The Interactive Databases Approach to the User Interface Modeling

Egor Bugaenko
MIKA Corporation
Ukraine, 320044,
g.Dnepropetrovsk,
ul.Dzershinskogo, 7/33
phone: (380 562) 461 058
fax: (380 562) 461 136
Email: egor@acm.org

March 14, 1998

Abstract

The proposed principle of Interactive Databases (IDB) is intended to improve client-server interaction and to avoid database integrity constraints violation. In order to make the client more clever and give it the opportunity to make preliminary decisions regarding the correctness of the transactions without the server, we define the formal mechanism of "step-queries" definition. This mechanism is used as a tool for the conversion of the knowledge the DBMS has into the concrete collection of DML queries. Step-queries are used by the client. It sends them to the server before the transaction and the retrieves lists of possible values for the fields of the visual form. Then, these lists are used in order to give the user an opportunity to choose correct value for the given field. After the form has been completed the transaction could be filled with entered data.

After the formal description of step-queries they must be mapped to DML. Here we will define the algorithm of such mapping into SQL and show how it could be used in order to implement the IDB principle.

Keywords: transactions verification, client-database interaction, visual form, predicate calculus, SQL, step-query.

1 Introduction

Consider the process of interaction between a *client* and a database *server*. In our case *interaction* means adding and retrieving information to/from the database. In order to add some data to the database, the following operations must be accomplished: 1) the client obtains data from the user and prepare it; 2) the client expresses the transaction in some data manipulation language (DML); 3) the client sends a transaction to the server; 4) the server commits or aborts the transaction; 5) the server returns a result or an error code to the client.

Database management systems (DBMS) can have data integrity constraints expressed in the schema which can restrict the addition of new data to the database. Not having the information which the DBMS has, the client can do almost nothing to avoid cases when data is not accepted by the server because of an integrity constraint violation.

The client interacts with the user by means of the *visual form* - the window with several fields for the information entering. The user can enter data by means of "edit boxes", "list boxes" or "combo boxes". It means that the value of the field could be new or may be selected from the "predefined" list.

For this reason the only way the client can work is to process error codes returned by the server and to rebuild the transaction, or to re-ask the user to enter the visual form. The latter way is not appropriate if the process of entering information has many steps and needs active interaction with the user (e.g., visual form has many interdependent fields).

For instance, we need a batch of add-operations to be submitted by the DBMS. Each operation of the batch concerns different objects of the database, e.g., several tuples must be added to different tables in the relational database. The batch is to be accepted as one transaction: everything or nothing will be processed. The client gets the data for each operation from the user in a visual form. At the same time, the server checks the conformity of the data only after the transaction has been built. With a large amount of data and a large number of operations in the batch it would be rather difficult to reenter the whole visual form by the user. Thus, it would be better if the client was aware of whether or not the data is appropriate during the creation of the transaction.

We give it such awareness by means of step-queries definition mechanism. The client could interact with the server during the period of the visual form completeness. We name the system which uses this paradigm *Interactive Data Base* (IDB). The algorithm of interaction, formal approach to step-queries definition and examples are discussed further.

The paper is organized as follows. In the second section we overview the background of the problem and related works. The third section gives several formal definitions. The forth section is dedicated to the general description of the IDB principle. The fifth section gives a short description of the client's algorithm in the IDB systems. Advantages and performance of the example are described in the sixth section. Then we conclude.

2 Background and Related Works

The database is a collection of objects also called data [7]. The database has a conceptual schema — data model. There are several modern data models: relational citecodd, object-oriented [3], object-relational and others.

Besides data objects the data storage always contains meta information, concerning data structure and logical constraints — knowledge. In order to represent them different approaches were proposed. KIF [10] as a format for knowledge interchange is based on Lisp and first-order predicate calculus. It provides the ability for disparate programs to communicate [13]. We propose the method of knowledge representation which has much in common with KIF because predicate calculus is used in both. There are many other systems for knowledge processing: LOOM [17] (a KL-ONE style system), Epikit [9] (a predicate calculus system), Algernon [6] (a frame system), CycL [16], KEE [8], and others. Almost all of them use predicate calculus as a tool to express meta information.

First-order predicate calculus [22, 12, 4] is used as a formal ground in the majority of meta data manipulating systems [15]. At the first part of the IDB principle the mechanism of knowledge representation is used for the expressing of the meta information. We use a "one moment" insight to the database.

Thus we miss the information about temporal properties of the database [7] and we do not take into account triggers and rules within the DBMS [27]. The first disadvantage could not be avoided by means of the IDB approach. The second one is a problem for the future work.

After the formal definitions are completed we propose the interaction with the database in transaction mode. The domain of transaction processing [11] is used as a ground for the IDB principle. While the client enters information in the visual form it interacts with the server by means of SQL queries [18, 14]. Transaction management is not in the frame of the IDB principle and thus not discussed.

