Faculteit der Economische Wetenschappen en Econometrie

Serie Research Memoranda

Automatic Relational Database Restructuring

M. Boogaard
R.J. Veldwijk

Research Memorandum 1992-4
maart 1992

vrije Universiteit *amsterdam*

Automatic Relational Database Restructuring

M. Boogaard[1]
R.J. Veldwijk

Vrije Universiteit
De Boelelaan 1105
1081 HV AMSTERDAM
The Netherlands
Phone: (+31) - 20 - 5487081
Fax: (+31) - 20 - 6462645

Abstract

When data requirements change and the existing database structure no longer reflects the current state of the Universe of Discourse, transformation of the information system is inevitable. Although required changes usually involve the addition or deletion of information, part of the process can be considered rule-changing but information-preserving. This part of the transformation process, i.e. restructuring, can be partially accomplished by an algorithm executing several adjustments on a working information system. The premise of this approach is formed by the statement that there is no fundamental distinction between data and metadata. The purpose of this paper is to introduce an elementary operations approach aimed at the automatic reorganization of the structure and contents of a relational database, and the affected application programs.

Keywords

relational database restructuring, maintenance, Imploded Relation

---

[1]. The authors are members of the MESDAG Research Group, a joint project of the N.V. Nederlandse Spoorwegen, RAET N.V., and the Vrije Universiteit Amsterdam. The first author is also affiliated with the Tinbergen Institute.

Automatic Relational Database Restructuring

1. INTRODUCTION

The underlying database schema of an information system models a Universe of Discourse (UoD). The schema should adequately capture the relevant objects, object structure, and rules that apply to the objects of the environment to be modelled. The applying rules can either be reflected in constraints or in the structure of the underlying relational database. Actually, there is no syntactic or semantic difference between rules that are represented as constraints and rules that are appearing in the database structure. The pondering of where to incorporate the rules is, however, of significant relevance because representing rules in the database structure is lucrative during the development of the information system but has negative repercussions on maintenance. The inverse argumentation applies for rules that are reflected as constraints.

Because the structure of the database schema itself stands for certain rules, modifications of rules can bring about schema alterations. However, the observation that the structure of a relational database reflects time-varying rules is normally overlooked. In dynamic environments, the extent of consequences for the information system as a result of alterations in the UoD is an important factor. In case of information-preserving alterations[2], these changes can be considered database *restructuring*. The process of restructuring is defined as modifying the logical structure of an existing source database schema into the desired structure of the target database schema. The process also involves the conversion of the contents of the source database so that it adjusts to the target database. Furthermore, restructuring requires modifications of existing application programs. Whereas logical data independence is still an unfulfilled claim of the relational model (see Graaff et al. 1992), restructuring of relational databases remains a complicated activity.

Several authors have introduced an automatic database restructuring mechanism (see, e.g., Navathe and Fry 1976, Shu et al. 1977, and Navathe 1980). These restructuring systems automate the restructuring of the database itself and mainly focus on hierarchical or network databases. Restructuring of relational databases is seldom taken into consideration. Furthermore, the adjustment of affected application programs that run against the database remain rather underexposed and must be executed manually. On the physical level this would be comparable to rewriting application programs following the alteration of an index. Shneiderman and Thomas (1982) describe the influence of restructuring of a database system on relational algebra. Their purpose is to prove to implementers that automatic database system conversion is feasible.

The purpose of this paper is to describe a restructuring approach using a powerful set of elementary operations that not only automates the restructuring of the database, but also automatically adjust the affected embedded queries of application programs. The approach is supported by an

---

[2] What constitutes the precise information contents of a database is difficult to determine (see Navathe and Fry 1976, p.140). In this paper, information-preserving alterations are defined as modifications of a source database structure with no information explicitly supplied or deleted.

advanced data dictionary and uses an canonical relation that contains both the database contents and description. This canonical relation demonstrates that there is no fundamental distinction between data and metadata. The influence on embedded relational language is described by means of an exemplary relational query facility, i.e. SQL. The restructuring procedure to be described is based on the following three concepts: implosion, conversion and explosion (see Veldwijk et al. 1991a).

The second section formally describes the restructuring process and considers the basic assumptions of database restructuring. The next section introduces our proposed architecture and algorithm for restructuring due to changes in rules which apply in the UoD. The three concepts of the algorithm, i.e. implosion, conversion and explosion, are briefly discussed. Section four through six illuminate these concepts in depth on the basis of an exemplary database restructuring process. The paper concludes with some critical notes and conclusions on the subject.

2. RESTRUCTURING PROCESS

Normally, an information system (IS) is seen to consist of the following three relevant elements: a database schema or structure (DS), data contents[3] (DC), and a collection of application programs (AP). Changing UoD rules affect each of these elements specifically. There are at least three reasons to restructure the underlying database schema of an information system:

a. correcting poor design of an underlying database (e.g., normalization);
b. improving application program performance (e.g., denormalization);
c. adjusting the information system to evolving rules.

The consequences of restructuring on information system elements can be formally described as follows (see Shneiderman and Thomas 1982). In the original situation an information system is represented by IS (DS, DC, AP), where AP = ap_1, ap_2, ..., ap_n. A specific application program (ap_i) that runs against the database yields an output (o_i) such that ap_i (DC) = o_i, for i = 1, 2, ..., n. This implies that the application programs include the description of the database structure in their code. However, the current tendency to eliminate this structural definition from the application program's code also requires the utilization of the database schema as input. Hence, the formal representation would be ap_i (DS, DC) = o_i, for i = 1, 2, ..., n.

The restructuring of the information system, R (IS), results in a new information system (IS'): R (IS) = IS' (DS', DC, AP'). It is important to note that the data contents did not change. Only the way the data contents are stored is logically altered (DS'). Consequently, the influence of restructuring on the data contents consists in re-storing the same data into a different logical data structure. However, the application programs are in effect influenced by restructuring. Database schema alteration requires a comparable restructuring of the embedded DML in the application programs in order to retain their syntactic and semantic validity. The

---

[3]. The term 'data contents' is adopted from Shneiderman and Thomas (1982). As will become clear, the exact data contents of a database are subject to change due to the restructuring process. It is rather the information contents of the database that remain the same.

application program restructured (ap_i') now yields ap_i' (DS', DC) = o_i', for i = 1, 2, ..., n. The restructuring process should maintain input/output equivalence, i.e. $o_i = o'_i$. In fact, input/output equivalence is another criterion for restructuring besides its information-preserving nature.

