Knowledge Transfer in Database Reverse Engineering - A Supporting Case Study

Hainaut, Jean-Luc; Englebert, Vincent; Hick, Jean-Marc; Henrard, Jean; Roland, Didier

Published in:
Proc. of the 4th IEEE Working Conference on Reverse Engineering

Publication date:
1997

Database Reverse Engineering - A Case Study

J-L. Hainaut, D. Roland, V. Englebert, J-M. Hick, J. Henrard
Institut d'Informatique, University of Namur, rue Grandgagnage, 21 - B-5000 Namur
jlh@info.fundp.ac.be

Abstract. This paper presents a generic methodology for database reverse engineering comprising two main steps, namely Data structure extraction and Data structure conceptualization. The first process tries to elicit the physical data structures of the database, while the second one tries to recover the semantics of these structures. This methodology is illustrated by a small, but non-trivial, example through which a set of COBOL files is transformed into a relational database.

Keywords: database, reverse engineering, methodology, CASE tool

1. Introduction

Database reverse engineering is a software engineering process through which the analyst tries to understand and redocument the files and/or the database of an application. More precisely, this process is to yield the complete logical schema and the conceptual schema of this database. The problem is particularly complex when old, ill-designed and poorly (if any) documented applications are processed. Grossly speaking, the most frequent problems can be classified as follows.

- *Implicit structures*. Such constructs have not been explicitly declared in the DDL specification of the database. We distinguish two kinds of implicit structures: hidden structures which have been intentionally undeclared (e.g. compound fields in COBOL files), and untranslatable structures which cannot be expressed due to the expressive weakness of the DBMD (e.g. foreign keys in Oracle 5 databases). In favorable situations, these lost constructs are managed in procedural components of the application: programs, dialog procedures, triggers, etc, and can be recovered through procedural analysis.
- *Optimized structures*. For technical reasons, such as time and/or space performance optimization, many database structures include non semantic constructs. In addition, redundant and unnormalized constructs are added to try to get better response time.
- *Awkward design*. Not all databases were built by experienced designers. Novice and untrained developers, generally unaware of database theory and database methodology, can produce poor or even wrong structures.
- *Obsolete constructs*. Some parts of a database can be abandonned, and ignored by the current programs.
- *Cross-model influence*. The cultural background of some designers can lead to very peculiar results. For instance, some relational databases actually are straighforward translations of IMS databases, of COBOL files or of spreadsheets.

Several authors have addressed the problem of non-standard data structures [1,2,14,28,29]. The Database Research Group of the University of Namur has proposed a general methodology for tackling this problem [13], and has developed a generic CASE tool [??] to support reverse engineering processes. The problem of recovering implicit structures have been particularly developed in [??]

In this paper, we first recall the main aspects of the methodology, then we develop a small case study which illustrates, despite its very small size, the problems encountered in actual situations, and which shows the reasonings and CASE features that can help solve these problems. This work is a part of the DB-MAIN[1] project, dedicated to Database Application Evolution and Maintenance [15].

2. A Generic Methodology for Database Reverse Engineering

The problems that arise when one tries to recover the documentation of the data naturally fall in two categories that are addressed by the two major processes in DBRE, namely data structure extraction and data structure conceptualization. By naturally, we mean that these problems relate to the recovery of two different schemas, and that they require quite different concepts, reasonings and tools. In addition, each of these processes grossly appears as the reverse of a standard database design process (resp. physical and logical design). We will describe briefly some of these processes and the problems they try to solve.

This methodology is generic in two ways. First, its architecture and its processes are largely DMS[2]-independent. Secondly, it specifies what problems have to be solved, and in which way, rather than the order in which the actions must be carried out. Its general architecture is outlined in figure 1.

2.1 The Data Structure Extraction Process

This phase consists in recovering the complete DMS schema, including all the implicit and explicit structures and constraints. True database systems generally supply, in some readable and processable form, a description of this schema (data dictionary contents, DDL texts, etc). Though essential information may be missing from this schema, it is a rich starting point that can be refined through further analysis of the other components of the application (views, subschemas, screen and report layouts, procedures, fragments of documentation, database content, program execution, etc).

The problem is much more complex for standard files, for which no computerized description of their structure exists in most cases. The analysis of each source program provides a partial view of the file and record structures only. For most real-world applications, this analysis must go well beyond

[1] This research is a part of the DB-MAIN project, which is partly supported by a consortium comprising ACEC-OSI (Be), ARIANE-II (Be), Banque UCL (Lux), BBL (Be), Cap Gemini (Lux), Centre de recherche public H. Tudor (Lux), Cliniques Univ. St-Luc (Be), CGER (Be), Cockerill-Sambre (Be), CONCIS (Fr), D'Ieteren (Be), DIGITAL (Be), EDF (Fr), EPFL (CH), GEDIS (Be), Groupe S (Be), IBM (Be), OBLOG Software (Port), ORIGIN (Be), TEC Charleroi, Ville de Namur (Be), Winterthur (Be), 3 Suisses (Be). The DB-PROCESS subproject is supported by the *Communauté Française de Belgique*.

[2] A Data Management System (DMS) is either a File Management System (FMS) or a Database Management System (DBMS).

the mere detection of the record structures declared in the programs, as will be shown in the case study.

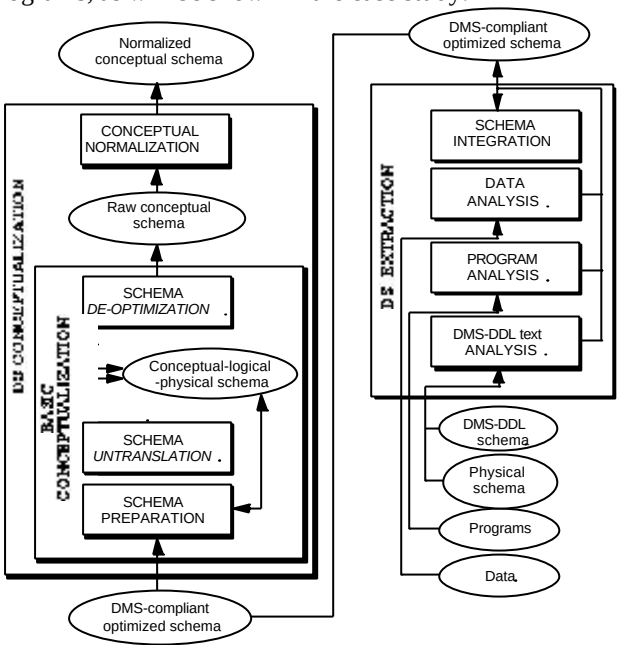


Figure 1: Main components of the generic DBRE methodology. Quite naturally, this reverse methodology is to be read from right to left, and bottom-up.

The main processes of DATA STRUCTURE EXTRACTION are the following :

- DMS-DDL text ANALYSIS. This rather straighforward process consists in analyzing the data structures declaration statements (in the specific DDL) included in the schema scripts and application programs. It produces a first-cut logical schema.
- SCHEMA REFINEMENT. This process is much more complex. Non declarative sources of information are analyzed in order to elicit implicit constructs and constraints.
  - ♦ PROGRAM ANALYSIS. It consists in analyzing the other parts of the application programs, a.o. the procedural sections, in order to detect evidence of additional data structures and integrity constraints. The first-cut schema can therefore be refined through the detection of hidden, non declarative structures.
  - ♦ DATA ANALYSIS. This refinement process examines the contents of the files and databases in order (1) to detect data structures and properties (e.g. to find the unique fields or the functional dependencies in a file), and (2) to test hypotheses (e.g. "could this field be a foreign key to this file ?").
  - ♦ Other sources ANALYSIS.
- SCHEMA INTEGRATION. When more than one information source has been processed, the analyst is provided with several, generally different, extracted (and possibly refined) schemas. Let us mention some common situations : base tables and views (RDBMS), DBD and PSB (IMS), schema and subschemas (CODASYL), file structures from all the application programs (standard files), etc. The final logical schema must include the specifications of all these partial views, through a *schema integration* process.

