An Algebra and Calculus for Multidatabases with Integrity Constraints *

Kasım S. Candan and **V.S. Subrahmanian**
Department of Computer Science
Institute for Advanced Computer Studies &
Institute for Systems Research
University of Maryland
College Park, Maryland 20742.
{candan, vs}@cs.umd.edu

Abstract

Litwin et. al. have developed a language called MSQL for query multidatabases. Subsequently, Grant, Litwin, Roussopolous and Sellis have developed a calculus and algebra associated with MSQL that facilitates querying and interoperation in a multidatabase environment. In this paper, we build upon their framework by assuming that a set of integrity constraints must be satisfied. Even though each individual database in a multidatabase may satisfy the integrity constraints, the entire multidatabase itself may not satisfy the constraints. We propose three new data retrieval notions based on whether the constraint semantics is "naive", "skeptical" or makes "choices." We propose a semantics for these operations, and develop an algebra and calculus based on these operators. We prove that the algebra can be embedded within the calculus – however, the calculus is strictly more powerful than the algebra. We study various algebraic properties linking the newly defined operators together and show how these algebraic properties can be used for query optimization.

1 Introduction

With the rapid growth of the information superhighway, it is fast becoming possible for individuals and businesses located across the world to access a wide variety of databases located at different points on the Internet. This brings to the user, a vast wealth of information stored in a multiplicity of databases and data structures. Extracting information from multiple databases often allows the user to draw conclusions that would not be deducible from one single database.

In this paper, we study the problem of integrating a set of relational databases when integrity constraints are present. Various aspects of integrating multiple relational databases have been studied by a number of authors. Litwin et. al. [9] were one of the first to develop a formal language, called MSQL, for querying multidatabases. Grant, et. al. [3] have extended these ideas and developed an algebra and calculus based on MSQL that can be used to query a multidatabase. We extend their work in the following ways:

*This work was supported by the Army Research Office under grant number DAAL-03-92-G-0225, by ARPA/Rome Labs contract F30602-93-C-0241 (ARPA Order Nr. A716) and by a National Science Foundation Young Investigator Award IRI-9357756.

1. We assume that integrity constraints are present, and that these integrity constraints must be satisfied by the multidatabase.

2. As a consequence of 1, it follows that even though an individual database satisfies the integrity constraints, the multidatabase, as a whole, may violate them. We present three versions of SQL's data retrieval operators which we call the `NAIVE`, `SKEPTICAL` and `CHOICE` operators, respectively.

3. Based on these new extensions to SQL, we propose a new relational algebra called ICBM-algebra (for "Integrity-Constraint Based Multidatabase" algebra). We study various properties of the ICBM-algebra and establish various kinds of equalities. We discuss how these relationships may be used in optimizing queries.

4. We propose a natural extension of the relational calculus called the ICBM-calculus. We prove that the ICBM-algebra can be embedded within the ICBM-calculus, i.e. all ICBM-algebra expressions can be expressed within the ICBM-calculus. However, the converse does not hold – the ICBM-calculus is strictly more expressive than the ICBM-algebra. The reasons for this are discussed.

2 Motivating Example

In this section, we present an intuitive example of a banking scenario where a bank has its head office in Washington DC (Main Center) and two branches – one in Boston and the other in Kansas. These offices maintain the following information about their clients:

- Account #: Every account in the bank has a unique account number.
- Client Name: The name of the client holding this account. No client can have more than one account for a given currency (i.e. he may have three accounts – one in US dollars, one in Yen, and one in Swiss Francs; however, he may not have two US Dollar accounts and one Austrian Schilling account).
- Balance: The value of the money in the account.
- Currency: The currency associated with the account.

The Main Center of the bank may access accounts in both Kansas and Boston. The Boston office is able to access accounts in Kansas, but not in Washington. The Kansas office is not able to access either Washington or Boston accounts.

Furthermore, the Washington DC office (Main Center) keeps track of currency fluctuations by maintaining an `exchange` relation. This relation has three attributes:

- Currency1 : First currency
- Currency2 : Second currency
- Rate: The exchange rate between first and second currencies.

The following tables show the current data in these databases :

1. Main Center :

(a) Local information :

Account #	Client Name	Balance	Currency
010386	Leonard	25000	Yen
283642	Sally	250	Dollar
534861	Alfred	2860	Pound
010098	Jack	750	Pound

Accounts

(b) Exchange Rates:

Exchange

Currency1	Currency2	Rate
Dollar	Pound	2.00
Dollar	Yen	50.00
Yen	Dollar	0.02
Yen	Pound	0.04
Pound	Dollar	0.5
Pound	Yen	25.00

In the above table, the tuple (Dollar,Pound,2.00) means that 2.00 Pounds equal one Dollar[1].

2. Kansas center :

(a) Local information :

Accounts

Account #	Client Name	Balance	Currency
689965	Susan	4000	Dollar
010067	Jack	1000	Dollar
323345	Chris	12000	Yen
623342	Peter	15000	Pound
764832	Mary	500	Dollar
079832	John	2500	Dollar

3. Boston center :

(a) Local information :

Accounts

Account #	Client Name	Balance	Currency
534861	Alfred	3500	Pound
572625	Ken	350	Yen
285361	Sally	500	Pound
976382	Peter	17000	Dollar
765532	Mary	2600	Dollar

[1] Please note that no effort has been made to verify the accuracy of these currency rates.

In addition, there are certain constraints that must be satisfied. Following the notation of Gupta, et. al. [5], we express integrity constraints in the form

$$A_1 \,\&\, \ldots \,\&\, A_n \quad \rightarrow \quad \textbf{panic}.$$

Intuitively, this constraint says that we should panic if all the conditions $A_1, \ldots, A_n$ are satisfied simultaneously. Thus, satisfaction of this constraint requires that **panic** never be derivable in any given database state, i.e. the conditions $A_1, \ldots, A_n$ are never simultaneously satisfied.

The constraints we will use in the banking example are the following:

(IC-1) "If two tuples have the same account number, then these two tuples should be identical." This can be stated as the constraint:

$$\text{Account}\#(\text{x}) = \text{Account}\#(\text{y}) \,\&\, \text{x} \neq \text{y} \quad \rightarrow \quad \textbf{panic}$$

(IC-2) "No client may have two or more accounts with the same currency."

$$\text{ClientName}(\text{x}) = \text{ClientName}(\text{y}) \,\&\, \text{Currency}(\text{x}) = \text{Currency}(\text{y}) \,\&\, \text{x} \neq \text{y} \quad \rightarrow \quad \textbf{panic}$$

(IC-3) "No account may have a balance worth less than 100 U.S. Dollars."

$$\text{Balance}(\text{x}) = \text{Amt} \,\&\, \text{Currency}(\text{x}) = \text{Cur} \,\&\, \text{Conversion}(\text{Dollar}, \text{Cur}) = \text{Rate} \,\&\, \frac{\text{Amt}}{\text{Rate}} < 100 \quad \rightarrow \quad \textbf{panic}.$$

or assuming the existence of a virtual attribute "DBal" (i.e. Dollar Balance):

$$\text{DBal}(\text{x}) < 100 \quad \rightarrow \quad \textbf{panic}.$$

The followings are sample queries to the database :

Query 1 : Get all the account information of the customer whose name is "Mary". The query is directed to the MainCenter.

Query 2 : Get all the account information of the customer whose name is "Mary". The query is directed to the Kansas database.

Query 3 : Get all the account information of the customer whose name is "Alfred". The query is directed to the MainCenter database.

Query 4 : Find the dollar value of every account. The query is directed to the MainCenter database.

We will use the above queries throughout the paper to illustrate the various formal definitions we introduce.

3 Integrity-Constraint Based Multidatabases

In this section, we will develop three versions of SQL's `SELECT` operator. In each case, we will explain the intuition underlying that operator, give the formal syntax, and explain the semantics using simple logic based rules. Prior to doing so, we first explain some notation.

3.1 Notation

A *multidatabase* $\mathbf{D}$ is a finite set of relational databases together with a reflexive and transitive accessibility relation `ACCESS` and a directed graph **NET** representing the physical connections linking databases. Intuitively, suppose D_1, D_2 are relational databases in $\mathbf{D}$ and $\texttt{ACCESS}(D_1, D_2)$ is true. In this case, it means that database D_1 is *authorized* to access the relations in database D_2. The graph **NET** represents a network such as the Internet. Intuitively, if there is a path between databases D_1 and D_2 in **NET**, then it means that database D_1 is physically in contact (via, say, a telecommunication link) with database D_2. Thus, `ACCESS` determines who is authorized to access information, while **NET** determines when these accesses can be physically realized across the communication network.

In the context of the banking example, the multidatabase consists of three databases – MainCenter, Boston and Kansas. The `ACCESS` relation consists of the reflexive transitive closure of the pairs:

$\texttt{ACCESS}(\text{MainCenter}, \text{Boston})$,
$\texttt{ACCESS}(\text{MainCenter}, \text{Kansas})$,
$\texttt{ACCESS}(\text{Boston}, \text{Kansas})$.

The graph **NET** for the banking example is shown in Figure 1. In this example, we will assume that the **NET** structure and the `ACCESS` structure are identical, i.e. whenever a physical connection exists between databases D_1 and D_2, then D_1 is authorized to access D_2.

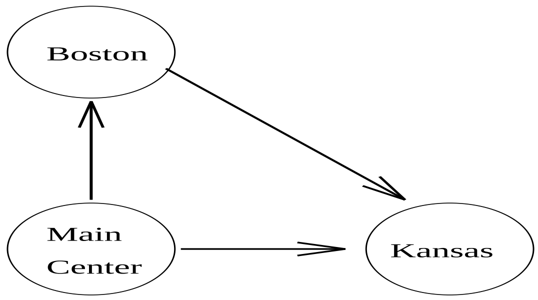

Figure 1: Structure of **NET**

Notice that the `ACCESS` relation in a multidatabase $\mathbf{D}$ may also be viewed as a directed graph (called the authorization graph) whose vertices consist of the databases in $\mathbf{D}$ and such that there is an edge from database D to D' iff $\texttt{ACCESS}(D, D')$ is true.

We say that database $D^\star$ is *reachable from* database D, denoted $reachable\_from(D^\star, D)$ iff there is: (1) a path in the authorization graph from D to $D^\star$ and (2) there is path in **NET** from D to $D^\star$.

Last, suppose $\mathbf{D_1} \subseteq \mathbf{D}$ is a set of databases in our multidatabase system and $D \in \mathbf{D}$ is a single database. We say that D is reachable from $\mathbf{D}_1$, denoted $\mathbf{reachable}(D) : [\mathbf{D}_1]$ iff D is reachable from some individual database in $\mathbf{D}_1$. In general, if A is an atom other than a **reachable_from(-,-)** atom, and $\mathcal{D}$ is a set of databases, then $A : [\mathcal{D}]$ means that atom A is true in some database in $\mathcal{D}$.

3.2 Integrity-Constraint Based Query Language

In this section, we will define certain new operators that allow integration of information from multiple databases, especially in those cases where conflicts arise. We will use the expression *selection criterion* to denote any disjunctive normal form formula, i.e. any statement of the form $\Theta_1 \vee .. \vee \Theta_c$ where each Θ_i, $1 \leq i \leq c$, is a conjunction of literals.

3.2.1 Preanswer

A tuple is qualified as a preanswer to the selection criteria $\Theta_1 \vee .. \vee \Theta_c$ iff it satisfies the given selection criteria. However, this does not guarantee that the tuple will be returned in the result list. A preanswer will be omited from the resulting set of tuples if it fails to satisfy the constraints associated with the query. Hence, the existence of a tuple in the result depends on the constraint semantics of the operator applied (we will define certain new selection operators and their semantics later on in the paper).

We define the concept of a *preanswer* as follows: A set of attributes ($A_{1,1}, .., A_{1,\phi_1}, .., A_{r,1}, .., A_{r,\phi_r}$ -where $A_{i,j}$ is the j^{the} attribute of i^{the} relation) is a *preanswer* to a query ($\Theta_1 \vee .. \vee \Theta_c$) asked of a set $\mathcal{D}$ of databases if the following holds:

$$\begin{aligned} preanswer\ (A_{1,1}, .., A_{1,\phi_1}, .., A_{r,1}, .., A_{r,\phi_r}) : [\mathcal{T}_q] \leftarrow\ & \Theta_1 \vee .. \vee \Theta_c \ \| \\ & \textbf{reachable}\ (T_1) : [\mathcal{D}], \\ & .. \\ & \textbf{reachable}\ (T_r) : [\mathcal{D}], \\ & R_1\ (A_{1,1}, .., A_{1,\phi_1}) : [\{T_1\}], \\ & .. \\ & R_r\ (A_{r,1}, .., A_{r,\phi_r}) : [\{T_r\}]. \end{aligned}$$

Here, T_i corresponds to a database which includes data from the relation R_i. A tuple will appear in the *preanswer* list if it satisfies the selection criterion and if each part of the tuple $< A_{i,1}, .., A_{i,\phi_1} >$ is satisfied by a relation R_i located in database T_i which is reachable from $\mathcal{D}$.

3.2.2 Naive Operator

The `NAIVE OPERATOR` is one which returns all the preanswers to a query. The syntax of the this operator is as follows:

$$\begin{aligned} & ASK_N\ \mathcal{D}_i \\ & SELECT\ \mathcal{A}_n \\ & FROM\ R_1, ..., R_r \\ & WHERE\ \Theta_1 \vee ... \vee \Theta_c \end{aligned}$$

Here, $\mathcal{D}_i$ denotes the set of databases on which the operator is allowed to work, and $\mathcal{A}_n$ denotes the set of attributes that will appear in the result (i.e. $\{A_{1,1}, .., A_{r,\phi_r}\}$) The intended meaning of the `NAIVE OPERATOR` operator is "use relations $R_1, ..., R_r$ that are reachable from $\mathcal{D}_i$ to find the tuples $< A_{1,1}, .., A_{r,\phi_r} >$, which satisfy the condition $\Theta_1 \vee ... \vee \Theta_c$".

Since there are no integrity constraints specified in the query, all the tuples satisfying the selection criteria will be returned to the user. The answer set may contain inconsistent information, but, the system does not care about these inconsistencies, and no attempt is made to resolve them. The handling of such violations is left to the user. We can state the semantics of the naive operator in logic as follows:

$$\begin{aligned} & Q_N : \\ & \quad answer_N\ (A_{1,1}, .., A_{r,\phi_r}) : [\mathcal{T}_q] \leftarrow preanswer\ (A_{1,1}, .., A_{r,\phi_r}) : [\mathcal{T}_q]. \end{aligned}$$

The above statement means that any tuple qualified as a *preanswer* is returned in the result of the following query:

$$\leftarrow answer_N\ (A_{1,1}, .., A_{r,\phi_r}) : [\mathcal{D}_i].$$

Answers to the Sample Queries

Query 1 :

ASK_N MainCenter
$SELECT$ $*$
$FROM$ `Accounts`
$WHERE$ $ClientName = \text{``}Mary\text{''}$

Result

Account #	Client Name	Balance	Currency
764832	Mary	500	Dollar
765532	Mary	2600	Dollar

Note that the result does not satisfy constraint **(IC-2)**.

