Conflict Tolerant Queries in AURORA

Ling Ling Yan and M. Tamer Özsu
Laboratory for Database Systems Research
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada T6G 2H1
{ling, ozsu}@cs.ualberta.ca

Abstract

Conflict tolerant queries are a new way of dealing with instance level conflicts in data integrated from multiple sources. In contrast to the traditional approach of resolving such conflicts during schema integration using aggregation functions, we establish a query model and processing techniques to tolerate these conflicts at query time to a degree specified by the users. Resolutions are only performed to produce conflict-free results. Currently we support 3 levels of conflict tolerance: HighConfidence, RandomEvidence, and PossibleAtAll and allow user-defined functions to be used for conflict resolution. The approach reduces the overhead of conflict detection and resolution and lends itself to new query optimization techniques. Fundamentally, our approach allows users to handle conflict at a coarse granularity to achieve better query performance when conflict resolution requirements are relaxed and when data contain occasional conflicts.

1. Introduction

Vast amount of digital information is stored in a variety of data sources. With the advent of the Internet, the way people use information is changing rapidly; integrated access to heterogeneous sources is required. When integrating data from such sources, two types of conflicts may arise: *semantic conflicts*, which happen when sources model the same application differently, and *instance level conflicts*, which happen when sources record inconsistent values on the same objects. In this paper, we propose a technique for querying data in the presence of instance level conflicts. Traditionally, these conflicts are resolved at schema integration time using aggregation functions [2]. For instance, one may specify that when multiple sources record different age values for a person, the "correct" age be computed as the average of these values. Queries are written as if data are conflict-free. Conceptually, instance level conflicts are resolved *before* queries are evaluated; users have no say over resolution policies at query time. We refer to this approach as the *static resolution* approach. These resolutions are realized during materialization or query processing. If integrated data are materialized, instance level conflicts are removed before any query is processed. If data are *virtual*, enough data must be retrieved for conflict detection and resolution at query time; this may incur significant performance penalty as illustrated below:

EXAMPLE 1.1 Assume that sources A and B provide data on `Person` and conflicts on `Age` are to be resolved by taking the average of all `Age` values. Consider query:

$Q_0 =$ **select** `ID`, `Name`, `Address`
from `Person`
where `Age`>30

It is not sufficient to retrieve only persons with `Age`>30; we must retrieve all Person data from both A and B, compute all `Age` values, and evaluate the query. This cost stays the same even when no `Age` conflict actually occurs. Optimization strategies have been proposed but cases such as Q_0 are fundamentally difficult to optimize. This drawback becomes significant when more sources contribute large volumes of Person data. □

In a dynamic data integration system where large numbers of data sources come and go, materialization may not be desirable. It may also be difficult to foresee when and where instance level conflicts are likely to happen; adding a new source may give rise to new conflicts. Specifying a resolution for conflicts that do not really happen incurs unnecessary performance penalties if data are virtual. On the other hand, applications vary in requirements for conflict handling. For Q_0 in Example 1.1, the exact age of a person does not matter so long as he/she is older than 30. When multiple sources offer different `Age` values of a person, one user may consider him to be older than 30 if *some* sources say so, while another may require that *all* sources say so. Conflict resolutions on Name and Address can be performed only for persons who qualified as older than 30. Conflicts on `Person.Age` are not resolved, but rather *tolerated* by the system during query processing. We refer to this approach of instance level conflict handling *conflict tolerant (CT) querying*.

Query Evaluation \ Conflicts	Statically Resolved	Tolerated
On Materialized Data	1	3
On Virtual Data	2	4

Table 1. Querying Integrated Data

Depending on whether integrated data are materialized and how instance level conflicts are handled, we distinguish among 4 cases of querying integrated data, as shown in Table 1. Cases 1 and 2 raise no new issues in query semantics; these are well-studied domains. Case 1 requires maintenance of materialized data. Query optimization issues in case 2 has been studied. The CT query model applies to cases 3 and 4. Optimizing queries on materialized data leverages existing techniques and is not discussed.

The CT query model enables users to resolve instance level conflicts to a desired degree and let the system "tolerate" the rest; it allows flexible conflict handling and better query performance for users who do not require static resolutions. Consider the following CT query:

$Q'_0 =$ **select** `PIN`, `Name`[ANY], `Age`[ANY], `Address`[DISCARD]
from `Person`
where `Age` > 30 **with** *HighConfidence*

HighConfidence in the "with" clause specifies that if inconsistent age values exist, a person qualifies as `Age`>30 only if *all* sources say so. *After* a person qualifies, if there is conflict on `Name`, `Age`, or `Address`, the functions ANY, ANY, and DISCARD, respectively, are used to remove these conflicts to produce a conflict-free query answer. Given a set of values S, function ANY returns a random value from S, function DISCARD returns a null value if S contains more than one distinct value, otherwise it returns the only value in S. These resolutions do not affect predicate evaluation; they are only used to construct conflict-free query results. If all sources record that Fred is younger than 30, then he does not have to be retrieved even if there is conflict on his Age. The framework described in Section 6 enables such optimized processing.

The CT query model and its optimization framework are developed as part of AURORA system [8], which provides a mediation framework for scalable integration of large number of sources. Section 2 describes the relevant aspects of AURORA. Section 3 defines conflicts and resolutions in AURORA terms. Section 4 defines the CT query model. Section 5 describes primitive CT query evaluation. Section 6 describes a query optimization framework. Section 7 reviews related work. Section 8 contains conclusions and future work.

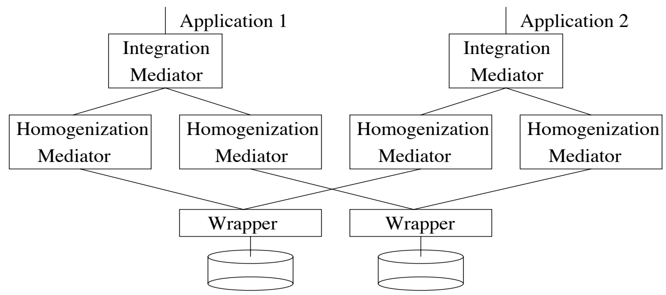

Figure 1. The AURORA Mediation Framework

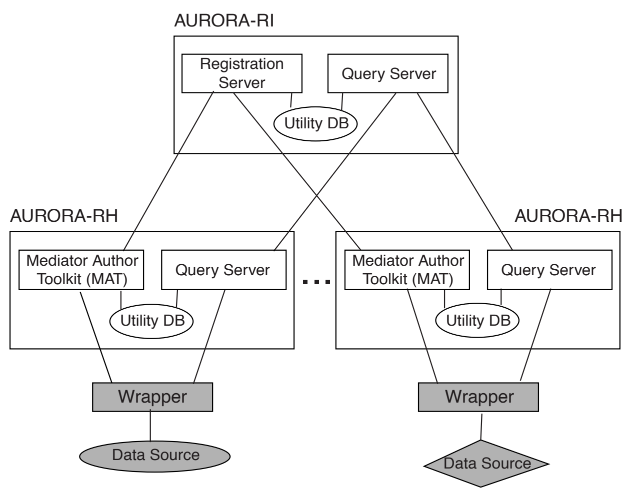

Figure 2. The AURORA Architecture

Notation. $t.A$ denotes the value of attribute A in tuple t, and $R\{A\}$ denotes all values of attribute A in relation R, Given a collection of relations, $Y = \{F_1, ..., F_m\}$, and an attribute B, $Y\{B\} = F_1\{B\} \cup ... \cup F_m\{B\}$. $ATTR(R)$ is the set of attributes of relation R, $ATTR(p)$ is the set of attributes referenced by predicate p.

