Functions of the database workbench

by YAHIKO KAMBAYASHI*
Kyushu University
Fukuoka, Japan

ABSTRACT

A powerful database system can be developed by a combination of a central relational database system and intelligent terminals. In such an organization a typical function of a terminal is to offer high-level user interfaces. In this paper the concept of the database workbench is introduced and shown to be suitable for development by such terminals. As design problems usually require a large amount of interaction, typical functions of the workbench are (1) the design of database schemas, (2) the design of conversion procedures between real-world data and data in the system, and (3) the design of queries. For the first function we focus on the relational database design under the assumption that set values are permitted. Problems of set values, especially conversion problems of dependencies, are discussed. Various facilities for design conversion procedures and the design of queries are also discussed.

*This paper was written when the author was at Kyoto University.
INTRODUCTION

As a result of the recent availability of large-scale commercial relational database systems and powerful microprocessor-based systems, a user-oriented database system can be economically created by a combination of a central large relational database system and an intelligent microprocessor-based terminal (Figure 1). High-level user interfaces are provided at the terminal, including graphics, very-high-level languages, and so on. In this paper the concept of a database workbench is introduced and shown to be very suitable for development by such terminals. The major functions of the workbench are the design of database schemas, the design of the conversion procedure between real-world data and data in the system, and the design of queries. Since such design processes involve a large amount of person-machine communication, to realize these functions at terminals saves communication cost as well as shortening response time. Although the motivation of the database workbench is similar to the programmer’s workbench concepts, functions are completely different because of the difference between programs and databases. Since some problems appearing in these processes have already been published by various authors, we will emphasize new problems in this paper.

Among other topics, facilities for database schema design will be discussed. Instead of reviewing high-level models for design, we will focus on problems of schema design for the relational model. As set values are very often used in the real world, we permit set values for database design. When we permit set values, we need to distinguish ordered sets from unordered sets and one-to-one correspondence of sets from direct products of sets. Another problem is the conversion of values to attribute names.

As checking of constraints satisfied by a schema is very important in database schema design, a procedure for schema checking will be discussed also. A database schema is defined by (1) a set of relation schemas, each of which corresponds to an attribute set; (2) constraints on attributes; and (3) constraints on sets of attributes. Each attribute is defined according to whether it corresponds to atomic values, unordered set values, ordered set values, or relation names. Attributes that will be used by selection and join operations should be distinguished. An example of constraints defined on a set of attributes is a dependency. One interesting problem discussed is the relationship among dependencies defined on attributes corresponding to atomic values as well as set values. An efficient procedure to check the existence of a join dependency is also given.

Next facilities for query design are discussed, including (1) query design using sample data; (2) query analysis facilities; and (3) a query database. We will show a practical method for obtaining a proper set of sample data as an alternative to the Armstrong relation approach, which usually produces relations with too many tuples. There are two approaches for query analysis: syntactic analysis and run-time analysis. Using a query database, a user can compose a query by a Boolean combination of retrieved queries as well as a user-specified query.

We have been developing a database workbench on a Z-80-based microprocessor system.

DATABASE SCHEMA DESIGN FACILITIES

The design of a database schema suitable for data to be stored is very important. The problem can be handled by the following steps: (1) By analyzing the real-world data, obtain a preliminary design for the database schema. (2) Using the preliminary design, convert the real-world data to relations in unnormalized form. (3) Find functional and join dependencies in the relations. (4) Design database schemas that are a collection of relation schemas. In the design, new attributes can be added, a set of values can be combined, data value can be converted to attribute names, and so on. (5) Find a suitable database schema among candidates designed in Step 4. (6) Design a procedure to convert real-world data to data represented by the database schema designed in Step 5. In Step 1 we need to find the correspondences between attributes and their values. For example, if we have the following data,


we can determine the correspondences between subsequences in the above text and attribute AUTHOR, TITLE, PUBLI-
CATION, DATE, and KEYWORD. In step 2 the data can be expressed by a relation shown in Figure 2(a).

For example, we can assume that TITLE represents the entity; i.e., TITLE is the key for the relation. In Step 3 dependencies are examined. If there are papers with the same title (published as reports and also as journals), then we have to change the key as a combination of TITLE and PUBLICATION. For sets we have to distinguish direct product and one-to-one correspondence. In Figure 2(b), there exists a one-to-one correspondence between elements in AUTHOR and AFFILIATION, and there exists a direct product property between elements in AUTHOR and KEYWORD. Whether there exist values with direct product property or not can be checked by the existence of join dependencies.

In Step 4, various schemas can be designed with dependencies and property-of-set values. Dependencies are used to determine how to decompose relations. We will define two typical properties of sets as follows: (4-a) unordered set and ordered set; and (4-b) Set to be decomposed into values.