The end product of this phase is the (hopefully) complete logical schema. This schema is expressed according to the specific model of the DMS, and still includes possible optimized constructs, hence its name : the *DMS-compliant optimized schema*, or *DMS schema* for short.

2.2 The Data Structure Conceptualization Process

This second phase addresses the conceptual interpretation of the DMS schema. It consists for instance in detecting and transforming or discarding non-conceptual structures, redundancies, technical optimization and DMS-dependent constructs. It consists of two sub-processes, namely *Basic conceptualization* and *Conceptual normalization*.

- BASIC CONCEPTUALIZATION. The main objective of this process is to extract all the relevant semantic concepts underlying the logical schema. Two different problems, requiring different reasonings and methods, have to be solved: *schema untranslation* and *schema de-optimization*. However, before tackling these problems, we often have to *prepare* the schema by cleaning it.
  - ♦ *SCHEMA PREPARATION*. The schema still includes some constructs, such as files and access keys, which may have been useful in the Data Structure Extraction phase, but which can now be discarded. In addition, translating names to make them more meaningful (e.g. substitute the file name for the record name), and restructuring some parts of the schema can prove useful before trying to interpret them.
  - ♦ *SCHEMA UNTRANSLATION*. The logical schema is the technical translation of conceptual constructs. Through this process, the analyst identifies the traces of such translations, and replaces them by their original conceptual construct. Though each data model can be assigned its own set of translating (and therefore of untranslating) rules, two facts are worth mentioning. First, the data models can share an important subset of translating rules (e.g. COBOL files and SQL structures). Secondly, translation rules considered as specific to a data model are often used in other data models (e.g. foreign keys in IMS and CODASYL databases). Hence the importance of generic approaches and tools.
  - ♦ *SCHEMA DE-OPTIMIZATION*. The logical schema is searched for traces of constructs designed for optimization purposes. Three main families of optimization techniques should be considered : denormalization, structural redundancy and restructuring.
- CONCEPTUAL NORMALIZATION. This process restructures the basic conceptual schema in order to give it the desired qualities one expects from any final conceptual schema, e.g. expressiveness, simplicity, minimality, readability, genericity, extensibility. For instance, some

entity types are replaced by relationship types or by attributes, is-a relations are made explicit, names are standardized, etc.

3. A small case study

This section is dedicated to a typical application of database reverse engineering, namely *database conversion*. The source application is a small COBOL program which uses three files. The objective of the exercise is to produce a relational schema which translates the semantics of these source files. This can be done in two steps: first we elaborate a possible conceptual schema of the three files, then we translate this schema into relational structures. Due to space limit, we will simplify the relational translation, that is now considered as standard. For the same reason, we will develop the first main process (Data Structure Extraction) of the reverse engineering step in more detail than the second one (Data Structure Conceptualization) which has been more extensively treated in the literature.

4. The DATA STRUCTURE EXTRACTION process

The only source of information that will be considered is the COBOL program listed in appendix. The case has been solved with the help of the DB-MAIN CASE tool [20].

4.1 DMS-DDL text ANALYSIS

This operation is carried out by a *COBOL parser* which extracts the file and record descriptions, and expresses them as a *first cut schema* in the repository. The resulting schema is given in figure 2.

Each record type is represented by a *physical entity type*, and each field by a *physical attribute*. Record keys are represented by *identifiers* when they specify uniqueness constraints and by *access keys* when they specify indexes. Files are represented as *physical entity collections* (cylinder icons).

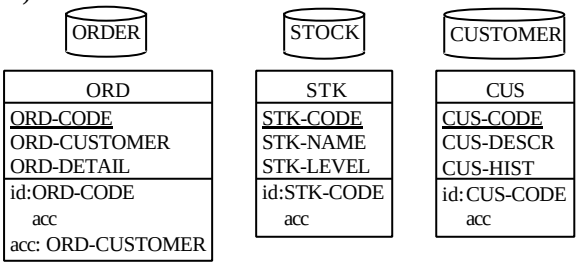


Figure 2 : The first cut file and record schema as produced by the COBOL parser of DB-MAIN.

The textual view shows the type and length of the fields :

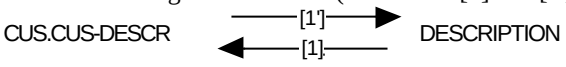
```
Schema CUST-ORD/1stCut-Logical

 collection CUSTOMER  ORD
  CUS                    in ORDER
 collection ORDER       ORD-CODE numeric(5)
  ORD                   ORD-CUSTOMER char(12)
 collection STOCK       ORD-DETAIL char(200)
  STK                   id:ORD-CODE
                        access key
 CUS                    access key:ORD-CUSTOMER
   in CUSTOMER
  CUS-CODE char(12)    STK
  CUS-DESCR char(80)      in STOCK
  CUS-HIST char(1000)    STK-CODE num(5)
  id:CUS-CODE            STK-NAME char(100)
     access key          STK-LEVEL num(5)
                        id:STK-CODE
                           access key
```

4.2 PROGRAM ANALYSIS

This schema will be refined through an in-depth inspection of the ways in which the program uses and manages the data. Through this process, we will detect additional structures and constraints which were not explicitly declared in the file/record declaration sections, but which were expressed in the procedural code and in local variables. We will consider four important processes, namely *Field refinement*, *Foreign key elicitation*, *Attribute identifier elicitation*, and *Field cardinality refinement*.

4.2.1 Field refinement

First observation : some fields are unusually long (CUS-DESCR, CUS-HIST, ORD-DETAIL, STK-NAME). Could they be further refined ? Let us consider CUS-DESCR first. We build the *variable dependency graph*, which summarizes the dataflow concerning CUS-DESCR (statements [1] and [1']) :



This graph clearly suggests that CUS-DESCR and DESCRIPTION should have the same structure, i.e. :

```
01 DESCRIPTION.
  02 NAME PIC X(20).
  02 ADDRESS PIC X(40).
  02 FUNCTION PIC X(10).
  02 REC-DATE PIC X(10).
```

This structure is associated with the field CUS-DESCR in the logical schema. We proceed in the same way for CUS-HIST, ORD-DETAIL and STK-NAME. The analysis shows that only the first two need be refined, as illustrated in figure 3.

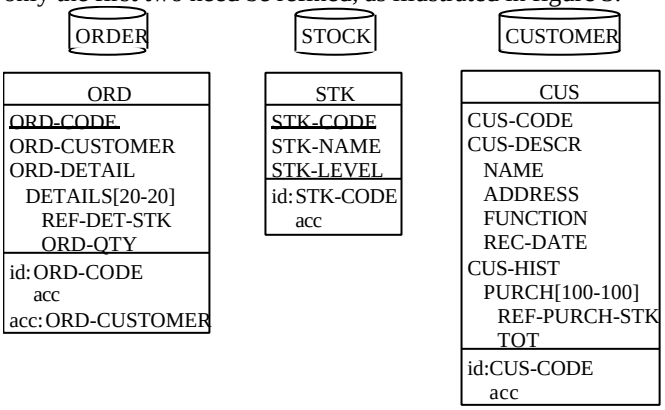


Figure 3. Result of the field refinement process.

