., is-a relation). A complete description of transformations can be found in [12], [14], [17]. The main transformations are:

- Merging entity types.
- Transforming entity type into attribute.
- Transforming entity type into relationship type.
- Transforming relationship types into multi-domain role.
- Transforming relationship types into is-a relation.
- Connecting entity types with is-a relation. If is-a relation of type disjoint or partition is found, it must be presented to the user for a final validation.
- Name processing. Some names can be changed to be more meaningful. For example, the attribute "TOT" can be changed to "TOTAL", "ADDR" - "ADDRESS". A usual naming rule is to use lowercase for relationship type names, uppercase for the entity type names and capitalized for the attribute names.

Result. Transformations are as follows:
- Entity type „PHONE“ is transformed into attribute „CLIENTPHONE“.
- Entity type „DETAIL“ is transformed into relationship type „detail“. Schema after this transformation is presented in Figure 16.
- Relationship types „customer“,”supplier“ are transformed into is-a relation of type partition and constraint exact-1 of entity „CLIENT“ is removed. Schema after this transformation is presented in Figure 17.
- Attribute „CLIENT.AADDR“ is renamed into „CLIENT.ADDRESS“. Attribute names are capitalized. After this transformation final conceptual schema is composed (Figure 18).

Figure 16. Conceptual Schema after Entity Types Transformation into Attribute and Relationship Type.
3.7 Business Knowledge Representation

Six steps of DBRE enable the extraction of the following schema information from the legacy database:

- Entities;
- Compound, multivalued and atomic attributes;
- Attribute constraints;
- Primary and secondary identifiers;
• Relationships;
• Is-a relations;
• Multi-domain roles;
• Business rules.

A conceptual overview of the extracted schema is represented by extended entity-relationship diagram shown in Figure 18.

4 Conclusion

Legacy information systems contain incredible detailed business rules and form the backbone of the information flow of organization, but their maintenance is very expensive and it is very difficult, if not impossible, to expand them. Reverse engineering is the essential part of process of changing and replacing legacy systems. Its main objective is to discover and extract business knowledge from legacy sources. Reverse engineering builds the powerful foundation for renovation of IT systems that enables the application of new technologies and programs.

Database reverse engineering algorithm provided in this article includes two main database reverse engineering processes: data structure extraction and data structure conceptualization. The data structure extraction is the most crucial and difficult part of DBRE. Data structure extraction analyzes the existing legacy system to recover complete logical schema that includes implicit and explicit structures and constructs. Various program understanding and data analysis techniques are used to recover implicit structures and constraints.

The data structure conceptualization interprets logical schema in a conceptual view and recovers conceptual schema. It detects and transforms or discards non-conceptual structures, redundancies, technical optimizations and DMS-dependent constructs.

Six steps of database reverse engineering algorithm enable the extraction of entities, compound, multivalued and atomic attributes, attribute constraints, primary and secondary identifiers, relationships, as-a relations, multi-domain roles and business rules. This knowledge then could be used when changing or replacing legacy systems, i.e. when redeveloping, wrapping, or migrating legacy systems.

References