2. Data Integration in AURORA

AURORA employs a two-tiered, plug-and-play mediation model depicted in Figure 1. This model is designed to facilitate dynamic and scalable data integration [8].

Sources are first *homogenized* and then *integrated*. Homogenization removes idiosyncrasies of individual sources, which is done independently and, possibly, in parallel. A homogenized source describes its content to the integration mediator to which it contributes data. The integration mediator deals with a large number of homogenized sources and is fully automatic. All mediators are data model specific. Currently the system has relational and object-oriented mediators. This paper considers only the relational ones. Figure 2 shows the current architecture of AURORA. AURORA-RH [10] is the Relational Homogenization mediator and AURORA-RI is the Relational Integration mediator. Wrappers support relational interface to sources. In the current implementation, we use OLE-DB providers supporting SQL as wrappers, such wrappers are readily available for a large variety of data sources. Mediators and wrappers cooperate via COM/DCOM.

The integration mediator, AURORA-RI, maintains a pre-defined *service view*, a usual relational schema that can be queried by applications. Sources wishing to participate in the service view S maintained by an AURORA-RI mediator M must be homogenized against S using an AURORA-RH mediator, which also communicates with M to describe the content of the source in the context of the service view. At query time, AURORA-RI merges data from relevant sources and deals with instance level conflicts using the CT query model.

2.1. Service View

For applications, the service view is a relational schema that can be queried. For sources that provide data through this view, it is a pre-defined relational schema where each relation, called a *global relation*, specifies a group of attributes as its *plug-in identifier* (PID). The PID is used by AURORA-RI for object matching, to identify tuples from different sources that describe the same object so that they can be combined to form tuples in the global relation; a source tuple must carry relevant PID in order to "fit" into the service view. Intuitively, the PID is a "ticket" to the service view. We use $PID(R)$ to denote the PID of relation R. To simplify presentation, we also assume that $PID(R)$ is a single attribute. For $t \in R$, its PID value is denoted as $t.PID$. For example, a service view may contain relation Person described below with PID(Person) = "`ID`":

`Person(ID, Name, Age, HomeNo, WorkNo, Employer, NoWorkYear, NoSchoolYear)`

2.2. Registrations, Fragments, and Match Join

A data source must register the data it provides to a target AURORA-RI mediator. A *registration* is a 3-tuple:

$$REG =< DSN, SR, GRN >$$

where DSN is the data source name, SR is a source relation schema, and GRN is a global relation name. Once this registration goes through, SR becomes a *registered fragment* of GRN. The attribute set of SR must include the PID of GRN, that is, $PID(GRN) \subseteq ATTR(SR)$. For any attribute $B \in ATTR(GRN)$, if $B \in ATTR(SR)$, we say that source relation SR *supports* B. A registered fragment of a global relation often supports some, but not all, of its attributes.

AURORA-RI uses the *Match Join* (MJ) operator to "manufacture" tuples in global relations using registered fragments based on PID values. Consider two registered fragments, $F1(P, A, B)$ and $F2(P, B, C)$, of global relation $R(P, A, B, C)$ with PID P. If $< p, a, b >\in F1$, $< p, b, c >\in F2$, then $< p, a, b, c >\in R$. If $< p, b', c >\in F2$ and $b \neq b'$, then both $< p, a, b, c >$ and $< p, a, b', c >$ are in R. MJ can be expressed using outer-joins.

DEFINITION **2.1** Let $Y = \{F_1, ..., F_M\}$ be a set of fragments with a common PID P. Let A_i be a non-PID attribute. The *value set* of A_i given Y is defined as:

$$VALset(A_i|Y) = \bigcup_{j=1}^{M_i} \pi_{P,A_i}(F_{i_j})$$

where F_{i_j}'s ($1 \leq j \leq M_i$) are all the fragments in Y supporting A_i. □

$VALset(A_i|Y)$ is a binary relation (P, A_i) containing all the A_i-values from the fragments in Y and the related PID value. These binary relations are then outer-joined to derive a global relation.

DEFINITION **2.2** Let $Y = \{F_1, ..., F_M\}$ be a set of fragments with a common PID P. Let $S = \{P, A_1, ..., A_g\}$ be a set of attributes, $\forall 1 \leq i \leq g, A_i \neq P$. The *Match Join (MJ)* of relations in Y based on P in regard to S is defined as:

$$\begin{aligned} MJ(P, S, Y) = \quad & VALset(A_1|Y) ⟗_P VALset(A_2|Y) ⟗_P \\ & ... ⟗_P VALset(A_g|Y) \end{aligned}$$

where $⟗_P$ denotes outer-equi-join on P. □

Let R be a global relation and let Y_R be the set of all fragments registered with R, $Y_R = \{F_1, ..., F_M\}$. Then relation R is derived as: $R = MJ(PID(R), ATTR(R), Y_R)$. Global relations thus computed may contain null values. For any tuple t and predicate p, we assume that $p(t)$ is true if and only if t contains no null values on all attributes referenced by p and p holds on t.