SQL is used in the example to show how step-queries could be mapped from predicate calculus to the real data manipulating language. Several approaches to the problem, concerning this translation, were given [21]. Any of them could be used in concrete IDB principle implementation system. In the industrial application "N96" we use simplified algorithm of the predicate calculus to SQL mapping described below. Complete description of the algorithm could be found in [2].

Besides data management the IDB principle concerns human-computer interaction [19] and bases the idea of "step-queries" upon the Dynamic Queries [24] and Dynamic Query Interfaces [25] approaches. The main goal of these paradigms is to design the communication with the user clearly and improve the cost of access to large data repositories. The same result must be accomplished by means of the proposed IDB principle.

3 Formal Definitions

In this section several formal definitions are given. Data schema, transaction, invariant and step-query are defined in terms of predicate calculus [22]. The following definitions express simple approach to the formal database theory [7]. Here we do not take into account temporal properties of the database and predict that everything is done at once. At the end of the paper we will describe the situation when the database is changeable in time.

The section contains only definitions without any concretization. The definitions are given for the sake of uniformity within the problem. Many of them are not identical with the well know existing definitions.

3.1 Data Schema

The database is a collection of objects, along with some invariants or integrity constraints on these objects. Data schema is logical equivalent of the database. Both database and data schema contains objects. But the object of the database is an element of data, when the object of data schema is a piece of knowledge.

For example, the database objects are: string "Egor Bugaenko", number $25.000, table SUPPLIERS, domain NAMES, constraint HAS_NAME, etc. The data schema objects are: the description of the relation SUPPLIERS, the definition of the domain NAMES. Only data integrity constraints are at the same time elements of data and of knowledge.

Definition 1. We define **data schema** as an aggregate:

$$Sch(A, T, C), \text{ where}$$

A — is a set of objects of schema, which include data objects:

$$A = \{a_1, a_2, \ldots, a_n\};$$

T — is a collection of structures of data schema;

C — is a collection of integrity constraints.

In the relational database a_i is a domain or a data type. In an object-oriented database a_i could be an abstract data type (ADT) or a class. Obviously, each element of A is a set:

$$a_i = \{\alpha_1, \alpha_2, \ldots, \alpha_{D_i}\}, \text{ where}$$

$\alpha_1, \alpha_2, \ldots, \alpha_{D_i}$ — are concrete data elements, e.g. integer and real numbers or strings.

D_i — is a size of the i-th domain or the number of objects in class "i".

Endless set

$$T = \{t_1, t_2, \ldots, t_i, t_{i+1}, \ldots\}$$

is a collection of sets, which include all structures of the schema. Each element t_i of T consists of i-arity structures. Thus,

$$t_i = \{\tau_1, \tau_2, \ldots, \tau_j, \ldots, \tau_{K_i}\},$$

where

$$\tau_j = \{a_{x_1}, a_{x_2}, \ldots, a_{x_i}\}.$$

Again, set t_1 contains all unary relations (relational database) or all classes (object-oriented database). Set t_2 consists of all binary relations or unary attributes etc. K_i is a number of structures of arity i in the schema (e.g., K_1 is a quantity of unary relations). Indexes $x_1, x_2, \ldots, x_i$ define which schema objects are used in the structure τ_j.

The third component of the schema definition is a collection of integrity constraints $C = \{c_1, c_2, \ldots, c_Z\}$. C is a set of predicate calculus expressions (*invariants*). The necessary condition of the database integrity could be defined as follows:

$$\forall \tau (\forall c((c \in C) \rightarrow c(A_\tau))),$$

where

$$A_\tau = A(time)$$

It means that every time τ all elements of C must be true on the set A.

3.2 Transaction

A transaction is an operation that transforms one database state to another. Transaction consists of several operations which must be committed or rejected atomically (as one unit of job). We do not take into account the state before the transaction. We treat the transaction as a condition which the database must satisfy after the execution of the operations. The conditions must be expressed in predicate calculus.

Definition 2. The transaction E is a couple consisting of a collection of variables (set V) and a conjunction of predicates (expression D). Each element is an element of T with variables from V as arguments:

$$E(V, D),$$

where

$$V = \{v_1, v_2, \ldots, v_F\},$$

and D is an expression in predicate calculus and must be a conjunction of predicates. F is a number of variables.

Thus, D must be a function of F arguments. We could state that the transaction will be accepted iff D is true. Certainly, if the database is stable in time. The result of the transaction is a logical value, which is equal to:

$$E(V, D) = \begin{cases} \mathit{true} & \text{if } D(v_1, v_2, \ldots, v_F) \text{ is true;} \\ \mathit{false} & \text{otherwise.} \end{cases}$$

Moreover, the transaction is a formal equivalent of the visual form. The transaction represents the sequence of fields of the form and logically defines the procedure of data insertion. Thus, F is a number of fields within the form.