In our example, AUTHOR is an ordered set, but KEYWORD is not. To normalize the relation we need to introduce a new attribute, AUTHOR-O, which shows the ordering. Figure 2(c) shows a normalized form for the AUTHOR, TITLE part of Figure 2(a).

There are two elementary cases in which attribute corresponds to sets. First is (4-b-1), a set of values corresponding to the same attribute. AUTHOR and KEYWORD in Figure 2(a) are examples of this case. Second is (4-b-2), a set that consists of values of more than one attribute. DATE in Figure 2(a) is an example of this case. The value 8209 can be decomposed into two values, YEAR = 82 and MONTH = 09.

In the case of (4-b-2), we need to introduce new attributes to decompose the set. Whether or not we have to decompose a set is determined by the usage of values. There are cases in which both (4-b-1) and (4-b-2) are mixed.

Another problem of designing a database schema is that we have to determine whether a value should be an attribute value, an attribute name, or a relation name. Figure 3 shows a timetable represented by three views. In Figure 3(a) MON-

DAY is an attribute value, and in Figure 3(b) it is regarded as attribute name, COURSES ON MONDAY. It also can be a relation name, as shown in Figure 3(c).

In Step 6, a procedure must be designed that will convert the real-world data to the data for the database schema we have designed. The procedure must handle the following problems: First (6-a), is a format conversion. In Figure 1(a) the following conversions are required: Lai, H. C. → Lai, H. C., Sept. 1982 → 8209, etc.

Second is (6-b), error checking. Typing errors and simple logical errors must be checked. For example, p. i-j satisfies j > i. Next is (6-c), value addition. If keywords are not supplied in the data and users must add keywords, we need to design a system for such data addition. Finally (6-d) is schema conversion. A schema conversion procedure from the real-world data to data in the target database schema must be created.

We need the following programs to realize such database design facilities: (1) A powerful partial matching program is required to analyze the real-world data efficiently. (2) A format conversion program is required by Problem (6-a). A powerful schema conversion program is required in Steps 4 and 6. (4) A schema-check program is needed in Step 4 after we design a new schema, we need to check whether it is suitable.

The program can produce sample outputs and statistical data...
such as number of tuples. (5) A dependency check program is needed at Step 3 to check whether a functional dependency or a join dependency is satisfied. (6) An error-checking program. By preparing a domain-value dictionary for each attribute, error checking as well as word control is realized. For example, to put keywords to papers, control of words to be used is required. Finally, (7) a flexible data preparation program is required in Step 6 for the purpose of solving Problem (6-c).

Since procedures for some programs are obvious or discussed elsewhere, we will discuss Programs 4 and 5 in the next section. For Programs 6 and 7, see the reference by Kambayashi and others.4

SCHEMA-CHECKING PROCEDURE

A database schema is defined by a set of relation schemas, a set of constraints on attributes, and a set of constraints defined on attribute sets. The schema-checking procedure consists of a syntactic check and a check by examining data.

A relation schema is given by a set of attributes. There are the following constraints on attributes: (1) Attributes corresponding to atomic values (atomic-value attribute for short) and attributes corresponding to sets and relations must be distinguished. (2) Attributes corresponding to set values (set-value attributes, for short) are characterized as ordered or unordered, as sets corresponding to values of the same attribute, or as sets corresponding to more than one attribute. (3) Attributes that are not used in selection, join, and division are distinguished, since such attributes can have set and relation values.

We can use attributes corresponding to set and relation values if atomic values of each set and relation value are not required to be handled separately by database operations. For example, if the relation in Figure 2 has the attribute COMMENT, we can permit sets for its values, since we are not interested in retrieving papers by specifying COMMENT values or by joining an attribute COMMENT. Values of COMMENT are only used in the result of queries.

Constraints defined on attribute sets are as follows: (1) For set-value attributes, one-to-one correspondence and direct product correspondence must be distinguished. (2) Dependencies such as functional and join dependencies exist. (3) Existence constraints are constraints such that if a value of Attribute A is not null, then a value of Attribute B is not null in every tuple. (4) Value-dependent dependencies. When a relation can be regarded as a union of subrelations and each subrelation is identified by values of some attribute set, we can permit different constraints on each subrelation. (5) There must be a set of attributes that is not handled separately by database operations.

A functional dependency \( X \rightarrow Y \) is said to be satisfied if a set of values corresponding to attribute set \( X \) uniquely determines a set of values corresponding to attribute set \( Y \). A join dependency \(*[X_1, X_2, \ldots, X_n]*\) is said to be satisfied in relation \( R \) if \( R \) is always expressed by a join of \( R[X_1], R[X_2], \ldots, R[X_n] \). Existence constraints can be expressed by a set of objects such that for any tuple there exists an object that is an attribute set corresponding to non-null values of the tuple.5

If MONTH and YEAR are not separately handled, we can use a combined attribute, DATE, to replace them. If YEAR and the combination of MONTH/YEAR are required but MONTH is not handled separately, we can use YEAR and DATE, although this representation is redundant.

