

Integrity Constraints in Multidatabase Systems

Randy Ho Deirdre Kong May Wong
{hopa, kong, wongmay}@cs.ucdavis.edu

1 Introduction

Database systems (DBSs) do not have a uniform design paradigm. It is in part due to the fact that different organizations emphasize on different requirements and specifications during the design phase of the DBS. These traditional DBSs function individually on their own set of operational rules. During the last two decades, the functionality and complexity of computer applications, which utilize the existing database systems, have grown tremendously. At the same time, heavy deployment of the Internet has provided a better networking infrastructure and a more mature distributed computing environment. As a result, information sharing across dissimilar database systems becomes a highly desired goal [ERS99].

A multidatabase system (MDBS) is one of the approaches to provide information sharing across different databases. MDBS rests upon the idea of integrating existing local schemata of the component databases into a global conceptual schema that provides transparent access of users. The global conceptual schema serves as the entry point to the system and the users are unaware of the existence of different local databases. Autonomy and heterogeneity are the two most important characteristics of MDBS. Due to autonomy and heterogeneity of the component databases, the specification, maintenance, and use of integrity constraints in query processing become a challenging issue.

The objective of this paper is to report the study of integrity constraints in MDBS, with a particular focus on the issues and approaches in specifying integrity constraints, maintaining integrity constraints, and using integrity constraints in databases operations. We have also studied the interactions between the existing integrity constraints at the local level and derived global integrity constraints. The rest of the paper is structured as follows. Section 2 will present the overview of MDBS. Section 3 discusses the general idea of integrity constraints. Section 4 describes issues of semantic heterogeneity in the context of integrity constraints. Section 5 contains a discussion of integrity constraints during schema integration. Section 6 considers the role of integrity constraints during global query processing in MDBS. We present an approach to model global and local integrity constraints in Section 7. Section 8 will summarize and conclude what has been done in this paper.

2 Overview of MDBS

One of the approaches to support information sharing among multiple database systems is a multidatabase system (MDBS). In the current literatures, there is not a precise and standard definition of a MDBS. It is mainly due to the lack of a completely standardized framework in this area. Moreover, different perspectives are placed on MDBSs that implement various features in different contexts. In general, a MDBS refers to a facility that

supports the interoperation of multiple autonomous database systems [ERS99, BGMS92], where each local DBS operates under separate and independent control. Interoperation must allow user access, which can be initiated in any local DBS of the MDBS, to data and functionalities located in the remote database systems.

2.1 Characteristics and Features of MDBS

The ultimate goal of a MDBS is to provide transparent access to all the DBSs while each local DBS needs to maintain the highest degree of autonomy. Autonomy is of the biggest concern in MDBS design. Other approaches to support interoperation, such as federated database systems (FDBS), sacrifices the autonomy of local DBSs in order for users to have more control over the sharable information. Figure 1 shows the metrics that describe the characteristics of implementation alternatives [ERS99]. In this paper, we will focus on the autonomy and heterogeneity aspects of MDBS.

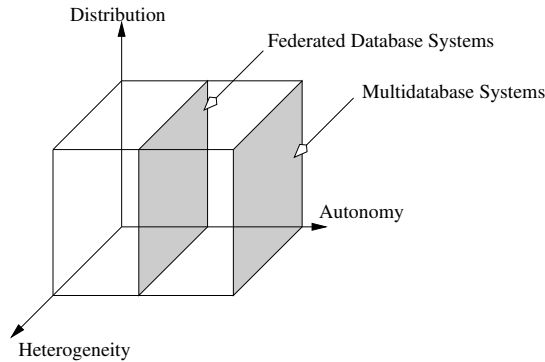


Figure 1: Implementation alternatives of distributed database systems.

As shown in figure 1, a MDBS is located at the farthest plane of the x-axis while FDBS has a lower degree of autonomy. Different aspects of autonomy are classified into four categories in [SL90]: design autonomy, communication autonomy, execution autonomy, and association autonomy. Design autonomy is the choice of design and implementation details such as data model, query language, and integrity constraints. Communication autonomy controls when and how to respond to requests from other databases. Execution autonomy refers to the power to constitute the execution of local transaction independently. Association autonomy allows the local database to decide the amount of information to be shared by other DBSs.

Different degree of heterogeneity contributes to where a particular MDBS resides in the plane. Heterogeneity refers to any dissimilarity at all levels of multiple database systems. Heterogeneity can be classified into two main categories in the context of interoperation: schematic and semantic. Schematic heterogeneity is present when the structure of the data model or the formats of data are different. It mostly occurs at the schema level of a DBS. For instance, a local DBS might specify an ADDRESS as a complex attribute; whereas in the remote DBS, ADDRESS is structured as a relation. Semantic heterogeneity refers to the different interpretation of data. The same piece of data can have different meanings in dissimilar environments. For instance, the computer science department database maintains the profiles of each student including an attribute of GPA that records the major grade-point-average of a student. The campus database also maintains the GPA of a student but is referred to as the overall grade-point-average of a student. The attribute GPA carries two different meanings in different contexts.

2.2 Architecture of MDBS

As described in [Bob96], there are many different architectures of MDBS. One of the most commonly used paradigms is based on the ANSI/SPARC data model, which is shown in figure 2. The architecture provides an abstraction of how a MDBS should be built. The corresponding layering view is also shown in the figure.

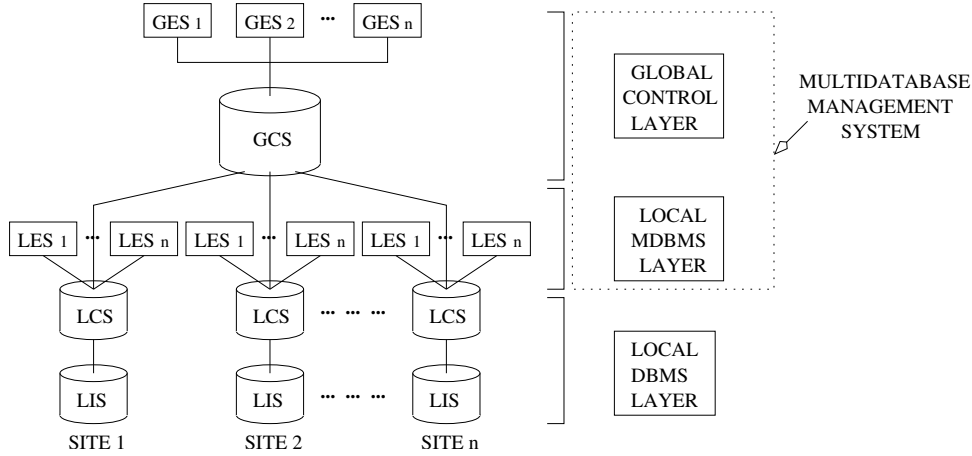


Figure 2: ANSI/SPARC data model for MDBS.