Query 2 :

ASK_N Kansas
$SELECT$ $*$
$FROM$ `Accounts`
$WHERE$ $ClientName = \text{``}Mary\text{''}$

Result

Account #	Client Name	Balance	Currency
764832	Mary	500	Dollar

In this case all the constraints are satisfied.

Query 3 :

ASK_N MainCenter
$SELECT$ $*$
$FROM$ `Accounts`
$WHERE$ $ClientName = \text{``}Alfred\text{''}$

Result

Account #	Client Name	Balance	Currency
534861	Alfred	2860	Pound
534861	Alfred	3500	Pound

The result does not satisfy constraints **(IC-1)** and **(IC-2)**.

Query 4 :

ASK_N MainCenter
$SELECT$ $Account\#, DBal$
$FROM$ `Accounts,Exchange`
$WHERE$ `Accounts`.$Currency$ = `Exchange`.$Currency1$,

$\texttt{Exchange.}Currency2 = \text{"}Dollar''$,
$DBal = \texttt{Accounts.}Balance \times \texttt{Exchange.}Rate$

Result

Account #	DBal
010386	500
283642	250
534861	1430
010098	325
689965	4000
010067	1000
323345	240
623342	7500
764832	500
079832	2500
534861	1750
572625	7
285361	250
976382	17000
765532	2600

Tuple with *Account#* = "572625" does not satisfy constraint **(IC-3)**. In addition to this, there are two different tuples with *Account#* = "534861" which is a violation of the constraint **(IC-1)**.

3.2.3 Skeptical Operator

The `NAIVE OPERATOR` may return an inconsistent set of tuples. The `SKEPTICAL OPERATOR` removes all the tuples that are involved in such a constraint violation and returns the remaining tuples.

$ASK_S\ \mathcal{D}_i$
$SELECT\ \mathcal{A}_n$
$FROM\ R_1, ..., R_r$
$WHERE\ \Theta_1 \vee ... \vee \Theta_c$
$PANIC\ ON\ C_K$

The main difference between the `NAIVE OPERATOR` and the `SKEPTICAL OPERATOR` is that the latter contains a set of integrity constraints encapsulated in the *PANIC ON* clause. This set represents the constraints that should be enforced by the operator. Hence, if the set of tuples satisfying the selection criteria includes any tuples that violate these constraints, these tuples will be automatically omitted from the resulting set. If the user wishes to specify that all constraints are to be enforced, then he may say *PANIC ON ALL*.

In many applications, integrity violations point to an error in the system, and usually the user wants to be warned about the existence of such errors. Furthermore, the tuples causing this error may be poisonous for the other parts of the system, and hence it may be desirable to avoid them.

Therefore, it may be wise to omit these tuples from the resulting relation and return an error flag to the user in the case of a constraint violation.

There are two possible ways of warning the user about the existence of a constraint violation. The first way is to return a special "error" tuple in the resulting tuple set. The second way is to activate a global panic variable. An advantage of the second approach is that if there is an error in the system, all users can be made aware of this error by setting the global panic flag. Furthermore, there will be no need to worry about the handling of a special tuple.

We design the `SKEPTICAL OPERATOR` so that it behaves in the second way. That is, if there is a constraint violation, it omits those tuples involved in the violation and sets the truth value of a special predicate called *panic* to "true".

There are many types of constraint violations possible in a multidatabase:

1. **Definition 3.1 ("Type 1 violations" or "intra-tuple violations")** *These violations occur if and only if there is an inconsistency in the values of the attributes of a single given tuple. The corresponding constraints are called "Type 1 constraints" or "intra-tuple constraints".*

   An example of a "Type 1 constraint" is the constraint

   $$((Currency(t) = Dollar)\ \&\ (Balance(t) < 100)) \rightarrow \textbf{panic}$$

   which says that no dollar account can have a balance under 100 dollars.

2. **Definition 3.2 ("Type 2 violations" or "inter-tuple violations")** *This type of a violation is caused by a mismatch amongst two tuples in the relation. The corresponding constraints are called "Type 2 constraints" or "inter-tuple constraints".*

   The following is an example of a "Type 2 constraint"

   $$((Account\#(t) = Account\#(u))\ \&\ (t \neq u)) \rightarrow \textbf{panic}$$

   which says that two different accounts may not have the same account number.

3. **Definition 3.3 ("Type n violations")** *In general if the violation is caused exactly by "n" tuples in the relation, we call it a "Type n violation", and the corresponding constraint is called "Type n constraint".*

   In a "Type n constraint", exactly "n" different tuples are addressed.

In this paper, we will deal only with "Type 1" and "Type 2" constraints; it is easy to see that the generalization of "Type 2" constraints to "Type n" constraints is straightforward.

The following sequence of first order logic statements give the semantics of the `SKEPTICAL OPERATOR`. In this logic program, $notsatisfy_1$ takes a constraint and a set of attributes as the input, and it returns true if the corresponding tuple causes an "intra-tuple violation". Similarly, $notsatisfy_2$ takes a constraint and two sets of attributes, and returns true if the corresponding tuples cause a "Type 2" violation with respect to the input constraint. In the sequel, $\mathcal{C}$ denotes a *set* of integrity constraints.

Q_S :

$$violation_1\ (A_{1,1}, .., A_{r,\phi_r}, C) : [\mathcal{T}_q] \leftarrow notsatisfy_1\ (C, A_{1,1}, .., A_{r,\phi_r}) : [\mathcal{T}_q].$$

$$violation_2\ (A_{1,1}, .., A_{r,\phi_r}, C) : [\mathcal{T}_q] \leftarrow notsatisfy_2\ (C, A_{1,1}, .., A_{r,\phi_r}, X_{1,1}, .., X_{r,\phi_r}) : [\mathcal{T}_q],$$

$$preanswer\ (X_{1,1},..,X_{r,\phi_r}) : [\mathcal{T}_q].$$

$$violation\ (A_{1,1},..,A_{r,\phi_r},\mathcal{C}) : [\mathcal{T}_q] \leftarrow C \in \mathcal{C} \ || \ violation_1\ (C, A_{1,1},..,A_{r,\phi_r}) : [\mathcal{T}_q].$$

$$violation\ (A_{1,1},..,A_{r,\phi_r},\mathcal{C}) : [\mathcal{T}_q] \leftarrow C \in \mathcal{C} \ || \ violation_2\ (C, A_{1,1},..,A_{r,\phi_r}) : [\mathcal{T}_q].$$

$$answers_S\ (A_{1,1},..,A_{r,\phi_r},\mathcal{C}) : [\mathcal{T}_q] \leftarrow preanswer\ (A_{1,1},..,A_{r,\phi_r}) : [\mathcal{T}_q], \ \neg\ (\ violation\ (A_{1,1},..,A_{r,\phi_r},\mathcal{C}) : [\mathcal{T}_q]\ ).$$

$$panic : [\mathcal{T}_q] \leftarrow preanswer\ (X_{1,1},..,X_{r,\phi_r}) : [\mathcal{T}_q], \ \neg\ (\ answers_S\ (X_{1,1},..,X_{r,\phi_r},) : [\mathcal{T}_q]\ ).$$

and the corresponding query will be expressed as follows :

$$\leftarrow answers_S\ (A_{1,1},..,A_{1,\phi_1},..,A_{r,1},..,A_{r,\phi_r},C_K) : [\mathcal{D}_i].$$

Answers to the Sample Queries

Query 1 :

ASK_S MainCenter
$SELECT\ *$
$FROM$ `Accounts`
$WHERE\ ClientName = $ "$Mary$"
$PANIC\ ON\ (IC-1),(IC-2),(IC-3)$

Result

Account #	Client Name	Balance	Currency

Since the corresponding naive result consists of two tuples involved in a constraint violation, the skeptical result is empty.

Query 2 :

ASK_S Kansas
$SELECT\ *$
$FROM$ `Accounts`
$WHERE\ ClientName = $ "$Mary$"
$PANIC\ ON\ (IC-1),(IC-2),(IC-3)$

Result

Account #	Client Name	Balance	Currency
764832	Mary	500	Dollar

Since the corresponding naive result satisfies all the constraints, the skeptical result is the equal to the naive result.

Query 3 :

ASK_S MainCenter
$SELECT$ $*$
$FROM$ `Accounts`
$WHERE$ $ClientName = \text{“}Alfred''$
$PANIC\ ON$ $(IC-1),(IC-2),(IC-3)$

Result

Account #	Client Name	Balance	Currency

Again, the result is empty because the tuples in the corresponding naive result were violating the constraints **(IC-1)** and **(IC-2)**.

Query 4 :

ASK_S MainCenter
$SELECT$ $Account\#, DBal$
$FROM$ `Accounts, Exchange`
$WHERE$ `Accounts`.$Currency$ = `Exchange`.$Currency1$,
`Exchange`.$Currency2 = \text{“}Dollar''$,
$DBal$ = `Accounts`.$Balance$ $\times$ `Exchange`.$Rate$
$PANIC\ ON$ $(IC-1),(IC-2),(IC-3)$

Result

Account #	DBal
010386	500
283642	250
010098	325
689965	4000
010067	1000
323345	240
623342	7500
764832	500
079832	2500
285361	250
976382	17000
765532	2600

The tuples with $Account\#$ = “572625” and $Account\#$ = “534861” are omited from the list because they were violating the constraints enforced by the query.

3.2.4 Choice Operator

Whenever two tuples are involved in a “Type 2” violation, the choice operator will discard one of the two and “choose” the other[2]. The syntax of the choice operator is given below.

[2]If three tuples t_1, t_2, t_3 are involved in an inconsistency, the choice operator will pick one tuple from each of the three pairs (t_1, t_2), (t_1, t_3) and (t_2, t_3), and that tuple will be omited from the result. In principle, this could lead to all three of t_1, t_2, t_3 being discarded – later, we will show how this situation can be avoided.

$ASK_C\ \mathcal{D}_i$
$SELECT\ \mathcal{A}_n$
$FROM\ R_1, ..., R_r$
$WHERE\ \Theta_1 \vee ... \vee \Theta_c$
$CHOICE\ ON\ C_K$

The syntax of this operator is very similar to the syntax of the `SKEPTICAL OPERATOR`; the only difference is the replacement of the $PANIC\ ON$ clause by the $CHOICE\ ON$ clause. On the other hand, these two operators differ dramatically in their semantics.

In the `CHOICE OPERATOR`, if there is an inconsistency amongst two tuples of the resulting relation, then the operator chooses one of the conflicting tuples and omits the other one. Hence the "chosen" tuple has a chance to appear in the result (if the operator does not omit it due to another violation), whereas the other one is definitely pushed out of the resulting relation.

The selection of the tuple can be totally non-deterministic or it can be determined through some fixed procedure (for instance priority assignments or time stamps can be used to determine such procedures). In a later section, we are going to define a concept called $LMCAS^*$ which will give a more detailed semantics to this operator.

The `CHOICE OPERATOR` does not deal with "Type 1" violations, because in the case of intra-tuple violations making a choice does not have any meaning. The violation is caused exactly by "one" tuple, and this tuple can go into the result unless it is discarded by another constraint violation (i.e. a tuple which has a "Type 1" violation is "chosen" by default by the corresponding "Type 1" constraint). Since only "Type 2" constraints are allowed to omit tuples from the result list in the case of the `CHOICE OPERATOR`, we only need to worry about the this type of violations[3].

The semantics of the `CHOICE OPERATOR` is the following :

Q_C :

$$
\begin{array}{lll}
chosen\ (A_{1,1}, .., A_{r,\phi_r}, \mathcal{C}) : [\mathcal{T}_q] & \leftarrow & preanswer\ (A_{1,1}, .., A_{r,\phi_r}) : [\mathcal{T}_q], \\
 & & \neg\ (\ diffchoice\ (A_{1,1}, .., A_{r,\phi_r}, \mathcal{C}) : [\mathcal{T}_q]\ ). \\
 & & \\
diffchoice\ (A_{1,1}, .., A_{r,\phi_r}, \mathcal{C}) : [\mathcal{T}_q] & \leftarrow & \exists_{i,j}\ (\ X_{i,j} \neq A_{i,j}\ ) \wedge (\ C \in \mathcal{C}\ )\ || \\
 & & notsatisfy_2\ (C, A_{1,1}, .., A_{r,\phi_r}, \\
 & & \qquad X_{1,1}, .., X_{r,\phi_r}) : [\mathcal{T}_q], \\
 & & chosen\ (X_{1,1}, .., X_{r,\phi_r}, \mathcal{C})\ [\mathcal{T}_q]. \\
 & & \\
answer_C\ (A_{1,1}, .., A_{r,\phi_r}, \mathcal{C}) : [\mathcal{T}_q] & \leftarrow & chosen\ (A_1, .., A_{r,\phi_r}, \mathcal{C}) : [\mathcal{T}_q].
\end{array}
$$

and the corresponding query is :

$$\leftarrow answer_C\ (A_{1,1}, .., A_{r,\phi_r}, C_K) : [\mathcal{D}_i].$$

[3] An alternative semantics for the `CHOICE OPERATOR` would, in addition to making a choice between tuples involved in "Type 2" violations, delete all tuples involved in a "Type 1" modification. That is, it would compute the `CHOICE OPERATOR` as defined in this paper, and *subsequently* delete all tuples involved in a "Type 1" inconsistency. It is easy to see that this may be viewed as first performing a `CHOICE SELECT` and then performing a `SKEPTICAL SELECT`. Hence, this alternative semantics may be captured within the current definitions, and hence we do not discuss this further.

Answers to the Sample Queries

Query 1 :

ASK_C MainCenter
$SELECT\ *$
$FROM$ `Accounts`
$WHERE\ ClientName = \text{“}Mary''$
$CHOICE\ ON\ (IC-1),(IC-2),(IC-3)$

Result

Account #	Client Name	Balance	Currency
764832	Mary	500	Dollar

Here, the operator made a choice among two tuples which caused an integrity violation and omited the other from the result.

Query 2 :

ASK_C Kansas
$SELECT\ *$
$FROM$ `Accounts`
$WHERE\ ClientName = \text{“}Mary''$
$CHOICE\ ON\ (IC-1),(IC-2),(IC-3)$

Result

Account #	Client Name	Balance	Currency
764832	Mary	500	Dollar

Since the naive result satisfies all the constraints, the choice result is equal to the naive result.