EXAMPLE 2.1 Assume that at one mediator we have defined a global relation `Person` as

`Person(ID, Name, Age, HomeNo, WorkNo, Employer, NoWorkYear, NoSchoolYear)`

with PID "`ID`". Also assume that Person has four registered fragments, as shown in Figure 3. AURORA-RI will derive Person as shown in Figure 4. The column tid is not part of the result but is used later to refer to tuples. □

Fragment 1

ID	Name	Age	HomeNo
001	Peter	20	1001
002	Mary	26	2000
003	Fred	32	3000
004	James	30	4000

Fragment 2

ID	Name	HomeNo	WorkNo
002	Mary	2000	2002
003	Fred	3003	3000
004	James	4000	4000
005	Joan	5005	5005

Fragment 3

ID	WorkNo	Employer
002	2000	Company2
003	3003	Company3
004	4000	Company4
005	5005	Company5
006	6000	Company6

Fragment 4

ID	Name	Age	NoWorkYear	NoSchoolYear
001	Peter	20	1	12
002	Mary	28	5	7
003	Fred	34	8	7
004	James	40	9	3
005	Joan	52	15	15
006	Julie	34	16	2
007	Alan	46	10	14

Figure 3. Registered Fragments for Global Relation Person

3. Instance Level Conflicts and Resolutions

In Figure 3, Fragment 1 records that Fred is 32 years old while Fragment 4 indicates that Fred's age is 34. This conflict is reflected in Figure 4 as a violation of key constraint, since there are more than one tuple with ID 003; these tuples form the *alternative tuple set* for 003, denoted as $ATset(Person, 003)$. An alternative tuple set containing more than one distinct value indicates an instance level conflict.

Formally, for global relation R and a PID value k. The alternative tuple set of R at k, $ATset(R, k)$, is defined as:

$$ATset(R, k) = \{t \mid t \in R, t.PID = k\}$$

For example, we have the following in Figure 4:

$ATset$(Person, 001) = $\{t_1\}$,
$ATset$(Person, 002) = $\{t_2, t_3, t_4, t_5\}$,
$ATset$(Person, 004) = $\{t_{14}, t_{15}\}$

If $|ATset(R, k)| \geq 2$, we say there is a *conflict* in R at k. Relations that *may* contain conflicts are called *conflict-accommodating relations*, or *CA-relations*.

Global relations derived using MJ are CA-relations; this is because we make no effort to remove any conflicts during this derivation. Conflicts are caused by inconsistencies among registered fragments and demonstrate themselves as $ATset$s with cardinalities larger than 1. $ATset$ describes conflicts at tuple level. These conflicts are caused by one or more conflicts at attribute level, For global relation R, non-PID attribute A and PID value k, we say there is an *attribute level conflict* on $R.A$ at k if $|ATset(\pi_{PID,A}(R)), k)| \geq 2$.

Even with the conflict tolerant query model, conflict resolution must still be performed, although delayed and relaxed in a controlled way. AURORA provides two operators, *Resolve Tuple level Conflict (RTC)* and *Resolve Attribute level Conflict (RAC)*, for conflict resolution at tuple and attribute levels, respectively. These operators are used for defining the CT query model later. Both operators take a *resolution* as a parameter. As defined below, a resolution is a function with an appropriate signature; it can be system-defined or provided by users.

DEFINITION **3.1** Given a global relation R and its attribute A, an *attribute conflict resolution* on $R.A$ is a function f:$setof(T) \rightarrow T$, where T is the type of $R.A$. A *tuple conflict resolution* on R is a function g such that, given a set of tuples $S = \{t_1, ..., t_n\} \subseteq R$, $t_i.PID = k$ for $1 \leq i \leq n$, $g(S)$ = t where $ATTR(t) = ATTR(R)$, $t = null$ or $t.PID = k$. □

AURORA provides common resolution functions such as SUM, AVG, MAX, MIN, ANY, DISCARD, but also allows user-defined functions. If we resolve conflicts on all attributes, we effectively have defined a tuple conflict resolution. This relationship between the two types of resolutions is captured by the concept of *equivalent tuple conflict resolution* (ETCR) given below. This concept allows us to build the CT query model only based on tuple-level conflict resolutions, although the users can still specify attribute level conflict resolutions if they wish. Traditionally, attribute level conflicts are the only type of conflicts discussed [2]; users may be more comfortable with them.

DEFINITION **3.2** Let R be a global relation and X = $\{A_1, ..., A_n\}$ be all the non-PID attributes of R over which there may be conflicts. Let $f_1, ..., f_n$ be attribute conflict resolutions on $A_1, ..., A_n$, respectively. Let S be a set of tuples of R that have the same PID value. A tuple conflict resolution of R, g, is the *Equivalent Tuple Conflict Resolution (ETCR)* of $f_1, ..., f_n$, denoted as $g = ETCR(f_1, ..., f_n)$, if for any set of R-tuples with a common PID value, S, $g(S) = t$ where t satisfies the following:

tid	ID	Name	Age	HomeNo	WorkNo	Employer	NoWorkYear	NoSchoolYear
t_1	001	Peter	20	1001	null	null	1	12
t_2	002	Mary	26	2000	2000	Company2	5	7
t_3	002	Mary	28	2000	2000	Company2	5	7
t_4	002	Mary	26	2000	2002	Company2	5	7
t_5	002	Mary	28	2000	2002	Company2	5	7
t_6	003	Fred	32	3000	3003	Company3	8	7
t_7	003	Fred	32	3000	3003	Company3	8	7
t_8	003	Fred	32	3003	3000	Company3	8	7
t_9	003	Fred	32	3003	3003	Company3	8	7
t_{10}	003	Fred	34	3000	3003	Company3	8	7
t_{11}	003	Fred	34	3000	3003	Company3	8	7
t_{12}	003	Fred	34	3003	3000	Company3	8	7
t_{13}	003	Fred	34	3003	3003	Company3	8	7
t_{14}	004	James	30	4000	4000	Company4	null	null
t_{15}	004	James	40	4000	4000	Company4	null	null
t_{16}	005	Joan	52	5000	5005	Company5	15	15
t_{17}	006	Julie	34	null	6000	Company6	16	2
t_{18}	007	Alan	46	null	null	null	10	14

Figure 4. Derived Population of Global Relation Person

1. $\forall i, 1 \leq i \leq n$, $t.A_i = f_i(S_i)$ where $S_i = \{v \mid \exists r \in S, r.A_i = v\}$; and

2. $\forall B \in ATTR(R) - X, t.B = r_1.B$, where $r_1 \in S$.

□

DEFINITION **3.3** Let R be a CA-relation and $f_1, ..., f_n$ be conflict resolutions on non-PID attributes $A_1, ..., A_n$. Operator *Resolve Attribute Conflict*, *RAC*, is defined as

$$RAC(\ R, A_1{:}f_1, ..., A_n{:}f_n) = \\ \{t' \mid \exists k, t, t \in R,\ t.PID = k,\ t'.PID = k, \\ \forall i, 1 \leq i \leq n,\ t'.A_i =, f_i(S(R, A_i, k)), \\ \forall B, B \in ATTR(R) - \{A_1, ..., A_n\}, t'.B = t.B\}$$

where $S(R, A, k) = \{a \mid\ < k, a > \in \pi_{PID,A}(R)\}$. □

DEFINITION **3.4** Let R be a CA-relation and F be a tuple conflict resolution of R. Operator *Resolve Tuple Conflict*, *RTC*, is defined as:

$$RTC(R, F) = \{t \mid \exists k, t = F(ATset(R, k))\}$$

□

Intuitively, RAC removes conflicts on attributes $A_1, ..., A_n$ of R using functions $f_1, ..., f_n$; RTC removes tuple level conflicts using function F. These operators are illustrated in Figures 5 and 6. Given C, a set of conflict resolution functions for all the non-PID attributes of R over which there may exists conflicts, we have $RTC(R, ETCR(C)) = RAC(R, C)$.