The ANSI/SPARC multidatabase architecture could be viewed as a bottom-up design approach. The local internal schema (LIS), local conceptual schema (LCS), and local external schema (LES) are the pre-existing abstraction at each of the local DBSs. Given the existing LIS, LCS, and LES, the goal of a MDBS architecture is to build a global conceptual schema (GCS) to bridge the LCS and the global external schema (GES). GES is the only external views of a MDBS, which also serves as the entry point to the entire system. GCS acts as the middleware between GES and LCS, which provides transparent and uniform access from the GES to the LCS. In this architecture, the only physical connections are GES-to-GCS and GCS-to-LCS. All other components are logically connected. When designing the GCS to integrate the LCS, autonomy of each local system needs to be preserved. It implies that the implementation of GCS should not change any schematic or semantic information that is associated with the existing LES. If such information is altered, the existing LES might not function properly after the integration.

2.3 Design Issues of MDBS

Due to the different design of the DBSs and the lack of support of MDBS in the commercial database packages, the design and implementation of MDBS becomes a challenging issue. Different applications have been developed specifically for the local DBSs that make use of the existing environment. MDBS needs to provide uniform and transparent access to the resources in multiple databases and maintain the existing integrity and investments of the local database systems.

2.3.1 Schema Translation and Schema Integration

Local DBSs have different LCS that describe the structure of the databases. The MDBS has to provide an accurate mapping of the data models between the global and local levels. Schematic heterogeneity is the major factor that needs to be considered during schema translation. A common approach is to translate the local schemata into

a common data model [BE96]. An object-oriented model has become the favored model for schema translation. The major advantage of an object-oriented model is its flexibility and ability to represent schematically different classes of objects. For instance, a virtual class can be created to bridge two heterogeneous schemata that contain similar information.

After the schemata have been translated to a common data model, the next step is schema integration. Due to the schematic and semantic heterogeneity, integrating the local external schemata to generate the global conceptual schema becomes a very challenging issue. During schema integration, one needs to carefully consider the local integrity constraints. There are many different types of integrity constraints such as domain constraint, primary key constraint, and referential constraint [BE96]. The different types of integrity constraints will be discussed in later sections of this paper. These integrity constraints might also conflict with each other. For instance, consider a local DB has a semantic integrity constraint of $SALARY > 1500$ and a remote DB has a semantic integrity constraint of $SALARY > 2000$. How should one integrate the schema containing the two integrity constraints and to generate a global integrity constraint? Does the data in a local DB violate the global integrity constraint? As a result of the different local integrity constraints and their conflicts, there is no complete automation in schema integration. The degree of which one should relax or integrate the integrity constraints totally depends on the context and environment.

2.3.2 Integrity Constraints in MDBS

In the process of integrating LCS to generate a GCS, integrity constraints have to be carefully taken into consideration. Integrity constraints are one of the major principles to protect the quality of data. In the context of multidatabases, integrity constraints can roughly be classified into two categories: local integrity constraints and global integrity constraints. Local integrity constraints are specified at the existing component databases. From the local integrity constraints, one could specify derived local integrity constraints, which further restrict the behavior and properties of data. Global integrity constraints are generated in the process of schema integration, and are located at the GCS. They place restrictions of what users can perform against the view. Additional constraints can be added at the GCS to increase the protections for the data. All of the integrity constraints have to be enforced in order to ensure the quality of data during write-operations. These constraints, if enforced completely and successfully, can provide valuable information during query processing and updates of component databases.

In a MDBS, enforcing integrity constraints at all levels becomes a very challenging issue because of semantic and schematic heterogeneities in the system, and each component database has to maintain its highest degree of autonomy. For instance, a semantically equivalent integrity constraint can be specified differently at different local databases. One needs to develop a robust technique to enforce the integrity constraints, detect violation of constraints, and to possibly repair the data. For the constraints that are not specified with the data definition, triggers are used to capture the integrity constraints. In a MDBS, triggers are usually not used efficiently because of the high autonomy of component databases, redundant information exchange, and its overhead.

3 Overview of Integrity Constraints

3.1 Motivation of Integrity constraint

The goal of a multidatabase system is to provide users global access via a single-system interface. Moreover, the quality of the data is essential. Users would like to obtain data that can be trusted. Therefore, an important functionality in database is to maintain consistency. This functionality is generally ensured by integrity constraints. Integrity constraints are what conditions must hold for data, such as the age of an employee must be between 20 and 30. Integrity constraints are concerned with single entity, attribute, as well as attributes of several relations. When all the integrity constraints are satisfied, the database is consistent. A multidatabase system is composed by a set of heterogeneous and autonomous databases, so each of these local databases has its own local constraints. When the local schemas are integrated to a global schema, it presents a challenge.

As data is stored on several sites and different data has various constraints in different databases, the question of what roles do integrity constraints play during integration arises. Do integrity constraints play any role in determining whether two relations or attributes are related? If two attributes are related, how should the two local constraints integrate into one global constraint? Should the one with lower bound be used or should new global constraints be introduced? These integrity constraints questions are important because they allow users to obtain consistent data.

Integrity constraints also play important roles in query processing and optimization. What kind of techniques can be used to process and optimize the queries? Utilize the constraints can improve the efficiency of getting the results. Before further discussion on the role of integrity constraints in a multidatabase, a classification of different kinds of integrity constraints is needed.

3.2 Classification of Integrity Constraints

Local Constraints: They are constraints to ensure the local coherence of the entities in a databases and independently of the other databases.

Global Constraints: They are constraints on the distributed data. These constraints can be newly introduced or induced from the local database.

The followings can be either a local or a global constraint:

Implicit Constraints: They are constraints of the integral part of structures of a data model, such as embedded constant attributes where all the tuples in the relation have the same constant attribute value implicitly. Consider an EMPLOYEE database of a company. Since all employees work for the same company, one would usually not include an attribute of *company_name* for which an employee works for.

Explicit Constraints: They are specified separately from the structures on the data model. They are usually called semantic integrity constraints and can be classified as dynamic and static constraints.

Dynamic Constraints: They involve a comparison of the states of the database before and after a transaction.

Static Constraints: They concern whether a given database state is valid, independent of all other database. Static Constraints consist of tuple constraints, relation constraints, and interrelation constraints.

Tuple Constraints: These constraints refer only to individual tuples of a relation, such as attribute constraints or any predicate on the attributes of the tuple.

Attribute Constraints:

Multivaluation Differences. An attribute may only allow one kind of value in one database where a corresponding attribute may allow multivalues in another database. Furthermore, an attribute may only allow to have a certain number of values, like a min and a max value or an attribute may only allow the value to lie within a specify range. That is called cardinality constraint.

Null Differences. An attribute in one database may allow null values while the corresponding attribute in another database does not allow null values.