3.3 Invariant

Definition 3. **Invariant** is an expression in predicate calculus which must be a true for all objects of some set.

For example, the following invariant requires, that each element of set BOYS must also be a member of set CHILDREN:

$$I = \forall x(\text{BOYS}(x) \to \text{CHILDREN}(x)).$$

Further the following abbreviation is used to state that the element belongs to some set:

$$S(x) \Leftrightarrow x \in S, \text{ where}$$

S — is a set, e.g., a member of set A or one of sets $t_1, t_2, \ldots, t_i, \ldots$ of set T.

3.4 Step-query

The query to the database is a declarative definition of a set. Step-query does not differ from the usual query. Step-queries are used in order to show the difference between alone queries and queries within the IDB principle. Further, it is shown how they distinguish.

Definition 6. **Step-query** is a formal definition in terms of predicate calculus of a set:

$$Q_S = \{x \ : \ P(x)\},$$

where

S — is a result of the step-query Q_S;

$P(x)$ — is a predicate calculus expression with one unbounded variable.

For example, the following expression is a step-query:

$$Q_{\text{GUYS}} = \{x \ : \ \text{CHILDREN}(x) \,\&\, \text{BOYS}(x)\},$$

and the result of this step-query is a sequence of all elements of set CHILDREN, which at the same time are elements of the set BOYS. Moreover, the result of the step-query could contain elements of different sets.

4 The IDB Principle

Consider the example of an information system, consisting of one server and several clients, which send and receive data. Each client prepares data for the server and sends it as a transaction. The server accepts either all operations in the batch or none of them. The source of information is a user, who in interactive mode enters data into the relative fields of the visual form. The client interprets the form and converts it to the transaction of some data manipulation language (e.g., SQL).

The transaction contains several simple *independent* operations, which concern different structures in the database. For example, the transaction could consist of two operations, which add data to different tables of the relational database. Operations are independent only from the clients' point of view, because the DBMS can contain structures which bind these two tables.

After populating the form, the client builds the transaction and sends it to the DBMS. The server checks whether it is possible to accept the entire transaction. If it is not acceptable for any reason (e.g, violation of at least one of the integrity constraints, blocking of some structure, permission denying etc.) the server returns an error code. According to the error code, the client returns a message to the user and proposes either to correct the form or to eject it.

4.1 The Solution

We propose the following idea: we give the client the possibility to check data at each step when populating the form and before sending the information to the server. This can be achieved by duplicating the meta information of the database (data schema and integrity constraints) within the client's local memory and exploiting this information to build step-queries at each step of the data entry.

As a result, we provide the client with the ability to decide at each step which data is correct and correlative to the other information in the database.

To do this, we define the principle of Interactive Data Bases (IDB) which is based on the paradigm above. The decision concerning data correctness and correlativeness is done at each step of the form completion by sending step-queries to the server and obtaining lists of acceptable values which can be entered into the given field. Step-queries are built using the formal definition of the data schema, including integrity constraints which the client has.

Thus, the interaction between the client and DBMS consists of the following steps:

- Formal definition in predicate calculus of the data schema of the database, with which the interaction will be done. The definition is needed to obtain the necessary meta information for building step-queries.
- Formal definition of the transaction.
- Formal definition of step-queries for each step of the form completion. These definitions will provide the possibility to formally process queries and eliminate redundancy.
- Mapping the step-query expression in predicate calculus into the query with a data manipulation language (e.g., SQL).
- Sending the query to DBMS and translating the result.

The step-query is sent to the DBMS to be processed. The result is used as a list of possible values for the first field of the form. The user selects an item from the list and the program fills the first field of the form with its value. Then this value is taken into account when creating the other step-queries. The transaction is completed as a result of consecutive requests to the DBMS and the insertion of these values from the form into the text of the transaction.

All formal definitions are done in first-order predicate calculus. Data schema and transaction must be defined in logic because of the necessity of formal translation method, which could be based only upon formal definitions. Step-queries are defined in predicate calculus for the following two reasons: 1) the use of unified tool for the description of the step-query allows us to make many different translators into different data manipulation languages (e.g., SQL, OQL etc.); 2) data schema and transaction are defined in predicate calculus and thus the most appropriate for step-queries definition language is first-order predicate calculus.

Very important to note, that the proposed IDB principle does not have anything in common with the multilevel transaction model found in object-oriented DBMS. In advanced database systems such mechanism allow to commit or abort a sub-transaction without committing or aborting the whole transaction. We do not propose to divide the transaction into smaller ones and to proceed them independently. The IDB principle improves the process of transaction verification at the time of the visual form populating. While the user enters data into the given field the system (client) checks whether the information is correct. At that time the client does not have rights to enter anything into the database. Only to check the conformity and relativeness.

4.2 Formal Schema and the Transaction Definition

For the description of the IDB principle we will use the following simple example. Data schema and the visual form follow (in the relational database).

