http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 5 (2): 329-331, 2006
ISSN 1812-5638
© 2006 Asian Network for Scientific Information

Normalizer: A Case Tool to Normalize Relational Database Schemas

Nabil Arman
Palestine Polytechnic University, Hebron, Palestine

Abstract: Relational Database Schemas represent the database schema as a collection of relation schemas. Sometimes, these relation schemas are poorly designed and need to be decomposed in an attempt to choose "good" relation schemas. In this study, we used the theory of relational database normalization to develop a case tool, called Normalizer, to automate the process of relational database normalization.

Key words: Case Tools, normalization, relational database design

INTRODUCTION

Normalization in relational databases is an important step in database design. The process of doing that manually, as we see in these days, makes it difficult and takes so much time, in addition to that, the human may make mistakes in doing normalization.

Normalizer is designed as a case tool that helps the database designers to perform the relational database schemas normalization quickly and accurately. This saves the time and effort of database designers and thus frees them to focus on other aspect of the database design process. The main objectives of Normalizer are:

- To perform normalization automatically and accurately.
- To reduce the time needed to perform the process of normalization.
- To avoid human error in the normalization process.

The single most important concept in relational schema design is that of a functional dependency. A functional dependency is a constraint between two sets of attributes from the database. Suppose that our relational database schema has N attributes $A_1, A_2, \ldots, A_n$. Let us think of the whole database as being described by a single universal relation schema $R = \{A_1, A_2, \ldots, A_n\}$. We do not imply that we will actually store the database as a single universal table; but we use this concept only in presenting the formal theory of data dependencies. A functional Dependency, denoted by $X \rightarrow Y$, between two sets of attributes X and Y that are subsets of R specifies a constraint on the possible tuples that can form a relation state r of R. The constraint is that, for any two tuples t_1 and t_2 in r that have $t_1[X] = t_2[X]$, we must also have $t_1[Y] = t_2[Y]$. This means that the values of the Y component of a tuple in r depend on, or are determined by, the values of the X component, or alternatively, the values of the X component of a tuple uniquely (or functionally) determine the values of the Y component. The abbreviation for functional dependency is FD or f.d. The set of attributes X is called the left-hand side of the FD and Y is called the right-hand side. Thus X functionally determines Y in a relation schema R if and only if, whenever two tuples of r(R) agree on their X-value, they must necessarily agree on their Y-value[1-3].

NORMALIZER DEVELOPMENT

Normalizer depends on two major algorithms to perform the normalization to the Second Normal Form (2NF) and Third Normal Form (3NF). Since we assume that we are dealing with relations, the definition of relation guarantees that the relation is in 1NF.

The formal definition of 2NF states that a relation schema R is in 2NF if every non-prime attribute is fully functionally dependent on every key in R. Therefore, the algorithm first determines the non-prime attributes and then tests for partial functional dependencies. Non-prime attributes are those attributes that are not part of the Candidate Keys (CK) attributes.
The algorithm is presented below:

```
Procedure Second_Normal_Form_Normalization(
                    R: Relation Schema,
                    SCK: Set of Candidate Keys,
                    SFD: Set of Functional Dependencies)
{
    determine the set of non-prime attributes (SNPA)
    for each attribute A in SNPA do
        {
            for each FD in SFD do
                {
                  if A is a right-hand side of FD
                  && the left-hand side is not one of the
                  candidate keys
                  && the left hand side is a proper subset of a
                  candidate key then
                  {
```

```
        Split R into two relations R1 and R2 such that R1 contains the
        attributes of FD and attributes that are dependent on A and R2
        contains the remaining attributes and the left-hand side of FD
        CallSecond_Normal_Form_Normalization(R2,SCK,SFD-FD)
      } // end if
    } // end inner for
  } // end outer for
} // end procedure
```

The formal definition of 3NF states that a relation schema R is in 3NF if whenever a functional dependency X→A holds in R, then either X is a candidate key or A is a prime attribute. Therefore, the algorithm needs to determine the set of prime attributes and then tests the functional dependencies to determine the violations. Although Third_Normal_Form_Normalization algorithm can be used to perform the normalization up to the 3NF directly, without having to go through the 2NF normalization, we have chosen to develop an algorithm to perform 2NF normalization since some database designer prefer to stop at 2NF (for performance reasons).
The Third_Normal_Form_Normalization algorithm is presented below:

```
Procedure Third_Normal_Form_Normalization(
            R: Relation Schema,
            SCK: Set of Candidate Keys,
            SFD: Set of Functional Dependencies)
{
      determine the set of prime attributes (SPA)
      for each FD in SFD do
          {
           if the right-hand side of FD is not in SPA
          && the left-hand side is not one of the candidate keys
          then
           {
              Split R into two relations R1 and R2 such that
              R1 contains the attributes of FD and R2 contains
              the remaining attributes and the left-hand side of FD
              Call Third_Normal_Form_
              Normalization(R2,SCK,SFD-FD)
           } // end if
          } // end for
} // end procedure
```

The two algorithms are recursive in nature, since relation splitting or decomposition should continue until there is no violations of the 2NF and 3NF.

NORMALIZER DEMO

Normalizer is a Case Tool that has a GUI interface that is very simple and easy to use. To use Normalizer, and after launching the application:

- The user should first enter the relation name in the appropriate text box.
- The use should enter the relation attributes one by one, pressing INSERT button to add the attribute to the attribue list.
- The user should enter the Candidate keys one by one, pressing INSERT button to add the key to the candidate keys list. If a candidate key contains more than one attribute, the user should separate the attributes by commas. For example, A_1, A_2 represents a candidate key that is composed of attributes A_1 and A_2.
- The user should enter the functional dependencies by inserting the FD left-hand side in the textbox berfore the arrow and the FD right-hand side in the text box after the arrow.