Because the restructuring process is information-preserving, restructuring is invertible as long as no data updates activities (update, insert, and delete) take place. However, for a certain category of restructuring processes structural invertibility is preserved even after the data contents is altered. This category consists of restructuring processes that tense the rules that apply to the UoD of the information system. In other words, they narrow the allowed states of the information system. For example, the transformation of the rule *'a part can be delivered by any number of suppliers'* into *'a part can be delivered by at most one supplier'* is always revertible independent of the contents of the information system. The only difference is that the information contents can be changed in comparison with the initial situation. The same, however, does not apply for restructuring processes that relax the rules of an information system.

In conclusion, restructuring does not involve adding or destroying information, but alters the logical database structure, relaxing or constraining the permissible states of the database. The database contents must be re-stored in accordance with the new logical structure. Restructuring also affects application programs that run against the database. Hence, the restructuring process should provide support for the modification of the all three information system elements.

3. ELEMENTARY DATABASE STRUCTURE OPERATIONS

Support of the restructuring process is a cumbersome activity because the specific characteristics of the process depend on the source information system and the desired target information system. Therefore, each restructuring process is unique and must be analyzed individually. This observation points to a well-known problem which can be solved by splitting up specific actions into a certain amount of smaller generic instructions. The sequential execution of these instructions assemble the specific action desired. Hence, the design of a generally applicable procedure to support restructuring requires the definition of a preferably small closed set of fundamental operations, i.e. Elementary Database Structure Operations (EDSOs). EDSOs can be considered 'building blocks' with which a specific restructuring of an information system can be described.

Furthermore, each EDSO follows a standard procedure which makes it possible to automate the restructuring process to a large extent. The decision whether or not to restructure can now be merely based on considerations of desirability rather than technical and economical feasibility. In other words, the short term incentive not to alter a database structure because of its widespread repercussions to many application programs disappears. Comparable problems at the physical level have long been resolved by the use of RDBMSs.

Figure 1 charts this restructuring architecture, inspired by the conversion system architecture of Shneiderman and Thomas (1982).

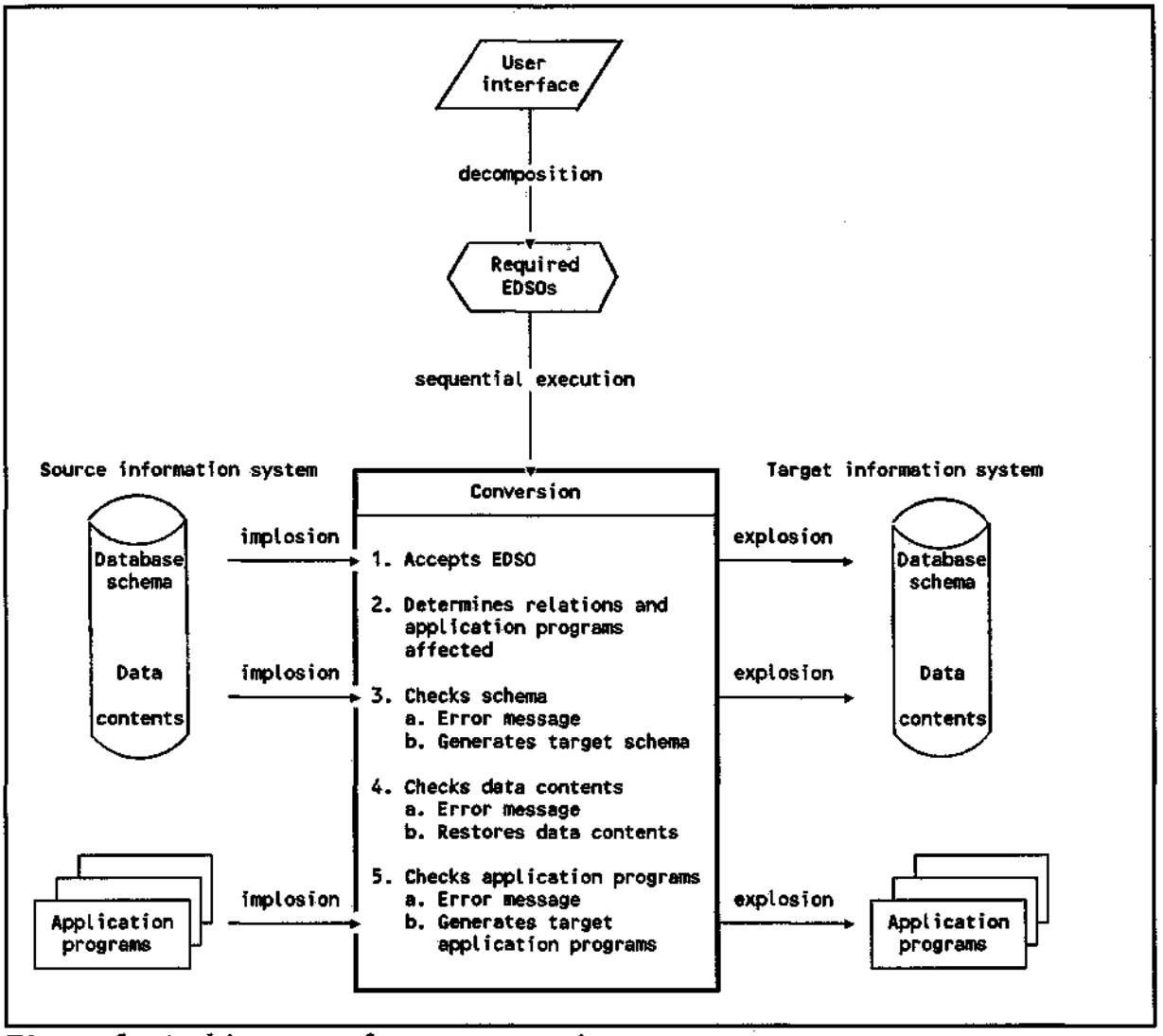


Figure 1. Architecture for restructuring

In essence, the restructuring system proposed follows a prevalent transformation procedure. The procedure hinges on the translation of a specific source system to a canonical form. Consequently, the source system with individual features is transformed to a standard lay-out. The result is that operations specific for the source system can now be realized using generalized and straightforward procedures. The desired target system will finally be derived from the re-arranged canonical form. The application of such a translation procedure in the database area has been described by Shoshani and Brandon (1975). Their conversion system consists of the following three components:

1) Source reformatter; which translates the source database into a predefined standard data form;
2) Converter; which logically converts the data from the source standard form to a target standard form, and
3) Target reformatter; which translates the target standard data form into the target database.