4. Conflict Tolerant Query Model

We define the semantics of single relation CT queries. A CT query over global relations $R_1, ..., R_n$ is semantically equivalent to a single relation CT query over $R_Q = R_1 \times ... \times R_n$. The PID of R_Q includes PIDs of all involved relations. Single relation CT queries are in the following form:

Q_{CT} = **select** L **from** R **where** p **with** c_1

where L is in one of the following forms:

1. $L = E_1, ..., E_m$ where $E_i = R.B_i$ $(1 \leq i \leq m)$ if $B_i = PID(R)$; $E_i = R.B_i[d_i]$ if $B_i \neq PID(R)$, d_i is an attribute conflict resolution for $R.B_i$.

2. $[D]R.B_1, ..., R.B_m$ where D is a tuple conflict resolution for $\pi_{PID(R),B_1,...,B_n}(R)$.

c_1 is called the *predicate evaluation parameter*, or *PE-parameter*, $c_1 \in \{$*HighConfidence*, *RandomEvidence*, *PossibleAtAll*$\}$; d_i's and D specify how conflicts are removed to produce a conflict-free query answer. Q_1 and Q_2 are example CT queries:

Q_1 : **select** `PIN`, `Name`[ANY], `Age`[ANY], `Address`[DISCARD]
from `Person`
where `Age` > 30
with *HighConfidence*

Q_2 : **select** [ANY] `PIN`, `Name`, `Age`, `Address`
from `Person`
where `Age` > 30
with *RandomEvidence*

ID	Name	Age	HomeNo	WorkNo	Employer	NoWorkYear	NoSchoolYear
001	Peter	20	1001	null	null	1	12
002	Mary	27	2000	null	Company2	5	7
003	Fred	33	3000	null	Company3	8	7
004	James	35	4000	4000	Company4	null	null
005	Joan	52	5000	5005	Company5	15	15
006	Julie	34	null	6000	Company6	16	2
007	Alan	46	null	null	null	10	14

Figure 5. RAC**(Person, Age:AVG, HomeNo:ANY, WorkNo:DISCARD)**

ID	Name	Age	HomeNo	WorkNo	Employer	NoWorkYear	NoSchoolYear
001	Peter	20	1001	null	null	1	12
002	Mary	26	2000	2002	Company2	5	7
003	Fred	34	3000	3003	Company3	8	7
004	James	30	4000	4000	Company4	null	null
005	Joan	52	5000	5005	Company5	15	15
006	Julie	34	null	6000	Company6	16	2
007	Alan	46	null	null	null	10	14

Figure 6. RTC**(Person, ANY)**

Both queries retrieve `PIN`, `Name`, `Age` and `Address` of persons older than 30. When there is conflict on `Age`, Q_1 selects persons for whom all `Age` values available are >30, while Q_2 randomly sample one `Age` value and if it is >30, then the person is selected. After a person qualifies as >30, there may still be conflicts on `Name`, `Age` or `Address`; these conflicts are resolved using the resolutions specified in the selection clause. Q_1 resolves conflicts on attribute level while Q_2 does it on tuple level. We support a few default forms of L. $L = A_1, ..., A_n$, where A_is are attributes, is the same as $L =$ [ANY]$A_1, ..., A_n$. If at least one attribute resolution is specified in L, the default resolution for all other non-PID attributes with no specified resolution is ANY. Fundamentally, no matter which form L takes, it specifies a tuple conflict resolution, $DE(L)$, referred to as the *data extraction parameter*, the *DE-parameter*. If L is in form 2, $DE(L) = D$. A form 1 select clause can be rewritten into form 2 with $D = ETCR(d_1, ..., d_m)$. We only consider form 2 select clause in the rest of the presentation.

Semantics of Q_{CT} is defined in two steps. First, we define how to find all the PID values that identify objects in R that satisfy the query predicate p, this set is called the *contributing PID set (CSET)*. Since R is a CA-relation, a given PID value, k, may identify a set of tuples, namely $ATset(R, k)$. Whether k identifies an object that satisfies p must be determined by properties of $ATset(R, k)$ against p given PE-parameter c_1. Second, we have to remove any conflicts on data related to PIDs in the CSET computed earlier according to the DE-parameter of Q_{CT}. Semantics of Q_{CT} is formally defined below.

DEFINITION **4.1** Given a CA-relation R, a predicate p and a PE-parameter c_1, the *contributing PID set* of R in regard to p under c_1, $CSET(R, p, c_1)$, is defined as follows:

1. For any $k \in R\{PID\}$ such that $|ATset(R, k)| = 1$, $k \in CSET(R, p, c_1)$ if and only if $p(t) = true$, where $t \in ATset(R, k)$.

2. For any $k \in R\{PID\}$ such that $|ATset(R, k)| \geq 2$:
   - If $c_1 = RandomEvidence$, $k \in CSET(R, p, c_1)$ if and only if $p(t) = true$, where $t \in ATset(R, k)$ is selected by a function at query evaluation time.
   - If $c_1 = PossibleAtAll$, $k \in CSET(R, p, c_1)$ if and only if $\exists t \in ATset(R, k), p(t) = true$.
   - If $c_1 = HighConfidence$, $k \in CSET(R, p, c_1)$ if and only if $\forall t \in ATset(R, k), p(t) = true$.

□

A $CSET$ contains PIDs identifying tuples that satisfy a predicate under a given PE-parameter; these tuples will contribute to the query result. When the PE-parameter is *RandomEvidence*, the value of $CSET$ depends on the run-time function used to choose a tuple from an $ATset$ based on which the query predicate is evaluated. Thus more than one *CSET* can be considered valid. Such variations are captured by the following definition.

DEFINITION **4.2** Let R be a CA-relation, p a predicate and c a PE-parameter. A set of PID values C is a *valid CSET* of R in regard to p under c_1 if:

- $c \neq RandomEvidence$ and $C = CSET(R, p, c)$; or
- $c = RandomEvidence$ and $\forall k \in C$, such that $k \notin CSET(R, p, HighConfidence)$, there exist tuples $t_1, t_2 \in R$, such that $t_1.PID = t_2.PID = k$, $p(t_1) = false, p(t_2) = true$.