Query 3 :

ASK_C MainCenter
$SELECT\ *$
$FROM$ `Accounts`
$WHERE\ ClientName = \text{“}Alfred''$
$CHOICE\ ON\ (IC-1),(IC-2),(IC-3)$

Result

Account #	Client Name	Balance	Currency
534861	Alfred	3500	Pound

As in the first query, the operator made a choice amongst the tuples that cause inter-tuple violations and put it in the result.

Query 4 :

ASK_C MainCenter
$SELECT\ Account\#, DBal$
$FROM$ `Accounts,Exchange`

$WHERE$ $\texttt{Accounts}.Currency = \texttt{Exchange}.Currency1,$
$\texttt{Exchange}.Currency2 = \text{“}Dollar''$,
$DBal = \texttt{Accounts}.Balance \times \texttt{Exchange}.Rate$
$CHOICE\ ON\ (IC-1), (IC-2), (IC-3)$

Result

Account #	DBal
010386	500
283642	250
534861	1430
010098	325
689965	4000
010067	1000
323345	240
623342	7500
764832	500
079832	2500
572625	7
285361	250
976382	17000
765532	2600

The tuple with *Account#* = “572625” has a “Type 1” violation, so it is left intact; on the other hand, tuples with *Account#* = “534861” were yielding to a “Type 2” violation, hence a choice has been made amongst them and only one of them survived in the result.

4 ICBM-Algebra

In the previous sections, we have defined three new operators for the new extended multidatabase language we are proposing. We called these operators `NAIVE OPERATOR`, `SKEPTICAL OPERATOR` and `CHOICE OPERATOR`, and we have presented the corresponding semantics in the form of first order logic statements.

In this section we are going to devise the corresponding extended multidatabase algebra which will be built upon the standard relational algebra. The algebraic operators proposed in this section will be capable of capturing the semantics of the new SQL operators defined in the previous sections.

An algebraic operator which operates over a multidatabase environment must include a database domain set (D_D) and a database range set (D_R) in addition to the standard attributes it has. The domain set includes the set of databases which will be used in answering the query, whereas the range set includes the set of databases which will use the results. In a distributed environment where the answers of the queries are going to be materialized, it may be very helpful to specify which sites will have the copies of the materialized results, since the answer may later be used to process other queries without traversing the network.

In the following subsections we are going to define the multidatabase algebraic operators.

4.1 Selection

We are going to propose three new select operators each reflecting one of the three different constraint semantics of the multidatabase language we have presented in the previous section. These operators are *Naive Select*, *Skeptical Select* and *Choice Select* operators.

4.1.1 Naive Select

This operator (denoted as σ_n) selects the tuples satisfying the selection criteria Θ and returns them without performing any constraint checks.

$$a \, \epsilon \, \sigma_n \, (D_R, D_D, R, \Theta) \longleftrightarrow R[D_D] \models \Theta(a)$$

$$a \, \epsilon \, \sigma_n \, (D_R, D_D, R, \Theta) \longrightarrow \forall_{i \epsilon D_R} \, (\, D_i \models a \,)$$

The first statement means that tuple a will be in the result if and only if it satisfies the selection criteria Θ and is in the relation R that is reachable from D_D.

The second statement means that all the databases in D_R will be made aware of the tuple a after the application of the operator (i.e. the result will be copied to all the databases in D_R).

4.1.2 Skeptical Select

This operator (denoted σ_s) picks the tuples which satisfy the selection criteria Θ and returns those that do not violate any integrity constraints in C_K . The other tuples are simply omited from the result.

$$\begin{aligned} a \, \epsilon \, \sigma_s \, (D_R, D_D, R, \Theta, C_K) \longleftrightarrow \; & R[D_D] \models \Theta(a) \, \wedge \\ & \forall_{c \epsilon C_K} \, (\, \neg \mathbf{notsatisfy_1} \, (a, c) \,) \, \wedge \\ & \forall_b \forall_{c \epsilon C_K} \, [\, (\, R[D_D] \models \Theta(b) \, \wedge \\ & \qquad \mathbf{notsatisfy_2} \, (a, b, c) \,) \rightarrow a = b \,] \end{aligned}$$

$$a \, \epsilon \, \sigma_s \, (D_R, D_D, R, \Theta, C_K) \longrightarrow \forall_{i \epsilon D_R} \, (\, D_i \models a \,)$$

Notice that, here $\mathbf{notsatisfy_1}$ corresponds to $notsatisfy_1$ and similarly $\mathbf{notsatisfy_2}$ corresponds to $notsatisfy_2$ defined in the previous section. The only difference between them is that $\mathbf{notsatisfy_1}$ and $\mathbf{notsatisfy_2}$ take a tuple, whereas $notsatisfy_1$ and $notsatisfy_2$ take individual attributes as their *inputs*.

The first statement means that tuple a will go to the result if and only if all the following conditions are met:

- it satisfies the selection criteria Θ and is in the relation R that is reachable from D_D.
- it satisfies all the "Type 1" constraints.
- there is no other tuple satisfying the selection criteria, which will cause a "Type 2" violation with tuple a.

The meaning of the second statement was already explained.

4.1.3 Choice Select

Similar to the *Skeptical Select*, this operator (σ_c) picks the tuples satisfying the selection criteria and eliminates the ones which lead to integrity constraint violations. But unlike the *Skeptical Select* operator, instead of eliminating all the malevolent tuples, it makes a choice amongst the tuples causing inter-tuple inconsistencies and omits the other tuples. This means that it picks one tuple from each chunk of tuples involved in an inter-tuple violation and allows that tuple to stay in the result. If this tuple does not get omited by another inter-tuple violation then it appears in the resulting relation[4].

Note that intra-tuple violations do not need to be handled by this operator, because in this type of violation there is only one tuple involved and there is no other tuple which will compete with it to go into the resulting relation. Hence, the operator simply discards any intra-tuple constraints and allow those tuples to appear in the result.

$$
\begin{aligned}
a \in \sigma_c\ (D_R, D_D, R, \Theta, C_K) \longleftrightarrow\ & R[D_D] \models \Theta(a) \wedge \\
& \forall_b \forall_{c \in C_K}\ [\ (\ R[D_D] \models \Theta(b) \wedge a \neq b \wedge \\
& \qquad \mathbf{notsatisfy_2}\ (a, b, c)\ ) \rightarrow \mathbf{chosen}\ (a, c)\ ]
\end{aligned}
$$

$$
a \in \sigma_c\ (D_R, D_D, R, \Theta, C_K) \longrightarrow \forall_{i \in D_R}\ (\ D_i \models a\ )
$$

where $(\mathbf{chosen}(a, c) \wedge \mathbf{chosen}(b, c)) \rightarrow \neg\mathbf{notsatisfy_2}(a, b, c)$.

Notice that, the **chosen** predicate is not a standard predicate in first order logic, because given a tuple a and a constraint c, it is not possible to say whether $\mathbf{chosen}(a, c)$ will return *true* or *false*. The truth value of this predicate not only depends on its attributes but also depends on the other tuples in the relation. The notion of a *choice* predicate is due to Zaniolo and his co-workers [10, 14] and was originally used for characterizing the semantics of nonmonotonic negation in deductive databases. In contrast, our use of this construct is to enable the resolution of inconsistencies in a multidatabase setting.

Hence, tuple a will go to the result if and only if all the following conditions are met:

- it satisfies the selection criteria Θ and is in the relation R that is reachable from D_D.
- if tuple a violates a "Type 2" constraint c in C_K with an other tuple b satisfying the selection criteria Θ then a must be "chosen" by the constraint c.

4.2 Projection

Unlike the select operators there is no need to have different semantics for the project operator. Hence we define only one project operator, which is semantically equivalent to the classical project operator except it that is working over a multidatabase domain.

$$
\begin{aligned}
a \in \pi\ (D_R, D_D, R, A_\pi) \longleftrightarrow\ & R[D_D] \models b \wedge \\
& a = b[A_\pi]
\end{aligned}
$$

$$
a \in \pi\ (D_R, D_D, R, A_\pi) \longrightarrow \forall_{i \in D_R}\ (\ D_i \models a\ )
$$

Tuple a appears in the result if and only if it is the projection of a tuple b in the relation R that is reachable from D_D.

[4] As in the case of the CHOICE OPERATOR, such tuples involved in "Type 1" inconsistencies can be eliminated by application of the *Skeptical Select* operator.

4.3 Join

As in the case of the select operations, we need to have different semantics in the case of joins. We are going to propose three new join operators, namely *Naive Join*, *Skeptical Join* and *Choice Join*. These three operators are similar in meaning to their select counterparts. Actually, each of these operators can be written in terms of corresponding select operators and ordinary join operators. However, we are still going to present these new operators because we believe that it is helpful to observe the existence of the different semantics for join.

4.3.1 Naive Join

In this case (the operator is denoted as $\bowtie_n$), the input relations are simply joined and all the resulting tuples are returned. That is, no constraint checks are performed.

$$\begin{aligned} a \in \bowtie_n (D_R, D_{D1}, R_1, D_{D2}, R_2) \longleftrightarrow\ & R_1[D_{D1}] \models d \wedge \\ & R_2[D_{D2}] \models e \wedge \\ & a = d \bowtie e \end{aligned}$$

$$a \in \bowtie_n (D_R, D_{D1}, R_1, D_{D2}, R_2) \longrightarrow \forall_{i \in D_R} \ (\ D_i \models a \ )$$

The above statement says that tuple a will go to the result if an only if it is equal to the join (cartesian product) of two other tuples (d and e) from the corresponding relations R_1, R_2 which are reachable from the databases in D_{D1} and D_{D2}.

4.3.2 Skeptical Join

The *Skeptical Join* operator (denoted $\bowtie_s$) performs integrity constraint checks over the resulting tuples of the join operation, and picks those that are not involved in any integrity constraint violation. Again, those tuples leading to a violation are simply omited from the result.

$$\begin{aligned} a \in \bowtie_s (D_R, D_{D1}, R_1, D_{D2}, R_2, C_K) \longleftrightarrow\ & R_1[D_{D1]} \models d \wedge \\ & R_2[D_{D2}] \models e \wedge \\ & a = d \bowtie e \wedge \\ & \forall_{c \in C_K} \ (\ \neg \mathbf{notsatisfy_1} \ (a, c) \ ) \wedge \\ & \forall_b \forall_{c \in C_K} \ [\ (\ R_1[D_{D1}] \models f \wedge \\ & \qquad\qquad R_2[D_{D2}] \models g \wedge \\ & \qquad\qquad b = f \bowtie g \wedge \\ & \qquad\qquad \mathbf{notsatisfy_2} \ (a, b, c) \ ) \\ & \qquad\qquad\qquad\qquad \rightarrow a = b \ ] \end{aligned}$$

$$a \in \bowtie_s (D_R, D_{D1}, R_1, D_{D2}, R_2, C_K) \longrightarrow \forall_{i \in D_R} \ (\ D_i \models a \ )$$

Similar to the *Skeptical Select*, tuple a goes to the result if and only if all the followings are satisfied:

- it is equal to the join (cartesian product) of two other tuples (d and e) from the corresponding relations R_1, R_2 which are reachable from the databases in D_{D1} and D_{D2}.
- it satisfies all the "Type 1" constraints.

- there is no other tuple in the join which causes a "Type 2" violation with tuple a.

Note that
$a \,\epsilon\, (\ R_1 \bowtie_s R_2\ ) \equiv a \,\epsilon\, \sigma_s(\ R_1 \bowtie R_2\ )$
i.e. we can write the *Skeptical Join* operator in terms of a *Skeptical Select* operator and the ordinary join operator. This fact will be very useful in proving some properties of this operator.

4.3.3 Choice Join

The *Choice Join* (denoted $\bowtie_c$)operator returns a tuple if and only if the tuple is in the result of the join, and one of the following holds :

1. the tuple is not involved in any integrity constraint violation.
2. the tuple has only "Type 1" violations.
3. the tuple is involved in a "Type 2" violation but it is "chosen" by all the constraints it violated.

In all other cases the tuple will be omited from the resulting relation.

$$
\begin{aligned}
a \,\epsilon \bowtie_c (D_R, D_{D1}, R_1, D_{D2}, R_2, C_K) \longleftrightarrow\ & R_1[D_{D1}] \models d \ \wedge \\
& R_2[D_{D2}] \models e \ \wedge \\
& a = d \bowtie e \ \wedge \\
& \forall_b \forall_{c \epsilon C_K}\ [\ (\ R_1[D_{D1}] \models f \ \wedge \\
& \qquad\qquad R_2[D_{D2}] \models g \ \wedge \\
& \qquad\qquad b = f \bowtie g \ \wedge \\
& \qquad\qquad \mathbf{notsatisfy_2}\ (a, b, c)\ ) \\
& \qquad\qquad\qquad\qquad \rightarrow \mathbf{chosen}\ (a, c)\ ]
\end{aligned}
$$

$$a \,\epsilon \bowtie_c (D_R, D_{D1}, R_1, D_{D2}, R_2, C_K) \longrightarrow \forall_{i \epsilon D_R}\ (\ D_i \models a\ )$$

Tuple a will go to the result if and only if all the following conditions are satisfied:

- it is equal to the join (cartesian product) of two other tuples (d and e) from the corresponding relations R_1, R_2 which are reachable from the databases in D_{D1} and D_{D2}.
- if tuple a causes a "Type 2" violation with another tuple of the join then a is "chosen".

Like the *Skeptical Join* operator, it is possible to write the *Choice Join* using the corresponding select operator and the ordinary join operator.
$a \,\epsilon\, (\ R_1 \bowtie_c R_2\ ) \equiv a \,\epsilon\, \sigma_c(\ R_1 \bowtie R_2\ )$

4.4 Union

We define the *Union Operator* without diverging from the standard semantics of this operation. Let $R1$ and $R2$ be two union compatible relations. Then,

$$
\begin{aligned}
a \,\epsilon \bigcup (D_R, D_{D1}, R1, D_{D2}, R2) \longleftrightarrow\ & R1[D_{D1}] \models a \ \vee \\
& R2[D_{D2}] \models a
\end{aligned}
$$

$$a \,\epsilon \bigcup (D_R, D_{D1}, R1, D_{D2}, R2) \longrightarrow \forall_{i \epsilon D_R}\ (\ D_i \models a\ )$$

i.e., tuple a is in the result if and only if it is in at least one of the input relations.

4.5 Intersection

We define the *Intersection Operator* in the usual way too. A tuple will appear in the result iff it appears in both of the input relations.