These authors apply this transformation procedure to database conversion, for example from a hierarchical database into a network database.

The EDSO approach employs an analogous translation procedure to the

restructuring of the database, and the adjustment of the affected embedded queries of application programs. After the required EDSOs are induced from the information system restructuring needed, each EDSO must be executed sequentially on the basis of three steps. The first step, *implosion*, is the same for every EDSO. Both structure and contents of every involved relation of the database is translated to the contents of a single relation with an invariant structure, i.e. the Imploded Relation. Consequently, implosion blurs the distinction between instance data and structural data (meta data) because both types of data are now contents of the Imploded Relation. Furthermore, every affected embedded query of application programs that run against the database is re-formulated in accordance to the Imploded Relation. The imploded version of the information system yields the same result as in the pre-implosion situation. Every query upon this canonical relation looks quite the same as far as form is concerned.

After this first step, nothing has actually changed. Both the relations of the database and the affected queries of application programs are transformed to a standard form which still reflects the same information system. The concrete alteration of the information system is instituted during the second step, *conversion*, which is specific for the EDSO to be executed. As a result of implosion, database schema modifications can easily be realized by means of DML statements on the Imploded Relation because the original structure of the database is now represented by contents. Furthermore, the required adjustments of the queries of application programs follow a standard procedure which can be automated because each query on the Imploded Relation looks quite alike. Before the conversion is actually enforced, the schema, data contents and/or the application programs are checked if necessary. If none of the constraints has been violated, both the imploded relations and the imploded queries are converted conform to the specifications of the schema modification.

The last step, *explosion*, is again the same for every EDSO. During the explosion process the new database schema is physically created, the original data contents are re-stored into this new schema, and the restructured queries are embedded into their original application programs in a 'normal' form. The restructuring process is completed when all required EDSOs have been executed.

It should be noted that this three step procedure is conceptual. It is possible to implement the restructuring procedure in a different manner, using this framework. In conclusion, the elimination of the need to logical data independence as far as restructuring is concerned can be realized by means of a substitute that restructures both the database and the application programs.

The subsequent sections illuminate each of the three steps of the EDSO procedure by means of an exemplary restructuring process. The example considers a personnel information system. In the initial situation the rule *'an employee can work for more than one department'* applies. This rule represents a many-to-many relationship between the objects employee and department. In a relational environment, many-to-many relationships between two relations are structured by means of two one-to-many relationships and an associative relation (see Figure 2). (The primary key of each relation is underlined. The arrows represent foreign-to-primary-key relationships.)

Source database structure:

Source database description and contents:

DEPARTMENT

DEPNO	DEPNAME	BUDGET
D1	STAFF	150000
D2	SALES	100000

ASSIGNMENT

DEPNO	EMPNO
D1	E1
D2	E2
D1	E3

EMPLOYEE

EMPNO	EMPNAME
E1	SMITH
E2	JONES
E3	BLACK

Exemplary information request on the source database (stated in SQL):

```
select   EMPNAME, DEPNAME
from     DEPARTMENT DEP, ASSIGNMENT ASG, EMPLOYEE EMP
where    ASG.DEPNO = DEP.DEPNO
and      ASG.EMPNO = ASG.EMPNO
order by ASG.DEPNO
```

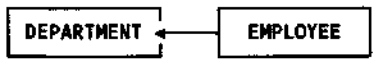
Results into:

DEPNAME	EMPNAME
STAFF	SMITH
STAFF	BLACK
SALES	JONES

Figure 2. Source database

Suppose, personnel management decides to modify the assignment policy so that an employee can work for only one department. As a result, the source database no longer reflects the rules that apply to its UoD. To be more specific, the many-to-many relationship must be changed into a one-to-many relationship. The target database, therefore, has to consist of two relations and one one-to-many relationship (see Figure 3).

Target database structure:



Target database description and contents:

DEPARTMENT

DEPNO	DEPNAME	BUDGET
D1	STAFF	150000
D2	SALES	100000

EMPLOYEE

EMPNO	EMPNAME	DEPNO
E1	SMITH	D1
E2	JONES	D2
E3	BLACK	D1

Exemplary information request on the target database (stated in SQL):

```
select   EMPNAME, DEPNAME
from     DEPARTMENT DEP, EMPLOYEE EMP
where    EMP.DEPNO = DEP.DEPNO
order by EMP.DEPNO
```

Results into:

DEPNAME	EMPNAME
STAFF	SMITH
STAFF	BLACK
SALES	JONES

Figure 3. Target database

In order to proceed the restructuring process, the EDSO procedure is supported by an advanced data dictionary. This data dictionary holds information about the current structure of the database. In fact, it captures all relevant elements of the relational model, for example the primary key of the relations, the foreign keys of relations, and

relationships between relations (see Veldwijk et al. 1991b). Figure 4 depicts a part of the data dictionary relevant for the EDSO procedure. The data dictionary relations contain the exemplary source database.