□

EXAMPLE 4.1 Examine relation Person in Figure 4, we see the following:

$CSET$(Person, "Age>33", *PossibleAtAll*) $= \{003, 004, 005, 006, 007\}$
$CSET$(Person, "Age>33", *HighConfidence*) $= \{005, 006, 007\}$
$CSET$(Person, "Age>33", *RandomEvidence*) $= \{004, 005, 006, 007\}$
$CSET$(Person, "Age>33", *RandomEvidence*) $= \{003, 004, 005, 006, 007\}$

The last two $CSET$s given above are both valid. 003 does not satisfy `Age`>33 under $HighConfidence$ because there is evidence that 003 is 32 years old. □

DEFINITION **4.3** [Answer Set.] The answer to query Q_{CT} given earlier is defined as:

$$A = \pi_{B_1,\dots,B_m}[RTC(R \ltimes_{PID} CSET(R, p, c_1),\ DE(L))]$$

□

Table 2 shows 12 CT queries and results. These queries vary in PE-parameter and DE-parameter. We use two DE-parameters: ANY and DISCARD but they can be any function defined by the system or user. We vary the select clause to demonstrate how CT query model tolerates conflicts. Results of queries involving $RandomEvidence$ or ANY may vary with the selection function used at run-time. By specifying these parameters, one accepts such variations.

EXAMPLE 4.2 First examine Q_1-Q_6 shown in the left columns of Table 2. The most stringent control appears in Q_2. This query has one of the smallest results. We next observe that queries in the 3rd column often have larger results than those in the 1st column. For example, Q_4' and Q_4. This is because relation Person contains no conflicts over `Name` but it contains conflicts over `Age`. When a query retrieves only conflict-free attributes, conflicts on other attributes are often hidden from the users altogether; the system does not resolve conflicts on them either. □

5. Primitive CT Query Evaluation

Algorithm *CT-QP-NoOpt* is an unoptimized algorithm that directly implements the CT query semantics given earlier. Correctness of this algorithm is straightforward.

ALGORITHM *CT-QP-NoOpt* $(R, Q, F_1, \dots, F_n)$
input:

R: Global relation involved in the query.

Q: Q = **select** L **from** R **where** p **with** c_1.

F_i: All the fragments registered with R, $F_1, \dots, F_n$.

output: A: the query answer.
begin

1. $R = \pi_{L_1} MJ(PID(R), ATTR(R), F_1, \dots, F_n)$, where $L_1 = \{PID(R)\} \cup ATTR(p) \cup ATTR(L)$.
2. $C = ComputeCSET(R, p, c_1)$.
3. $A = \pi_{ATTR(L)}[RTC(\pi_{ATTR(L)\cup\{PID\}}(R \ltimes_{PID} C), DE(L))]$;

end of algorithm.

In step 1, *CT-QP-NoOpt* retrieves all fragments and performs a match join. This can be expensive when fragments are large and numerous. When query selectivity is low, large portion of the retrieved data is discarded in step 2 where $CSET$ is computed with the algorithm given below; it is desirable to not retrieve these data in step 1. In step 3, we apply operator RTC to produce a conflict-free result. RTC is a direct implementation of Definition 3.4 and is not given here.

ALGORITHM *ComputeCSET*(R, p, c_1)
input:

R: A CA-relation, sorted on $PID(R)$.

p: A predicate.

c_1: A PE parameter.

output: C: $CSET(R, p, c_1)$.
begin

1. Let $R' = \sigma_p(R)$.
2. If $c_1 = RandomEvidence$ or $c_1 = PossibleAtAll$, then $C = \pi_{PID}(R')$.
3. If $c_1 = HighConfidence$, then $C = \pi_{PID}(R') - \pi_{PID}(R - R')$.

end of algorithm.

In the next Section, we establish techniques to use query predicate p to derive conditions based on which the amount of fragment data retrieved in step 1 can be reduced without impacting on the correctness of the query result. This technique will reduce both communication cost and the volume of data manipulated.

6. Optimizing CT Query Processing

For predicate p over a global relation R and a fragment of R, F, if $ATTR(p) \subseteq ATTR(F)$, we say p is *applicable* to F. CT query optimization aims at using applicable predicates to reduce the volume of fragment data retrieved into the mediator while preserving query semantics.

Query			Answer	Query			Answer
Q_1 :	**select**	[ANY] Name, Age	$< Fred, 32 >$	Q'_1 :	**select**	[ANY] Name	$< Fred >$
	from	Person	$< James, 40 >$		**from**	Person	$< James >$
	where	$27 < Age < 45$	$< Julie, 34 >$		**where**	$27 < Age < 45$	$< Julie >$
	with	$HighConfidence$			**with**	$HighConfidence$	
Q_2 :	**select**	[DISCARD] Name, Age		Q'_2 :	**select**	[DISCARD] Name	$< Fred >$
	from	Person			**from**	Person	$< James >$
	where	$27 < Age < 45$	$< Julie, 34 >$		**where**	$27 < Age < 45$	$< Julie >$
	with	$HighConfidence$			**with**	$HighConfidence$	
Q_3 :	**select**	[ANY] Name, Age	$< Mary, 28 >$	Q'_3 :	**select**	[ANY] Name	$< Mary >$
	from	Person	$< Fred, 32 >$		**from**	Person	$< Fred >$
	where	$27 < Age < 45$	$< James, 40 >$		**where**	$27 < Age < 45$	$< James >$
	with	$PossibleAtAll$	$< Julie, 34 >$		**with**	$PossibleAtAll$	$< Julie >$
Q_4 :	**select**	[DISCARD] Name, Age		Q'_4 :	**select**	[DISCARD] Name	$< Mary >$
	from	Person			**from**	Person	$< Fred >$
	where	$27 < Age < 45$			**where**	$27 < Age < 45$	$< James >$
	with	$PossibleAtAll$	$< Julie, 34 >$		**with**	$PossibleAtAll$	$< Julie >$
Q_5 :	**select**	[ANY] Name, Age		Q'_5 :	**select**	[DISCARD] Name	
	from	Person	$< Fred, 34 >$		**from**	Person	$< Fred >$
	where	$27 < Age < 45$	$< James, 40 >$		**where**	$27 < Age < 45$	$< James >$
	with	$RandomEvidence$	$< Julie, 34 >$		**with**	$RandomEvidence$	$< Julie >$
Q_6 :	**select**	[DISCARD] Name, Age		Q'_6 :	**select**	[DISCARD] Name	$< Mary >$
	from	Person			**from**	Person	$< Fred >$
	where	$27 < Age < 45$			**where**	$27 < Age < 45$	$< James >$
	with	$RandomEvidence$	$< Julie, 34 >$		**with**	$RandomEvidence$	$< Julie >$