The syntactic check compares sets of constraints satisfied by two given database schemas. There are the following major cases: First, if sets of attributes and constraints on the attributes are the same in both schemas, we can use dependency theory to check the equivalence.6,7 Next, if sets of attributes in both schemas are the same and there are attributes that correspond to sets in one schema and atomic values in the other schema, we need to develop a procedure to compare constraints satisfied by both schemas. Finally, when the sets of attributes are different, comparison of constraints can be realized by dividing into basic steps.

The following theorem can be used for the second case.

Theorem 1: If each subrelation of \( R \) obtained by setting values in attribute set \( X \) be constant, satisfies JD *[\( Y_1, \ldots, Y_n \)*], and \( R \) satisfies JD *[\( X_1, \ldots, X_n \)*], where \( XY \) means the union of \( X \) and \( Y \).

This theorem is obvious, but we have the following useful corollary, which establishes the correspondence of dependencies on attributes defined on atomic values and on set values.

Corollary 1: If the relation has attributes \( XY \) where each attribute in \( X \) is an atomic-value attribute and each attribute \( A_i \), in \( Y \) is a set-value attribute (i = 1, \ldots, n), then the equivalent relation on \( XY \), where all atomic-value attributes satisfy the following join dependency is as follows: *[\( X_1, X_2, \ldots, X_n \)*], where U is the set of attributes of the relation schema: *[\( X\cup A \)*].

Since functional dependency \( X \rightarrow A_1, A_2, \ldots, A_n \) can be decomposed as \( X \rightarrow A_1 \), \( X \rightarrow A_2 \), \( X \rightarrow A_n \), the above theorem can be applied to the case \( X \rightarrow Y \).

In Case 2, we can apply Corollary 1 and Theorem 2 to both schemas and compare the dependency set on the schema defined on the same set of attributes where all attributes correspond to atomic values.

Case 3 can be handled by the following cases:

(3-1) Conversion of attribute sets.
(3-2) Conversion among attribute values, attribute names, and relation names.
(3-3) Conversion of case 2.

For (3-1) we have the following cases:

(3-1-a) A new attribute is introduced that corresponds to a set of attributes.
(3-1-b) An attribute is decomposed into a set of attributes.
(3-1-c) A new attribute is introduced for an ordered set in order to store the order explicitly (see Figure 2(c)).
(3-1-d) A new attribute is introduced as a set identifier.
(3-1-e) A new attribute is introduced to handle the problem caused by the dependency set.

For Cases (3-1-a), (3-1-b), and (3-1-c), we can handle the dependency conversion very easily. An example of Case (3-1-d) is as follows: If (i) A is an attribute corresponding to a set, (ii) we need to change A to be an attribute corresponding to atomic values, and (iii) there is a requirement that we need to see the original set, then we can add a new attribute B, which is a set identifier. An example is shown in Figure 4.

This kind of conversion is required when we need to introduce an attribute corresponding to an entity or we need to keep the structure of attribute values—for example, a set of set values.

Corollary 2: The attributes A and B satisfy the following dependency, where A is originally a set-value attribute and B is introduced as a set identifier in order to make A be an atomic-value attribute: *[AB, U - A]*

It is obvious from Theorem 2, since in the original relation B⇒A is satisfied.

Some conditions for conversion of attribute values and attribute names (3-2) are shown in Reference 10.

Checking by examining data is also important. There are the following problems: (1) Checking of a functional dependency, and, if it is not satisfied, finding a set of data that violate it. (2) Checking of a join dependency, and, if it is not satisfied, finding a set of data that violate it. (3) For finding constraints we need a facility to handle small sets of example data. (4) To evaluate a database schema we need to get statistical data such as the number of tuples satisfying the given condition.

Checking of a functional dependency X → Y can be done by sorting tuples by values of X. The following theorem can be used for efficient checking of a join dependency.

Theorem 3: When JD *[X₁, X₂, ..., Xₙ]* is satisfied in R, then each subrelation of R obtained by setting values in attribute set X as a constant satisfies JD *[X₁ - X, X₂ - X, ..., Xₙ - X]*.

For checking of the existence of JD *[X₁, X₂, ..., Xₙ]* we select X as a set of attributes contained in at least two components of the JD: (1) Sort the tuples by the values of X. (2) For each subrelation having the same values for X, examine whether the JD *[X₁ - X, X₂ - X, ..., Xₙ - X]* is satisfied.