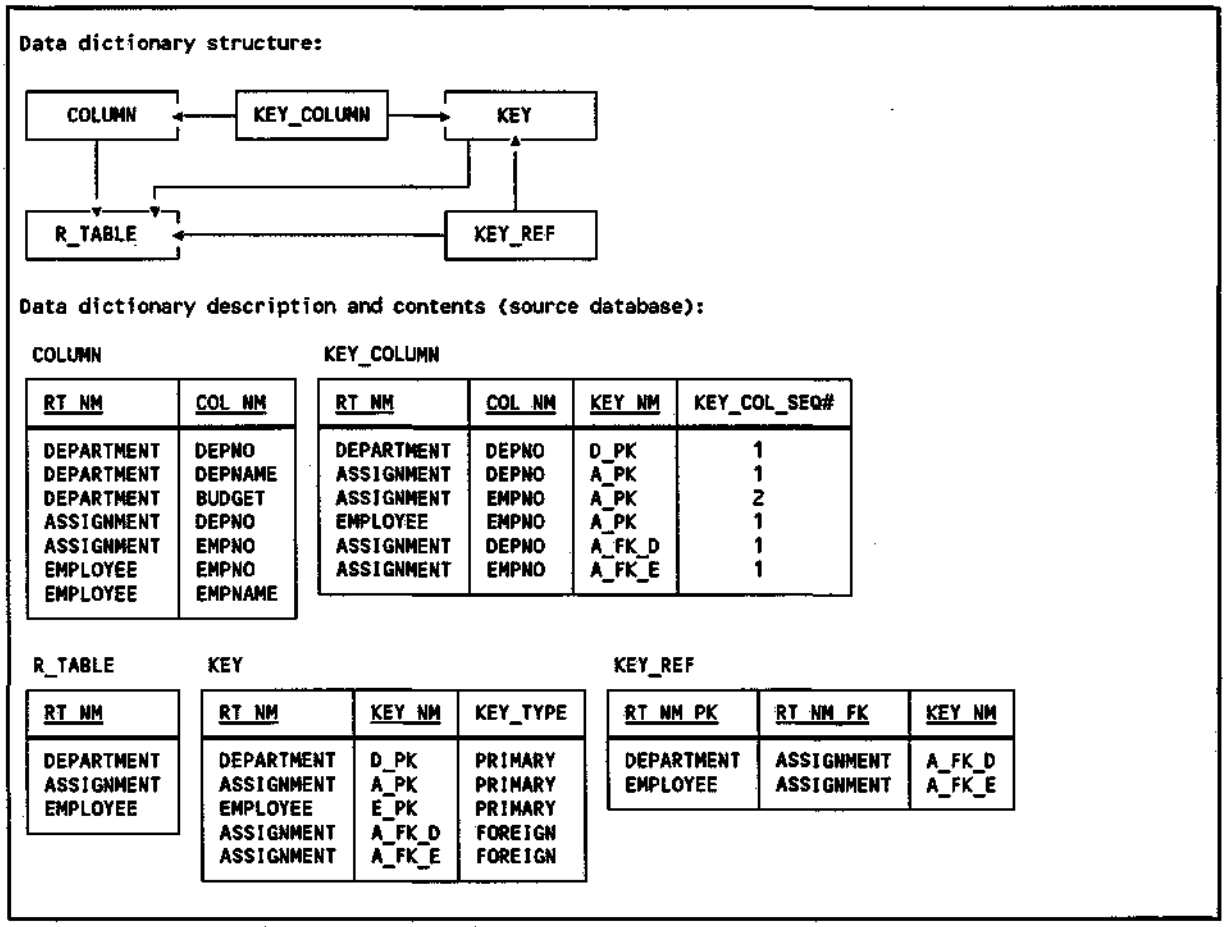


Figure 4. Data dictionary model

4. IMPLOSION

In this section, the exemplary source database will be translated to the Imploded Relation. Furthermore, the exemplary query will be re-formulated on the Imploded Relation in order to yield the same result as in the initial situation. This section addresses our statement that there is no fundamental distinction between data and metadata.

The underlying contemplation for the design of the Imploded Relation is to consider the description of the database relations as instances of the Imploded Relation (see Boogaard et al. 1992). Curtice (1981) labels this approach the "self-defining data element technique". The Imploded Relation is, therefore, structured on the basis of elements that characterize relational databases, namely: (1) the name of the relation, (2) the name of the column, and (3) the value of the column. However, these elements will result into a list rather than a relation. Moreover, the implicit relationship between the column values that represent an original tuple is lost. Hence, the Imploded Relation requires a surrogate identifier that explicitly reflects the implicit relationship between the column

values of a tuple. In conclusion, the Imploded Relation can be described as follows (the primary key columns are underlined):

I_RELATION (RT_NM, COL_NM, T_CODE, VALUE)

Where:

RT_NM - Name of the relation;
COL_NM - Name of the column;
T_CODE - Identifier of the tuple, relating the column values occurring in the same initial tuple, and
VALUE - Value of the column.

During the translation procedure, or *implosion*, each value of the involved database relations becomes a single tuple in the Imploded Relation. This value, together with the name of both column and relation in which the value participates and an identification of the tuple to which the value belongs, is stored in the Imploded Relation. For example, the tuple "'D1' 'STAFF' 150000" is transformed into three tuples in the Imploded Relation connected having the same relation name and tuple identifier. The application of the translation procedure for the exemplary source database results in the (see Figure 5).

I_RELATION

RT_NM	COL_NM	T_CODE	VALUE
DEPARTMENT	DEPNO	1	D1
DEPARTMENT	DEPNAME	1	STAFF
DEPARTMENT	BUDGET	1	150000
DEPARTMENT	DEPNO	2	D2
DEPARTMENT	DEPNAME	2	SALES
DEPARTMENT	BUDGET	2	100000
ASSIGNMENT	DEPNO	1	D1
ASSIGNMENT	EMPNO	1	E1
ASSIGNMENT	DEPNO	2	D2
ASSIGNMENT	EMPNO	2	E2
ASSIGNMENT	DEPNO	3	D1
ASSIGNMENT	EMPNO	3	E3
EMPLOYEE	EMPNO	1	E1
EMPLOYEE	EMPNAME	1	SMITH
EMPLOYEE	EMPNO	2	E2
EMPLOYEE	EMPNAME	2	JONES
EMPLOYEE	EMPNO	3	E3
EMPLOYEE	EMPNAME	3	BLACK

***Figure 5.** Imploded Relation*

Obviously, the structure of the source database is hard to distinguish for humans[4]. This information, however, can be found in the supporting data dictionary. The description of the values, for instance data type and format, are also stored in the data dictionary. This part of the data dictionary is not reflected in Figure 4 for reasons of simplicity. Both structure and contents of the source database occur as contents of the Imploded Relation. The Imploded Relation, therefore, shows that the

4. The original database relations can always be re-established using the Imploded Relation and the data dictionary relations which can also be included in the Imploded Relation.

distinction between data and metadata is arbitrary and not fundamental.