Let $R1$ and $R2$ be two union compatible relations. The following is the multidatabase semantics for the *Intersection Operator* :

$$\begin{aligned} a \epsilon \bigcap (D_R, D_{D1}, R1, D_{D2}, R2) \longleftrightarrow\ & R1[D_{D1}] \models a \wedge \\ & R2[D_{D2}] \models a \end{aligned}$$

$$a \epsilon \bigcap (D_R, D_{D1}, R1, D_{D2}, R2) \longrightarrow \forall_{i \epsilon D_R} (\ D_i \models a\ )$$

4.6 Difference

The last algebraic operator we will define is the (standard) $Difference\ Operator$. Let $R1$ and $R2$ be two union compatible relations We define the $Difference\ Operator$ as:

$$\begin{aligned} a \epsilon\ Diff\ (D_R, D_{D1}, R1, D_{D2}, R2) \longleftrightarrow\ & R1[D_{D1}] \models a \wedge \\ & \neg\ (\ R2[D_{D2}] \models a\ ) \end{aligned}$$

$$a \epsilon\ Diff\ (D_R, D_{D1}, R1, D_{D2}, R2) \longrightarrow \forall_{i \epsilon D_R} (\ D_i \models a\ )$$

Note that, tuple a goes to the result if and only if it is in the first relation but not in the second.

5 ICBM-Calculus

In the previous section we have defined a new relational algebra capable of capturing the semantics of the extended multidatabase SQL operators with background integrity constraints. In this section we will define a calculus capable of capturing both multidatabase and integrity constraint semantics.

5.1 ICBM-Calculus Formulas

Before defining the syntax of formulas in the calculus, we present the alphabet that we will use:

- A set, U, of attributes
- A set Θ of binary comparative operators on domains
- A set Q of relation names $\{r_1, r_2, \ldots, r_r\}$ on schemes $\{R_1, R_2, \ldots, R_r\}$, where each R_i is a subset of U
- A set, D, of databases
- The power set of D, $D^* = \{d_1, d_2, \ldots, d_d\}$
- A set, C, of constraints,
- The power set of C, $C^* = \{c_1, c_2, \ldots, c_c\}$
- A set of special predicates $\mathcal{P} = \{\mathbf{chosen}, \mathbf{satisfy_1}, \mathbf{satisfy_2}\}$.

There are two types of variables :

- "Tuple Variables" which range over the tuples,
- "Constraint Variables" which range over the set C.

Note that we do not define the concept of "Database Variables' as it is easy to extend the definition of formulas to capture "Database Variables".

An *atom* is defined to be one of the followings:

1. For any relation name r in Q, for any database sets $d_1, d_2 \in D^*$ and for any tuple variable x, $r\{d_1, d_2\}(x)$ is an atom (*annotated predicate*). The corresponding meaning is "tuple x is in relation r which is located in a database that is 'reachable' from d_1, and the result will be reflected to the databases in d_2". The special symbol "$\emptyset$" is going to be used to denote "any" set of databases reachable from the database processing the query.

2. $X \Theta Y$ is an atom, where Θ is an arithmetic comparison operator, X and Y are either constants, tuple variables or component references of the form $x[i]$, here x is a tuple variable and $i \in U$.

3. $\mathbf{satisfy_1}(x, w)$, $\mathbf{satisfy_2}(x, y, w)$ and $\mathbf{chosen}(x, w)$ are atoms where x and y are tuple variables and w is a constraint variable. The corresponding meanings are:

   (a) $\mathbf{satisfy_1}(x, w) \leftrightarrow \neg\mathbf{notsatisfy_1}(x, w)$

   (b) $\mathbf{satisfy_2}(x, y, w) \leftrightarrow \neg\mathbf{notsatisfy_2}(x, y, w)$

   (c) $\mathbf{chosen}(x, w)$ has the same meaning it had in the algebra.
   Remember that: $(\mathbf{chosen}(x, w) \wedge \mathbf{chosen}(y, w)) \rightarrow \neg\mathbf{notsatisfy_2}(x, y, w)$
   and that the **chosen** predicate is a special predicate.

A *formula* can be defined recursively as follows:

1. an atom is a formula,

2. if F_1 and F_2 are formulas then

   (a) $F_1 \wedge F_2$

   (b) $F_1 \vee F_2$

   (c) $\neg F_1$

   are formulas,

3. if F is a formula with at least one free occurrence of x, where x is a tuple variable, then $(\exists x)F$ is a formula,

4. if F is a formula with at least one free occurrence of x, where x is a tuple variable, then $(\forall x)F$ is a formula,

5. if F is a formula with at least one free occurrence of x, where x is a constraint variable, then $(\exists x \in c_i)F$, where c_i is in C^*, is a formula,

6. if F is a formula with at least one free occurrence of x, where x is a constraint variable, then $(\forall x \in c_i)F$, where c_i is in C^*, is a formula,

7. if F is a formula then (F) is a formula,

8. if F is a formula with only one free tuple variable x, and no free constraint variables, then $F\{d_1, d_2\}(x)$ is also a formula, where $d_1, d_2 \in D^*$. The corresponding meaning is: "The formula F will be evaluated in databases that are reachable from d_1, and the result will be reflected to the databases in d_2". The special symbol "$\emptyset$" is going to be used to denote "any" set of databases reachable from the database processing the query.

Nothing else is a formula. The order of evaluation of a formula is as follows:

1. () has the highest precedence,
2. $\neg$, $(\exists x)$, $(\forall x)$, $(\exists x \in c_i)$, $(\forall x \in c_i)$ are of the next precedence
3. $\wedge$ is of the third precedence
4. $\vee$ is of the fourth precedence.
5. $\{\}()$ is of the lowest precedence.

5.2 ICBM-Calculus Expressions

$ICBM - Calculus$ expressions are of the form:

$$\{x | F\{d_1, d_2\}(x)\}$$

where $d_1, d_2 \in D^*$, x is the only free tuple variable in the formula F, and there are no free constraint variables.

5.3 Safe ICBM-Calculus

As in the case of standard database calculis [13], we define the notion of safe $ICBM - Calculus$ expressions to obey the following limitations:

1. $(\forall x)$ is not used at all (notice that here x is a tuple variable)
2. if the $\vee$ operator connects two formulas F_1 and F_2, then these formulas each have only one free tuple variable, and that variable is common to both of them.
3. If a subformula is a conjunction of the form $F_1 \wedge \ldots \wedge F_m$ then all components of free tuple variables are *limited*:
   (a) if F_i is not negated and if it is not an arithmetic comparison and has a free tuple x, then all the components of x are limited.
   (b) if F_i is $x[j] = a$ or $a = x[j]$, where a is a constant, then $x[j]$ is limited.
   (c) if F_i is $x[j] = y[k]$ or $y[k] = x[j]$, where y[k] is a limited component, then $x[j]$ is limited.
4. A $\neg$ operator can only be applied to a term in a conjunction of type described in the third item.

These constraints prevents the existence of infinite relations in the calculus expressions, because such relations are not allowed by the datalog rules.

6 ICBM-Algebra vs. ICBM-Calculus

In the preceding sections, we have presented a query language, an algebra and a calculus which reflects the semantics of the integrity constraint based multidatabase relational operations. In this section we are going to compare the calculus and the algebra.

6.1 ICBM-Algebra $\leq$ Safe ICBM Calculus

In this section, we prove that the ICBM-algebra can be embedded within the ICBM-calculus. The reader who is not interested in details of the proof may skip ahead to the next section.

Theorem: For any ICBM-Algebra expression E_a there exists a corresponding safe ICBM-Calculus expression E_c.

Proof: Our proof will be based on the number of algebraic operators used in the query.

- Base Case : E_a contains "0" operators.

    1. if E_a is a constant annotated relation name of the form $r[d]$ where r is any relation in the multidatabase and d is any subset of the multidatabase, then the formula $r\{d, \emptyset\}(x)$ corresponds to $r[d]$.
    2. if E_a is a set of tuples say $\{t_1, \ldots, t_t\}$, then we use one free variable x and create the following formula:
    $$(v = t_1 \vee v = t_2 \vee \ldots \vee v = t_t)$$

- Inductive Hypothesis : For all the algebraic expressions containing $\leq n$ operators, there is an equivalent tuple calculus expression.

    Let $r[d_r]$ be the result of a query whose algebraic expression contains $n+1$ operators. This means that the expression E_a is either $\{\bigoplus(E_1, E_2)\}$ or $\{\bigoplus(E_1)\}$ where both E_1 and E_2 are subexpressions each consisting of $\leq n$ operators, and $\bigoplus$ is one of the algebraic operators (binary or unary) we have defined in the previous sections. Therefore by applying the following cases inductively we can prove our claim :

    Let F_c, F_1, F_2 be the formulas corresponding to E_c, E_1 and E_2 respectively, then we have

    1. Naive Select

    $$\begin{aligned}
    &\sigma_n \; (D_R, D_1, E_1, \Theta) \\
    &\longrightarrow E_1[D_1] \models \Theta(a) \\
    &\longrightarrow F_c' = \; (\exists u)(F_1\{D_1, \emptyset\}(u) \;\wedge\; \Theta(u) \;\wedge\; (t = u)) \\
    &\longrightarrow F_c = \; F_c'\{\emptyset, D_R\}(t)
    \end{aligned}$$

    2. Skeptical Select

    $$\begin{aligned}
    &\sigma_s \; (D_R, D_1, E_1, \Theta, C_K) \\
    &\longrightarrow E_1[D_1] \models \Theta(a) \wedge \\
    &\qquad \forall_{c \epsilon C_K} \; (\; \neg \mathbf{notsatisfy_1} \; (a, c) \;) \wedge \\
    &\qquad \forall_b \forall_{c \epsilon C_K} \; [\; (\; E_1[D_1] \models \Theta(b) \wedge \\
    &\qquad\qquad\qquad \mathbf{notsatisfy_2} \; (a, b, c) \;) \rightarrow a = b \;] \\
    &\longrightarrow F_c' = (\exists u)(\forall c \epsilon C_K)(\; F_1\{D_1, \emptyset\}(u) \;\wedge\; \Theta(u) \;\wedge\; (t = u) \;\wedge\; \mathbf{satisfy_1} \; (u, c) \wedge \\
    &\qquad\qquad\qquad \neg(\exists v)(F_1\{D_1, \emptyset\}(v) \;\wedge\; \Theta(v) \;\wedge\; (u \neq v) \wedge \neg \mathbf{satisfy_2} \; (u, v, c))) \\
    &\longrightarrow F_c = \; F_c'\{\emptyset, D_R\}(t)
    \end{aligned}$$

3. Choice Select

$$
\begin{aligned}
&\sigma_c\ (D_R, D_1, E_1, \Theta, C_K)\\
&\longrightarrow E_1[D_1] \models \Theta(a) \wedge\\
&\qquad \forall_b \forall_{c \epsilon C_K}\ [\ (\ E_1[D_1] \models \Theta(b) \wedge\\
&\qquad\qquad\qquad \mathbf{notsatisfy_2}\ (a,b,c)\ ) \rightarrow \mathbf{chosen}(a,c)\ ]\\
&\longrightarrow F'_c = (\exists u)(\forall c \epsilon C_K)(\ F_1\{D_1, \emptyset\}(u)\ \wedge\ \Theta(u)\ \wedge\ (t = u) \wedge\\
&\qquad\qquad\qquad\qquad (\ \mathbf{chosen}\ (u,c) \vee\\
&\qquad\qquad\qquad\qquad\quad \neg(\exists v)(F_1\{D_1, \emptyset\}(v)\ \wedge\ \Theta(v)\ \wedge\ (u \neq v) \wedge\ \neg\mathbf{satisfy_2}\ (u,v,c)))\\
&\longrightarrow F_c =\ F'_c\{\emptyset, D_R\}(t)
\end{aligned}
$$

4. Projection

$$
\begin{aligned}
&\pi\ (D_R, D_1, E_1, A_\pi)\\
&\longrightarrow E_1[D_1] \models b \wedge a = b[A_\pi]\\
&\longrightarrow F'_c =\ (\exists u)(F_1\{D_1, \emptyset\}(u)\ \wedge\ (t = u[A_\pi]))\\
&\longrightarrow F_c =\ F'_c\{\emptyset, D_R\}(t)
\end{aligned}
$$

5. Naive Join

$$
\begin{aligned}
&\bowtie_n\ (D_R, D_1, E_1, D_2, E_2)\\
&\longrightarrow E_1[D_1] \models d \wedge\\
&\qquad E_2[D_2] \models e \wedge\\
&\qquad a = d \bowtie e\\
&\longrightarrow F'_c =\ (\exists u)(\exists v)(F_1\{D_1, \emptyset\}(u)\ \wedge\ F_2\{D_2, \emptyset\}(v) \wedge\\
&\qquad\qquad\qquad t[1] = u[1]\ \wedge \ldots \wedge\ t[n] = u[n]\ \wedge\ t[n+1] = v[1]\ \wedge \ldots \wedge\ t[n+m] = v[m])
\end{aligned}
$$

Here, n is the number of attributes in tuple u and m is the number of attributes in tuple v.

6. Skeptical Join

$$
\begin{aligned}
&\bowtie_s\ (D_R, D_1, E_1, D_2, E_2, C_K)\\
&\longrightarrow E_1[D_1] \models d \wedge\\
&\qquad E_2[D_2] \models e \wedge\\
&\qquad a = d \bowtie e \wedge\\
&\qquad \forall_{c \epsilon C_K}\ (\ \neg\mathbf{notsatisfy_1}\ (a,c)\ ) \wedge\\
&\qquad \forall_b \forall_{c \epsilon C_K}\ [\ (\ E_1[D_1] \models f \wedge\\
&\qquad\qquad\qquad E_2[D_2] \models g \wedge\\
&\qquad\qquad\qquad b = f \bowtie g \wedge\\
&\qquad\qquad\qquad \mathbf{notsatisfy_2}\ (a,b,c)\ ) \rightarrow a = b\ ]\\
&\longrightarrow F'_c = (\exists u)(\exists v)(\forall c \epsilon C_K)(F_1\{D_{D_1}, \emptyset\}(u)\ \wedge\ F_2\{D_{D_2}, \emptyset\}(v) \wedge\\
&\qquad\qquad\qquad\qquad t[1] = u[1]\ \wedge \ldots \wedge\ t[n] = u[n]\ \wedge\\
&\qquad\qquad\qquad\qquad t[n+1] = v[1]\ \wedge \ldots \wedge\ t[n+m] = v[m]) \wedge\\
&\qquad\qquad\qquad\qquad \mathbf{satisfy_1}\ (t,c) \wedge\\
&\qquad\qquad\qquad\qquad \neg(\exists j)(\exists k)(\exists l)(F_1\{D_{D_1}, \emptyset\}(k)\ \wedge\ F_2\{D_{D_2}, \emptyset\}(l) \wedge\\
&\qquad\qquad\qquad\qquad\qquad j[1] = k[1]\ \wedge \ldots \wedge\ j[n] = k[n]\ \wedge\\
&\qquad\qquad\qquad\qquad\qquad j[n+1] = k[l]\ \wedge \ldots \wedge\ j[n+m] = l[m]) \wedge\\
&\qquad\qquad\qquad\qquad\qquad \wedge\ (t \neq j) \wedge\ \neg\mathbf{satisfy_2}\ (t,j,c)))\\
&\longrightarrow F_c =\ F'_c\{\emptyset, D_R\}(t)
\end{aligned}
$$

Here, n is the number of attributes in tuple u and m is the number of attributes in tuple v.