WAYBILL_SUPPLIER
WAYBILL SUPPLIER
WAYBILL_TRUCK SUPPLIER_TRUCK
TRUCK

The definition of the transaction follows from the definition of the visual form. Note that the succession of fields in the form is important.

Waybill number:	WAYBILL
Truck number:	TRUCK
Supplier name:	SUPPLIER

Formal schema definition $Sch(A, T, C)$ consists of the definition of A, T, and C. Set A must be defined as a sequence of objects of schema and include domains of all relations and data types used in the schema:

$$A = \{a_1 = \text{STRING}, \quad a_2 = \text{TWAYBILL}, \quad a_3 = \text{TSUPPLIER}, \quad a_4 = \text{TTRUCK}\}.$$

Set T must be defined as an infinite aggregate of sets, which include all structures of the schema:

$$
\begin{aligned}
T = \{t_1 = \{\emptyset\},\\
t_2 = \{&\tau_{2_1} = \text{WAYBILL}(\text{ID : TWAYBILL}, \text{NAME : STRING}),\\
&\tau_{2_2} = \text{SUPPLIER}(\text{ID : TSUPPLIER}, \text{NAME : STRING}),\\
&\tau_{2_3} = \text{TRUCK}(\text{ID : TTRUCK}, \text{NAME : STRING})\},\\
&\tau_{2_4} = \text{WS}(\text{W : TWAYBILL}, \text{S : TSUPPLIER}),\\
&\tau_{2_5} = \text{WT}(\text{W : TWAYBILL}, \text{T : TTRUCK}),\\
&\tau_{2_6} = \text{ST}(\text{S : TSUPPLIER}, \text{T : TTRUCK}),\\
t_3 = \{\emptyset\}, \ldots\}
\end{aligned}
$$

Further, we consider only those elements of T which are not equal to $\emptyset$. Set C must be defined as a collection of invariants in predicate calculus which are equivalents to data integrity constraints of the database:

$$
\begin{aligned}
C = \{c_1 &= \forall t(\text{TRUCK}(t, \text{TNAME}) \rightarrow (\exists s, w((\text{ST}(s,t) \ \& \ \text{WT}(w,t)) \rightarrow \text{WS}(w,s)))),\\
c_2 &= \forall s(\text{SUPPLIER}(s, \text{SNAME}) \rightarrow (\exists m, w((\text{ST}(s,t) \ \& \ \text{WS}(w,s)) \rightarrow \text{WT}(w,t))))\}.
\end{aligned}
$$

The transaction must be defined as $E(V, D)$, where $V = \{v_1, v_2, \ldots, v_F\}$. From the definition of the visual form follows the consequence of fields and variables in V:

$$
V = \{w : \text{TWAYBILL}, t : \text{TTRUCK}, s : \text{TSUPPLIER}\}.
$$

In order to define D we must select from $t_1, t_2, \ldots, t_i, t_{i+1}, \ldots$ those elements which include at least one element from V as an argument and insert them all into D as conjunctors:

$$
\begin{aligned}
D = \{&\text{TRUCK}(t, \text{TNAME}) \ \& \ \text{SUPPLIER}(s, \text{SNAME}) \ \& \ \text{WAYBILL}(w, \text{WNAME}) \ \&\\
&\text{WS}(w,s) \ \& \ \text{WT}(w,m) \ \& \ \text{ST}(s,m)\}.
\end{aligned}
$$

4.3 Step-queries definition

Step-queries are defined while the client program is being developed and must be used at run-time. They are defined by means of the proposed IDB principle on the base of the definitions of data schema and transaction as described above.

Step-queries must be defined according to the necessity of using the relative field in the visual form. Preliminarily, the query must be defined as a predicate calculus expression. This formulae must contain one of the elements of set V as bounded variable under universal quantifier and a conjunction D, which follows the quantifier. For example, the definition of the query for the list of possible values for field TRUCK looks like the following:

$$
Q_{\text{TTRUCK}} = \{t \ : \ P(t)\},
$$

where

$$
\begin{aligned}
P = &\text{TRUCK}(t, \text{TNAME}) \ \&\\
&\text{SUPPLIER}(s, \text{SNAME}) \ \&\\
&\text{WAYBILL}(w, \text{WNAME}) \ \&\\
&\text{WS}(w,s) \ \& \ \text{WT}(w,t) \ \& \ \text{ST}(s,t).
\end{aligned}
$$

After this, the predicate P must be expanded taking into account all expressions from the collection C.