Default Differences. A default value defined for two corresponding attributes are different. In addition, an attribute may not have a default value but the corresponding one has.

Relation Constraints. They are constraints refer to the entire set of tuples in a relation, such as aggregate constraints.

Aggregation Constraints: They are constraints that are defined on the basis of aggregate functions, such as avg, count, and sum. A local or a global schema may have a constraint such as $\text{sum}(\text{salary}) < 10000$.

Interrelation Constraints: They are constraints that refer to relationship between different relations. The most important type is the referential integrity.

Referential Integrity Constraints: They are constraints on the relationships between primary keys and foreign keys to ensure that they are data consistency. A primary key does not necessary need to be referenced by a foreign key. However, every foreign key has to match a primary key value.

4 Detection and Reconciliation of Semantic Heterogeneity in the Context of Integrity Constraints

4.1 Approach of Maintaining Referential Integrity Constraints

In multidatabase systems, data are distributed in different databases. Objects in different databases are represented and identified differently. In order to identify and relate objects in different databases, referential integrity constraint is necessary. Referential integrity constraints are a mechanism to ensure relationships between tables in different databases are consistent.

4.1.1 Structure Model

In [Mar91], a mechanism is proposed to ensure referential integrity constraints in a multidatabase. Two additional components, metadatabase and meta-multidatabase, are necessary in his approach. A metadatabase is used to keep relation names in each site, and a meta-multidatabase stores the information on local-objects and foreign-objects relations. In the model of the proposed mechanism, there are local-objects relation and foreign objects relation. A foreign-object relation is stored under metadatabase. A surrogate object identifiers (OIDs) is added in the relational databases as well. These OIDs do not contain any information. Users can delete or create OIDs but cannot modify it. The local-object relation records local objects, which are referenced by other databases. It contains three attributes: L_OID, Key, and InRelation. L_OID refers to the unique local object identifier where the object is located in a local database, and it is referenced by other databases. Key is the primary key of a tuple. InRelation is the name of the relation that contains the tuple. Moreover, foreign-objects relation needs to be added to the proposed model. It is used to keep track of the foreign objects that are represented in other databases and referenced in the local database. It also has three attributes: F_OID, L_OID, InRelation. F_OID

(foreign object identifier) is the local object ID of a foreign object. L_OID (local object identifier) refers to the local object that references to the foreign object, and InRelation refers to the name of the relation that contains the tuple of the foreign object.

4.1.2 Rules for Referential Integrity Constraints

In order to maintain consistency in multidatabase, four cases of referential integrity should be enforced. These four cases are variable foreign-relation referential integrity, foreign keys in the local-objects relation, foreign keys in the foreign-objects relation, and referencing foreign objects in a multidatabase.

4.1.2.1 Variable Foreign-Relation Referential Integrity Variable foreign-relation referential integrity ensures that the foreign key in the foreign-object relation matches the primary key of the foreign relation, which is indicated in InRelation, in another database. For instance, in order to ensure variable foreign-relation referential integrity in figure 3, every F_OID of F_OBJECTS needs to match with the L_OID in the relation, which is specified in InRelation. In this case, In MDB1, F_OID, which has the value @1, needs to check if it exists as a L_OID value in F_OBJECT relation table in MDB2. Moreover, we also need to check if F_OID in MDB2, that has the value #1, exists as a L_OID value in the MDB1.L_OBJECT relation table. Since @1 is the value of L_OID in F_OBJECTS relation table in MDB2 and #1 exists in the L_OID in F_OBJECTS relation table in MDB1, the variable foreign-relation referential integrity is satisfied.

Metadatabase MDB1:

RELATIONS	L.OBJECTS		
Name	L_OID	Key	InRelation
.....
EMPLOYEE	#1	123-45-6789	EMPLOYEE
.....

F.OBJECTS		
F_OID	L_OID	InRelation
...
@1	#1	MDB2-L.OBJECTS
...

Database DB1:

EMPLOYEE
SSN
.....
123-45-6789
.....

Metadatabase MDB2:

RELATIONS	L.OBJECTS		
Name	L_OID	Key	InRelation
.....
EMPLOYEE	@1	Alice Brown	EMPLOYEE
.....

F.OBJECTS		
F_OID	L_OID	InRelation
...
#1	@1	MDB1-L.OBJECTS
...

Database DB2:

EMPLOYEE	
Firstname	Lastname
.....
Alice	Brown
.....

Meta-Multidatabase:

LO_RELATIONS
Name
MDB1.L.OBJECTS
MDB2.L.OBJECTS
.....

Figure 3: Metadatabase and databases.

4.1.2.2 Foreign Keys in the Local-Objects Relation This constraint ensures that each relation name in the local-object relation exists in the database and each primary key in the local-object relation exists as a primary key in the database. For instance, if EMPLOYEE is stored in InRelation of L_OBJECTS table in MDB1, then it should exist in RELATIONS in MDB1. Moreover, if EMPLOYEE in InRelation of L_OBJECTS table in MDB2, then it should exist in DB2. Since both cases are true, the first condition of the constraint is satisfied. Then, one needs to check if Key in L_OBJECTS exists in InRelation. In this case, the value 123-45-6789 in Key of L_OBJECTS table in MDB1 exists in the EMPLOYEE table in DB1. Moreover, the value Alice Brown in Key of L_OBJECTS table in MDB2 exists in the EMPLOYEE table in DB2. As a result, the second condition of the constraint is satisfied. With the satisfaction of both conditions of the constraint, the constraint is satisfied.

If a relation is deleted in the database, all local object IDs, which are associated with the same relation in the local-objects relation table, have to be deleted in order to maintain consistency in the database. Moreover, if the name of a relation is changed, all attributes referring to the same relation have to be renamed. If a tuple is deleted from the database, the object ID associated with this tuple has to delete from the local-objects relation table. Similarly, if the primary key in the database is updated, all keys in the local-objects relation that is associated with this primary key have to be updated, too.

4.1.2.3 Foreign Keys in the Foreign-Objects Relation This constraint ensures three conditions. The first condition is to ensure that the object ID of a local object in the foreign-objects relation exists in a local-objects relation in the same metadatabase. For example, in MDB1 from figure 3, the L_OID #1 in F_OBJECTS also appears in L_OID of L_OBJECTS. The second condition is to make sure that the relation name in the foreign-objects relation is the same as the name in a local-object relation in another database. The value MDB2.L_OBJECTS in InRelation of F_OBJECTS in MDB1 in figure 3 refers to the existing L_OBJECTS table in MDB2. The third condition is to ensure that each foreign object ID in the foreign-object relation appears as the local object ID of the relation table in another metadatabase that has the same name as InRelation of the same tuple. For example, F_OID of F_OBJECTS in MDB1 in figure 3 has the value @1. In the same tuple, InRelation indicates that @1 should appear in L_OBJECTS relation table in MDB2, that is true.