Figure 1 shows an example that explains the input of the Normalizer case tool. The main function for each button is:

Insert Button: Used to input the attributes of relations or Candidate keys or a functional dependency.

New Button: Used to begin a new normalization process.

Normalize Button: Used to perform normalization and generate the result of the normalization process.

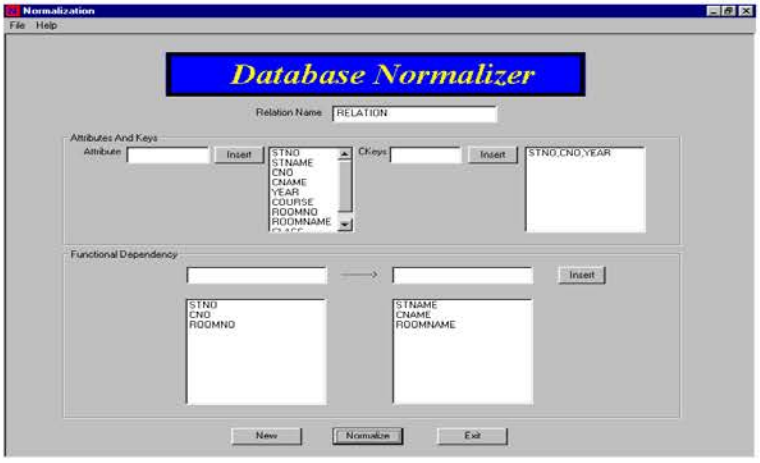

Fig. 1: Input relation example

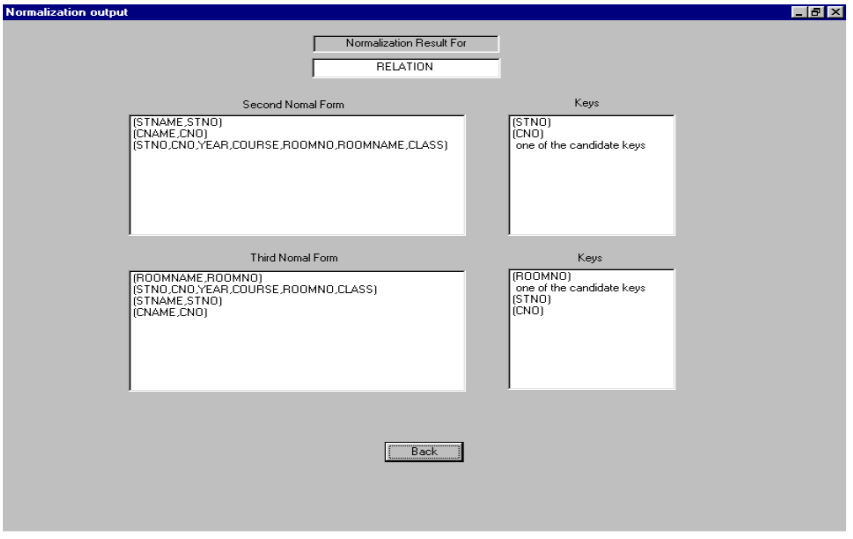

Fig. 2: Output relations from the normalization process

Exit Button: Used to exit the application (Normalizer). The File menu contains three menu items, which perform the same functions as (New, Normalize, and Exit) buttons. Figure 2 shows the output screen, which contains the new relations that result from the original relation after normalization to the 2NF and 3NF.

Back button: Used to return to the main screen.

Normalizer has been tested with this relation by entering the relation as shown in Fig. 1. Figure 2 shows the output screen, which contains the new relations that result from the original relation.

To apply Second_Normal_Form_Normalization algorithm, the input of the algorithm is given by:

RELATION(STNO, STNAME, CNO, CNAME, YEAR, COURSE, ROOMNO, ROOMNAME, CLASS)
SCK: {(STNO, CNO, YEAR)}
SFD:{STNO→STNAME, CNO→
CNAME, ROOMNO→ROOMNAME}

The algorithm first determines the set of non-prime attributes:
SNPA= {STNAME, CNAME, COURSE, ROOMNO, ROOMNAME CLASS}
The algorithm tests each functional dependency in SFD.

For FD1: STNO→ STNAME and since STNO is a proper subset of (STNO, CNO, YEAR), then this functional dependency violates 2NF, and the relation needs to be split into two relations:

R1(STNO, STNAME)
R2((STNO, CNO, CNAME, YEAR, COURSE, ROOMNO, ROOMNAME, CLASS).

The algorithm then recursively call itself with R as the input relation, SFD-FD1 as the new set of functional dependencies, and the same set of candidate keys. Doing the same for the other functional dependences produces the result shown in Fig. 2. The application of Third_Normal_Form_Normalization algorithm is very similar to the application of Second_Normal_Form_ Normalization algorithm.

CONCLUSIONS

In this study, we used the theory of relational database normalization to develop a case tool, called Normalizer, to automate the process of relational database normalization. Normalizer uses a GUI interface that is simple and easy to use. Normalizer is very useful for database designer who may be very busy to manually perform the normalization of the relational database schemas, and need to focus on other aspects of the database development process.

REFERENCES

1. Elmasri, R. and S. Navathe, 2003. Fundamentals of Database Systems. 4th Edn., Addison-Wesley,
2. Ullman, J., 1989. Principles of Database and Knowledge-Base Systems. Volume II, Computer Science Press, pp: 2.
3. Silberschatz, A., H. Korth and S. Sudarshan, 2003. Database System Concepts. 4th Edn., McGraw-Hill.