4.2.2 Foreign key elicitation

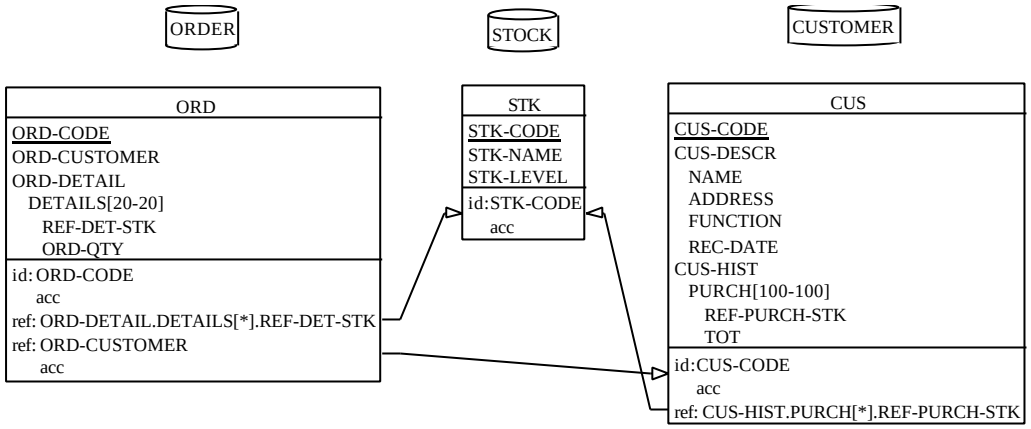


Figure 4. The foreign keys are made explicit.

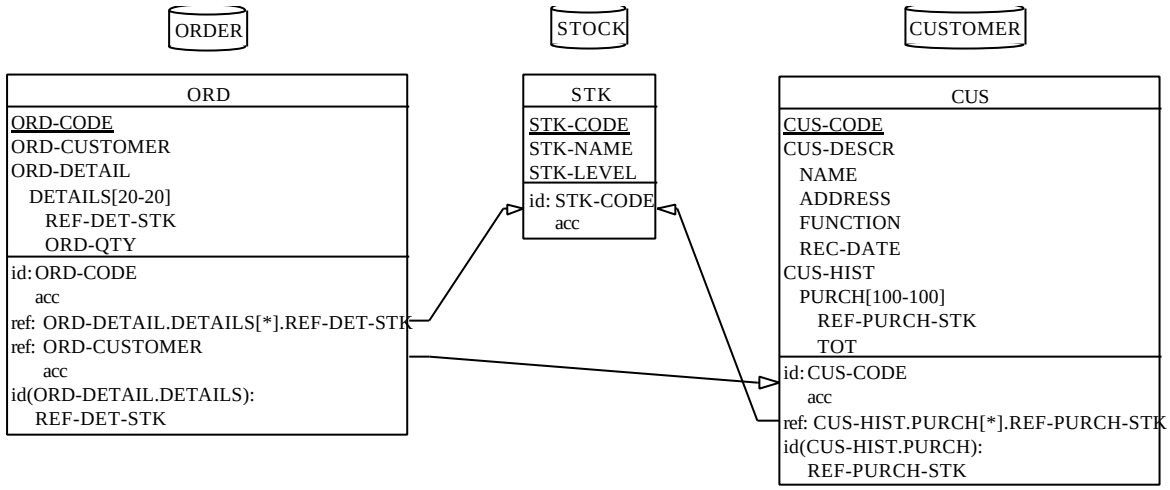


Figure 5. Identifiers of the multivalued fields.

There should exist reference links between these record types. Let us examine the field ORD-CUSTOMER for instance. We observe that :

- its name includes the name of another file (CUSTOMER);
- it has the same type and length as the record key of CUSTOMER;
- it is supported by an access key (= index);
- its dependency graph shows that it receives its value from the record key of CUSTOMER

CUS.CUS-CODE ——[4]——▶ ORD.ORD-CUSTOMER

- its usage pattern shows that, before moving it to the ORD record to be stored, the program verifies that ORD-CUSTOMER value identifies a stored CUS record :

```
NEW-ORD.
  ...
  MOVE 1 TO END-FILE.
  PERFORM READ-CUS-CODE UNTIL END-FILE = 0.
  ...
  MOVE CUS-CODE TO ORD-CUSTOMER.
  ...
  WRITE ORD INVALID KEY DISPLAY "ERROR".

READ-CUS-CODE.
  ACCEPT CUS-CODE.
  MOVE 0 TO END-FILE.
  READ CUSTOMER INVALID KEY
    DISPLAY "NO SUCH CUSTOMER"
    MOVE 1 TO END-FILE
  END-READ.
```

These are five positive evidences contributing to asserting that ORD-CUSTOMER is a foreign key. Data analysis could have added additional information. We decide to confirm the hypothesis. In the same way, we conclude that :

- ORD-DETAIL.DETAILS.REF-DET-STK is a multivalued foreign key to STOCK. Here the REF part of the name suggests a referential function of the field.
- CUS-HIST.PURCH.REF-PURCH-STK is a multivalued foreign key to STOCK

Now the schema looks like that in figure 4.

4.2.3 Elicitation of identifiers of multivalued fields

Compound multivalued fields in COBOL records often have an implicit identifier that makes their values unique. The

schema includes two candidate multivalued fields : ORD-DETAIL.DETAILS and CUS-HIST.PURCH.
By examining the way in which these fields are searched and managed, we isolate the following pattern (or *program slice* [38]):

```
  SET IND-DET TO 1.
UPDATE-ORD-DETAIL.
  MOVE 1 TO NEXT-DET.
  ...
  PERFORM UNTIL
       REF-DET-STK(NEXT-DET) = PROD-CODE
       OR IND-DET = NEXT-DET
    ADD 1 TO NEXT-DET
  END-PERFORM.
  IF IND-DET = NEXT-DET
    MOVE PROD-CODE TO REF-DET-STK(IND-DET)
    PERFORM UPDATE-CUS-HIST
    SET IND-DET UP BY 1
  ELSE
    DISPLAY "ERROR : ALREADY ORDERED".
```

It derives from this code section that the LIST-DETAIL.DETAILS array will never include twice the same REF-DET-STK value. Therefore, this field is the local identifier of this array, and of ORD-DETAIL.DETAILS as well. Through the same reasoning, we are suggested that REF-PURCH-STK is the identifier of LIST-PURCHASE.PURCH array. These findings are shown in figure 5.

4.2.4 Refinement of the cardinality of multivalued attributes

The multivalued fields have been given cardinality constraints derived from the occurs clause. The latter gives the maximum cardinality, but says nothing about the minimum cardinality.

Storing a new CUS record generally implies initializing each field, including CUS-HIST.PURCH. This is done through the INIT-HIST paragraph [13], in which the REF-DET-STK is set to 0. Furthermore, the scanning of this list stops when 0 is encountered [7]. Conclusion : there are from 0 to 100 elements in this list. A similar analysis leads to refine the cardinality of ORD-DETAIL.DETAILS). Hence the final schema of figure 6.