Table 2. Example Queries and Answers

6.1. CT Query Optimization Examples

Let p be a predicate over R and let $p = p_1 \wedge ... \wedge p_m$ be its conjunctive normal form. Given a registered fragment of R, F, the question is: *"if $p_x (1 \leq x \leq m)$ is applicable to F, can we retrieve only $\sigma_{p_x} F$ into the mediator and still evaluate $CSET(R, p, c)$ correctly?"*

In order to decide whether we can push a predicate onto a fragment, we have to consider the impact of such reductions on the query semantics. Consider the fragments shown in Figure 3 and C = $CSET$(Person, "Age > 33", c). Assume we retrieve only $\sigma_{Age>33}$ (Fragment 1) and $\sigma_{Age>33}$ (Fragment 4) into the mediator. t_{fred} = (003, "Fred", 32, 3000) in Fragment 1 will not be retrieved. This potentially excludes 003 from C. If c = *RandomEvidence*, it is valid to exclude 003 from C, according to Definition 4.2. If c = *HighConfidence* then it is necessary to exclude 003 from C. However, the mediator will retrieve t_{fred2} = (003, "Fred", 34, 8, 7) from Fragment 4 and algorithm *ComputeCSET* would include 003 in C, resulting in an incorrect CSET. To fix this problem, we can send 003 to the site of Fragment 1 to verify that Fred indeed has `Age`> 33. In our example, the verification fails and 003 is removed from C. This process is referred to as *PID verification*. Obviously, when Age is supported by only one fragment, PID verification is not needed. Assume we have derived a temporary $CSET$ value C' from reduced fragments. To perform PID verification, we send the following queries to the sites of Fragment 1 and 4, respectively:

$\delta_1 = C' \cap \pi_{PID}\sigma_{Age \leq 33}$(Fragment 1)

$\delta_4 = C' \cap \pi_{PID}\sigma_{Age \leq 33}$(Fragment 4)

PID values in δ_1 or δ_4 must be removed from C'. The cost of this approach is low when (1) query selectivity is low resulting in a small C'; and (2) Conflict rate is low resulting in small δs. When no conflict exists, all δs are empty. When C' is large, the cost of PID verification may offset the savings achieved by pushing selections onto fragments; a cost model is needed for strategy selection.

If c = *PossibleAtAll*, C can be computed by $ComputeCSET$ correctly from reduced fragments. However, we must be careful about pushing predicates that involve more than one attribute. Consider C_1 = *CSET(Person, "HomeNo=WorkNo", PossibleAtAll)*. In Figure 3, Fragment 2 contains tuple (003, Fred, 3003, 3000). If we retrieve only $\sigma_{HomeNo=WorkNo}$ (Fragment 2), 003 will be excluded from C_1, which is incorrect since combining Fragments 1 and 2, it is possible that Fred's HomeNo and WorkNo are the same, 3000. Generally, we can push a multi-attribute predicate p onto a fragment F only if no fragments other than F support any of the attributes involved in p.

CT query optimization possibilities as illustrated by the example above are summarized in Table 3. In the next section, we formally establish the above described optimization strategies. When c = *HighConfidence*, a cost model is needed to determine whether the strategies we devise actually reduce cost. This is a future research issue; we only establish the validity of the strategies in this paper.

	Can p_x be used for fragment reduction?
$c = RandomEvidence$	YES
$c = PossibleAtAll$	YES (Conditional)
$c = HighConfidence$	YES (with PID verification)

Table 3. Fragment Reduction with Selections

6.2. A Theory for CT query Optimization

The main theorems of our theory are Theorems 6.1 and 6.2, which allow us to push selections across MJ onto fragments to various degrees according to the PE-parameter.

THEOREM 6.1 *Let R be a CA-relation. Let* $p = p_1 \wedge p_2 \wedge ... \wedge p_x$ *be a predicate over* R *in conjunctive normal form. Let* $F_1, ..., F_n$ *be all fragments registered with* R *that contain no null values. Let* $p^i = p^i_1 \wedge ... \wedge p^i_{s_i}$, *where* $p^i_j \in \{p_1, ..., p_x\}, 1 \leq j \leq s_i$ *is applicable to* F_i. *Let* $F'_i = \sigma_{p^i}(F_i)$, $1 \leq i \leq n$. *Let* $T_i = \sigma_{\neg p^i}(F_i)$, $1 \leq i \leq n$. *Let* $W = T_1\{PID\} \cup ... \cup T_n\{PID\}$. *Let* $R' = MJ(PID(R), ATTR(R), F'_1, ..., F'_n)$. *Then we have the following:*

1. *CSET(R', p, RandomEvidence) is a valid value for CSET(R, p, RandomEvidence);*
2. *CSET(R, p, HighConfidence) = CSET(R', p, HighConfidence) − W.*

Note that 2 of Theorem 6.1 says that $CSET(R, p, HighConfidence)$ can be computed from reduced fragments but we must verify that PID values thus selected are not in any T_x. This process is the PID verification as described earlier.

THEOREM 6.2 *Let* R *be a CA-relation. Let* $p = p_1 \wedge p_2 \wedge ... \wedge p_x$ *be a predicate over* R *in conjunctive normal form. Let* $F_1, ..., F_n$ *be all fragments registered with* R. F_i*'s do not contain null values. Then*

$$CSET(R, p, PossibleAtAll) = CSET(MJ(R, F'_1, ..., F'_n), p, PossibleAtAll)$$

where $\forall i, 1 \leq i \leq n$, $F'_i = \sigma_{p^i}(F_i)$, $p^i = p^i_1 \wedge ... \wedge p^i_{s_i}$, $p^i_j \in \{p_1, ..., p_x\}, 1 \leq j \leq s_i$; p^i *satisfies the following:*

1. $ATTR(p^i) = \{PID\}$ *or* $ATTR(p^i) = ATTR(p) \cap ATTR(F_i)$*; and*
2. $p^i_j (1 \leq j \leq s_i)$ *involves at most one non-PID attribute or no registered fragment of R other than* F_i *supports any of the non-PID attributes in* $ATTR(p^i_j)$.

We omit the formal proof of these theorems due to limit in space. Interested readers can find these proofs in [9].

6.3. Optimized CT Query Evaluation

The following algorithm is directly based on Theorems 6.1 and 6.2.

<u>**ALGORITHM** *Optimized-CT-QP*</u> $(R, Q, F_1, ..., F_n)$
input:

R: Global relation R involved in the query.

Q: Q_{CT} = **select** L **from** R **where** p **with** c_1.