If the object ID is deleted in the local-objects relation, the same object ID in the foreign-objects relation will be set to null. When the object ID is updated in the local-objects relation, the same object ID in the foreign-objects relation should update as well. If the local-objects relation is deleted, all object IDs in the local-objects relation are deleted from all the foreign-objects relations of other databases of the multidatabase. If the object ID in the local-object relation is updated, the foreign-objects relations associated with this object ID will be updated. If the object ID is deleted from the local-objects relation, the same object IDs are also deleted from the foreign-objects relations.

4.1.2.4 Referencing Foreign Objects in a Multidatabase This constraint ensures that the foreign object ID in foreign-objects relation table also appears as a local object ID in foreign-objects relation table in another metadatabase. For example, @1 is F_OID of F_OBJECTS in MDB1 in figure 3. It also exists in the L_OID of F_OBJECTS relation table in MDB2. Therefore, this constraint is satisfied.

If a foreign object ID is deleted from the foreign-objects relation, the object IDs are delete or set to null in the relation. If the object ID in the foreign-objects relation is updated, the relations needs to be updated, too.

4.2 Practical Approach of Maintaining Referential Integrity

Assume FK is a foreign key in DB1 referencing a primary key PK in DB2. The value of each foreign key FK in DB1 needs to be equal to the value of a primary key PK in DB2 that the foreign key FK references. Sometimes, a relaxed form of referential integrity constraint is allowed. This relaxed form of referential integrity constraint allows FK to be NULL.

4.2.1 Action Rules

Modifications on the database for the primary key or the foreign key need some action rules in order to maintain referential integrity in the multidatabase. Two cases that the action rules need to enforce are modification, deletion, or creation on either primary key or foreign key.

1. When an entity is created in DB1, the foreign key is created as well. If the foreign key is created or its value is modified in DB1, the restrict rule has to be enforced. The restrict rule states that if the update to the foreign key causes a violation of the referential integrity, the update is not allowed. Therefore, it will result in a rollback.
2. When an entity is deleted in DB2, the primary key is deleted too. If the primary key is deleted or its value is modified in DB2, there are four different action rules.
 - (a) Restrict Rule – If the update of the primary key will violate the referential integrity, the update is not allowed.
 - (b) Cascade Rule – If the primary key in DB2 is deleted or modified, the foreign key in DB1 referencing that primary key has to be deleted or modified.
 - (c) Set NULL Rule – If the primary key in DB2 is deleted, the foreign key in DB1 referencing that primary key will be set to NULL. If the primary key in DB2 is modified, the foreign key in DB1 referencing that primary key will be set to NULL.
 - (d) Set Default Rule – If the primary key in DB2 is deleted or modified, the foreign key referencing that primary key in DB1 will be set to a designated default value.

There are two primary key restrict rules: hard restrict rule and soft restrict rule. The hard restrict rule checks the referential integrity before the deletion or modification of a primary key in DB2 if it matches any foreign key in DB1. If that is the case, the update is not allowed. On the other hand, the soft restrict rule checks the referential integrity after the update and cascaded action. If it fails, all actions will be rolled back.

4.2.2 Replication technique

A multidatabase can have no replication, primary key replication, and foreign key replication. In the model with no replication, none of the data are replicated in the database. In the model with primary key replication, all the primary keys in DB2 are replicated and stored in DB1. It shortens the time for creating or modifying a foreign key because it can check with the primary key locally instead of contacting the other database. If a new primary key is created, it needs to be propagated to the replicated database. Moreover, if a primary key is deleted or modified, it needs to be propagated to the replicated copy too. On the other hand, in the model with foreign key replication, all the foreign keys in DB1 are replicated and stored in DB2. It shortens the time for deleting or modifying the primary key because it can check its foreign key in the database at the same site instead of

contacting the other database. In addition, a deletion, modification, and creation of foreign key needs to be propagated to the replicated copy.

4.2.3 Various Practical Models

In [MT93], some techniques are discussed to maintain the referential integrity in different practical models such as queued message with or without replication, remote procedure call model, distributed transaction processing model, and local transactions dialog model.

4.2.3.1 Queued Message Model (QM) In the queued message model, each system has an input message queue and output message queue. Messages are stored-and-forwarded in the queue, and the request initiates a new transaction at the server. After the request message is sent, the response message will eventually return even if there are crashes at the client or server. Primary keys and the foreign keys can be replicated in the multidatabase. Therefore, different techniques are necessary to handle three different cases - no replication, primary key replication, and foreign key replication. Since the distributed database does not have distributed atomicity, it is difficult to check others databases after the updates as described in the soft restrict rule. Therefore, in the queued message model, hard restrict rules are used instead.

No Replication

Communication between databases is necessary when no replicated data exists in the databases. However, the referential integrity action rules might cause data inconsistencies sometimes because some distributed data might succeed in some databases while the others might fail and the locks might be dropped when transactions are committed at different databases.

For example, the user creates a new foreign key in DB1 and it refers to a primary key in DB2. Simultaneously, another user deletes the same primary key. When DB1 checks for the existence of the primary key in DB2, the primary key still exists. Similarly, when DB2 checks for the referencing foreign key of that primary key in DB1, the creating foreign key does not exist. At the end, DB1 creates a new foreign key and DB2 deletes the primary key. As a result, DB1 has a foreign key referencing a deleted primary key. This violates the referential integrity and it is known as a dangling reference. This violation occurs because the databases allow two conflicting operations to interleave with each other instead of executing sequentially. In [MT93], it suggests three possible solutions to this problem. The first possible solution is to use administrative policies and procedures to ensure that primary key deletion or modification and foreign key creation or modification cannot be processed concurrently. The second possible solution is to check the referential integrity after the execution. If the actions violate referential integrity, it will roll back the primary key and foreign key operations. However, temporary inconsistencies still occur. The last possible solution is to apply application level locking. If the user wants to delete a primary key, the database needs to mark that primary key for deletion with an application level tag. Then, it checks for the existence of referencing foreign keys. If this primary key does not have any referencing foreign keys, the user can delete the primary key. Otherwise, the database will unmark the application level tag for deletion. On the other hand, if the user wants to create a foreign key, the database needs to mark that foreign key as "pending" with an application level tag. Then, it checks for the existence of the referenced primary key. If the check succeeds, the system can remove the tag. Otherwise, it needs to delete the new foreign key. The drawback of this solution is the blocking requests from other users on the marked primary or foreign keys. To solve this problem, the system can allow requests to access the marked primary or foreign key with an indication showing this key is marked. In [MT93],

it also suggests avoiding cascade and NULL/default actions rules because if the secondary action fails, it is too late to roll back the previous actions. These actions will add a great burden on the application programmers.