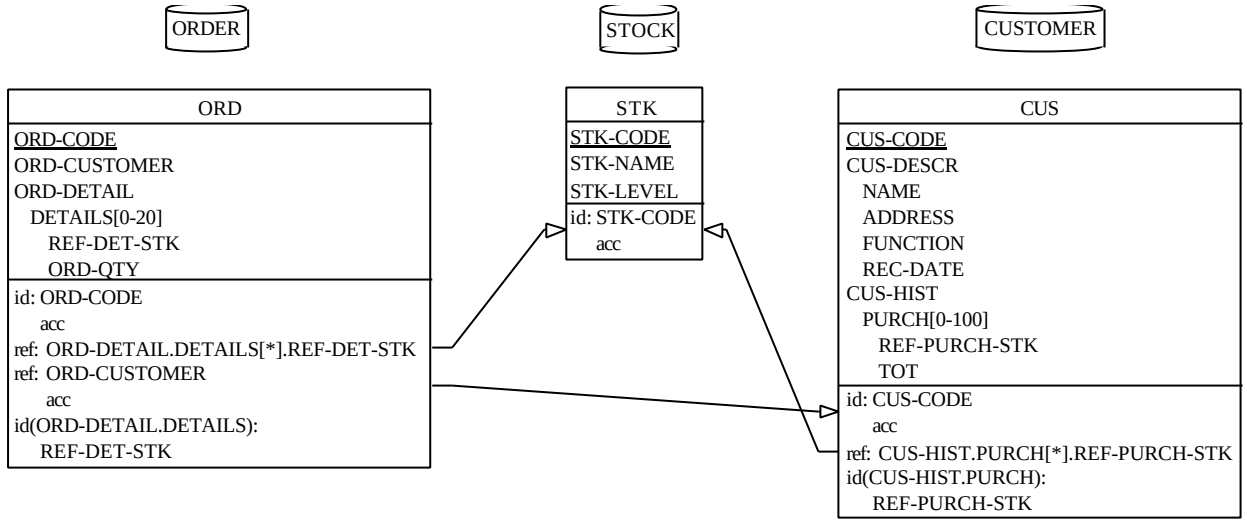


Figure 6. The exact cardinality (repeating factors) of attributes ORD-DETAIL.DETAILS and CUS.HIST.

5. The DATA STRUCTURE CONCEPTUALIZATION process

5.1 Schema preparation

The schema obtained so far describes the complete COBOL data structures. Before trying to recover the conceptual schema, we will clean this schema a little bit (figure 7).

5.1.1 Name processing

- The files have more meaningful names than their record types : we give the latter the name of their files.
- The fields of each record type are prefixed with a common short-name identifying their the record type, and which bears no semantics. We trim them out.
- Compound fields CUS.CUS-HIST and ORD.ORD-DETAIL have one component only, and can be disaggregates without structural or semantic loss.

5.1.2 Physical cleaning

The physical constructs, namely files and access keys, are no longer useful, and are removed.

5.2 Schema de-optimization

The attributes CUSTOMER.PURCH and ORDER.DETAILS have a complex structure : they are compound, they are multivalued, they have a local identifier and they include a foreign key. They obviously suggest a typical COBOL trick to represent *dependent entity types*. This very efficient technique

consists in representing such entity types by embedded multivalued fields. We transform the latter into entity types. The schema appears as in figure 8.

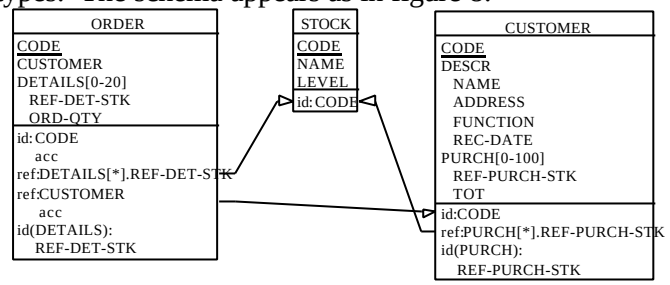


Figure 7 : Reducing the names and removing physical constructs from the logical schema.

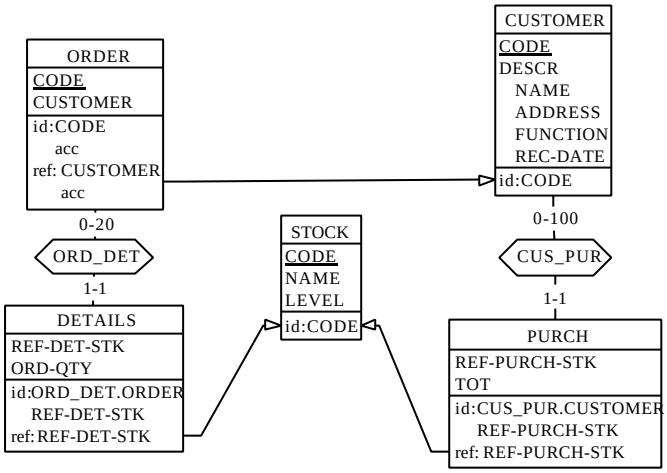


Figure 8 : Making dependent entity types explicit.

5.3 Schema untranslation

The foreign keys are the most obvious traces of the ER/COBOL translation. We express them as *one-to-many* relationship types (Fig. 9).

5.4 Conceptual normalization

We will only mention three elementary problems to illustrate the process.

5.4.1 Maximal cardinalities

Are the maximum cardinalities 100 and 20 of real semantic value, or do they simply describe obsolete technical limits from the legacy system? Considering their origin these constraints are dropped, and replaced with "N".

5.4.2 Rel-type entity types

PURCH and DETAILS could be perceived as mere relationships, and are transformed accordingly.

5.4.3 Names

Now, the semantics of the data structures have been elicited, and better names can be given to some of them. Since *customers pass orders*, we rename **ORD_CUS** as **passes**. In addition, **PURCH** is given the full-name **Purchase** (figure 10).

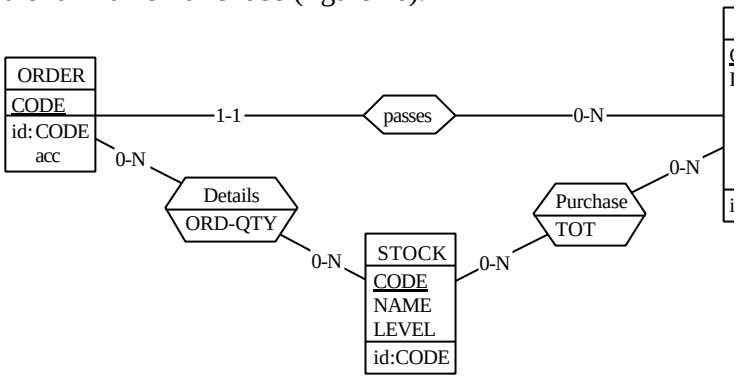


Figure 10. A normalized variant of the conceptual schema.

6. Relational database design

To complete the exercice, let us develop a new relational database schema from this conceptual specification. The process is fairly standard, and includes the *Logical design* and the *Physical design* phases. Due to the size of the problem, they are treated in a rather symbolic way.

6.1 Logical design

Transforming this schema into relational structures is fairly easy : we disaggregate the compound attribute DESC, we express the complex relationship types `passes` and `Purchase` into entity types, then we translate the one-to-many relationship types into foreign keys. The resulting schema comprises flat entity types, identifiers and foreign keys. It can be considered as a logical relational schema (Fig. 11).