7. Choice Join

$$
\begin{aligned}
&\bowtie_c (D_R, D_1, E_1, D_2, E_2, C_K)\\
&\longrightarrow E_1[D_1] \models d \wedge\\
&\qquad E_2[D_2] \models e \wedge\\
&\qquad a = d \bowtie e \wedge\\
&\qquad \forall_b \forall_{c \epsilon C_K} \; [\; (\; E_1[D_1] \models f \wedge\\
&\qquad\qquad\qquad E_2[D_2] \models g \wedge\\
&\qquad\qquad\qquad b = f \bowtie g \wedge\\
&\qquad\qquad\qquad \mathbf{notsatisfy_2}\ (a, b, c)\;) \rightarrow \mathbf{chosen}(a, c)\;]\\
&\longrightarrow F'_c = (\exists u)(\exists v)(\forall c \epsilon C_K)(\; F_1\{D_{D_1}, \emptyset\}(u) \;\wedge\; F_2\{D_{D_2}, \emptyset\}(v) \wedge\\
&\qquad\qquad t[1] = u[1] \;\wedge \ldots \wedge\; t[n] = u[n] \;\wedge\\
&\qquad\qquad t[n+1] = v[1] \;\wedge \ldots \wedge\; t[n+m] = v[m]) \wedge\\
&\qquad\qquad (\mathbf{chosen}\ (t, c) \vee\\
&\qquad\qquad \neg(\exists j)(\exists k)(\exists l)(F_1\{D_{D_1}, \emptyset\}(k) \;\wedge\; F_2\{D_{D_2}, \emptyset\}(l) \wedge\\
&\qquad\qquad\qquad j[1] = k[1] \;\wedge \ldots \wedge\; j[n] = k[n] \;\wedge\\
&\qquad\qquad\qquad j[n+1] = k[l] \;\wedge \ldots \wedge\; j[n+m] = l[m]) \wedge\\
&\qquad\qquad\qquad \wedge\; (t \neq j) \wedge\; \neg\mathbf{satisfy_2}\ (t, j, c))))\\
&\longrightarrow F_c = \; F'_c\{\emptyset, D_R\}(t)
\end{aligned}
$$

Here, n is the number of attributes in tuple u and m is the number of attributes in tuple v.

8. Union

$$
\begin{aligned}
&\bigcup (D_R, D_1, E_1, D_2, E_2)\\
&\longrightarrow E1[D_1] \models a \vee E2[D_2] \models a\\
&\longrightarrow F'_c = \; F_1\{D_1, \emptyset\}(t) \;\vee\; F_2\{D_2, \emptyset\}(t)\\
&\longrightarrow F_c = \; F'_c\{\emptyset, D_R\}(t)
\end{aligned}
$$

9. Intersection

$$
\begin{aligned}
&\bigcap (D_R, D_1, E_1, D_2, E_2)\\
&\longrightarrow E1[D_1] \models a \wedge E2[D_2] \models a\\
&\longrightarrow F'_c = \; F_1\{D_1, \emptyset\}(t) \;\wedge\; F_2\{D_2, \emptyset\}(t)\\
&\longrightarrow F_c = \; F'_c\{\emptyset, D_R\}(t)
\end{aligned}
$$

10. Difference

$$
\begin{aligned}
&Diff\ (D_R, D_1, E_1, D_2, E_2)\\
&\longrightarrow E1[D_1] \models a \wedge \neg\; (\; E2[D_2] \models a\;)\\
&\longrightarrow F'_c = \; F_1\{D_1, \emptyset\}(t) \;\wedge\; \neg F_2\{D_2, \emptyset\}(t)\\
&\longrightarrow F_c = \; F'_c\{\emptyset, D_R\}(t)
\end{aligned}
$$

As shown above, for each algebraic operator E_a, there exists an equivalent tuple calculus expression of the form:

$$E_c = \; \{t \mid F_c\{\emptyset, \emptyset\}(t)\}$$

6.2 Safe ICBM Calculus > ICBM-Algebra

In the previous subsection we have shown that for any ICBM-Algebra expression, there exists a corresponding safe ICBM-Calculus expression. On the other hand, the converse of this property does not hold, i.e. the ICBM-Calculus presented in this section is more powerful than the ICBM-Algebra presented earlier.

To see see this, observe that the following safe ICBM-Calculus expression E_c does not have any corresponding ICBM-Algebra expression :

$$E_c = \{t \mid (\; (\exists c \in C_K)(r\{d_1, d_2\}(t) \;\wedge\; \mathbf{satisfy_1}(t, c) \;) \; \{\emptyset, \emptyset\}(t) \; \}$$

The reason why this ICBM-calculus formula cannot be captured with the ICBM-algebra is that it involves an existential quantification over integrity constraints and the algebra always assumes a universal quantification over all integrity constraints specified in a query (algebraic expression).

7 LMCAS - Largest Maximally Consistent Answer Set

In the previous sections we have looked at different semantics for the select and join operations in multidatabases, and we have proposed new algebraic operators to deal with these new semantics.

We would like to study different properties of the ICBM-algebra in order to be able to optimize queries by the reformulation of algebraic expressions into computationally less formidable forms. However, in order to do this, we need to know more about the answers we expect from the execution of the operators. This is the subject of this section – the next section will contain results on properties (commutativity, associativity, etc.) of the ICBM-algebra.

7.1 *MCAS*

The structure of the answer sets of *Naive* and *Skeptical* operators are straightforward. On the other hand, because of the nondeterministic characteristic of the *Choice* operators, it is not very clear which tuples will go into the answer set and which ones will be omited.

We have already stated in the previous sections, that *Skeptical Join* and *Choice Join* operators can be expressed in terms of ordinary join operators and the corresponding select operators. Hence, it will be enough for us to study the answers of the select operators in more detail. Then, it is possible to use these results to determine the expected behavior of the join operators.

We define *MCAS* (Maximally Consistent Answer Set) for a select operation as follows :

Definition 7.1 (*MCAS*) *A set of tuples A is a MCAS for a selection query* Θ *iff every tuple in A satisfies the selection criteria and A is consistent with respect to the constraint semantics of the selection operator; furthermore, any tuple added to this set will either cause a violation of the selection criteria or will lead to a violation of an integrity constraint.*

Definition 7.2 (*MCAS – Naive*) *Since there are no integrity constraints specified by the Naive Select operator, in this case, the MCAS is simply the set of tuples which satisfy the selection criteria.*

Definition 7.3 (*MCAS – Skeptical*) *For the Skeptical Select operator, the MCAS is the set of tuples which satisfy the selection criteria minus those that are involved in an integrity constraint violation.*

MCAS for the *Choice Select* operator is more tricky, because in this case, for each “pair” of tuples leading to a “Type 2” integrity constraint violation, we keep at most one tuple in the result, and different choices of tuples may lead to different answer sets. Before presenting the formal definition of the *MCAS* for the *Choice Select* operator, we will present some informal discussion.

Informally, $MCAS$ for the *Choice Select* operator is a violation-free set, consisting of the tuples that satisfy the selection criteria and that fall in one of the following categories:

Definition 7.4 (0-safe tuples:) *do not cause any integrity constraint violation.*

Definition 7.5 (1-safe tuples:) *cause only intra-tuple ("Type 1") violations.*

Definition 7.6 (2-safe tuples:) *cause inter-tuple ("Type 2") integrity constraint violations, but are "chosen" by all the "Type 2" constraints they have violated.*

As usual, an $MCAS$ must satisfy the maximality condition (i.e. any tuple addition to this set will either cause a violation of the selection criteria or will lead to a violation of an integrity constraint of "Type 2").

We may now define $MCAS$ for the choice operator in a formal graph theoretical framework.

- Let R_Θ be the subset of the original relation which satisfies the selection criteria. Note that $MCAS$ is a subset of R_Θ.
- Let G be an undirected graph (V, E) where the vertices in V are named by the tuples of R_Θ and (v_1, v_2) is in E iff v_1 and v_2 violate a constraint "Type 2" together. Hence, at most one of these two tuples must appear in the $MCAS$; otherwise there will be an inter-tuple violation in the result.

Notice that in the graph G two vertices are connected by a "single" edge iff the corresponding tuples lead to at least one "Type 2" violation.

Example: Let us suppose that a relation called *Accounts* is distributed in two databases d_1 and d_2 as shown below:

$A\ (d_1)$

Account #	Client Name	Tupleid
010386	Leonard	1
283642	Sally	2
534861	Leonard	3
010098	Kate	4

$A\ (d_2)$

Account #	Client Name	Tupleid
236465	Peter	5
283642	Mary	6
936324	Jack	7
010098	Jack	8

Let the constraint set be

$$\phi : \{((Account\#(t) = Account\#(u)) \ \& \ (ClientName(t) \neq ClientName(u))) \ \rightarrow \ \mathbf{panic},$$
$$((ClientName(t) = ClientName(u)) \ \& \ (Account\#(t) \neq Account\#(u))) \ \rightarrow \ \mathbf{panic}\}$$

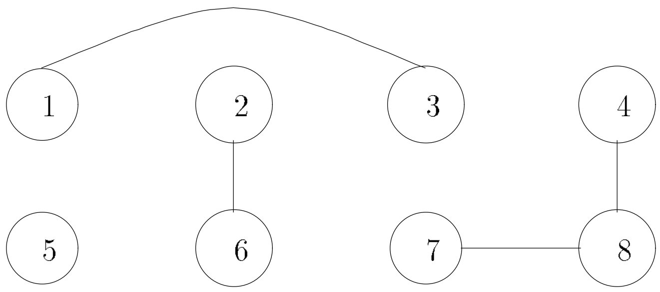

Figure 2: Graph G

The corresponding graph G is in Figure 2. Note that tuples 1 and 3 violate the second constraint, and hence, they are connected by an edge in the graph. Similarly, tuples 7 and 8 have the same violation and they are connected as well. The pairs 4-8 and 2-6, on the other hand, violate the first constraint. Hence, these pairs are connected too.
¿From this graph, it is easy to see that the possible $MCAS$' are: $\{1,2,4,5,7\}$, $\{1,2,5,8\}$, $\{1,4,5,6,7\}$, $\{1,5,6,8\}$, $\{3,2,4,5,7\}$, $\{3,2,5,8\}$, $\{3,4,5,6,7\}$ and $\{3,5,6,8\}$.

The following is the formal definition for the $MCAS$ for the choice operator.

Definition 7.7 ($MCAS - Choice$) *$MCAS$ for the $Choice\ Select$ operator is any "maximal independent set" [4] of the resulting graph (follows directly from the definition of the "maximal independent set"). That is, $MCAS$ is a subset of V such that any edge in E is incident on at most one vertex in $MCAS$ and such that you can not add another vertex to the set without violating this property.*

7.2 $LMCAS$

As can be seen in the above example, there may be many $MCAS$'s associated with a choice query and the cardinality of these may differ.

Definition 7.8 (LMCAS) *We define an $LMCAS$ (Largest MCAS) to be an $MCAS$ of the largest possible cardinality.*

Example: Hence, the $LMCAS$' for our example are: $\{1,2,4,5,7\}$, $\{1,4,5,6,7\}$, $\{3,2,4,5,7\}$ and $\{3,4,5,6,7\}$.

For the $Naive\ Select$ and the $Skeptical\ Select$ operators $LMCAS$ is identical to $MCAS$, because there is only one $MCAS$ in each of these two cases.

7.3 $LMCAS^*$

Definition 7.9 ($LMCAS^*$) *There may be more than one $LMCAS$ of a query. Hence we will assume that one of the $LMCAS$'s is chosen in some arbitrary, but fixed manner, to reflect the "**semantics of the select operation**". We call this set "$LMCAS^*$".* What this means is that for our purposes, we will assume that there is some agreed upon method to select an $LMCAS$ from the set of $LMCAS$s associated with a given query.

We observe that for the *Naive Select* and the *Skeptical Select* operators $LMCAS^*$ is equal to the only $LMCAS$. We now study various properties of the $LMCAS$'s and $LMCAS^*$ semantics for choice select.

7.4 Properties of $LMCAS$ and $LMCAS^*$

7.4.1 Complexity

An important property of any semantics is its complexity. It is well known that finding a "largest independent set" of a graph is an NP-Complete problem. As $LMCAS$'s are defined directly in terms of the largest independent sets of graphs, it follows immediately that finding an $LMCAS$ of a given *Choice Select* query is also NP-Complete (as is finding $LMCAS^*$).

7.4.2 Update properties-Size of the $LMCAS$

In this section, we briefly discuss how the size of an $LMCAS$ varies when insertions are made to one of the databases in our multidatabase system.

Suppose Q is a choice query involving relation R Adding a new tuple to the relation R may have two effects on the graph associated with this query:

- The graph stays the same because the tuple does not satisfy the selection criteria expressed in the query, or
- A new vertex is added to the graph, i.e. the tuple satisfies the selection criteria, and the new vertex represents this new tuple.

In the latter case, the new tuple may lead to new "Type 2" inconsistencies with already existing tuples. These new inconsistencies will be manifest themselves in the graph associated with the query in the form of new edges from the tuple to (old) tuples.

1. For *Naive Select* and *Choice Select*:
   $LMCAS$ is non-decreasing. That is, when you add a new vertex to the graph (and possibly new edges between new vertex and the old vertices), the size of the $LMCAS$ does not decrease. Hence $LMCAS^*$ is also non-decreasing.
2. For *Skeptical Select*:
   $LMCAS$ is unpredictable. That is, when you add a new vertex to the graph (and possibly new edges between new vertex and the old vertices), the size of the $LMCAS$ may increase, decrease or stay the same. Hence $LMCAS^*$ is also unpredictable.

7.5 Update properties-Monotonicity of the $LMCAS$

IN this section, we briefly study monotonicity properties of the $LMCAS$-semantics. Suppose we have inserted a new tuple, as before, into a relation R that occurs in a query Q, and suppose this new tuple satisfies the selection criteria specified in query Q.

1. For *Naive Select*:
   $LMCAS$ is monotonic in this case because the addition of the new tuple causes the addition of a new vertex to the graph (and possibly new edges between the new vertex and old vertices). It is easy to see that the new $LMCAS$ is a superset of the old $LMCAS$. Hence $LMCAS^*$ is also monotonic.