In order to yield the same result, the exemplary query has to be re-formulated to the Imploded Relation. Because the Imploded Relation is the only relation involved, several joins over the Imploded Relation itself are required to re-formulate the exemplary query. Therefore, aliases are designated to the columns of the exemplary source database. (These aliases do not affect the assignment of the aliases of the original query.) Re-formulation of the exemplary query results in the imploded query of Figure 6.

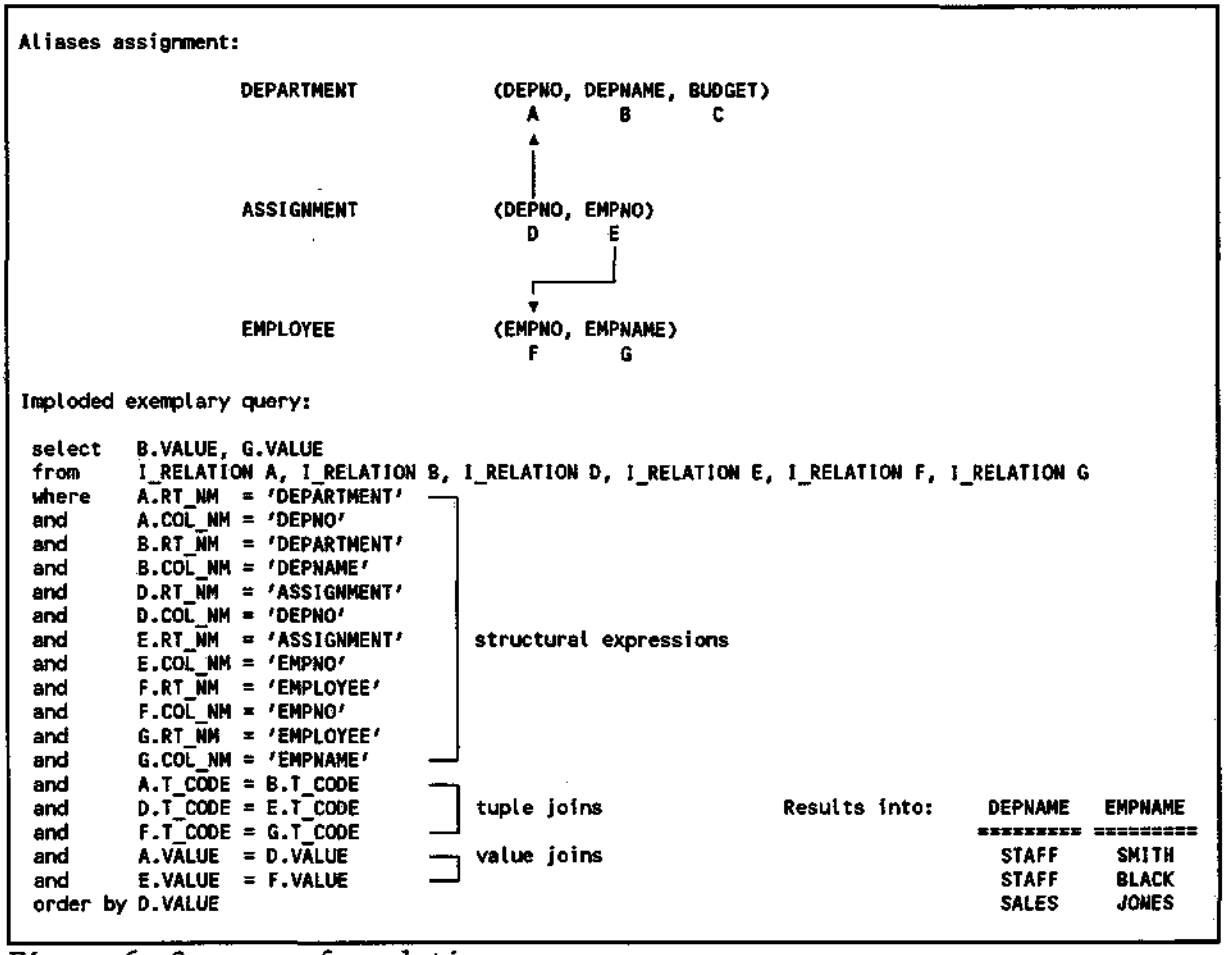


Figure 6. Query re-formulation

The re-formulation of the query has three important effects on the WHERE clause. First, the WHERE clause contains structural declarations of the involved columns. The structural expressions defines the involved relations (e.g. "A.RT_NM = 'DEPARTMENT'") and columns (e.g. "A.COL_NM = 'DEPNO'"). Originally, these structural elements are part of both the FROM and the WHERE clause. Second, because the implicit relationships between the column values of a tuple are replaced by an explicit relationship, so-called tuple joins are necessary. In our example, the column values of DEPNO and DEPNAME are related by means of the tuple join "A.T_CODE = B.T_CODE". Because these implications apply to every conceivable query, imploded queries display a simple and regular structure. As a result, the imploded queries can be modified rather plainly. Finally, the normal joins appear as value joins in the imploded queries ("A.VALUE = D.VALUE").

5. CONVERSION

The required exemplary EDSO can only be executed if the database satisfies the following three conditions: (1) the associative relation does not contain columns that do not belong to the primary key of the other two relations, (2) the associative relation does not participate in any other relationships than the two relationships considered, and (3) the contents of the associative relation do not reflect multiple values in the primary key columns of the upcoming many-side relation.

This EDSO requires an outer natural join between the associative relation (ASSIGNMENT) and the upcoming many-side relation (EMPLOYEE). (The relational algebra notation is used to ensure the information-preserving nature of the EDSO.) The columns of the two relations that are needed for this outer join are the primary key columns of the upcoming many-side relation and the foreign key columns of the associative relation to the many-side relation. The resulting relation of this outer natural join replaces the upcoming many-side relation. Furthermore, this EDSO results in the elimination of the associative relation. The data dictionary information request to support conversion is depicted in Figure 7.