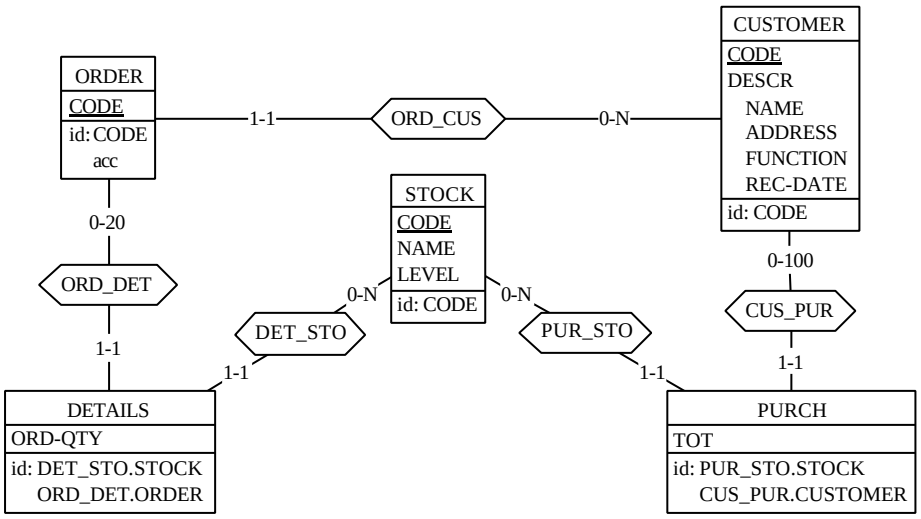


Figure 9. The first cut conceptual schema.

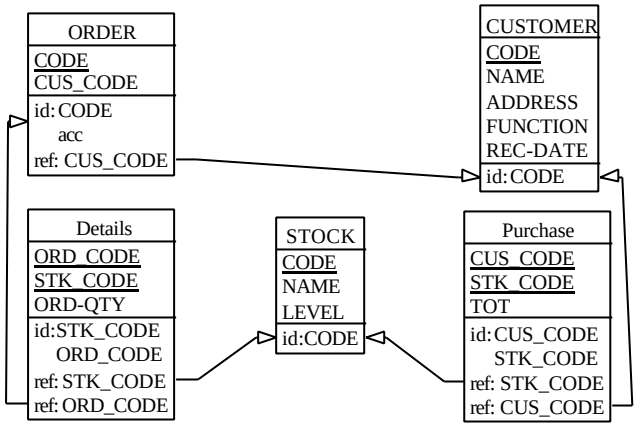


Figure 11. The relational logical schema.

6.2 Physical design

We reduce this phase to processing the names according to SQL standard (e.g. all the names in uppercase, no "-", no reserved words, etc) and defining the physical spaces and the access keys (indexes) which support identifiers and foreign keys (Fig. 12). As a symbolic touch of optimization, we remove all the indexes which are a prefix of another index (i.e. no index on PURCHASE.CUS_CODE and on DETAILS.ORD_CODE).

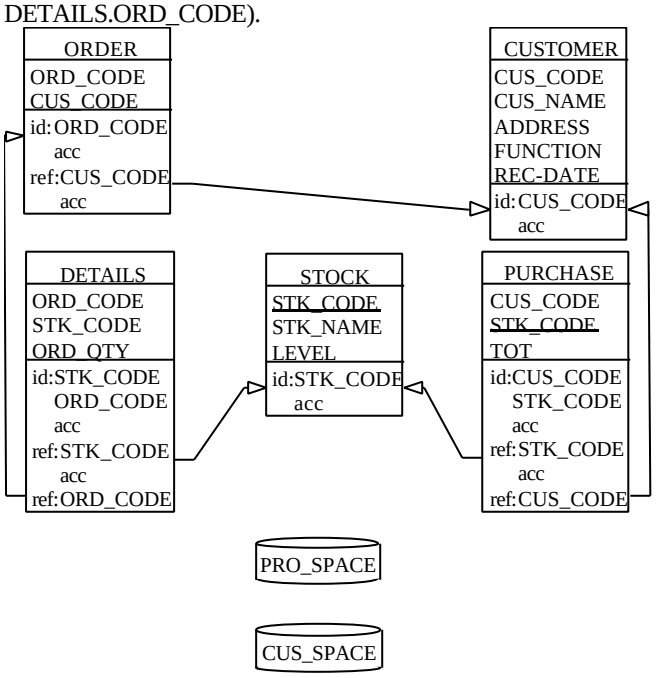


Figure 12 :The relational physical schema

6.3 Code generation

The production of the SQL code is straightforward:

```
create database CUS-ORD;

create dbspace PRO_SPACE;
create dbspace CUS_SPACE;
```

```
create table CUSTOMER (
  CUS_CODE char(12) not null ,
  CUS_NAME char(20) not null ,
  ADDRESS char(40) not null ,
  FUNCTION char(10) not null ,
  REC-DATE char(10) not null ,
  primary key (CUS_CODE))
     in CUS_SPACE;
create table DETAILS (
  ORD_CODE numeric(10) not null,
  STK_CODE numeric(5) not null,
  ORD-QTY numeric(5) not null,
  primary key (STK_CODE,ORD_CODE))
    in CUS_SPACE;
create table ORDER (
  ORD_CODE numeric(10) not null,
  CUS_CODE char(12) not null ,
  primary key (ORD_CODE))
   in CUS_SPACE;
create table PURCHASE (
  CUS_CODE char(12) not null ,
  STK_CODE numeric(5) not null,
  TOT numeric(5) not null ,
 primary key(STK_CODE,CUS_CODE))
   in CUS_SPACE;
create table STOCK (
  STK_CODE numeric(5) not null ,
  STK_NAME char(100) not null ,
  LEVEL numeric(5) not null ,
  primary key (STK_CODE))
   in PRO_SPACE;
alter table DETAILS add
  constraint FKDET_STO
  foreign key (STK_CODE)
  references STOCK;
alter table DETAILS add
  constraint FKDET_ORD
  foreign key (ORD_CODE)
  references ORDER;
alter table ORDER add
  constraint FKO_C
  foreign key (CUS_CODE)
  references CUSTOMER;
alter table PURCH add
  constraint FKPUR_STO
  foreign key (STK_CODE)
  references STOCK;
alter table PURCH add
  constraint FKPUR_CUS
  foreign key (CUS_CODE)
  references CUSTOMER;
create unique index CUS-CODE
  on CUSTOMER (CUS_CODE);
create unique index IDDETAILS
  on DETAILS (STK_CODE,ORD_CODE);
create index FKDET_ORD
  on DETAILS (ORD_CODE);
```

```
create unique index ORD-CODE
  on ORDER (ORD_CODE);
create index FKO_C
  on ORDER (CUS_CODE);
create unique index IDPURCH
  on PURCHASE (STK_CODE,CUS_CODE);
create index FKPUR_CUS
  on PURCHASE (CUS_CODE);
create unique index STK-CODE
  on STOCK (STK_CODE);
```

The project window of DB-MAIN shows the engineering products that have been used and produced in this conversion (figure 13).