2. For *Skeptical Select*:
   $LMCAS$ is non-monotonic. This follows directly from the fact that $LMCAS$ exhibits unpredictable when new vertices are added to the graph associated with the query. Therefore, $LMCAS^*$ is non-monotonic too.

3. For *Choice Select*:
   In this case, $LMCAS$ exhibits partial monotonicity properties. If the newly inserted tuple satisfies the selection criteria expressed in the query, then this corresponds to adding a new vertex to the graph (and possibly new edges between new vertex and the old vertices). Each of the new $LMCAS$s' is a superset of an old $LMCAS$.

   However, the converse is not true, i.e. there may be old $LMCAS$'s that cannot be "expanded" to a corresponding new $LMCAS$).

   For this reason, it follows that $LMCAS^*$ is non-monotonic. It is not possible to guarantee that the new $LMCAS^*$ after the addition of a new vertex (and possibly new edges between new vertex and the old vertices), will be a superset of the old one. Nor can this be guaranteed in the case of tuple (i.e. vertex) deletions. However, in the case of tuple (vertex) deletions, we can make the following statements:

   (a) If the removed vertex was not in the old $LMCAS^*$, then the new $LMCAS^*$ is equivalent to the old $LMCAS^*$.

   (b) If the removed vertex was in the old $LMCAS^*$, then we can pick the new semantics to be a superset (not necessarily proper) of the remaing part of the old $LMCAS^*$.

   Figure 3, shows that the removal of the vertex c from the graph leads to an increase in the number of $LMCAS$'s. Initially there was only one $LMCAS$ ($= LMCAS^*$); however, after the removal of the vertex c, this number increases to two. Either of these $LMCAS$'s can be chosen to be the new $LMCAS^*$. We assume here that, $LMCAS_1$ will be chosen as the new $LMCAS^*$, because it is a equal to the remaing part of the old $LMCAS^*$.

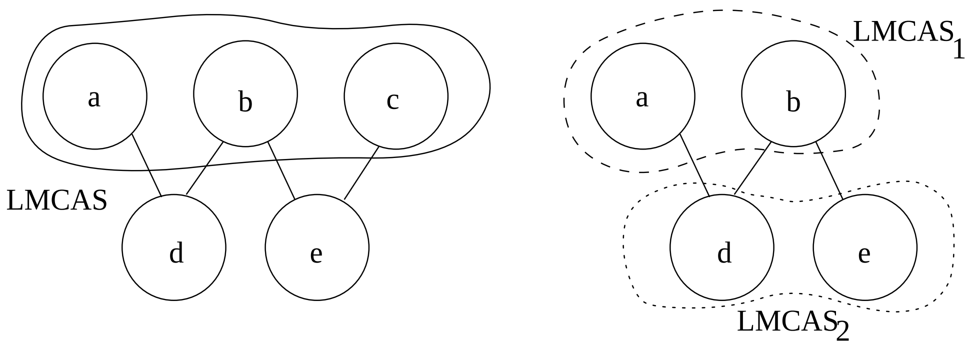

Figure 3: Vertex c is removed from the initial graph

8 Some Properties of the ICBM-Algebra

In this section we study various algebraic properties (such as commutativity, associativity etc.) of the ICBM-Algebra that we have developed in this paper. When convenient, we will use ICBM-Calculus equivalents of the algebraic operators to prove these properties. In many cases, these properties will lead to strategies for optimizing queries. Before we proceed to study the various properties, we set out certain basic assumptions and notation.

- $LMCAS^*$ be the semantics of the operators,
- a, b, c, d, e and f be relations,
- Θ_1 and Θ_2 be selection criteria of the corresponding operators,
- ϕ_1 and ϕ_2 be constraint sets of the corresponding operators,
- k, j, l, t, u, y and z be tuple variables,
- c be a constraint variable,
- and the operators use the same set of databases.

In the sequel, given two algebraic operators op_1, op_2, $op_1 op_2$ means: *apply op_2 first, and then apply op_1.*

As *Naive Select* is the same as ordinary select, it is easy to see that these operators commute; and hence, when applying two *Naive Select* operators one after the other, it may be better to apply the operator that returns "fewer" tuples first.

Property 1 $\sigma_{n1}\sigma_{n2} = \sigma_{n2}\sigma_{n1}$

□

The following example shows that *Skeptical Select* operators do not commute.

Example 8.1 $\sigma_{s1}\sigma_{s2} \neq \sigma_{s2}\sigma_{s1}$

Name	*Salary*	*Age*	*Position*
John	60K	32	Asst. Manager
John	70K	35	Manager

Let $\Theta_1 = (Age(t) > 33)$ and $\Theta_2 = (Salary(t) < 50K)$, and let the constraint set be

$\phi_1 : \{((Name(t) = Name(u))\ \&\ (t \neq u)) \ \rightarrow$ **panic**$\}$. Note that this is a "Type 2" constraint.

Then

$\sigma_{s1}\sigma_{s2} = \{\}$

$\sigma_{s2}\sigma_{s1} = \{<\text{John,70K,35}>\}$

$\diamondsuit$

The following example shows that the *Choice Select* operation does not commute either, and hence, when computing algebraic expressions involving multiple *Choice Select*s, we must perform them in the given order.

Example 8.2 $\sigma_{c1}\sigma_{c2} \neq \sigma_{c2}\sigma_{c1}$

Name	*Salary*	*Age*	*Position*
John	60K	32	Asst. Manager
John	70K	35	Manager
Peter	100K	45	Manager

Let $\Theta_1 = True$ and $\Theta_2 = True$ (That is there exists no selection criteria) Let the constraint set for σ_{c1} be

$\phi_1 : \{((Name(t) = Name(u)) \;\&\; (t \neq u)) \;\rightarrow\; \textbf{panic}\}$. Note that this is a "Type 2" constraint. And let the constraint set for σ_{c2} include another "Type 2" constraint

$\phi_2 : \{((Position(t) = \text{“}Manager\text{”} \wedge Position(u) = \text{“}Manager\text{”}) \;\&\; (t \neq u)) \;\rightarrow\; \textbf{panic}\}$.

Then, the following is a valid set of answers for the above queries:

$\sigma_{c1}\sigma_{c2} = \{<\text{John,60K,32}>, <\text{Peter,100K,45}>\}$

$\sigma_{c2}\sigma_{c1} = \{<\text{John,70K,35}>\}$

$\diamondsuit$

In situations where either there are no type 2 constraints specified, or where type 2 constraints are specified, but not violated, the following result says that the order of *Naive* and *Skeptical Select* operators may be interchanged.

Property 2 *If either there are no type 2 constraints specified, or where type 2 constraints are specified, but not violated, then* $\sigma_{n1}\sigma_{s2} = \sigma_{s2}\sigma_{n1}$.

Proof : If there are no constraint violations in the *Skeptical Select* then both are equivalent to cascaded *Naive Select* operations. We have already proven that *Naive Select* operators commute. Furthermore, if there are only "Type 1" violations, then the violating tuples will be eliminated in the $\sigma_{n1}\sigma_{s2}$ computation as well as in the $\sigma_{s2}\sigma_{n1}$ computation, and hence, the computations are equivalent. $\square$

The preceding proposition applies when type 2 violations are not encountered. The following result says that even if such violations do occur, the result of performing a *Skeptical Select* first and then a *Naive Select* is a subset of the the result of performing a *Naive Select* first and then a *Skeptical Select*.

Property 3 $\sigma_{n1}\sigma_{s2} \subseteq \sigma_{s2}\sigma_{n1}$.

Proof : If we apply σ_{n1} first, and if (t_1, t_2) is pair of tuples involved in a constraint violation, then it is possible that only one of this pair (say t_1) survives, and that this tuple goes into the answer set. However, if we apply σ_{s2} first, the violation will be observed by the *Skeptical Select* operator and both t_1, t_2 will be omitted from the answer set. Therefore the above property holds.

$\square$

In situations where either there are no type 2 constraints specified, or where type 2 constraints are specified, but not violated, the following result says that the order of *Naive* and *Choice Select* operators may be interchanged.

Property 4 *If either there are no type 2 constraints specified, or where type 2 constraints are specified, but not violated, then* $\sigma_{n1}\sigma_{c2} = \sigma_{c2}\sigma_{n1}$.

Proof : Similar to the proof of $\sigma_{n1}\sigma_{s2} = \sigma_{s2}\sigma_{n1}$.

$\square$

The preceding proposition applies when type 2 violations are not encountered. The following result says that even if such violations do occur, the result of performing a *Naive Select* first and then a *Choice Select* is a subset of the the result of performing a *Choice Select* first and then a *Naive Select*.

Property 5 $\sigma_{n1}\sigma_{c2} \subseteq \sigma_{c2}\sigma_{n1}$.

Proof : In this case, since, $LMCAS^*$ is non-monotonic for *Choice Select*, it is not possible to satisfy the equality. Using properties 3a and 3b of $LMCAS$, we have the following: if we apply *Choice Select* first and *Naive Select* next, then the semantics S will be equal to a subset of the $LMCAS^*$ of the *Choice Select*. On the other hand, if we apply *Naive Select* first then some of the tuples may get removed from the relation, hence due to 3a and 3b, the new semantics S' will be a superset (not necessarily proper) of S.

□

In situations where either there are no type 2 constraints specified, or where type 2 constraints are specified, but not violated, the following result says that the result of performing a *Skeptical Select* first and then a *Choice Select* is a subset of the the result of performing a *Choice Select* first and then a *Skeptical Select*.

Property 6 *If either there are no type 2 constraints specified, or where type 2 constraints are specified, but not violated, then* $\sigma_{c1}\sigma_{s2} \subseteq \sigma_{s2}\sigma_{c1}$.

Proof : If there are no "Type 2" violations, the expression reduces to $\sigma_{n1}\sigma_{s2} \subseteq \sigma_{s2}\sigma_{n1}$ which is already proven. In the preceding proposition, if type 2 violations exist, then *Choice Select* and *Skeptical Select* may not commute as shown in the following example, i.e. it is possible that $\sigma_{c1}\sigma_{s2} \neq \sigma_{s2}\sigma_{c1}$.

Name	*Salary*	*Age*	*Position*
John	60K	32	Asst. Manager
John	70K	35	Manager
Peter	100K	45	Manager

Let $\Theta_1 = True$ and $\Theta_2 = True$, i.e. there are no selection criteria. Let the constraint set for σ_{c1} be
$\phi_1 : \{((Name(t) = Name(u))\ \&\ (t \neq u)) \rightarrow \textbf{panic}\}$. Note that this is a "Type 2" constraint.
And let the constraint set for σ_{s2} include another "Type 2" constraint
$\phi_2 : \{((Position(t) = \text{“}Manager\text{”} \wedge Position(u) = \text{“}Manager\text{”})\ \&\ (t \neq u)) \rightarrow \textbf{panic}\}$.
Then, the following answer sets are possible:
$\sigma_{c1}\sigma_{s2} = \{<\text{John,60K,32}>\}$
$\sigma_{s2}\sigma_{c1} = \{<\text{John,60K,35}>, <\text{Peter,100K,45}>\}$
or,
$\sigma_{c1}\sigma_{s2} = \{<\text{John,60K,32}>\}$
$\sigma_{s2}\sigma_{c1} = \{\}$.

□

The following (straightforward) properties says that a naive (resp. skeptical, resp. choice) join operation can be thought of as a naive (resp. skeptical, resp. choice) select applied after an ordinary join. In fact, a *Naive Join* is just an ordinary join.

Property 7 *The followings hold:*
$(a \bowtie_{n1} b) = \sigma_{n1}(a \bowtie b)$.
$(a \bowtie_{s1} b) = \sigma_{s1}(a \bowtie b)$.
$(a \bowtie_{c1} b) = \sigma_{c1}(a \bowtie b)$.

□

The following result says that in cases where it still makes sense to apply a selection criterion w.r.t. the relation resulting from a project, the project and *Naive Select* operators commute.

Property 8 *If the selection criterion* Θ *applies to the relation produced by a projection operation, then* $\underset{A_1}{\pi}\sigma_{n2} = \sigma_{n2}\underset{A_1}{\pi}$.

$t \in \underset{A_1}{\pi}\sigma_{n2} \longleftrightarrow (t = k[A_1] \wedge \Theta(k)) \longleftrightarrow t \in \sigma_{n2}\underset{A_1}{\pi}$

□

The preceding property is not true when we consider *Skeptical Select* in place of the *Naive Select* considered above. However, it is indeed the case that in cases where it still makes sense to apply a selection criterion w.r.t. the relation resulting from a project, the result of applying a *Skeptical Select* followed by a project is a subset of the result of applying a project operation first followed by the *Skeptical Select*. As a consequence, it may be beneficial, for query optimization purposes, to first perform the project, followed by the selection.

Property 9 *If the selection criterion* Θ *applies to the relation produced by a projection operation, then* $\underset{A_2}{\pi}\sigma_{s1} \subseteq \sigma_{s1}\underset{A_2}{\pi}$.

Proof : If there are no violations then the operation is equivalent to cascaded projection and *Naive Select* operations. We have already proven that *Naive Select* and projection operators commute.

In the case of a violation, on the other hand, if we perform projection first, it may be possible that the violation is prevented. Hence the result will be a superset of $\underset{A_2}{\pi}\sigma_{s1}$ which fails to prevent the violation.

□

The following result says that the same situation applies when we consider *Choice Select*s instead of *Skeptical Select*s.

Property 10 *If the selection criterion* Θ *applies to the relation produced by a projection operation, then* $\underset{A_2}{\pi}\sigma_{c1} \subseteq \sigma_{c1}\underset{A_2}{\pi}$.

Proof : If there are no "Type 2" violations then the operation is equivalent to cascaded projection and *Naive Select* operations. We have already proven that *Naive Select* and projection operators commute.
In the case of a "Type 2" violation, on the other hand, if we perform projection first, it may be possible that the violation is prevented. Hence the result will be a superset of $\underset{A_2}{\pi}\sigma_{s1}$ which fails to prevent the violation.

□

The following result says that when we wish to execute select operations on the result of *Naive Join* operations, then this is the same as first performing the *Naive Select* operations on the relations being joined, and then performing the *Naive Join*. The latter approach of doing the join on the (smaller) derived relations obtained after executing the *Naive Select* operations will lead to greater efficiency, and hence, when performing query optimizations, it is better to rewrite equations of the form shown on the left hand side into the form shown on the right hand side.

Property 11 *If* $\Theta_1(t) = \Theta_1(k) \wedge \Theta_1(l)$ *where* $t = k * l$ *and* $k \in a$ *and* $l \in b$*, then* $\sigma_{n1}(a \bowtie_{n2} b) = (\sigma_{n1}a) \bowtie_{n2} (\sigma_{n1}b)$.