Data dictionary information request to compose the outer natural join:

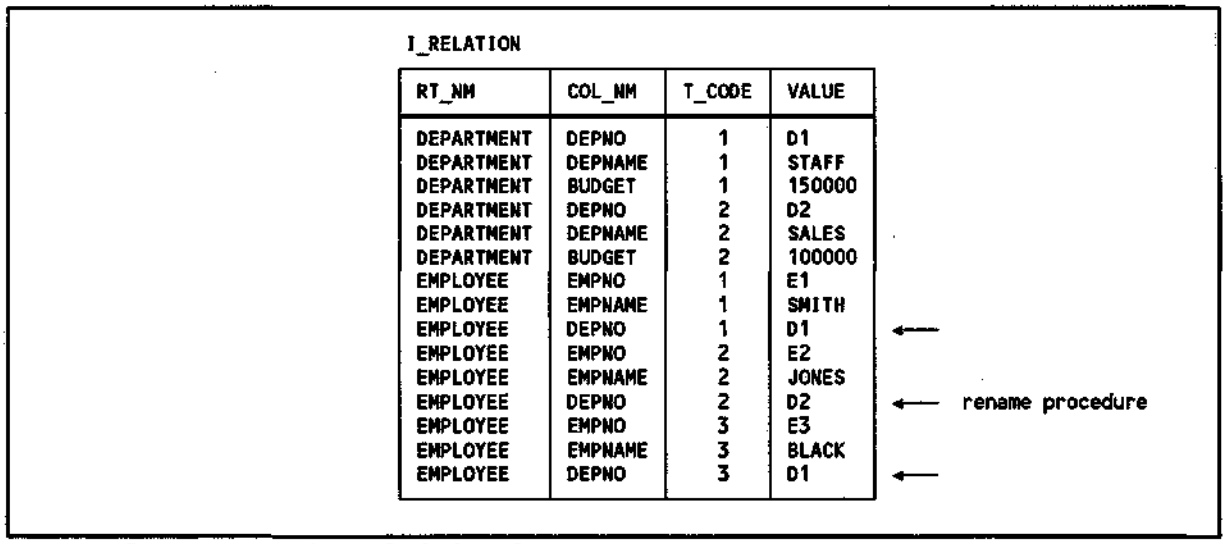
```
select distinct PKC.COL_NM, FKC.COL_NM
from   KEY_COLUMN PKC,
       KEY_COLUMN FKC,
       KEY PK,
       KEY FK,
       KEY_REF KR
where  KR.RT_NM_PK  = 'EMPLOYEE'
and    KR.RT_NM_FK  = 'ASSIGNMENT'
and    KR.RT_NM_FK  = FK.RT_NM
and    KR.KEY_NM    = FK.KEY_NM
and    FK.RT_NM     = FKC.RT_NM
and    FK.KEY_NM    = FKC.KEY_NM
and    KR.RT_NM_PK  = PK.RT_NM
and    PK.KEY_TYPE  = 'PRIMARY'
and    PK.RT_NM     = PKC.RT_NM
and    PK.KEY_NM    = PKC.KEY_NM
and    PKC.KC_SEQNO = FKC.KC_SEQNO;

Results into:  PKC.COL_NM   FKC.COL_NM
               ============ ============
                  EMPNO        EMPNO
```

This result in the following outer natural join:

EMPLOYEE (EMPNO, EMPNAME, DEPNO) = ASSIGNMENT [EMPNO ⊛ EMPNO] EMPLOYEE

Figure 7. Data dictionary support

These required alterations can be executed by means of two procedures on the Imploded Relation. First, a rename procedure is required to modify the associative relation name of tuples into the upcoming many-side relation name for the column names of the associative relation that do not participate in the foreign key to the primary key of the upcoming many-side relation (DEPNO). The tuple identifier of these tuples must be in accordance with the tuple identifiers of the corresponding values of the primary key of the upcoming many-side relation. This rename procedure simulates the outer natural join (possible null values are added during the explosion). Second, a delete procedure is necessary to eliminate the other tuples with the associative relation name.



I_RELATION

RT_NM	COL_NM	T_CODE	VALUE	
DEPARTMENT	DEPNO	1	D1	
DEPARTMENT	DEPNAME	1	STAFF	
DEPARTMENT	BUDGET	1	150000	
DEPARTMENT	DEPNO	2	D2	
DEPARTMENT	DEPNAME	2	SALES	
DEPARTMENT	BUDGET	2	100000	
EMPLOYEE	EMPNO	1	E1	
EMPLOYEE	EMPNAME	1	SMITH	
EMPLOYEE	DEPNO	1	D1	←
EMPLOYEE	EMPNO	2	E2	
EMPLOYEE	EMPNAME	2	JONES	
EMPLOYEE	DEPNO	2	D2	← rename procedure
EMPLOYEE	EMPNO	3	E3	
EMPLOYEE	EMPNAME	3	BLACK	
EMPLOYEE	DEPNO	3	D1	←

Figure 8. *Converted Imploded Relation*

This EDSO impair queries because now a join over the new many-side relation (EMPLOYEE) is required instead of the deleted associative relation (ASSIGNMENT) in order to obtain the same result. Conversion of the exemplary query starts with the replacement of ASSIGNMENT by EMPLOYEE in the structural expressions where "D.RT_NM = 'ASSIGNMENT'" and "D.COL_NM = 'DEPNO'". The remaining structural expressions concerning ASSIGNMENT and the columns of ASSIGNMENT can be removed ("E.RT_NM = 'ASSIGNMENT'" and "E.COL_NM = 'EMPNO'"). Furthermore, the columns of EMPLOYEE that constitute the foreign-to-primary-key relationship between EMPLOYEE and ASSIGNMENT can also be deleted ("F.RT_NM = 'EMPLOYEE'" and "F.COL_NM = 'EMPNO'"). The value join that reflects this relationship can be removed too (E.VALUE = F.VALUE). The declaration of the aliases of the structural expressions that are no longer part of the imploded exemplary query can be deleted from the FROM clause (I_RELATION E and I_RELATION F). The only adjustment that is left is the alteration of the tuple joins. Because the columns with aliases E and F are removed, the tuple joins involving these aliases can be deleted. The tuple join "D.T_CODE = G.T_CODE" has to be included in the imploded exemplary relation. The exemplary query is now adjusted to the re-arranged converted Imploded Relation (see Figure 9).