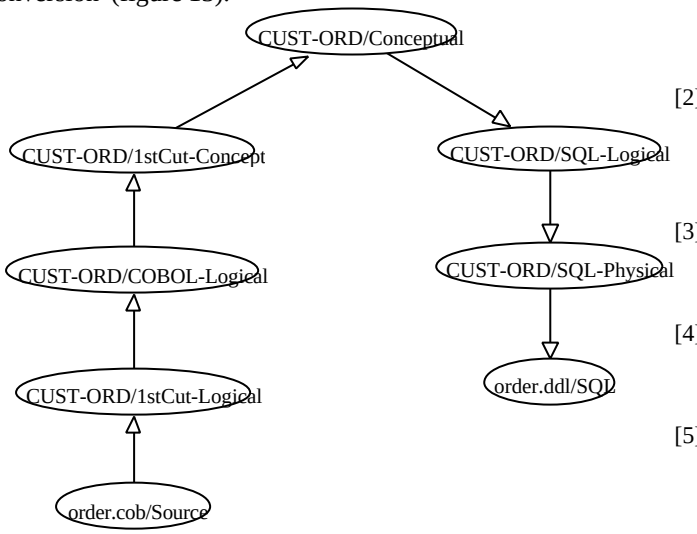


Figure 13 : The products (program texts and schemas) of the conversion project, and their dependency relations.

7. Conclusions

This case study was a toy application only. A typical, medium size, real-world project could included, for instance:

- 250 program units, with a size ranging from 100 to 20,000 lines;
- 1,000 files, 50 of which being relevant;
- these 50 files include 100 record types and 3,000 fields;
- more than one DMS (for instance COBOL files + a CODASYL database);
- more sophisticated constructs such as : alternate field structures, overlapping foreign keys, transitive and embedded foreign keys, conditional identifier and foreign keys, field redundencies, explicit NEXT pointers [2,29];
- conflicting structures and views
- usage of data dictionaries, CASE tools, generators.

It is important to notice that we have translated the data structures only. We have to mention two additional problems: converting the COBOL data into relational data, and converting the COBOL programs into COBOL/SQL programs. But this is another, more complex, story.

At the present time, practically no commercial CASE tool can cope with even such a simple application. Obviously, much work must be devoted to the fundamentals of Database reverse engineering, and to CASE tools that can be used not only by their developers, but also by analysts and programmers. The DB-MAIN project aims, among others, at developing methodologies and CASE tools to help solve these problems. Its reverse engineering functions have been used in several industrial projects in redocumentation, conversion and migration projects. An *Education version* is available for non-profit organizations.

8. References

[1] Andersson, M. 1994. Extracting an Entity Relationship Schema from a Relational Database through Reverse Engineering, in *Proc. of the 13th Int. Conf. on ER Approach*, Manchester, Springer-Verlag

[2] Blaha, M.R., Premerlani, W., J. 1995. Observed Idiosyncracies of Relational Database designs, in *Proc. of the 2nd IEEE Working Conf. on Reverse Engineering*, Toronto, July 1995, IEEE Computer Society Press

[3] Bolois, G., Robillard, P. 1994. Transformations in Reengineering Techniques, in *Proc. of the 4th Reengineering Forum "Reengineering in Practice"*, Victoria, Canada

[4] Casanova, M., Amarel de Sa, J. 1983. Designing Entity Relationship Schemas for Conventional Information Systems, in *Proc. of ERA*, pp. 265-278

[5] Casanova, M., A., Amaral De Sa. 1984. Mapping uninterpreted Schemes into Entity-Relationship diagrams : two applications to conceptual schema design, in *IBM J. Res. & Develop*., Vol. 28, No 1

[6] Chiang, R., H., Barron, T., M., Storey, V., C. 1994. Reverse Engineering of Relational Databases : Extraction of an EER model from a relational database, *Journ. of Data and Knowledge Engineering*, Vol. 12, No. 2 (March 94), pp107-142

[7] Davis, K., H., Arora, A., K. 1985. A Methodology for Translating a Conventional File System into an Entity-Relationship Model, in *Proc. of ERA*, IEEE/North-Holland

[8] Davis, K., H., Arora, A., K. 1988. Converting a Relational Database model to an Entity Relationship Model, in *Proc. of ERA : a Bridge to the User*, North-Holland

[9] Edwards, H., M., Munro, M. 1995. Deriving a Logical Model for a System Using Recast Method, in *Proc. of the 2nd IEEE WC on Reverse Engineering*, Toronto, IEEE Computer Society Press

[10] Fong, J., Ho, M. 1994. Knowledge-based Approach for Abstracting Hierarchical and Network Schema Semantics, in *Proc. of the 12th Int. Conf. on ER Approach*, Arlington-Dallas, Springer-Verlag

[11] Fonkam, M., M., Gray, W., A. 1992. An approach to Eliciting the Semantics of Relational Databases, in *Proc. of 4th Int. Conf. on Advance Information Systems Engineering* - CAiSE'92, pp. 463-480, May, LNCS, Springer-Verlag

[12] Hainaut, J-L., Cadelli, M., Decuyper, B., Marchand, O. 1992. Database CASE Tool Architecture : Principles for Flexible Design Strategies, in *Proc. of the 4th Int. Conf. on Advanced Information System Engineering* (CAiSE-92), Manchester, May 1992, Springer-Verlag, LNCS

[13] Hainaut, J-L., Chandelon M., Tonneau C., Joris M. 1993a. Contribution to a Theory of Database Reverse Engineering, in *Proc. of the IEEE Working Conf. on Reverse Engineering*, Baltimore, May 1993, IEEE Computer Society Press

[14] Hainaut, J-L, Chandelon M., Tonneau C., Joris M. 1993b. Transformational techniques for database reverse engineering, in *Proc. of the 12th Int. Conf. on ER Approach, Arlington-Dallas*, E/R Institute and Springer-Verlag, LNCS

[15] Hainaut, J-L, Englebert, V., Henrard, J., Hick J-M., Roland, D. 1994. Evolution of database Applications : the DB-MAIN Approach, in *Proc. of the 13th Int. Conf. on ER Approach*, Manchester, Springer-Verlag

[16] Hainaut, J-L. 1995. *Database Reverse Engineering - Problems, Methods and Tools*,Tutorial notes, CAiSE•95, Jyväskylä, Finland, May. 1995 (available at jlh@info.fundp.ac.be)

[17] Hainaut, J-L. 1995b. *Transformation-based Database Engineering*,Tutorial notes, VLDB-95, Zürich, Switzerland, , Sept. 1995 (available at jlh@info.fundp.ac.be)

[18] Hainaut, J-L. 1996. Specification Preservation in Schema transformations - Application to Semantics and Statistics, *Data & Knowledge Engineering*, Elsevier (to appear)

[19] Hainaut, J-L, Roland, D., Hick J-M., Henrard, J., Englebert, V. 1996b. Database design recovery, *in Proc. of CAiSE·96*, Springer-Verlag, 1996

[20] Hainaut, J-L, Roland, D., Hick J-M., Henrard, J., Englebert, V. 1996c. Database Reverse Engineering : from Requirements to CARE tools, *Journal of Automated Software Engineering*, Vol. 3, No. 1 (1996).

[21] Hall, P., A., V. (Ed.) 1992. *Software Reuse and Reverse Engineering in Practice*, Chapman&Hall

[22] IEEE, 1990. *Special issue on Reverse Engineering*, IEEE Software, January, 1990

[23] Johannesson, P., Kalman, K. 1990. A Method for Translating Relational Schemas into Conceptual Schemas, in *Proc. of the 8th ERA*, Toronto, North-Holland,

[24] Joris, M., Van Hoe, R., Hainaut, J-L., Chandelon M., Tonneau C., Bodart F. et al. 1992. *PHENIX : methods and tools for database reverse engineering*, in Proc. 5th Int. Conf. on Software Engineering and Applications, Toulouse, December 1992, EC2 Publish.