$$
\begin{aligned}
P = {} & \textsc{truck}(t, \textsc{tname})\ \& \\
& \textsc{supplier}(s, \textsc{sname})\ \& \\
& \textsc{waybill}(w, \textsc{wname})\ \& \\
& \textsc{ws}(w, s)\ \&\ \textsc{wt}(w, t)\ \&\ \textsc{st}(s, t)\ \& \\
& \forall t(\textsc{truck}(t, \textsc{tname}) \rightarrow \\
& \quad (\exists s, w((\textsc{st}(s, t)\ \&\ \textsc{wt}(w, t)) \rightarrow \textsc{ws}(w, s))))\ \& \\
& \forall s(\textsc{supplier}(s, \textsc{sname}) \rightarrow \\
& \quad (\exists m, w((\textsc{st}(s, t)\ \&\ \textsc{ws}(w, s)) \rightarrow \textsc{wt}(w, t)))).
\end{aligned}
$$

Still, the unbounded variables s and w are concretized relatively to the already entered fields. Either they are changed to concrete values which have been entered before. Or, if such values do not still exist, all predicates with undefined variables as arguments must be deleted from the list of conjunctors of the expression.

Then, all predicates without field variable within the list of arguments must be excluded from the list of conjunctors. Thus, from P_{TRUCK} all predicates without variable t in the list of arguments must be deleted. Consequently, the predicate P_{TRUCK} must be transformed to the following:

$$
\begin{aligned}
P = {} & \textsc{truck}(t, \textsc{tname})\ \& \\
& \textsc{wt}(w1234, t)\ \& \\
& \forall t(\textsc{truck}(t, \textsc{tname}) \rightarrow \\
& \quad (\exists s, w((\textsc{st}(s, t)\ \&\ \textsc{wt}(w1234, t)) \rightarrow \textsc{ws}(w1234, s)))).
\end{aligned}
$$

where $w1234$ is a concrete value from the domain TWAYBILL.

When the formulae of the step-query contains only predicates with bounded variables and constants as arguments, it is possible to prove the following theoreme:

Theorem 1. Any element from set R, received in the moment t as a result of applying of Q_R to the database with schema $Sch(A, T, C)$, will be relevant to all invariants of C in the moment $t + \tau$ iff $A_t = A_{t+\tau}$.

Proof. The theorem can be proved by induction. First, when C is equal to $\emptyset$, the result of the query will always be equal to R, and all elements will be relevant to the empty set C.

Secondly, let any element r from the subset of set R be relevant to all invariants of the set $C = \{c_1, c_2, \ldots, c_Z\}$. Thus, we should prove that r will be relevant to all invariants of set $C_1 = \{c_1, c_2 \ldots, c_Z, c_{Z+1}\}$, if it is relevant to the invariant c_{Z+1}.

Define J as a conjunction of all invariants of set C: $J = c_1 \,\&\, c_2 \,\&\, \ldots \,\&\, c_Z$. Then, exclude from J all invariants, which do not concern set R, (i.e are always a true value whatever r is). We will obtain a conjunction J_1 which is a part of the expression of the query Q_R, except for static definitions.

By definition, r is relevant to Q_R and thus is relevant to J_1, too. Consequently, r is relevant to J because all excluded invariants $(J_1 - J)$ are always a true value no matter what r is.

Because r is relevant to J and by the definition to c_{Z+1}, it is also relevant to $J \,\&\, c_{Z+1}$ (i.e. r is relevant to all invariants of $C = c_1 \,\&\, c_2 \,\&\, \ldots \,\&\, c_Z \,\&\, c_{Z+1}$). ■

As shown in the proof for the theorem, all elements of set R will satisfy all invariants from C iff the data is constant. It means that the following event could violate the correctness of the IDB principle and invalidate the proof: the concurrent access to the given set. It could really happen, because we do not lock the database while the user populates the visual form. If do it, than all advantages of the IDB principle will be lost.

4.4 Predicate Calculus Queries Mapping to SQL

The predicate calculus expression of the query could be mapped to the data manipulation language SQL. Further we assume that a relational database is used. According to this assumption all elements of T are *tables* and all elements of all τ_i are *attributes*. All elements of A are *domains*.

As was defined above, the query

$$Q_S = \{x \ : \ P(x)\}$$

is a formal definition of the result set. The predicate calculus expression $P(x)$ must be converted to SQL conditional expression. Predicate calculus contains several operators and quantifiers, each of which could be expressed by means of some another [22]. In order to describe the principle of "predicate calculus to SQL" conversion, we must define several rules for this "primitive" operators and quantifier mapping.

The definition of the formulae in predicate calculus follows:

1. Predicate is a formulae and looks like the following:

   $$P(d_1, d_2, \ldots, d_n),$$

   where P is a table (element of T) and $d_1, d_2, \ldots, d_n$ are attributes of this table;

2. If x and y are formulas then

   $$\overline{x}, \quad x \vee y, \quad \forall x(P(x))$$

   — also are formulas;

3. Thus all formulas in predicate calculus are defined.

We treat $x \wedge y$ as an abbreviature for $\overline{\overline{y} \vee \overline{y}}$; $\exists x(P(x))$ for $\overline{\forall x(\overline{P(x)})}$; and $x \rightarrow y$ for $\overline{x} \vee \overline{\overline{x} \vee \overline{y}}$. Using the rules above, any expression in predicate calculus could be simplified to the formulae with OR and NOT operators and a universal quantifier. Thus, we must define the rule for the conversion of these "primitive" constructs.