Proof : $t \in \sigma_{n1}(a \bowtie_{n2} b) \longleftrightarrow (\Theta_1(t) \wedge (t = k * l) \wedge (k \in a) \wedge (l \in b))$
$\longleftrightarrow (\Theta_1(k) \wedge \Theta_1(l) \wedge$

$$(t = k * l) \wedge (k \in a) \wedge (l \in b))$$
$$\longleftrightarrow t \in (\sigma_{n1} a) \bowtie_{n2} (\sigma_{n1} b)$$

□

When we attempt to replace the naive join in the preceding result with a *Skeptical Join* instead, we observe that the result does not hold.

Example 8.3 To see this consider the following:

a

Name	*Position*
John	Asst. Manager
John	Manager

b

Position	*Salary*
Asst. Manager	70K
Manager	100K

Let $\Theta_1 = (Position(t) = \text{"}Manager\text{"})$ and let the constraint set be
$\phi_1 : \{(((Name(t) = Name(u)) \; \& \; (t \neq u)) \;\rightarrow\; \textbf{panic})\}$.
The followings are the answer sets for the corresponding queries :
$\sigma_{n1}(a \bowtie_{s2} b) = \{\}$
$(\sigma_{n1} a) \bowtie_{s2} (\sigma_{n1} b) = \{< John, Manager, 100K >\}$

◊

However, it is true that the result of performing the select operation *after* taking the *Skeptical Join* of relations a and b is a subset of the result of first performing the selections on a and b, and then taking the *Skeptical Join* of the resulting relations. This is established below.

Property 12 *If* $\Theta_1(t) = \Theta_1(k) \wedge \Theta_1(l)$ *where* $t = k * l$ *and* $k \in a$ *and* $l \in b$ *then* $\sigma_{n1}(a \bowtie_{s2} b) \subseteq (\sigma_{n1} a) \bowtie_{s2} (\sigma_{n1} b)$.

Proof : When the skeptical operators are applied first it is possible to omit tuples due to the violations; on the other hand, applying the naive operators first may remove some of the tuples involved in these violations from the database so that the result may be larger than it is in the former case. More formally :

$$\begin{aligned}
t \in \sigma_{n1}(a \bowtie_{s2} b) \longrightarrow\; & (\exists u)(\exists v)(\forall c \epsilon \phi_2)(a(u) \;\wedge\; b(v) \;\wedge\; \Theta_1(t) \;\wedge \\
& (t = u \bowtie v) \;\wedge\; \textbf{satisfy}_\mathbf{1}\; (t, c) \;\wedge \\
& \neg(\; (\exists k)(\exists y)(\exists z)(a(y) \;\wedge\; b(z) \;\wedge (k = y \bowtie z) \;\wedge \\
& (\neg \textbf{satisfy}_\mathbf{2}\; (t, k, c)) \;\wedge\; (t \neq k)\;)) \\
\longrightarrow\; & (\exists u)(\exists v)(\forall c \epsilon \phi_2)(a(u) \;\wedge\; b(v) \;\wedge\; \Theta_1(t) \;\wedge \\
& (t = u \bowtie v) \;\wedge\; \textbf{satisfy}_\mathbf{1}\; (t, c) \;\wedge \\
& \neg(\; (\exists k)(\exists y)(\exists z)(a(y) \;\wedge\; b(z) \;\wedge\; \Theta_1(k) \;\wedge (k = y \bowtie z) \;\wedge \\
& (\neg \textbf{satisfy}_\mathbf{2}\; (t, k, c)) \;\wedge\; (t \neq k)\;)) \\
\longrightarrow\; & (\exists u)(\exists v)(\forall c \epsilon \phi_2)(a(u) \;\wedge\; b(v) \;\wedge\; \Theta_1(u) \;\wedge\; \Theta_1(v) \;\wedge \\
& (t = u \bowtie v) \;\wedge\; \textbf{satisfy}_\mathbf{1}\; (t, c) \;\wedge \\
& \neg(\; (\exists k)(\exists y)(\exists z)(a(y) \;\wedge\; b(z) \;\wedge\; \Theta_1(y) \;\wedge\; \Theta_1(z) \;\wedge (k = y \bowtie z) \;\wedge \\
& (\neg \textbf{satisfy}_\mathbf{2}\; (t, k, c)) \;\wedge\; (t \neq k)\;)) \\
\longrightarrow\; & t \in (\sigma_{n1} a) \bowtie_{s2} (\sigma_{n1} b)
\end{aligned}$$

□

A result analogous to the preceding holds when we consider *Choice Join* instead of *Skeptical Join*.

Property 13 *If* $\Theta_1(t) = \Theta_1(k) \wedge \Theta_1(l)$ *where* $t = k * l$ *and* $k \in a$ *and* $l \in b$ *then* $\sigma_{n1}(a \bowtie_{c2} b) \subseteq (\sigma_{n1}a) \bowtie_{c2} (\sigma_{n1}b)$.

Proof : With the assumption that the selection criteria can also be applied to the initial relations, we have the following:

$$
\begin{aligned}
t \in \sigma_{n1}(a \bowtie_{c2} b) \longrightarrow\ & (\exists u)(\exists v)(\forall c \epsilon \phi_2)(\ a(u) \wedge b(v) \wedge \\
& \quad \Theta_1(t) \wedge (t = u \bowtie v) \wedge \\
& \quad (\mathbf{chosen}\ (t,c) \vee \\
& \quad \neg(\exists j)(\exists k)(\exists l)(a(k) \wedge b(l) \wedge \\
& \qquad (j = k \bowtie l) \wedge \\
& \qquad \wedge (t \neq j) \wedge \neg \mathbf{satisfy_2}\ (t,j,c)))) \\
\longrightarrow\ & (\exists u)(\exists v)(\forall c \epsilon \phi_2)(\ a(u) \wedge b(v) \wedge \\
& \quad \Theta_1(u) \wedge \Theta_1(v) \wedge (t = u \bowtie v) \wedge \\
& \quad (\mathbf{chosen}\ (t,c) \vee \\
& \quad \neg(\exists j)(\exists k)(\exists l)(a(k) \wedge b(l) \wedge \\
& \qquad (j = k \bowtie l) \wedge \\
& \qquad \wedge (t \neq j) \wedge \neg \mathbf{satisfy_2}\ (t,j,c)))) \\
\longrightarrow\ & (\exists u)(\exists v)(\forall c \epsilon \phi_2)(\ a(u) \wedge b(v) \wedge \\
& \quad \Theta_1(u) \wedge \Theta_1(v) \wedge (t = u \bowtie v) \wedge \\
& \quad (\mathbf{chosen}\ (t,c) \vee \\
& \quad \neg(\exists j)(\exists k)(\exists l)(a(k) \wedge b(l) \wedge \\
& \qquad \Theta_1(k) \wedge \Theta_1(l) \wedge (j = k \bowtie l) \wedge \\
& \qquad \wedge (t \neq j) \wedge \neg \mathbf{satisfy_2}\ (t,j,c)))) \\
\longrightarrow\ & (\sigma_{n1}a) \bowtie_{c2} (\sigma_{n1}b)
\end{aligned}
$$

□

For the sake of completeness, we remark that *Naive Joins* (which are really the same as standard joins) have the associativity property.

Property 14 $a \bowtie_{n1} (b \bowtie_{n2} c) = (a \bowtie_{n1} b) \bowtie_{n2} c$

□

However the following examples shows that when one of the cascaded join operators is *Skeptical* or *Choice*, it is not possible to alter the oder of execution.

Example 8.4 To see that it may be the case that $a \bowtie_{n1} (b \bowtie_{s2} c) \neq (a \bowtie_{n1} b) \bowtie_{s2} c$ consider the following:

a

Salary
60K

b

Name	*Salary*
John	60K
John	70K

c

Name	*Age*
John	32

Let the constraint set for $\bowtie_{s2}$ (i.e. ϕ_2) be

$\phi 2 : \{((Name(t) = Name(u)) \;\&\; (t \neq u)) \;\rightarrow\; \textbf{panic}\}$.

Then,

$a \bowtie_{n1} (b \bowtie_{s2} c) = \{\}$

$(a \bowtie_{n1} b) \bowtie_{s2} c = \{<\text{John,60K,32} >\}$

$\diamondsuit$

Example 8.5 This example shows that it may be the case that $a \bowtie_{n1} (b \bowtie_{c2} c) \neq (a \bowtie_{n1} b) \bowtie_{c2} c$. To see this, reconsider the preceding example with the following change :

a	*Salary*
	60K
	70K

Then some possible answers to the queries are:

$a \bowtie_{n1} (b \bowtie_{c2} c) = \{\}$

$(a \bowtie_{n1} b) \bowtie_{c2} c = \{<\text{John,60K,32} >\}$

or

$a \bowtie_{n1} (b \bowtie_{c2} c) = \{<\text{John,70K,32} >\}$

$(a \bowtie_{n1} b) \bowtie_{c2} c = \{<\text{John,60K,32} >\}$

It is easy to see that equality cannot be guaranteed.

$\diamondsuit$

Example 8.6 This example shows that it may be the case that $a \bowtie_{c1} (b \bowtie_{s2} c) \neq (a \bowtie_{c1} b) \bowtie_{s2} c$. Example 8.4 works for this case too. The following is an answer to the queries:

$a \bowtie_{c1} (b \bowtie_{s2} c) = \{\}$

$(a \bowtie_{c1} b) \bowtie_{s2} c = \{<\text{John,60K,32} >\}$

$\diamondsuit$

Example 8.7 This example shows that it may be the case that $a \bowtie_{c1} (b \bowtie_{c2} c) \neq (a \bowtie_{c1} b) \bowtie_{c2} c$. The counterexample of example 8.5 is applicable to this case. The following is a possible answer to the queries:

$a \bowtie_{c1} (b \bowtie_{c2} c) = \{<\text{John,70K,32} >\}$

$(a \bowtie_{c1} b) \bowtie_{c2} c = \{<\text{John,60K,32} >\}$

$\diamondsuit$

Example 8.8 To see that it may be the case that $a \bowtie_{s1} (b \bowtie_{s2} c) \neq (a \bowtie_{s1} b) \bowtie_{s2} c$, the case of Example 8.4 may be reconsidered. The following is the answer to the queries:

$a \bowtie_{s1} (b \bowtie_{s2} c) = \{\}$

$(a \bowtie_{s1} b) \bowtie_{s2} c = \{<\text{John,60K,32} >\}$

$\diamond$

As *Naive Join* is identical to standard join, we state, for the sake of completeness that *Naive Join* is distributive. The impact of this result is that if the relation c is small, and $|(a \bowtie_{n2} c)| << |a|$ and/or $|(b \bowtie_{n2} c)| << |b|$ then it is better to compute $(a \bowtie_{n1} b) \bowtie_{n2} c$ by distributing c over the inner join, and computing the equivalent expression $(a \bowtie_{n2} c) \bowtie_{n1} (b \bowtie_{n2} c)$ instead.

Property 15 $(a \bowtie_{n1} b) \bowtie_{n2} c \ = \ (a \bowtie_{n2} c) \bowtie_{n1} (b \bowtie_{n2} c)$

$\square$

A similar property holds for the distribution of the *Skeptical Join* over *Naive Join.*

Property 16 *Suppose the join criteria between c and* $(a \bowtie b)$ *can also be applied to the join operations* $(a \bowtie c)$ *and* $(b \bowtie c)$ *and suppose there are no inter-tuple constraints which uses attributes from both a and b. Then:* $(a \bowtie_{n1} b) \bowtie_{s2} c \ = (a \bowtie_{s2} c) \bowtie_{n1} (b \bowtie_{s2} c)$

Proof :
$$
\begin{aligned}
(a \bowtie_{n1} b) \bowtie_{s2} c &= \sigma_{s2}((a \bowtie_{n1} b) \bowtie c) \\
&= \sigma_{s2}(a \bowtie_{n1} b \bowtie c) \\
&= \sigma_{s2}\sigma_{n1}(a \bowtie b \bowtie c)
\end{aligned}
$$

We have already proven that $\sigma_{s2}\sigma_{n1} \subseteq \sigma_{n1}\sigma_{s2}$, but here we are facing a special case of it. Now, the select operations do not have any selection criterion, i.e. they are used only in enforcing the integrity constraints on the results of the joins. A *Naive Select* with no selection criterion is nothing but a *NULL* operation. Hence we have:

$$
\begin{aligned}
&= \sigma_{s2}((a \bowtie c) \bowtie (b \bowtie c)) \\
&= (a \bowtie_{s2} c) \bowtie (b \bowtie_{s2} c) \\
&= \sigma_{n1}((a \bowtie_{s2} c) \bowtie (b \bowtie_{s2} c)) \\
&= (a \bowtie_{s2} c) \bowtie_{n1} (b \bowtie_{s2} c)
\end{aligned}
$$

We here assume that all the "Type 2" violations can be detected just by performing the *Skeptical Join* operations with the initial relations (i.e. a and b). This means that, there are no inter-tuple constraints which uses attributes from both a and b, hence performing the join operation $(a \bowtie b)$ adds no extra "Type 2" violations.

$\square$

The following result shows that the above proposition holds if we replace *Skeptical Joins* by *Choice Joins*.

Property 17 *With the same assumptions as in the preceding proposition:* $(a \bowtie_{n1} b) \bowtie_{c2} c \ = (a \bowtie_{c2} c) \bowtie_{n1} (b \bowtie_{c2} c)$

$$
\begin{aligned}
(a \bowtie_{n1} b) \bowtie_{c2} c &= \sigma_{c2}((a \bowtie_{n1} b) \bowtie c) \\
&= \sigma_{c2}(a \bowtie_{n1} b \bowtie c) \\
&= \sigma_{c2}\sigma_{n1}(a \bowtie b \bowtie c) \\
&= \sigma_{c2}((a \bowtie c) \bowtie (b \bowtie c)) \\
&= (a \bowtie_{c2} c) \bowtie (b \bowtie_{c2} c) \\
&= \sigma_{n1}((a \bowtie_{c2} c) \bowtie (b \bowtie_{c2} c)) \\
&= (a \bowtie_{c2} c) \bowtie_{n1} (b \bowtie_{c2} c)
\end{aligned}
$$

$\square$

The following result is from standard database theory and is stated here for the sake of completeness.

Property 18 $\sigma_{n1}(a \bigcup b) = \sigma_{n1}a \bigcup \sigma_{n1}b$

□

However, when we consider *Choice Select* instead of *Naive Select* in the above relationship, neither inclusion holds, and the only statement that can be made is the following.