[25] Markowitz, K., M., Makowsky, J., A. 1990. Identifying Extended Entity-Relationship Object Structures in Relational Schemas, *IEEE Trans. on Software Engineering*, Vol. 16, No. 8

[26] Navathe, S., B., Awong, A. 1988. Abstracting Relational and Hierarchical Data with a Semantic Data Model, in *Proc. of ERA : a Bridge to the User*, North-Holland

[27] Nilsson,E., G. 1985. The Translation of COBOL Data Structure to an Entity-Rel-type Conceptual Schema, in *Proc. of ERA*, IEEE/North-Holland,

[28] Petit, J-M., Kouloumdjian, J., Bouliaut, J-F., Toumani, F. 1994. Using Queries to Improve Database Reverse Engineering, in *Proc. of the 13th Int. Conf. on ER Approach*, Manchester, Springer-Verlag

[29] Premerlani, W., J., Blaha, M.R. 1993. An Approach for Reverse Engineering of Relational Databases, in *Proc. of the IEEE Working Conf. on Reverse Engineering*, IEEE Computer Society Press

[30] Rock-Evans, R. 1990. *Reverse Engineering : Markets, Methods and Tools*, OVUM report

[31] Rosenthal, A., Reiner, D. 1994. Tools and Transformations - Rigourous and Otherwise - for Practical Database Design, *ACM TODS*, Vol. 19, No. 2

[32] Sabanis, N., Stevenson, N. 1992. Tools and Techniques for Data Remodelling Cobol Applications, in *Proc. 5th Int. Conf. on Software Engineering and Applications*, Toulouse, 7-11 December, pp. 517-529, EC2 Publish.

[33] Selfridge, P., G., Waters, R., C., Chikofsky, E., J. 1993. Challenges to the Field of Reverse Engineering, in *Proc. of the 1st WC on Reverse Engineering*, pp.144-150, IEEE Computer Society Press

[34] Shoval, P., Shreiber, N. 1993. Database Reverse Engineering : from Relational to the Binary Relationship Model, *Data and Knowledge Engineering*, Vol. 10, No. 10

[35] Signore, O, Loffredo, M., Gregori, M., Cima, M. 1994. Reconstruction of ER Schema from Database Applications: a Cognitive Approach, in *Proc. of the 13th Int. Conf. on ER Approach*, Manchester, Springer-Verlag

[36] Springsteel, F., N., Kou, C. 1990. Reverse Data Engineering of E-R designed Relational schemas, in *Proc. of Databases, Parallel Architectures and their Applications*

[37] Vermeer, M., Apers, P. 1995. Reverse Engineering of Relational Databases, in *Proc. of the 14th Int. Conf. on ER/OO Modelling (ERA*)

[38] Weiser, M. 1984. Program Slicing, *IEEE TSE*, Vol. 10, pp 352-357

[39] Wills, L., Newcomb, P., Chikofsky, E., (Eds) 1995. *Proc. of the 2nd IEEE Working Conf. on Reverse Engineering*, Toronto, July 1995, IEEE Computer Society Press

[40] Winans, J., Davis, K., H. 1990. Software Reverse Engineering from a Currently Existing IMS Database to an Entity-Relationship Model, in *Proc. of ERA : the Core of Conceptual Modelling*, pp. 345-360, October, North-Holland