F_i: All the fragments registered with R.

output: A: the query answer.
begin
<u>**Compute CSET:**</u>

- Let $L_1 = ATTR(L) \cup \{PID(R)\} \cup ATTR(p)$. Write p into conjunctive normal form $p = p_1 \cap ... \cap p_x$. Let $X_p = \{p_1, ..., p_x\}$.
- For $i = 1, n$ do:
  - if $c_1 \neq PossibleAtAll$ then let p^i be the conjunction of all predicates in X_p that are applicable to F_i. If no such p^i is found, $p^i = true$;
  - if c_1 = *PossibleAtAll* then let p^i be the conjunction of all predicates in X_p such that (1) it involves at most one non-PID attribute; or (2) No fragments other than F_i supports any of the non-PID attributes involved. If *ATTR*(p^i) $\neq$ *ATTR(p)* $\cap$ *ATTR*(F_i), $p^i = true$.

  <u>**S1**</u> $F'_i = \pi_{L_1 \cap ATTR(F_i)} \sigma_{p^i}(F_i)$.
- R' = *MJ(PID(R)*, $L_1, F'_1, ..., F'_n$*)*;
- *C = ComputeCSET*(R', p, c_1*)*;

<u>**PID Verification:**</u>

- If c_1 = *HighConfidence* or *DE(L)* $\neq$ *ANY* then
  - Let L_2 = *ATTR(L)* $\cup$ *{PID(R)}*.
  - For $i = 1, n$ do:

    <u>**S2**</u> Let $\delta_i = \pi_{L_2 \cap ATTR(F_i)} \sigma_{\neg p^i}(F_i \bowtie_{PID(R)} C)$;
    - if c_1 = *HighConfidence*

      C = C $-\delta_i$*(PID(R))*; $\delta_i = \emptyset$;
- R' = $R' \bowtie_{PID(R)} C$;

<u>**Data Completion:**</u>

- if DE(L) $\neq$ ANY then R' = MJ(PID(R), L_2, R', $\delta_1, ..., \delta_n$);

<u>**Data Extraction:**</u>

- $A = \pi_{ATTR(L)}$*[RTC(* $R' \bowtie_{PID(R)}$ *C, DE(L))].*

$F_1' = \pi_{ID,Name,Age,HomeNo}\sigma_{Age>33}(F_1)$				$F_2' = \pi_{ID,Name,HomeNo,WorkNo}\sigma_{HomeNo=WorkNo}(F_2)$			
ID	Name	Age	HomeNo	ID	Name	HomeNo	WorkNo
				004	James	4000	4000
				005	Joan	5005	5005

$F_3' = \pi_{ID,WorkNo}(F_3)$		$F_4' = \pi_{ID,Name,Age,NoWorkYear,NoSchoolYear}$ $\sigma_{Age>33 \wedge NoWorkYear \geq NoSchoolYear}(F_4)$				
ID	WorkNo	ID	Name	Age	NoWorkYear	NoSchoolYear
002	2000	003	Fred	34	8	7
003	3003	004	James	40	9	3
004	4000	005	Joan	52	15	15
005	5005	006	Julie	34	16	2
006	6000					

$R' = MJ(PID, L, F_1', F_2', F_3', F_4')$						
ID	Name	Age	HomeNo	WorkNo	NoWorkYear	NoSchoolYear
002	null	null	null	2000	null	null
003	Fred	34	null	3003	8	7
004	James	40	4000	4000	9	3
005	Joan	52	5005	5005	15	15
006	Julie	34	null	6000	16	2

Figure 7. Compute CSET and content of R' when c_1 = RandomEvidence or HighConfidence

end of algorithm.

Steps S1 in Compute CSET and S2 in PID Verification are where queries are sent to the data sources that provide the respective fragments. These steps follow directly from Theorems 6.1 and 6.2. When the number of sources involved is large and data volume is large, cutting down on data retrieval at S1 and S2 improves query performance. We further observe the following:

Optimized-CT-QP is a 1- or 2-phase algorithm. The first phase retrieves enough data to compute $CSET$. Depending on the PE- and the DE-parameter, a second phase retrieves extra data for PID verification and/or data completion. PID verification is only needed if the PE-parameter is *HighConfidence*. Data completion is not needed when the DE-parameter is ANY.

Performance perspectives of Optimized-CT-QP. Step S1 is obviously a good move towards saving communication cost. At step S2, we could send the content of the computed $CSET$, C, to relevant data sources. This works well when C is small due to a low query selectivity, but may get expensive when C is large. A simple computation can be applied to restrict this cost. Consider performing step S2 against a data source supporting a fragment F_i. The purpose of sending a query to compute W_i is indeed to retrieve data related to PIDs in F_i that are in C but have not been retrieved in step S1. Thus we can compare the volume of $\pi_{L_2 \cap ATTR(F_i)}\sigma_{\neg p^i}(F_i)$ with the volume of C. If the former is smaller, then we simply retrieve it without sending C to the relevant data source, and proceed normally.

Optimized-CT-QP performs better when conflict rate is low. When conflict rate is low, the δ_i's will be empty or very small. This means the cost of PID verification and Data Completion becomes low. We expect *Optimize-CT-QP* to be most efficient against low conflict data.

EXAMPLE 6.1 We use the above given algorithms to evaluate the following 6 queries:

select $[d]$ `Name, Age`
from `Person`
where `Age` > 33 **and** `HomeNo = WorkNo`
and `NoWorkYear` $\geq$ `NoSchoolYear`
with c_1

where $c_1 \in$ {RandomEvidence, HighConfidence, PossibleAtAll} and $d \in$ {ANY, DISCARD}. Figure 7 shows the predicates pushed onto F_i's to compute F_i''s ($i = 1...4$) in the case of *HighConfidence* and *RandomEvidence*. It also shows the result of the match join producing R', from which we get: $CSET$*(Person, p, RandomEvidence)* = $\{004, 005\}$. Based on this result we perform PID verification. The $\delta_i' s$ are computed when c_1 = *HighConfidence* or d = DISCARD, shown in Figure 9. Based on this result we have: *CSET(Person, p, HighConfidence)* = $\{005\}$ Figure 8 shows the predicates pushed onto F_i's to compute of F_i''s, $i = 1...4$ in the case of c_1 = PossibleAtAll. It also shows the result of the match join producing R', from which we get: *CSET(Person, p, PossibleAtAll)* = $\{003, 004, 005\}$ Final results of the 6 queries are given in Figure 10. □

$F'_1 = \pi_{ID,Name,Age,HomeNo}(F_1)$				$F'_2 = \pi_{ID,HomeNo,WorkNo}(F_2)$			
ID	Name	Age	HomeNo	ID	Name	HomeNo	WorkNo
001	Peter	20	1001	002	Mary	2000	2002
002	Mary	26	2000	003	Fred	3003	3000
003	Fred	32	3000	004	James	4000	4000
004	James	30	4000	005	Joan	5005	5005