Property 19 $|\sigma_{c1}(a \bigcup b)| \leq |\sigma_{c1}a \bigcup \sigma_{c1}b|$

Proof : All "Type 2" violations in a and in b are also contained in $a \bigcup b$. In addition to these, $a \bigcup b$ may contain additional "Type 2" violations. Hence, the cardinality of the left-hand side of the above expression is smaller than the cardinality of the right-hand side.

□

In the case of *Skeptical Select*s though, it turns out that the result of applying the *Skeptical Select* operator to $a \bigcup b$ is a subset of the result of first applying the *Skeptical Select* operator to a, then applying it to b, and the taking the union of the two results.

Property 20 $\sigma_{s1}(a \bigcup b) \subseteq \sigma_{s1}a \bigcup \sigma_{s1}b$

Proof : All constraint violations in a and in b are also contained in $a \bigcup b$; in addition to these, $a \bigcup b$ may contain extra violations. Hence it is true that the cardinality of the left-hand side is smaller than the cardinality of the right-hand side. Furthermore since there exists only one *LMCAS* semantics for *Skeptical Select*, it is also true that the subset property holds in this case.

The following result is a standard result from database theory, stated for the sake of completeness.

Property 21 $\sigma_{n1}(a \bigcap b) = \sigma_{n1}a \bigcap \sigma_{n1}b$

□

However, the above equality does not hold in the case of *Choice Select*s as shown in the following example.

Example 8.9 To see that it may be the case that $\sigma_{c1}a \bigcap \sigma_{c1}b \neq \sigma_{c1}(a \bigcap b)$ consider the following relations:

	Name	*Position*
a	John	Asst. Manager
	John	Manager

	Name	*Position*
b	John	Asst. Manager
	John	Manager

Let $\Theta_1 = True$ (i.e. there exists no selection criterion), and let the constraint set be
$\phi_1 : \{((Name(t) = Name(u))\ \&\ (t \neq u)) \rightarrow \textbf{panic}\}$.
Some possible answer sets are:
$\sigma_{c1}a \bigcap \sigma_{c1}b = \{\}$
$\sigma_{c1}(a \bigcap b) = \{<\text{John,Manager}>\}$
or
$\sigma_{c1}a \bigcap \sigma_{c1}b = \{<\text{John,Asst. Manager}>\}$
$\sigma_{c1}(a \bigcap b) = \{<\text{John,Manager}>\}$

$\Diamond$

In the case of *Skeptical Select*, the equality does not hold; however, the intersection of applying *Skeptical Selects* on a and b independently is a subset of the result of applying *Skeptical Select* to the intersection of a and b.

Property 22 $\sigma_{s1}a \bigcap \sigma_{s1}b \subseteq \sigma_{s1}(a \bigcap b)$

Proof : Let t be a tuple which satisfies the selection criteria.Then :

1. $(t \notin a)$ or $(t \notin b) \longrightarrow (t \notin \sigma_{s1}a \bigcap \sigma_{s1}b)$ and $(t \notin \sigma_{s1}(a \bigcap b))$
2. $(t \in a)$ and $(t \in b)$ and t is not involved in any constraint violation $\longrightarrow (t \in \sigma_{s1}a \bigcap \sigma_{s1}b)$ and $(t \in \sigma_{s1}(a \bigcap b))$
3. $(t \in a)$ and $(t \in b)$ and t is involved in a "Type 1" constraint violation $\longrightarrow (t \notin \sigma_{s1}a \bigcap \sigma_{s1}b)$ and $(t \notin \sigma_{s1}(a \bigcap b))$
4. $(t \in a)$ and $(t \in b)$ and t is not involved in a "Type 1" constraint violation and t is involved in "Type 2" constraint violations in both $a, b \longrightarrow (t \notin \sigma_{s1}a \bigcap \sigma_{s1}b)$ but (t may be $\in \sigma_{s1}(a \bigcap b))$
5. $(t \in a)$ and $(t \in b)$ and t is not involved in a "Type 1" constraint violation and t is involved in "Type 2" constraint violations in either a or $b \longrightarrow (t \notin \sigma_{s1}a \bigcap \sigma_{s1}b)$ but (t may be $\in \sigma_{s1}(a \bigcap b))$

Hence, $\sigma_{s1}a \bigcap \sigma_{s1}b \subseteq \sigma_{s1}(a \bigcap b)$ holds.

The following result is a standard result linking and joins and differences and is stated for the sake of completeness.

Property 23 $\sigma_{n1}(a - b) = \sigma_{n1}a \; - \; \sigma_{n1}b$

$\square$

When we attempt to replace *Naive Selects* by *Choice Selects*, then the equality does not hold as is seen from the following example.

Example 8.10 To see that it may be the case that $\sigma_{c1}a \; - \; \sigma_{c1}b \neq \sigma_{c1}(a - b)$ consider:

a

Name	*Position*
John	Asst. Manager
John	Manager

b

Name	*Position*
John	Manager
James	Manager

Let $\Theta_1 = True$ (i.e. there are no selection criteria), and let the constraint set be

$\phi_1 : \{(((Name(t) = Name(u)) \;\&\; (t \neq u)) \;\rightarrow\; \textbf{panic}), (((Position(t) = Position(u)) \;\&\; (t \neq u)) \;\rightarrow \textbf{panic})\}$.

The following is a possible answer set :

$\sigma_{c1}a \; - \; \sigma_{c1}b = \{<\text{John,Manager}>\}$

$\sigma_{c1}(a - b) = \{<\text{John,Asst. Manager}>\}$

$\diamondsuit$

In the case of *Skeptical Select*, however, the result of performing a *Skeptical Select* on the difference of a and b results in a superset of the result of independently performing the *Skeptical Select* on a and b and then subsequently taking the difference.

Property 24 $\sigma_{s1}a - \sigma_{s1}b \subseteq \sigma_{s1}(a-b)$

Proof : Let t be a tuple which satisfies the selection criteria. Then :

1. $(t \notin a) \longrightarrow (t \notin \sigma_{s1}a - \sigma_{s1}b)$ and $(t \notin \sigma_{s1}(a-b))$
2. $(t \in a)$ and $(t \notin b)$ and t is not involved in any constraint violation $\longrightarrow (t \in \sigma_{s1}a - \sigma_{s1}b)$ and $(t \in \sigma_{s1}(a-b))$
3. $(t \in a)$ and $(t \notin b)$ and t is involved in a "Type 1" constraint violation $\longrightarrow (t \notin \sigma_{s1}a - \sigma_{s1}b)$ and $(t \notin \sigma_{s1}(a-b))$
4. $(t \in a)$ and $(t \notin b)$ and t is not involved in a "Type 1" constraint violation and t is involved in "Type 2" constraint violations in $a \longrightarrow (t \notin \sigma_{s1}a - \sigma_{s1}b)$ but (t may be $\in \sigma_{s1}(a-b)$)
5. $(t \in a)$ and $(t \in b)$ and t is not involved in any constraint violation $\longrightarrow (t \notin \sigma_{s1}a - \sigma_{s1}b)$ and $(t \notin \sigma_{s1}(a-b))$
6. $(t \in a)$ and $(t \in b)$ and t is involved in a "Type 1" constraint violation $\longrightarrow (t \notin \sigma_{s1}a - \sigma_{s1}b)$ and $(t \notin \sigma_{s1}(a-b))$
7. $(t \in a)$ and $(t \in b)$ and t is not involved in a "Type 1" constraint violation and t is involved in "Type 2" constraint violations in $a \longrightarrow (t \notin \sigma_{s1}a - \sigma_{s1}b)$ but (t may be $\in \sigma_{s1}(a-b)$)
8. $(t \in a)$ and $(t \in b)$ and t is not involved in a "Type 1" constraint violation and t is involved in "Type 2" constraint violations in b but not in $a \longrightarrow (t \in \sigma_{s1}a - \sigma_{s1}b)$ and$(t \in \sigma_{s1}(a-b))$

Hence, $\sigma_{s1}a - \sigma_{s1}b \subseteq \sigma_{s1}(a-b)$ holds.

$\square$

9 Related Work

There has been a great deal of work on multidatabases. In this section, we will quickly review and relate our work with existing methods in this field.

First and foremost, our work extends work by Grant, Litwin, Roussopolous and Sellis [3] who developed a calculus and algebra based on a query language called called MSQL (due to Litwin et. al. [9]) that extends SQL and which can be used to query multiple relational databases. This approach has the advantage that it extends, in a simple and easy to use way, the existing standard database query language, viz. SQL. In the same spirit, our work too can be viewed as a query language with certain new constructs that explain what to do when inconsistencies occur when integrating information from multidatabases. These methods to deal with inconsistencies in the presence of integrity constraints distinguish our work from that presented in [3]. Our work too builds upon SQL, and thus enjoys the ease of expression present in SQL.

Litwin and Abdellatif [8] develop methods to query multidatabases – in their approach, they define a tuple-calculus based query language in which they can query multiple databases. As in the case of [3], they do not deal with violations of integrity constraints which is the main focus of our paper. Conversely, they can "group" sets of databases (e.g. a Michelin database, a Gault Milan database, and a Baedekar database may all be grouped together as "Travel" databases) which we cannot do in our current framework.

Krishnamurthy, Litwin and Kent [7] define criteria that a language that a language that integrates multiple databases must satisfy. They show how to handle schematic discrepancies. Intuitively, two databases may contain the semantic information, but this information may be represented in different ways (e.g. in one database, all the information may be contained in one relation, while in the other, the same information may be spread over two relations). They define a query language that takes such schematic mismatches into account when handling queries. In a similar vein, Spaccapietra and Parent [12] structural conflicts that arise when multiple databases contain similar, but structurally different data. They show how data across multiple databases can be "linked" or "tied together" based on common attribute values (even though the attribute names may be different across different databases). In contrast, our focus is on how to proceed when integrity constraints are violated.

A good survey of database integration schemes using the federated approach is content in Sheth and Larson [11]. It is instructive to note that this survey does not specify what to do when integrity constraints are violated.

We should mention that the idea behind `CHOICE SELECT` and `CHOICE JOIN` was motivated, in part, by work by Zaniolo [14, 10] and his colleagues on nondeterministic choices in logic programming languages. The idea in their work was that in logic database languages, one may often wish to express the fact that one out of several possible ways of satisfying an atom is nondeterministically selected. This is then used by them to define a *choice* semantics for logic programs with negation. In contrast, we are not dealing with logic programming languages in this paper (except to specify the semantics of the special operators introduced by us). Instead, we are using choice mechanisms to *resolve* conflicts that arise in querying multidatabases.

Conflict resolution has also been studied in the context of active databases and production rule systems by many researchers [15, 2, 1]. Most of these study what to do when multiple active/production rules with conflicting heads requesting that an atom be both added and deleted, fire simultaneously. In contrast, we attempt to evaluate queries and resolve conflicts in answers to queries spanning multiple relational databases.

10 Conclusions

In this paper, we have developed an extension of SQL (as well as MSQL) that allows users to query multiple databases simultaneously in the presence of integrity constraints. The key feature of our work is the operators that facilitate the handling of inconsistencies which arise as a consequence of accessing multiple databases. We propose two new kinds of operations – `SKEPTICAL` operators and `CHOICE` operators. Skeptical operators throw away all tuples associated with an inconsistency – they follow the principle *when in doubt, throw it out.* Choice operators, on the other hand, attempt to restore consistency by throwing out a few tuples. In other words, it may not be necessary to throw out all tuples associated with an inconsistency. Eliminating a few may suffice. There may be multiple ways of determining which tuples to eliminate. The choice operators would non-deterministically pick one.

Based on these new operators, we define, in addition to the extension of SQL, an algebra and a calculus called the Integrity-Constraint Based Multidatabase (ICBM) algebra and calculus. We prove that the ICBM-algebra can be embedded within the calculus. The converse, however, does not hold because the ICBM-calculus allows arbitrary existentially quantified queries that the ICBM-algebra may be unable to express.

Subsequently, we have studied various algebraic relationships that exist between different permutations of the algebraic operators. These algebraic properties are, in some cases, useful in optimizing queries.

Acknowledgements. We would like to thank John Grant for his useful comments on the paper.

References

[1] R. Agrawal, R. Cochrane and B. Lindsay. (1991) *On Maintaining Priorities in a Production Rule System*, Proc. VLDB-91, pps 479–487.

[2] S. Ceri and J. Widom. (1990) *Deriving Production Rules for Constraint Maintenance*, Proc. 16th Intl. Conf. on Very Large Data Bases, Brisbane, pps 566-577.

[3] J. Grant, W. Litwin, N. Roussopoulos and T. Sellis. (1991) *An Algebra and Calculus for Relational Multidatabase Systems*, Proc. First International Workshop on Interoperability in Multidatabase Systems, IEEE Computer Society Press (1991) 118-124.

[4] M. Garey and D.S. Johnson. (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness.*

[5] A. Gupta, Y. Sagiv, J. Ullman and J. Widom. (1994) *Constraint Checking with Partial Information*, Proc. 1994 ACM Symp. on Principles of Database Systems, pps 45–55.

[6] Y. Ioannidis and T. Sellis. (1989) *Conflict Resolution of Rules Assigning Values to Virtual Attributes*, Proc. ACM SIGMOD Symp. on Management of Data.

[7] R. Krishnamurthy, W. Litwin and W. Kent. (1991) *Language Features for Interoperability of Databases with Schematic Discrepancies*, Proc. 1991 ACM SIGMOD Conf. on Management of Data, pps 40–49.

[8] W. Litwin and A. Abdellatif. (1986) *Multidatabase Interoperability*, IEEE Computer, Dec. 1986, pps 10–18.

[9] W. Litwin, A. Abdellatif, A. Zeroual, B. Nicolas and Ph. Vigier. (1989) *MSQL: A Multidatabase Language*, Information Sciences, 49, 1989.

[10] D. Sacca and C. Zaniolo. (1990) *Stable Models and Non-Determinism in Logic Programs with Negation*, Proc. 9th ACM Symp. on Principles of Database Systems.

[11] A. Sheth and J. Larson. (1990) *Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases*, ACM Computing Surveys, 22, 3, pps 183–236.

[12] S. Spaccapietra and C. Parent. (1994) *View Integration: A Step Forward in Solving Structural Conflicts*, IEEE Trans. on Knowledge and Data Engineering, 6, 2, pps 258–274.

[13] J. D. Ullman. (1988) *Database and Knowledge-Base Systems, Vol. 1*, Computer Science Press.

[14] C. Zaniolo. (1988) *Design and Implementation of a Logic-based Language for Data-Intensive Applications*, Proc. of the International Conference on Logic Programming (eds. K. Bowen and R. Kowalski), pps 1666-1687, MIT Press.

[15] C. Zaniolo. (1993) *A Unified Semantics for Active and Deductive Databases*, in "Rules in Database Systems," (N. Paton, ed.), Springer Verlag, 1994.