Appendix. The COBOL source text

```
IDENTIFICATION DIVISION.
PROGRAM-ID. C-ORD.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
  SELECT CUSTOMER ASSIGN
         TO "CUSTOMER.DAT"
    ORGANIZATION IS INDEXED
    ACCESS MODE IS DYNAMIC
    RECORD KEY IS CUS-CODE.
  SELECT ORDER ASSIGN TO "ORDER.DAT"
    ORGANIZATION IS INDEXED
    ACCESS MODE IS DYNAMIC
    RECORD KEY IS ORD-CODE
    ALTERNATE RECORD KEY
          IS ORD-CUSTOMER
    WITH DUPLICATES.
  SELECT STOCK ASSIGN TO "STOCK.DAT"
    ORGANIZATION IS INDEXED
    ACCESS MODE IS DYNAMIC
    RECORD KEY IS STK-CODE.

DATA DIVISION.
FILE SECTION.
FD CUSTOMER.
01 CUS.
  02 CUS-CODE PIC X(12).
  02 CUS-DESCR PIC X(80).
  02 CUS-HIST PIC X(1000).

FD ORDER.
01 ORD.
  02 ORD-CODE PIC 9(10).
  02 ORD-CUSTOMER PIC X(12).
  02 ORD-DETAIL PIC X(200).

FD STOCK.
01 STK.
  02 STK-CODE PIC 9(5).
  02 STK-NAME PIC X(100).
  02 STK-LEVEL PIC 9(5).

WORKING-STORAGE SECTION.
01 DESCRIPTION.
  02 NAME PIC X(20).
  02 ADDRESS PIC X(40).
  02 FUNCTION PIC X(10).
  02 REC-DATE PIC X(10).

01 LIST-PURCHASE.
  02 PURCH OCCURS 100 TIMES
     INDEXED BY IND.
    03 REF-PURCH-STK PIC 9(5).
    03 TOT PIC 9(5).

01 LIST-DETAIL.
  02 DETAILS OCCURS 20 TIMES
     INDEXED BY IND-DET.
    03 REF-DET-STK PIC 9(5).
    03 ORD-QTY PIC 9(5).

01 CHOICE PIC X.
01 END-FILE PIC 9.
01 END-DETAIL PIC 9.
01 EXIST-PROD PIC 9.
01 PROD-CODE PIC 9(5).

01 TOT-COMP PIC 9(5) COMP.
01 QTY PIC 9(5) COMP.
01 NEXT-DET PIC 99.

PROCEDURE DIVISION.
MAIN.
  PERFORM INIT.
  PERFORM PROCESS UNTIL CHOICE = 0.
  PERFORM CLOSING.
  STOP RUN.

INIT.
  OPEN I-O CUSTOMER.
  OPEN I-O ORDER.
  OPEN I-O STOCK.

PROCESS.
  DISPLAY "1 NEW CUSTOMER".
  DISPLAY "2 NEW STOCK".
  DISPLAY "3 NEW ORDER".
  DISPLAY "4 LIST OF CUSTOMERS".
  DISPLAY "5 LIST OF STOCKS".
  DISPLAY "6 LIST OF ORDERS".
  DISPLAY "0 END".
  ACCEPT CHOICE.
  IF CHOICE = 1
    PERFORM NEW-CUS.
  IF CHOICE = 2
    PERFORM NEW-STK.
  IF CHOICE = 3
    PERFORM NEW-ORD.
  IF CHOICE = 4
    PERFORM LIST-CUS.
  IF CHOICE = 5
    PERFORM LIST-STK.
  IF CHOICE = 6
    PERFORM LIST-ORD.

CLOSING.
  CLOSE CUSTOMER.
  CLOSE ORDER.
  CLOSE STOCK.

NEW-CUS.
  DISPLAY "NEW CUSTOMER :".
  DISPLAY "CUSTOMER CODE ?"
    WITH NO ADVANCING.
  ACCEPT CUS-CODE.

  DISPLAY "NAME DU CUSTOMER : "
    WITH NO ADVANCING.
  ACCEPT NAME.
  DISPLAY "ADDRESS OF CUSTOMER : "
```

```
    WITH NO ADVANCING.
  ACCEPT ADDRESS.
  DISPLAY "FUNCTION OF CUSTOMER : "
    WITH NO ADVANCING.
  ACCEPT FUNCTION.
  DISPLAY "DATE : " WITH NO ADVANCING.
  ACCEPT REC-DATE.
  MOVE DESCRIPTION TO CUS-DESCR.  [1]
  PERFORM INIT-HIST.
  WRITE CLI INVALID KEY DISPLAY "ERROR".


LIST-CUS.
  DISPLAY "LISTE DES CUSTOMERS".
  CLOSE CUSTOMER.
  OPEN I-O CUSTOMER.
  MOVE 1 TO END-FILE.
  PERFORM READ-CUS UNTIL END-FILE = 0.

READ-CUS.
  READ CUSTOMER NEXT
    AT END MOVE 0 TO END-FILE
    NOT AT END
      DISPLAY CUS-CODE
      DISPLAY CUS-DESCR
      DISPLAY CUS-HISTORY.

NEW-STK.
  DISPLAY "NEW STOCK".
  DISPLAY "PRODUCT NUMBER : "
    WITH NO ADVANCING.
  ACCEPT STK-CODE.

  DISPLAY "NAME : " WITH NO ADVANCING.
  ACCEPT STK-NAME.

  DISPLAY "LEVEL : " WITH NO ADVANCING.
  ACCEPT STK-LEVEL.

  WRITE STK INVALID KEY DISPLAY "ERREUR ".

LIST-STK.
  DISPLAY "LIST OF STOCKS ".

  CLOSE STOCK.
  OPEN I-O STOCK.

  MOVE 1 TO END-FILE.
  PERFORM READ-STK UNTIL END-FILE = 0.

READ-STK.
  READ STOCK NEXT
    AT END MOVE 0 TO END-FILE
    NOT AT END
      DISPLAY STK-CODE
      DISPLAY STK-NAME
      DISPLAY STK-LEVEL.

NEW-ORD.
  DISPLAY "NEW ORDER".
  DISPLAY "ORDER NUMBER : "
    WITH NO ADVANCING.
  ACCEPT ORD-CODE.
  MOVE 1 TO END-FILE.
  PERFORM READ-CUS-CODE
          UNTIL END-FILE = 0.
  MOVE CUS-DESCR TO DESCRIPTION.  [1']
  DISPLAY NAME.
  MOVE CUS-CODE TO ORD-CUSTOMER.  [4]
  MOVE CUS-HISTORY TO LIST-PURCHASE.

  SET IND-DET TO 1.
  MOVE 1 TO END-FILE.
  PERFORM READ-DETAIL
    UNTIL END-FILE = 0 OR IND-DET = 21.
  MOVE LIST-DETAIL TO ORD-DETAIL. [2]

  WRITE COM INVALID KEY DISPLAY "ERROR".

  MOVE LIST-PURCHASE
    TO CUS-HISTORY.  [3]
  REWRITE CLI
    INVALID KEY DISPLAY "ERROR CUS".
READ-CUS-CODE.
  DISPLAY "CUSTOMER NUMBER : "
    WITH NO ADVANCING.
  ACCEPT CUS-CODE.
  MOVE 0 TO END-FILE.
  READ CUSTOMER INVALID KEY
    DISPLAY "NO SUCH CUSTOMER"
    MOVE 1 TO END-FILE
  END-READ.

READ-DETAIL.
  DISPLAY "PRODUCT CODE (0 = END) : ".
  ACCEPT PROD-CODE.
  IF PROD-CODE = "0"
    MOVE 0
      TO REF-DET-STK(IND-DET)      [12]
    MOVE 0 TO END-FILE
  ELSE
    PERFORM READ-PROD-CODE.

READ-PROD-CODE.
  MOVE 1 TO EXIST-PROD.
  MOVE PROD-CODE TO STK-CODE.     [5]
  READ STOCK INVALID KEY
    MOVE 0 TO EXIST-PROD.
  IF EXIST-PROD = 0
    DISPLAY "NO SUCH PRODUCT"
  ELSE
     PERFORM UPDATE-ORD-DETAIL.

UPDATE-ORD-DETAIL.
  MOVE 1 TO NEXT-DET.
  DISPLAY "QUANTITY ORDERED : "
    WITH NO ADVANCING
  ACCEPT ORD-QTY(IND-DET).
  PERFORM UNTIL
    REF-DET-STK(NEXT-DET)
      = PROD-CODE    [9]
    OR IND-DET = NEXT-DET
    ADD 1 TO NEXT-DET
  END-PERFORM.
```

```
  IF IND-DET = NEXT-DET    [10]
    MOVE PROD-CODE
    TO REF-DET-STK(IND-DET) [6]
    PERFORM UPDATE-CUS-HISTO
    SET IND-DET UP BY 1
  ELSE
    DISPLAY "ERROR : ALREADY ORDERED".

UPDATE-CUS-HISTO.
  SET IND TO 1.
  PERFORM UNTIL
    REF-PURCH-STK(IND) = PROD-CODE
    OR REF-PURCH-STK(IND) = 0
    OR IND = 101     [7]
      SET IND UP BY 1
  END-PERFORM.

  IF IND = 101
    DISPLAY "ERR : HISTORY OVERFLOW"
    EXIT.
  IF REF-PURCH-STK(IND)
     = PROD-CODE     [11]
    ADD ORD-QTY(IND-DET) TO TOT(IND)
  ELSE
    MOVE PROD-CODE
    TO REF-PURCH-STK(IND)   [8]
    MOVE ORD-QTY(IND-DET) TO TOT(IND).

LIST-ORD.
  DISPLAY "LIST OF ORDERS ".
  CLOSE ORDER.
  OPEN I-O ORDER.
  MOVE 1 TO END-FILE.
  PERFORM READ-ORD UNTIL END-FILE = 0.

READ-ORD.
  READ ORDER NEXT
    AT END MOVE 0 TO END-FILE
    NOT AT END
      DISPLAY "ORD-CODE "
        WITH NO ADVANCING
      DISPLAY ORD-CODE
      DISPLAY "ORD-CUSTOMER "
        WITH NO ADVANCING
      DISPLAY ORD-CUSTOMER
      DISPLAY "ORD-DETAIL "
      MOVE ORD-DETAIL TO LIST-DETAIL
      SET IND-DET TO 1
      MOVE 1 TO END-DETAIL
      PERFORM DISPLAY-DETAIL.

INIT-HIST.          [13]
  SET IND TO 1.
  PERFORM UNTIL IND = 100
    MOVE 0 TO REF-PURCH-STK(IND)
    MOVE 0 TO TOT(IND)
    SET IND UP BY 1
  END-PERFORM.
  MOVE LIST-PURCHASE TO CUS-HISTORY.

DISPLAY-DETAIL.
  IF IND-DET = 21
    MOVE 0 TO END-DETAIL
    EXIT.
  IF REF-DET-STK(IND-DET) = 0
    MOVE 0 TO END-DETAIL
  ELSE
    DISPLAY REF-DET-STK(IND-DET)
    DISPLAY ORD-QTY(IND-DET)
    SET IND-DET UP BY 1.
```