The step-query must be converted to the SELECT operator. Any domain from any predicate of the query expression could be chosen as a table for the selection. Then the following SQL construct must be built:

SELECT	*attribute*
FROM	*table*
WHERE	*other-attributes-conditions*
	AND *conditions-list.*

Conditions-list is a list of conditions built from the predicate calculus formulae $P(x)$. *Other-attributes-conditions* are made according to the attributes of the table, used in the selection. In the example the conditions should look like

$$(\text{W} = \text{"w1234"}),$$

if table WS was chosen as a table for the selection. The rules for the *conditions-list* formulating are defined in several statements:

1. Predicate

$$P(d_1 = a_1, d_2 = a_2, \ldots, d_n = a_n)$$

must be converted to

$$\begin{array}{ll} (\text{SELECT} & d_1 \\ \text{FROM} & P \\ \text{WHERE} & \text{NOT } ((d_1 = a_1) \text{ AND } (d_2 = a_2) \text{ AND } \ldots \\ & \text{AND } (d_n = a_n)) \text{ IS NULL} \end{array}$$

2. Logical disjunction $x \vee y$ of two predicates must be converted to

$$x \text{ OR } y;$$

3. Negation $\overline{x}$ must be converted to

$$\text{NOT } x;$$

4. A universal quantifier

$$\forall x (P_1 \,\&\, P_2 \,\&\, \ldots P_i(d_{i_1}, d_{i_2}, \ldots, d_{i_j} = x, d_{i_{j+1}} \ldots, d_{i_m}) \,\&\, P_{i+1} \,\&\, \ldots \,\&\, P_n)$$

must be converted to

$$\begin{array}{ll} (\text{SELECT} & d_1 \\ \text{FROM} & P \\ \text{WHERE} & \text{NOT } ((d_{i_1} = a_1) \text{ AND } (d_{i_2} = a_2) \text{ AND } \ldots \\ & \text{AND } (d_{i_j} = x) \text{AND } \ldots \text{ AND } (d_{i_m} = a_m) \text{ AND} \\ & P_1 \text{ AND } P_2 \text{ AND } \ldots \text{ AND } P_{i-1} \text{ AND } P_{i+1} \text{ AND } \ldots \text{ AND } P_n)) \text{IS NULL} \end{array}$$

Where $a_1, a_2, \ldots, a_n$ are attributes of the table P.

Although, these four rules *completely* defines the mapping of the predicate calculus to the SQL expressions we define the following additional rules for the sake of convenience:

1. Logical conjunction $x \wedge y$ of two predicates must be converted to

$$x \text{ AND } y;$$

2. An existential quantifier

$$\exists x(P_1 \,\&\, P_2 \,\&\, \ldots P_i(d_{i_1}, d_{i_2}, \ldots, d_{i_j} = x, d_{i_{j+1}} \ldots, d_{i_m}) \,\&\, P_{i+1} \,\&\, \ldots \,\&\, P_n)$$

must be converted to

$$\begin{array}{ll}
(\text{SELECT} & d_1 \\
\text{FROM} & P \\
\text{WHERE} & (d_{i_1} = a_1) \text{ AND } (d_{i_2} = a_2) \text{ AND } \ldots \\
 & \text{AND } (d_{i_j} = x)\text{AND } \ldots \text{ AND } (d_{i_m} = a_m) \text{ AND} \\
 & P_1 \text{ AND } P_2 \text{ AND } \ldots \text{ AND } P_{i-1} \text{ AND } P_{i+1} \text{ AND } \ldots \text{ AND } P_n)\text{IS NOT NULL}
\end{array}$$

Thus, according to the rules above the query Q_{TTRUCK} could be mapped into the following SQL expression:

```
SELECT truck
FROM wt
WHERE
  (w = w1234) AND
  ((SELECT id
   FROM truck
   WHERE NOT (id = truck)) IS NULL) AND
  ((SELECT t
   FROM truck
   WHERE NOT (
     (SELECT s
      FROM st
      WHERE
        (NOT ((t = truck.t) AND
          ((SELECT w
           FROM wt
           WHERE NOT ((w = w1234) AND (t = truck.t))) IS NULL))) OR
        ((t = truck.t) AND
        ((SELECT w
         FROM wt
         WHERE NOT ((w = w1234) AND (t = truck.t))) IS NULL))) AND
        ((SELECT w
         FROM ws
         WHERE NOT ((w = w1234) AND (s = st.s))) IS NULL)))
      IS NOT NULL)
   IS NULL)
```

Certainly, the expression is not perfect in sense of the compactness but nevertheless it is absolutely correct in sense of logic (see Theorem 1). The problem of the expression optimization is for the future work.