$F'_3 = \pi_{ID,WorkNo}(F_3)$		$F'_4 = \pi_{ID,Name,Age,NoWorkYear,NoSchoolYear}$ $\sigma_{Age>33 \wedge NoWorkYear \geq NoSchoolYear}(F_4)$				
ID	WorkNo	ID	Name	Age	NoWorkYear	NoSchoolYear
002	2000	003	Fred	34	8	7
003	3003	004	James	40	9	3
004	4000	005	Joan	52	15	15
005	5005	006	Julie	34	16	2
006	6000					

$R' = MJ(PID, L, F'_1, F'_2, F'_3, F'_4)$						
ID	Name	Age	HomeNo	WorkNo	NoWorkYear	NoSchoolYear
001	Peter	20	null	null	null	null
002	Mary	26	2000	2002	null	null
002	Mary	26	2000	2000	null	null
003	Fred	32	3000	3000	8	7
003	Fred	32	3000	3003	8	7
003	Fred	32	3003	3000	8	7
003	Fred	32	3003	3003	8	7
003	Fred	34	3000	3000	8	7
003	Fred	34	3000	3003	8	7
003	Fred	34	3003	3000	8	7
003	Fred	34	3003	3003	8	7
004	James	40	4000	4000	9	3
004	James	30	4000	4000	9	3
005	Joan	52	5005	5005	15	15
006	Julie	34	null	6000	16	2

Figure 8. Compute CSET Phase for PossibleAtAll

7. Related Work

Projects that employ a similar style data integration model include DISCO [7] and Information Manifold (IM) [4]. AURORA differs from these systems in its 2-tiered mediation model, designed to make adding and removing data sources easier. Unlike previous systems, we do not force the adoption of a specific data model; above the wrapper level, relational and object-oriented mediators can be developed independently. A comparison between AURORA and other mediation models can be found in [8, 9]. Many integration systems with comparable model do not deal with instance level conflicts.

[1, 2] study algebraic rules for pushing selections across aggregation functions, under the assumption that schema integration is performed by an integration specification which resolves all potential instance level conflicts using various aggregation functions. AURORA integration mediators do not keep integration specifications, sources participate in the data service by registering with the mediator the data they can contribute. Conflicts are not resolved at schema integration time but rather tolerated at query time and resolved only upon returning of query results. In general, AURORA's approach towards instance level conflict handling offers a new way of querying potentially inconsistent data and new techniques for processing such queries efficiently.

The flexible relation model [3, 6] is designed to deal with instance level conflicts but it requires the applications to use a non-standard data model for data access. This approach only deals with conflicts at predicate evaluation time and the tolerance mode is always *HighConfidence*. Conflicts in query results are not removed. Multiplex [5] deals with instance level conflicts in the context of answering queries using given materialized views. Conflicts arise when the materialized views overlap and the same query can be evaluated in multiple ways resulting in multiple answers.

c_1 = HighConfidence	c_1 = RandomEvidence	c_1 = PossibleAtAll
$\delta_1 = < 004, \text{James}, 30 >$	$\delta_1 = < 004, \text{James}, 30 >$	$\delta_1 = \emptyset$
$\delta_2 = \emptyset$	$\delta_2 = \emptyset$	$\delta_2 = \emptyset$
$\delta_3 = \emptyset$	$\delta_3 = \emptyset$	$\delta_3 = \emptyset$
$\delta_4 = \emptyset$	$\delta_4 = \emptyset$	$\delta_4 = \emptyset$

Figure 9. Value of δ_i's with d = DISCARD

	c_1 = HighConfidence	c_1 = RandomEvidence	c_1 = PossibleAtAll
d = ANY	A = $\{< 005, \text{Joan}, 52 >\}$	A = $\{< 004, \text{James}, 40 >,$ $< 005, \text{Joan}, 52 >\}$	A = $\{< 003, \text{Fred}, 34 >,$ $< 004, \text{James}, 40 >,$ $< 005, \text{Joan}, 52 >\}$
d = DISCARD	A = $\{< 005, \text{Joan}, 52 >\}$	A = $\{< 005, \text{Joan}, 52 >\}$	A = $\{< 005, \text{Joan}, 52 >\}$

Figure 10. Query Results

A mechanism is proposed to derive an *approximate query answer* using these candidate answers. However, without any object matching assumption, it is not clear how conflicts can be detected.

8. Conclusion and Future Work

In this paper, we described AURORA's approach to instance level conflict handling. This approach differs from previous approaches in that we do not resolve conflicts at schema integration time with aggregation functions, we define a new model for querying possibly inconsistent data, the CT query model. With this model, conflicts are tolerated to a degree specified by the user at query time. The advantage of the CT query approach is that applications gain more control of the quality of the data access service they receive and the mediator gain more room for query optimization; we have developed techniques for optimized processing of such queries. We believe that the ability of optimizing query processing according to applications' requirements is a significant factor in deployment.

Future research involves development of a cost model for strategy selection and a detailed performance study of the query optimization techniques presented here. Since query processing is a multi-phase procedure, apart from the major transformations given in this paper, many smaller techniques for smart reuse of data retrieved in previous phases can be explored. These are engineering issues but may improve performance further.

References

[1] A. Chen. Outerjoin optimization in multidatabase systems. *in Proc. 2nd International Symposium on Distributed and Parallel Database Systems*, pages 211–218, 1990.

[2] U. Dayal. Query processing in a multidatabase system. *in Query Processing in Database Systems, W. Kim et.al Eds*, pages 81–108, 1985.

[3] L. Demichiel. Performing operations over mismatched domains. *ICDE*, 1989.

[4] A. Levy, A. Rajaraman, and J. Ordille. Querying heterogeneous information sources using source descriptions. *VLDB*, 1996.

[5] A. Motro. Multiplex: A formal model for multidatabases and its implementation. *Technical Report ISSE-TR-95-103, Department of Information and Software System Engineering, George Mason University*, 1995.

[6] S.Agarwal, A. Keller, G. Wiederhold, and K. Saraswat. Flexible relation: an approach for integrating data from multiple, possibly inconsistent databases. *ICDE*, pages 495–504, 1995.

[7] A. Tomasic, L. Raschid, and P. Valduriez. Scaling heterogeneous databases and the design of disco. *in Proceedings of the International Conference on Distributed Computer Systems*, 1996.

[8] L. L. Yan. Towards efficient and scalable mediation: the aurora approach. *in Proc. IBM CASCON Confernece*, pages 15–29, 1997.

[9] L. L. Yan. Building scalable and efficient mediation: the aurora approach. *PhD Thesis in preparation*, 1999.

[10] L. L. Yan, T. Ozsu, and L. Liu. Accessing heterogeneous data through homogenization and integration mediators. *in Proc. 2nd IFCIS Conference on Cooperative Information Systems (CoopIS-97)*, pages 130–139, 1997.