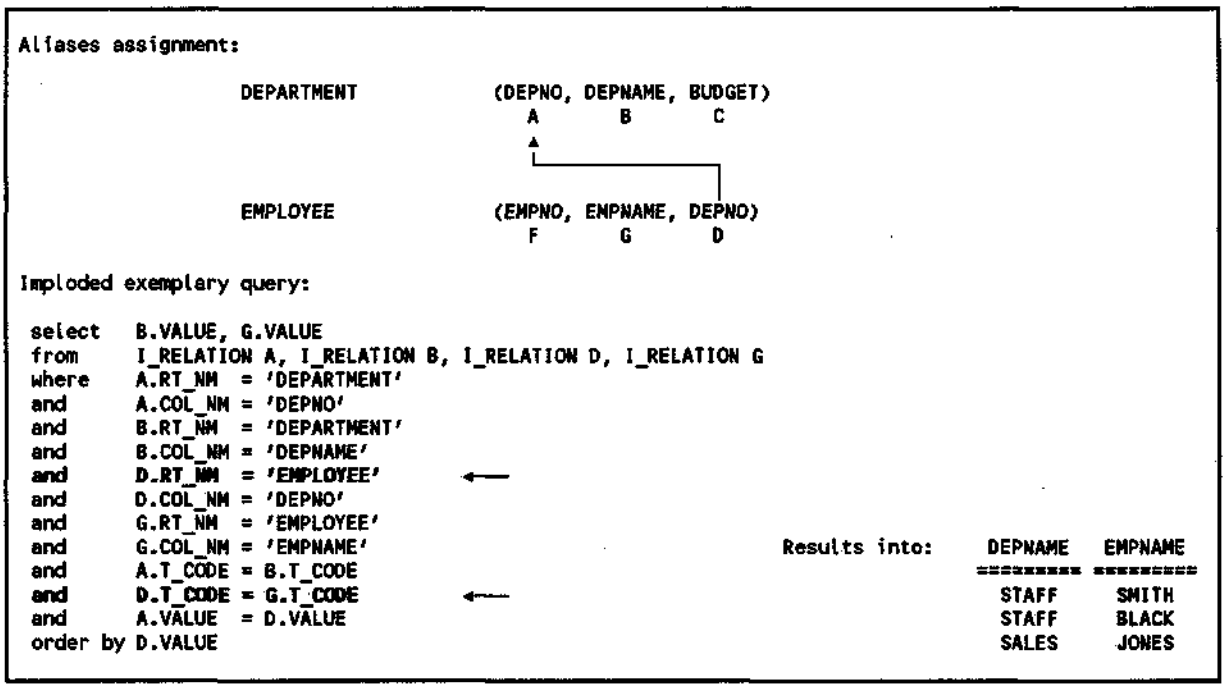


Figure 9. Coverted imploded query

6. EXPLOSION

The final step of the EDSO procedure is the translation, i.e. explosion, of the converted Imploded Relation to a normal form. This implies that the target database is created in physical database relations. The data contents are re-stored from the Imploded Relation into the corresponding database relations using the tuple identifiers. The missing column values in the Imploded Relation are marked null during this re-store process. The information whether or not the column can accept nulls is obtained from the data dictionary. The many-to-many relationship between DEPARTMENT and EMPLOYEE has been changed into a one-to-many relationship. Furthermore, the converted queries are transformed to their conventional form and re-positioned in the original places in the application programs. The exemplary target database and query were displayed in Figure 3. After the affected application programs are compiled and linked, and all selected EDSOs are executed, the restructuring process is completed. The information system now reflects its UoD adequately.

7. CONCLUDING REMARKS

The purpose of the paper has been to introduce an automatic relational database restructuring algorithm using EDSOs. Consequently, part of application maintenance due to changes in UoD rules can be eliminated. The premise of the proposed approach is formed by the statement that there is no fundamental distinction between data and metadata. This statement is founded by means of the translation of both contents and description of database relations to a canonical relation, the so-called Imploded Relation. It should be stressed that the EDSO procedure described is a

conceptual framework; performance is not the issue here.

In order to support restructuring processes resulting from changes in rules that apply to the UoD, it is crucial to determine a powerful or even a complete set of EDSOs. The completeness of the set elementary operations is currently being researched in both an empirical and formal way. Furthermore, the impact of the EDSO approach on the development and maintenance of information systems will be subjects for further research too.

ACKNOWLEDGEMENT

The authors would like to thank dr. Edu Spoor for his comments on earlier versions of this paper.

REFERENCES

Boogaard, M., Veldwijk, R.J., and Spoor, E.R.K. (1992), "An Alternative Approach for Generalization in the Relational Model," To appear in the *Proceedings of the IFIP WG 8.1 Working Conference*, Alexandria, Egypt, April 13-15, 1992.

Curtice, R.M, (1981), "Data Dictionaries: An Assessment of Current Practice and Problems," *Proceedings of the Seventh International Conference on Very Large Databases*, Cannes, France, 564-570.

Graaff, J. de, Veldwijk, R.J., and Boogaard, M., (1992) "Why Views Do Not Provide Logical Data Independence." Submitted for publication.

Navathe, S.B. and Fry, J.P. (1976), "Restructuring for Large Databases: Three Levels of Abstraction," *ACM Transactions on Database Systems*, 1, 2, 138-158.

Navathe, S.B. (1980), "Schema Analysis for Database Restructuring," *ACM Transactions on Database Systems*, 5, 2, 157-184.

Shneiderman, B and Thomas, G. (1982), "An Architecture for Automatic Relational Database System Conversion," *ACM Transactions on Database Systems*, 7, 2, 235-257.

Shoshani, A. and Brandon, K. (1975), "On the Implementation of a Logical Data Base Converter," *Proceedings of the First International Conference on Very Large Databases*, Massachusetts, USA, 529-531.