Primary Key Replication

In the model with primary key replication, creation or modification of foreign keys are not required to contact with the other database for checking. It only needs to check the local copy in the local database at the same site. If the user creates a primary key, the original copy should be consistent with the local copy of the primary key. Moreover, if the user deletes or updates a primary key, the action needs to be applied to both the local copy and the original copy of the primary key.

If the user tries to delete or update the primary key using the restrict rule, the database checks for the non-existence of the referencing foreign key at DB1, and DB2 notifies DB1 about the pending action (delete/update). If no referencing foreign key exists, the pending actions are marked on the local primary key at DB1. Afterwards, DB2 must notify DB1 if the action succeeds so that the action can be executed or cancelled.

The advantages of this approach are to increase the efficiency to deal with foreign key creation or modification and to solve the concurrency control issue using the application level lock.

Foreign Key Replication

In the model with foreign key replication, deletion or modification of a primary key does not need to be propagated to the other database for checking. It only needs to check with the local copy of the foreign key at the same site. If the user creates a foreign key, the original copy should be consistent with the local copy of the foreign key. Moreover, if the user deletes or updates a foreign key, the action needs to apply to both the local and original copy of the foreign key.

Inconsistent scenarios under the queued message model with foreign key replication might occur. For instance, the user tries to delete a primary key and create a foreign key, which is referencing that primary key using the cascading rule. When the users try to create a foreign key, the pending foreign key is created in DB2 and the system checks the existence of the referenced primary key in DB2. Simultaneously, another user wants to delete the primary key, which is referenced by the currently creation of foreign key. The system will mark all the foreign keys in DB2 for deletion if those foreign keys are referencing that primary key. Afterward, it deletes all these foreign keys while the pending foreign key is still under creation status. If the deletion of these foreign keys at DB1 succeeds, it will delete the local pending copy of foreign keys at DB2 as well. Because the previous check for the creation of the foreign key at DB1 is passed, the foreign key will be created in DB1 and the pending tag will be removed from the local copy of the foreign key in DB2. Therefore, it tries to remove the pending tag for a foreign key that does not exist anymore. If this action is allowed, it violates the referential integrity. To prevent this problem, the system can save the cascaded delete action on the foreign key and use it to cancel the creation action when it arrives later. This model still has the same atomicity problem and transient inconsistency problems as in the no replication and primary key replication model.

4.2.3.2 Remote Procedure Call Model In the remote procedure call model, the client sends message to the server in order to request the execution of operations. Each request initiates a new transaction at the server. There is no commit coordination and transaction semantics maintained between client and server.

In [MT93], restrict rules are suggested to be used in this model because the locks in this model can be held throughout the actions. For any of the operations, it performs the action first. Then, it will check for the existence or non-existence of the referenced keys and the operation will be either committed or aborted. However, the problem in this model is the deadlock when a foreign key is created and the primary key referencing the foreign

key is deleted. If a foreign key is created at DB1 and a primary key referencing the foreign key is deleted in DB2, then it will check for the non-existence of a foreign key referencing the primary key in DB1 because of the deletion of primary key. However, this check is blocked by the newly created foreign key. On the other hand, a check is made in DB2 for the existence of a primary key referenced by the foreign key. Similarly, this check is blocked because of the deletion of the primary key. As a result, both operations are deadlocked. As mention in [MT93], application level timeouts are usually used to detect deadlocks. Moreover, this model also has the same atomicity problems as the queued message model and the same solution for the queued message model can be used in this model.

4.2.3.3 Distributed Transaction Processing Model In this model, clients engage in dialogs with the servers. Several operations at the server are executed as a sub-transaction. The server needs to use the distributed commit protocol such as two-phase commit with the client in order to commit the transaction.

In [MT93], it favors the use of two-phase locking for concurrency control because it can ensure atomicity and serializability. However, this model has distributed deadlock problems. Consider the case where a user creates a foreign key referencing a deleting primary key in another database. If T1 is the transaction for checking the existence of the referenced primary key and T2 is the transaction for checking the non-existence of the referencing foreign key, inconsistency occurs if these transactions are allowed to release their locks before the commit points. However, if all locks are held until the commit point, deadlock problems might occur. As a result, the lock should ensure that no other transaction can create a data item which has been accessed by other transactions.

4.2.3.4 Local Transactions Dialog Model In this model, clients engage in dialogs with the servers, too. The transactions in the servers are executed as stand-alone transactions. When a client asks a server to commit the transaction, the server attempts to commit the transaction without coordination with the client or other servers.

In [MT93], it suggests not to use cascade or set NULL/default action rules because inconsistencies might occur when the secondary actions succeed while others actions fail because of no distributed atomicity. Moreover, the model using strict two-phase locking for concurrency control can ensure distributed serializability. However, deadlocks might occur in the model as well.

5 Schema Integration and Constraints Integration

Recall that one of the goals of a multidatabases system is to provide users transparent access to heterogeneous and distributed data. Thus, it is necessary to create a single global schema that represents the content of all the local databases in a multidatabase system. The process of creating the global schema is called schema integration. In this section, we will discuss the roles that integrity constraints play during schema integration and constraint integration.

5.1 Running Example

In this section, consider the following as the running example:

LDB1: Relation Employee

SSN	Salary	Height-in-Inches
111-11-1111	3000	67
222-22-2222	4000	69
333-33-3333	5000	63

LDB2: Relation Person

Id	Age	Height-in-Centimeters
1	30	170.18
2	28	175.26
3	32	160.02

5.2 Dealing with Integrity Constraints in Determining Attribute Equivalences

One of the contributions to the area of schema integration is the Theory of Attribute Equivalence presented in [LNE89]. The basic idea of their theory is to phrase conflicts that arise in the integration in terms of equivalence of attributes, equivalence of object classes, and equivalence of relationship. The term object class refers either to an entity set (relations) or a category. Once equivalent and extensional relationships (equal, contains, contained-in, overlaps, and disjoint) have been determined, integration can be performed based on the provided strategies.

5.2.1 Attribute Constraints and Characteristics

Characteristics of attributes are needed in order to determine equivalences among attributes. These characteristics describe the attribute with respect to the relation to which it belongs. The following are characteristics (constraints) of an attribute a with respect to relation C .

1. **Uniqueness:** If attribute a is a primary key, then no two values of attribute a are the same.
2. **Cardinality:** The number of values that attribute a can have for relation C .
3. **Static Semantic Integrity Constraints:** The constraints involving the attribute a , such as attribute a can only be increased or involving more than one attributes such as referential integrity constraints. Let $SIC(a)$ be the set of semantic integrity constraints that involves attribute a .
4. **Dynamic Semantic Constraints:** Constraints describe changes to which attribute values must adhere. For example, OLD.salary must be less than or equal to NEW.salary. Let $SSC(a)$ denotes the set of state-change constraint expressions involve attribute a .
5. **Domain:** A set of values that attribute a may take. Let $D(a)$ be the domain of attribute a .