Since X₁, X₂, ..., are disjoint, we can easily check the existence of the JD as follows:

(2-1) Let p be the number of tuples in the subrelation.
(2-2) Let q₁, q₂, ..., qₙ be the number of different tuples in R[X₁ - X], R[X₂ - X], ..., R[Xₙ - X], respectively.
(2-3) If p=q₁xq₂x...xₙ, then the JD is satisfied at this subrelation. Tuples violating the join dependencies can be checked at each subrelation.

For (3) we need a schema conversion program. For the schema evaluation, one possible method is to evaluate it by the number of data contained in the schema under various conditions. We select a database schema that requires less space.

QUERY DESIGN FACILITIES

There are the following facilities for query design: (1) Query design using sample data, (2) query analysis facility, and (3) query database.

For (1) we need a procedure to design a proper set of sample data. There are two kinds of query analysis facilities, (2-a), syntactic analysis; and (2-b), run-time analysis. Query database can be used to design a query using queries already used.

In Reference 3, Armstrong relations are used for sample data. An Armstrong relation for a set of dependencies is defined as a relation satisfying exactly the dependency set, in which any dependency not derivable from the set is not satisfied by the relation. The problems with Armstrong relations are that the number of tuples in a relation tends to be very large and an actual snapshot of the relation usually satisfies dependencies not derivable from the set. We propose a practical method for selecting sample data using Theorem 3. We assume that the dependency set satisfies the following condition, since it is regarded as a practical assumption.

We assume that the dependency set is equivalent to a set consisting of at most one join dependency and functional dependencies, where the left-side set of each functional dependency is contained in at least one component of the join dependency.

Let *[X₁, ..., Xₙ]* be the join dependency and F be the set of functional dependencies. Sample relations are designed as follows.

(1-a) Modification of the join dependency, so that every functional dependency is contained in one component of the resulting join dependency. If there exists a functional dependency Y → A such that Y is contained in X, and A is not, replace X by XₜA. This conversion corresponds to a join without loss of information by Y → A.

(1-b) Let Y be the set of attributes each of which is contained in at least two components of the join dependency. Let X be the set obtained by adding all possible A to Y such that
Y → A is satisfied. As shown in Theorem 3, in each sub-relation defined by one combination of values of X, the join dependency \(*[X_1 - X, \ldots, X_n - X]\) is satisfied and has the direct product property. Using this property, sample relations are designed as follows:

(1-c) In the following, values for each attribute are selected randomly from its domain (the set of values actually appearing in relations). For the given attribute set Y and a set F of functional dependencies satisfied in Y, we will design a sample relation on Y satisfying F as follows. For each functional dependency X → Y such that X is minimal, there are at least two tuples whose XY values are identical and others are different.

(1-d) Let F be the set of functional dependencies satisfied in X. Design a relation on X satisfying F under the condition of (1-c).

(1-e) For each tuple of the relation designed in (1-d), we can design n relations R_1, \ldots, R_n, where R_k is defined on the set X_k - X. The tuple is selected from the relation on X.

(1-f) Repeat (1-e) for every tuple of the relation designed at (1-d); the union of the tuples forms the set of sample data. For a different tuple at (1-e) should try to design different R_k, although we can use the same R_k’s for all tuples.

The method for creating a set of sample data is much simpler than preparing Armstrong relations. For the purpose of checking queries by sample data, our method seems to be adequate.

There are the following facilities for query analysis:

(2-a) Syntactic query analysis. As relational language offers wide freedom to users, sometimes semantically incorrect queries cannot be detected by conventional syntactic analysis. A proper warning message is printed in the following three cases: (1) The given query consists of two or more separated queries. (2) there exists a join of attributes that seems to be unnatural. For example, SALARY = YEAR is permitted in relational expressions, but usually queries containing such a join are wrong. For this purpose we can prepare a matrix showing the properness of joining two attributes for all possible combinations. (3) There may be an error in the value of the attribute used for a selection operation; it can be detected by checking the domain of the attribute.

(2-b) Run-time query analysis. Sometimes a user wants to get information during the execution of a query in order to improve the query. For example, if a query gives a null result, a user wants to know the number of tuples at each step of query processing. As the optimization process of the database system usually does not keep the order of the execution, the given query must be divided into subqueries corresponding to each step at the workbench and then transmitted to the main database system for stepwise execution.

The query database contains the following: (3-a) Meaning of the query. (3-b) Statistical data (number of uses, cost of processing, number of the result, etc.). (3-c) Information for optimization, to avoid recalculation of access path selection. A query is identified by specifying a set of relations to be used in the query. A user can design a query by a Boolean combination of retrieved queries as well as a user-defined query. Conversion of such a query into a simpler form should be done at the database workbench.

ACKNOWLEDGMENT

The author is grateful to Professor Shuzo Yajima of Kyoto University for helpful discussions.

REFERENCES