4.5 Query Results Processing

The client sends the query in SQL to the DBMS and obtains the result set. This set will be used as a list of possible values for the relative field of the visual form. Obviously, the form could contain fields with the combined mode of value selection (both from the list and edit box), e.g. combo-box. In such case, a new value could be inserted into the transaction without a verification. Meanwhile, the client also could verify the value by means of step-query. The definition of these queries is not in the frame of this paper.

5 The Client Algorithm

In this section we describe the example algorithm for the client. The IDB principle must be used at the time of the program building, and used by the developer of the program. Now we assume that all necessary step-queries have been already built and that we have the definitions: $\mathrm{Sch}(A, T, C)$, $E(V, D)$ and Q_{S_1}, Q_{S_2}, $\ldots$, and Q_{S_n}.

After the preparation stage completed the client can use the IDB principle by means of the visual form. The routine for the IDB processing must be built in the "window function" of the form. It could look like the following:

1. to **show** the visual form with N fields;
2. $i = 1$;
3. to **send** the step-query Q_{S_i};
4. to **receive** the list of possible values for the first field F_i;
5. the user chooses the item (f);
6. $F_i = f$; $i = i + 1$;
7. **if** $i < N$ **then goto** *3*;
8. to **fill** the transaction with $F_1, F_2, \ldots, F_N$.

6 Advantages and Performance of the Example

We implemented the IDB principle in the program "N96" — a part of the Automated System for Raw Materials Reception "EpMak-N96" [20].

The "EpMak-N96" system automates the processing of raw material reception at a factory by means of registration chip-cards with a re-programmable device inside. All requisites of the load are stored onto the chip-card when the truck arrives at the factory. Then, the gross weight is written onto the card by the electronic truck scales. After the out-loading, the net gross is calculated and written onto the card.

All information from the way-bill is moved to the card by the program "N96". To enter it, the program uses the visual form, which contains over a dozen fields and has to be edited rather quickly. Almost all of these fields have some dependency on the others. As a result, the operator must enter these values, so as not to violate these dependencies.

By means of using the IDB principle in the program "N96" we achieved several important results. First, the time needed to complete the necessary visual form was decreased. Secondly, a large amount of data sent from the server to the client and vice versa was excluded. Third, the work-load of the DBMS decreased due to the decrease in the number of rollbacks.

This section describes the advantages of using the IDB principle in the "EpMak-N96" system. The program used in the system was written first in a usual way, i.e. without improving any of the interaction process. In a very short time of the system's exploitation the lack of the approach emerged. The number of drivers became more than five hundred and the time of the selection from the menu came close to the critical point. At the same time, the number of suppliers was over one hundred.

We applied the IDB principle to the program and got the following results:

- The dataflow from the DBMS to the client was decreased due to the decrease in the number of elements of domains. The dataflow from the client to the DBMS was also slightly decreased, primarily because of the decreasing of the number of retries.
- The server overloading was decreased because of rollbacks avoidance.
- The effectiveness of the interface was increased and the time it took to populate the visual form was decreased.

Further, we explain each of these three items. As a result of applying the IDB principle to the client-server interaction programming, we received a system performance increase in the user interface and data interaction.

6.1 Dataflow

Dataflow means the quantity of information sent to/from the server. The sources of data are clients. They make the server process transactions and return result codes. First, the dataflow consists of packets sent from the client to the server (transactions). Each packet contains the text of the transaction with relative fields' values (e.g., strings, numbers etc.) Secondly, it consists of the result codes returned by the server. The size of these packets is not valuable but the number of them reflects the number of rollbacks. Third, it consists of step-queries and their results, returned by the server.

Without the IDB principle. Dataflow is increased considerably because of rollbacks and step-queries result lists. At the same time the size of the transaction is constant and step-queries are very small.

With the IDB principle. The number of rollbacks is much less and the quantity of transactions falls. Dataflow also is decreased because of the decrease in size of the step-queries results. The size of transaction is much bigger than without the IDB principle, though, and step-queries are slightly larger.

6.2 The DBMS Overloading

The DBMS can process the transaction (commitment) or reject it (rollback), depending upon whether all operations of the transaction are relevant to the content of the database or not. The server wastes time while working with rollbacks because no useful work is done during rollback processing. Thus, decreasing the number of rollbacks significantly decreases the overloading of the DBMS.

6.3 The User Interface Effectiveness

The user enters data by means of visual form. He or she populates fields with definite values and then sends the form for processing. In order to enter some value into the field, the user must select it from the list. If the list was shorter it would be more comfortable for the user. If it was necessary to populate the form only once (without re-entering) it would more appropriate. We call it user interface effectiveness.

7 Conclusion

The IDB principle was invented in order to simplify the process of transaction processing and to provide the client with the ability to verify the information. Before the data is sent to the server the client could obtain the list of possible data values from the server and could compare. These lists could be retrieved from the server by means of step-queries, which are generated using the IDB principle.