Shu, N.C., Housel, B.C., Taylor, R.W., Ghosh, S.P. and Lum, V.Y. (1977), "EXPRESS: A Data EXtraction, Processing, and REStructuring System," *ACM Transactions on Database Systems*, 2, 2, 134-174.

Veldwijk, R.J., Boogaard, M., Dijk, M.V. van, and Spoor, E.R.K., (1991a), "EDSOs, Implosion and Explosion: Concepts to Automate Part of Application Maintenance of Relational Database," *Information and Software Technology*, 33, 5, 343-350.

Veldwijk, R.J., Buitendijk, R.B., Spoor, E.R.K., and Boogaard, M., (1991b), "Towards a Catalog Standard for the Relational Model Version 2: A Manifesto," *Serie Research Memorandum*, Vrije Universiteit Amsterdam, 1991-50.

INTRODUCTION

The MESDAG project is a joint project endorsed by three organizations in the Netherlands: the N.V. Nederlandse Spoorwegen (The Netherlands Railways Company), RAET N.V. and the Vrije Universiteit of Amsterdam. The MESDAG project originated at RAET N.V. during the second half of 1989 as an outgrowth of research done in the field of active data dictionary models. This research and a prototype of an active data dictionary form the basis for the mission of the MESDAG project that officially started its activities in September 1990.

MESDAG is an abbreviation of:

MEta Systems Design And Generation

MISSION AND OBJECTIVES

The mission of the MESDAG project is to prove the feasibility of developing inherently flexible information systems by introducing higher levels of logical data independence.

Derived from this mission following are the two main objectives:

1. Examine the feasibility and initiate the development of an active, self-referential data dictionary model in which both a description of the database data and a description of all specifiable application design data can be stored. This data dictionary model should contain sufficient semantic aspects (like domains, constraints and time aspects) to assure the integrity, consistency and validity of the stored (meta) data, to avoid maintenance and to support query-formulation independent of current database structure.

2. Examine the feasibility and initiate the development of the possibilities of data dictionaries in general and the described data dictionary in specific. This analysis of possibilities is directed at the embedding in and developing methods, techniques, methodologic guidelines and automated tools for the design, implementation and maintenance of flexible information systems.

MEMBERS OF THE MESDAG RESEARCH GROUP

1. Dr. E.R.K. Spoor
Dr. E.R.K. Spoor is associate professor at the Vrije Universiteit Amsterdam. He teaches and consults in the area of database systems and database development with a focus on the use of these technologies in organizations. His eighteen years of experience with computer technology includes eight years with NCR and six years with the Vrije Universiteit, first as a systems engineer and later as a computer scientist. He is one of the founders and board members of two automation oriented organizations: PSB (Amsterdam) and VDA (Hilversum).

2. Drs. R.J. Veldwijk
Drs. R.J. Veldwijk graduated from the Vrije Universiteit Amsterdam in 1986. In his quality as consultant at RAET N.V. Utrecht, he is among others responsible for the design and implementation of advanced database architectures. His main interest lies in developing and implementing self-knowledgeable database models, aimed at reducing maintenance costs and at improving the accessibility of databases by end-users. Furthermore he teaches courses in data modelling.

3. Drs. M. Boogaard
Drs. M. Boogaard is assistant researcher at the Vrije Universiteit Amsterdam. Furthermore, he is part-time involved in projects by the Netherlands Railways Company. He graduated from the Vrije Universiteit Amsterdam, in August 1990. The objective of his research is to develop an approach to achieve higher levels of logical data independence for both end-users and application programs and to analyze the consequences of the level of logical data independence accomplished on the system development life cycle in general and on software maintenance and database inquiry in particular.

ACCOMMODATION ADDRESS

Vrije Universiteit
Faculteit Economie & Econometrie
Vakgroep BIK
De Boelelaan 1105
1081 HV Amsterdam Phone: +31-20-5487081
The Netherlands Fax : +31-20-6462645

1991-1	N.M. van Dijk	On the Effect of Small Loss Probabilities in Input/Output Transmission Delay Systems
1991-2	N.M. van Dijk	Letters to the Editor: On a Simple Proof of Uniformization for Continious and Discrete-State Continious-Time Markov Chains
1991-3	N.M. van Dijk P.G. Taylor	An Error Bound for Approximating Discrete Time Servicing by a Processor Sharing Modification
1991-4	W. Henderson C.E.M. Pearce P.G. Taylor N.M. van Dijk	Insensitivity in Discrete Time Generalized Semi-Markov Processes
1991-5	N.M. van Dijk	On Error Bound Analysis for Transient Continuous-Time Markov Reward Structures
1991-6	N.M. van Dijk	On Uniformization for Nonhomogeneous Markov Chains
1991-7	N.M. van Dijk	Product Forms for Metropolitan Area Networks
1991-8	N.M. van Dijk	A Product Form Extension for Discrete-Time Communication Protocols
1991-9	N.M. van Dijk	A Note on Monotonicity in Multicasting
1991-10	N.M. van Dijk	An Exact Solution for a Finite Slotted Server Model
1991-11	N.M. van Dijk	On Product Form Approximations for Communication Networks with Losses: Error Bounds
1991-12	N.M. van Dijk	Simple Performability Bounds for Communication Networks
1991-13	N.M. van Dijk	Product Forms for Queueing Networks with Limited Clusters
1991-14	F.A.G. den Butter	Technische Ontwikkeling, Groei en Arbeidsproduktiviteit
1991-15	J.C.J.M. van den Bergh, P. Nijkamp	Operationalizing Sustainable Development: Dynamic Economic-Ecological Models
1991-16	J.C.J.M. van den Bergh	Sustainable Economic Development: An Overview
1991-17	J. Barendregt	Het mededingingsbeleid in Nederland: Konjunktuurgevoeligheid en effektiviteit
1991-18	B. Hanzon	On the Closure of Several Sets of ARMA and Linear State Space Models with a given Structure
1991-19	S. Eijffinger A. van Rixtel	The Japanese Financial System and Monetary Policy: a Descriptive Review
1991-20	L.J.G. van Wissen F. Bonnerman	A Dynamic Model of Simultaneous Migration and Labour Market Behaviour