Other attribute characteristics are scale, allowed operations ($OP(a)$ denotes the set of allowable operations on the domain of attribute a), and security constraints ($SEC(a)$ be the set of security constraint expressions that involve attribute a). Attributes that have several characteristics in common are referred to as basic attribute equivalence properties.

5.2.2 Basic Attribute Equivalence Properties

Let a_i be an attribute of relation A and b_i be an attribute of relation B . Let D_i be the largest non-null subset of $DOM(a_i)$ and R_i be the largest non-null subset of $DOM(b_i)$ such that there exists a mapping $f_i: D_i \rightarrow R_i$ and its inverse, $f_i^*: R_i \rightarrow D_i$.

The function f_i is determined by the integrator and must have the following properties:

1. The function f_i is an isomorphism.
2. a_i has $OP(a_i)$ if and only if b_i has $OP(b_i)$

3. All semantic integrity constraints hold under f_i and its inverse
4. All state-change constraints hold under f_i and its inverse
5. All security constraints hold under the f_i and its inverse
6. f_i and its inverse preserve functional dependencies
7. The mapping function preserves unique identifiers

Using the basic attribute equivalence properties, the STRONG attribute equivalence is formed.

5.2.3 STRONG attribute equivalence

There are several classes of strong attribute equivalence, one is the STRONG α equivalences, and the other is the STRONG β equivalences. For each of the classes, extensional relationships are given. Some examples of STRONG equivalences types are:

STRONG α equivalences: Given an attribute a of relation A and an attribute b of relation B at some point in time, and $f: D \rightarrow R$:

1. If a and b obey the Basic Equivalence Properties, $D = \text{VALUES}(a)$, and $R = \text{VALUES}(b)$, then a STRONG α EQUAL b .
2. If a and b obey the Basic Equivalence Properties, $D = \text{VALUES}(a)$, and R is properly contained in $\text{VALUES}(b)$, then a STRONG α CONTAINS b .

STRONG β equivalences: Let a be an attribute of relation A , and b be an attribute of relation B then at all time

1. If a STRONG α EQUAL b holds, then a STRONG β EQUAL b .
2. If either a STRONG α EQUAL b , or a STRONG α CONTAINS b holds, then a STRONG β CONTAINS b .

For the entire set of STRONG α and β equivalences, see [LNE89].

5.2.4 Object Equivalence

After STRONG equivalences are defined, the concept of attribute β equivalence is used to define ρ (object) equivalence. In the paper, the authors propose that the relationship between the identifier attributes of relations constitute the basis to identify semantic equivalent relationships among relations and their extensional relationship [BE96]. Consider relation A has unique identifier k_1 and attributes a_1, a_2, \dots, a_n and relation B has unique identifier k_2 and attributes b_1, b_2, \dots, b_m . Then, the relations are integrated according to the different types of equivalence that exists between the key identifiers. For example, if k_1 STRONG β EQUAL k_2 , then relation A and B are integrated into AB' with identifier k_1 or k_2 . For other types of object equivalence, see [LNE89].

5.2.5 Strategies for Attribute Integration

After the attribute equivalences and object equivalences are determined, the integration can be performed. The authors propose four strategies for merging attributes based on equivalences.

In general, strategy 1 is used to integrate all non-disjoint attributes. When two relations are integrated, all equivalence attributes are integrated to form a new attribute c' . Strategy 2 is used to integrate only attributes that

are β Equal. Strategy 3 is used to integrate only attributes that are β equal and indicate relationships between nonintegrated similar attributes. This strategy captures containment constraints among multivalued attributes of the integrated relations. The fourth strategy is used to integrate all nondisjoint attributes and migrate values between attributes, that is, if there are two common attributes, then only one of them will be in the integrated version.

5.2.6 Application

Example 1: Consider the running example with the followings added:

A semantic integrity constraint requires that values of height-in-inches can only be increased in EMPLOYEE and so for the height-in-centimeters in PERSON.

Determine whether two relations are equivalent. Let's define f_1 between social-security-number and Id.

$$f_1: \text{DOM} = \{\text{SSN}\} \text{ and Range} = \text{DOM}(\text{Id})$$

$$f_1(111-11-1111) = 1, f_1(222-22-2222) = 2, f_1(333-33-3333) = 3$$

This illustrates SSN STRONG β EQUAL Id according to the definition. Thus, both classes can be integrated into one called E_P . Next, the integrator finds a function that maps between height-in-inches and height-in-centimeters,

$$f_2: \text{DOM} = \{\text{height-in-inches}\} \text{ and Range} = \text{DOM}(\text{height-in-centimeters});$$

$$f_2(x) = 2.54 * x$$

We can conclude that height-in-inches STRONG β EQUAL height-in-centimeters. Assume age and salary attributes are not related. By using strategy 2 to do integration, the result is:

E_P relation (integration of EMPLOYEE and PERSON) has the attributes of SSN (integration of SSN and Id), height (integration of height-in-inches and height-in centimeters), age, and salary, and all the cardinal constraints, static constraint are still valid in the E_P relation.

5.2.7 Final Thought

This integration approach depends on the set of attribute characteristics, which contain integrity constraints. Then properties are formed on top of those characteristics to determine whether two relation classes or attributes can be integrated. One question that arises is how to determine whether the set of attributes characteristics is complete. Are there any attributes constraints that are not included? Another inadequacy of this approach is that it is not reasonable to just use the primary keys to establish the relationship among relations because there are some semantics embodied in other aspects of the relations that cannot be ignored [BE96]. However, the advantage is that if an attribute is integrated, all the integrity constraints for the attribute are preserved in the global level since attribute equivalence is determined by 'semantics preserving' mappings.

5.3 Constraint Integration

A schema integrator may use different methods to determine object and attribute equivalences. Once the equivalence relations are determined, and domain conversion are done and have used Two-Dimensional Union method to integrated relations, see figure 4 and [GTE89], then one needs to consider how to determine what constraints exist on the integrated global database.

Consider the running example with the following tuples added in EMPLOYEE and PERSON respectively, (444-44-4444, 6000, 68) and (5, 25, 175.26) and assume domain conversions are done, such that SSN is converted to Id and height-in-centimeters are converted to height-in-inches. Then the two-dimensional union of the relations EMPLOYEE and PERSON, denoted by $E_P = \text{EMPLOYEE} \cup^2 \text{PERSON}$, is a relation schema on the attribute set $\{K, X, Y_1, Y_2, L\}$, where K is globally equivalent to keys (SSN and Id), X is globally equivalent to attributes, Y_i is the set of non-equivalent attributes, and L is the global locality attribute, which is the designation of the database(s) in which a copy of that tuple is stored. Here, L_3 means that the tuple occurs in both LDB1 and LDB2. Also, a locality attribute should be added in each of the relation in local database for the purpose of update operations.