Such approach provides developers with the opportunity to built intelligent clients, vs. simple stubs without any knowledge about server database. The principle was implemented and gave rather good results. The perspective of the work refinement and improving concerns on-line step-queries generation and "any values" queries making.

References

[1] E.Bugaenko, *IDB Principle as a Mechanism of Transactions Verification*, to be published in Proceedings of the First International Workshop on Verification, Validation and Integrity Issues in Expert and Database Systems in conjunction with DEXA98.

[2] E.Bugaenko, *Predicate Calculus Step-Queries to SQL Mapping*, MIKA, IDB-97-08, 1997, Dnepropetrovsk.

[3] Edited by R.G.G.Cattell, *The Object Database Standard ODMG-93: Release 1.2*, The Morgan Kaufmann Series in Data Management Systems, 1996.

[4] S.Ceri, G.Gottlob, and L.Tanca, *Logic Programming and Databases*, New York, Springer-Verlag, 1990.

[5] E.F.Codd, *A Relational Model of Data for Large Shared Data Banks*, CACM vol.13, no.6, 1970.

[6] J.M.Crawford and B.J.Kuipers, *Toward a theory of access-limited logic for knowledge representation*, in Proceedings of the First International Conference on Principles of Knowledge Representation, Morgan Kaufmann, 1990.

[7] C.J.Date, *An introduction to Database Systems*, Reading Mass., Addison-Wesley, 6th edition, 1995.

[8] R.Fikes and T.Kehler, *The role of frame-based representation in reasoning*, Communications of the ACM, vol.28, no.9, 1985, pp.904–920.

[9] M.Genesereth, *The Epikit manual*, 1990.

[10] M.R.Genesereth and R.E.Fikes, *Knowledge Interchange Format. Version 3.0. Reference Manual*, Logic-92-1, Stanford University, Stanford, June 1991.

[11] J.Gray and A.Reuter, *Transaction Processing: Concepts and Techniques*, San Mateo, Calif., Morgan Kaufmann, 1993.

[12] P.M.D.Gray, *Logic, Algebra and Databases*, Chichester, England: Ellis Horwood Ltd., 1984.

[13] T.R.Gruber, *A Translation Approach to Portable Ontology Specifications*, KSL-92-71, Stanford University, Stanford, Calif., 1993.

[14] International Organization for Standardization, *Database Language SQL*, Document ISO/IEC 9075, 1992.

[15] M.Kifer, G.Lausen, and J.Wu, *Logical foundations of object-oriented and frame-based languages*, Journal of the ACM, vol.42, no.4, pp.740–843, 1995.

[16] D.B.Lenat and R.V.Guha, *Building Large Knowledge-based Systems: Representation and Inference in the Cyc Project*, Addison-Wesley, 1990.

[17] R.MacGregor, *The evolving technology of classification-based knowledge representation systems*, In John Sowa, editor, Principles of Semantic Networks: Explorations in the Representation of Knowledge, Morgan Kaufmann Publishers, San Mateo, Calif., pp. 385–400.

[18] J.Melton and A.Simon, *Understanding the new SQL: a Complete Guide*, Morgan-Kaufmann Publishers, Inc., 1993.

[19] B.A.Myer, J.D.Hollan, and I.F.Cruz, eds, *Strategic Directions in Human Computer Interaction*, ACM Computing Surveys, vol.28, no.4, 1996.

[20] *Automatized System for Raw Materials Reception "EpMak-N96"*, MIKA, EM2501TO, Dnepropetrovsk, 1997.

[21] M.Negri, S.Pelagatti, and L.Sbattella, *Formal Semantics of SQL Queries*, ACM TODS vol.16, no.3, September 1991.

[22] B.I.Plotkin, *Universal Algebra, Algebraic Logic and Databases*, Moscow, Nauka, 1991.

[23] D.Sahlin, *An Automatic Partial Evaluator for Full Prolog*, The Royal Institute of Technology (KTH), Stockholm, Sweden, May 1991, available at *file://sics.se/pub/isl/papers/dan-sahlin-thesis.ps.gz.*

[24] B.Shneiderman, *Dynamic Queries for Visual Information Seeking*, IEEE Software, vol.11, no.6, 1994, pp.70–77.

[25] Egemen Tanin, Richard Beigel and Ben Shneiderman, *Incremental Data Structures and Algorithms for Dynamic Query Interfaces*, ACM SIGMOD Record, vol.25, no.4, December 1996, pp.21–24.

[26] P.Wegner, *Why Interaction Is More Powerful Than Algorithms*, Communications of the ACM, vol.40, no.5, May 1997, pp.80–91.

[27] J.Widom and S.Ceri, *Active Database Systems: Triggers and Rules for Advanced Database Processing*, Morgan-Kaufmann Publishers, Inc., San Francisco, California, 1996.