E_P Relation

Id	Age	Salary	Height-in-Inches	Locality
1	30	3000	67	L3
2	28	4000	69	L3
3	32	5000	63	L3
4	∅	6000	68	L1
5	25	∅	69	L2

Figure 4: $E_P = \text{EMPLOYEE} \cup^2 \text{PERSON}$

5.3.1 Integrating Tuple Constraints

Consider that a tuple t is acceptable with respect to tuple constraint P if $P(t)$ is either “true” or “unknown”. Let t_i be a tuple in relation i and $t_i(X_j)$ be the value of attribute X_j in a tuple in relation i . Two tuples are called compatible tuples when tuple t_1 ’s key attribute value equals to that of tuple t_2 ’s, $t_1(X_1) = t_2(X_2)$ if they are not-null, and X_1 and X_2 are globally equivalent attributes. Let R_1 and R_2 be two relations. Now, for all the compatible tuples in R_1 and R_2 , if constraint P is true on t_1 , then P is true on $t = t_1 \cup^2 t_2$ in $R_1 \cup^2 R_2$ (see proof in [GTE89]). P is now a temporary global tuple constraint. However, it is not true that all tuples in the integrated relation satisfy P . For example, let R_1 has tuple constraint P and all compatible tuples in R_1 satisfy P . For the integrated relation, t_2 in PERSON has a key value of 5 and t_1 does not have it. Thus, the integrated relation takes t_2 ’s height value as the integrated value and that value may not satisfy constraint P . So next, contradicting constraints need to be found, which are violated by any global tuple. Also find all tuple constraints P and Q that are totally contradicting, that is no tuple in the global domain satisfy both constraints. Then the induced global constraints are the union of the set of local constraints after excluding the global contradicting constraints.

5.3.2 Aggregate Constraint Integration

The five types of aggregate constraints that are considered are based on the five aggregate functions *min*, *max*, *sum*, *count* and *average*. Let A be a set of attribute values, M_1 , M_2 , S_1 , and S_2 are constants, and R_1 and R_2 are relations. The result for two-dimension union is:

1. If $\max(A) \leq M_1$ on R_1 and $\max(A) \leq M_2$ on R_2 , then clearly $\max(A) \leq \max(M_1, M_2)$. Similar results hold for *min*.
2. If $\text{sum}(A) \leq S_1$ on R_1 and $\text{sum}(A) \leq S_2$ on R_2 , then $\text{sum}(A) \leq S_1 + S_2$ on $R_1 \cup^2 R_2$. Similar results hold for *count*.

3. If $T_1 \leq avg(A) \leq U_1$ on R_1 and $T_2 \leq avg(A) \leq U_2$ on R_2 , then $min\{T_1, T_2\} \leq avg(A) \leq max\{U_1, U_2\}$.

Another work that has made a contribution in aggregation constraint integration is [BT99]. In [BT99], the authors propose two integration properties of aggregation constraints. With these properties of aggregate constraints, conclusions can be made whether the aggregation constraints can be integrated or not.

5.3.2.1 Decomposability and Composability In [BT99], the authors propose two aggregate constraint properties: decomposability and composability. Decomposability of the integrity constraint is when the integrity constraint holds in the original set as well as in the subset of the original set. This property can be utilized in the decomposing schema manipulation that is the results from the selection, difference and intersection of schema operation. Composability of integrity constraint is when the integrity constraint holds in the original set as well as in the superset of the original set. This property is utilized in the composed schema manipulation that results from the union schema operation.

5.3.2.2 Base Aggregate Constraints There are three types of aggregation constraints.

α -base aggregation constraints do not contain aggregate functions. For example, $salary < 5000$. α -base aggregation constraints are only decomposable but not composable because the decomposability and composability of these constraints depend on the relationship between attributes and constants. In the case of composing scheme manipulation, the properties of the added objects are unknown.

β -base aggregation constraints compare the result of the aggregate function with a constant value or an attribute value. For example, $sum(Salary) < 50000$ or $sum(Salary) < Budget$. β -base aggregation constraints depend on the comparison operator. Table 1 is a partial decomposability and composability of β -base aggregation constraints, where θ means comparison operator, c means constant and x and y means an attribute.

Aggregation Constraints	Decomposability	Composability
$\min(x) \theta c$	$\geq, >$	$\leq, <$
$\max(x) \theta c$	$\leq, <$	$\geq, >$
$\text{count}(x) \theta c$	$\leq, <$	$\geq, >$
$\min(x) \theta y$	$\geq, >$	–
$\max(x) \theta y$	$\leq, <$	–
$\text{count}(x) \theta y$	$\leq, <$	–

Table 1: Decomposability and composability of β -base aggregation constraints.

For example, $\min(x) > c$ is decomposable. Let $M = \{3, 4, 5, 6\}$ and $M' = \{4, 5\}$ and the β -base aggregation constraint is $\min(x) > 1$, where x can be M or M' . Obviously, $\min(M) > 1$ is true. Applying $\min(M') > 1$, it is still true. Since M' is the subset of M and the integrity constraint is valid in both sets, therefore $\min(x) > c$ is decomposable. However, the composability of those β -base aggregation constraints that compare the result of an aggregate function with an attribute value is unpredictable. For example, let $M = \{3, 4, 5, 6\}$ and the β -base aggregation constraint is $\min(x) < S$, where S has the value of 8. Obviously $\min(M) < S$ holds. Depending on what the composed schema manipulation is, if $M' = \{1, 3, 4, 5, 6\}$, then $\min(M') < S$ holds but if $M' = \{3, 4, 5, 6, 10\}$, then the constraint is invalid.

γ -base aggregation constraints compare the results of two aggregate functions. For example, $sum(Salary) < sum(Budget)$.

In [BT99], the authors also introduced the notion of behavior tendencies of aggregate functions that are used to describe an estimation of the result values of an aggregate function before and after a decomposed/composed

schema manipulation. For example, the aggregation function \min has an increasing behavior tendency for decomposition. Consider $M = \{3, 4, 5, 6\}$ and $M' = \{4, 5\}$. The following holds: $(\min(M)=3) \leq (\min(M')=4)$. This illustrates that the result of the aggregation function \min can only increase or remain equal after a decomposed schema manipulation. Table 2 summarizes the behavior tendencies for the aggregation functions in case of decomposition/composition.

Aggregate Function	Decomposition	Composition
{min avg- sum-}(x)	↑	↓
{max count avg+ sum+}	↓	↑

Table 2: Behavior tendencies of aggregate functions.

From the behavior tendencies and the comparison operations, conclusions of whether the aggregation constraints are decomposable or composable can be made. Let A and B be attributes, constants, or aggregate functions, and θ be a comparison operator. Then

$$[A \theta B] \text{ is decomposable if } \{A\uparrow, B\downarrow, \theta \in \{>, \geq\}\} \text{ or } \{B\uparrow, A\downarrow, \theta \in \{<, \leq\}\}$$

$$[A \theta B] \text{ is composable if } \{A\uparrow, B\downarrow, \theta \in \{>, \geq\}\} \text{ or } \{B\uparrow, A\downarrow, \theta \in \{<, \leq\}\}$$

From them, the properties of the γ -base aggregate constraints can be determined. Table 3 is the partial decomposability and composability of γ -based constraints.

5.3.2.3 Arithmetic Extension Four basic arithmetic operations, $+$, $-$, $*$ and $/$, are also considered as the extension of the aggregate functions. The authors restrict arithmetic extensions on constant c , attribute (x) , or aggregate functions $(f(x))$. If the extended term is a constant, then the addition and difference operations of constants do not cause any change in the relations between extended terms. If the operation is multiplication or division, then the non-constant factor of the aggregate constraint may have different signs before and following a schema manipulation. If the extended term is an attribute or an aggregate function, then the signs of both operands need to be considered. For more details on the behavior tendencies of arithmetically extended terms and the integration properties of the arithmetic extended constraints, see [BT99].

5.3.2.4 Application

Example 2: We now demonstrate how to use the proposed rules and properties in a small example.

Consider the running example and assume each relation satisfies the following constraints:

EMPLOYEE: $\max(\text{salary}) > 2000$ and $\text{count}(\text{SSN}) > \min(\text{height-in-inches})$

PERSON: $\max(\text{age}) < 65$

Suppose E_P is the composed schema manipulation of EMPLOYEE and PERSON. Therefore, any constraint that is composable can be adopted to the E_P relation. We can adopt $\max(\text{salary}) > 2000$ and $\text{count}(\text{SSN}) > \min(\text{height-in-inches})$ because they are composable, see Table 1 and 3, respectively. However, we cannot adopt $\max(\text{age}) < 65$ because it is a decomposable constraint.

Decomposability				Composability			
$f(x) \theta f'(x')$	min	max	count	$f(x) \theta f'(x')$	min	max	count
min	=	$\geq, >$	$\geq, >$	min	=	$\leq, <$	$\leq, <$
max	$\leq, <$	=	-	max	$\geq, >$	=	-
count	$\leq, <$	-	=	count	$\geq, >$	-	=

Table 3: Decomposability and composability of γ -base aggregate constraints.

5.3.3 Integrating Referential Integrity Constraints

According to [GTE89], it is straightforward to prove that if a referential integrity constraint exists in both local databases, then it should exist in the integrated databases. If it is not respected at one local database, then it is unlikely to be respected in the integrated database.

5.3.4 Update Operation

The reason to add the locality attribute to the relations is for update purposes [GETE88]. There are two situations:

1. If the user is allowed to specify the location(s) at which data items should be stored, then the values for the locality attribute are provided as an explicit part of updates.
2. If not, the storage location(s) of a data item is determined from the attribute-locality mappings derived from the global relation.

To perform the updates, the language QUELO, which is a modified and restricted form of QUEL, can be used. There is a where clause for each of the INSERT, DELETE, and REPLACE operation.

First, execute the where clause in the update command as a query. Second, check constraints at the global level and terminate the operation if the process is not accepted. Then modify the global tuples. Next, determine which local databases need to be updated by using the value of locality if explicit state in the update or using the mapping information in the distribution schema and covert each update operation into local update operation commands. Finally check local constraints and if all satisfied, perform the update. Notice that for deletion, if the given value of locality does not include all the local databases that contain the deleted tuples, then the deletion will be from some local databases only, and the tuple will continue to exist in the global relation. For a replacement operation, a sequence of two operations is needed: delete and insert. The overall operation can take place only if it does not violate any of the global and local constraints.

5.3.5 Final Thought

The work of constraint integration places a burden on the integrator, which requires large amount of human power. In addition, when the size of the multidatabase is huge, it is difficult for the integrator to have an intimate knowledge of the local databases. As for the update approach in [GETE88], the question of how a user can update local databases through local schemata, and how the updates will propagate throughout the system arises.

6 Integrity Constraints in Query Processing

Although MDBSs have been deployed for more than a decade, research in multidatabases query processing and optimization (MQO) has been scarce. MQO is perceived to be a very difficult task because of the lack of a reliable cost model for multiple autonomous databases [LOG93]. Autonomy and heterogeneity of the component databases are the major obstacles in MQO. A different approach, as opposed to rule-based query optimization, called semantic query optimization has been deployed. The core idea of semantic query optimization is to use constraints specified on the database schema in order to transform an input query into one that is more efficient to process [EN00]. In this section, we will briefly discuss the role of integrity constraints in the context of MQO.

6.1 Role of Integrity Constraints

Integrity constraints protect the quality of data in the databases. Integrity constraints exist at all levels of a MDBS. At the local level, there are integrity constraints for the local databases that ensure the quality of existing data. During schema integration, global integrity constraints are derived from the local constraints to protect the GCS. There are also different types of integrity constraints such as primary key, referential, and semantic constraints. Triggers and assertions are the usual approach for enforcement. All of these constraints provide valuable information and alternatives for query processing [GGGM98]. The goals of using semantic information are to remove unnecessary components of a query statement and to reduce the intermediate results' sizes.

6.1.1 Primary Key Constraint

Primary key constraints exist for all relations, and thereof at all levels of a MDBS. Consider a query with a *distinct* clause. Since a primary key is a unique identifier in nature, if the query optimizer knows about the distinct clause of a query is associated with the primary key, then it can remove all the effects caused by the distinct clause. Similarly, such a technique can also be used to remove the *group by* clause if the query tries to group the results by a primary key identifier. The two simple techniques also apply to the unique constraint of attributes.

6.1.2 Domain Constraint

Domain constraints refer to what values the attributes can have. In a MDBS, domain constraints exist at both the existing relations and the GCS, which is essentially a view. Utilizing domain constraints at the global level relies on an accurate mapping of attributes to resolve semantic conflicts such as naming conflicts. Since the restricted domain is always a subset of the possible domain, this type of IC is very useful in removing unnecessary predicates in a query statement and eliminating empty intermediate relations [Car94]. In a MDBS, elimination of redundant predicates and empty relations are of great concern because it can avoid unnecessary mapping of relations (and attributes) and the shipment of intermediate results. Consider the following query and an IC on the involved relation.

q_1 : select * from EMP where SALARY > 5000
 IC_1 on EMP: SALARY > 10000

The integrity constrain restricts that all employees must have a salary of greater than 10000. Obviously, the where-clause is redundant because the two predicates have a trivial implication relationship. A query processor needs to only ship the query request with the where-clause removed. It saves time for the local database to process the query since it does not need to evaluate any expression. The above scenario has shown how an IC can remove a redundant predicate.

On the other hand, consider IC_1 is modified as $SALARY < 3000$. If the query q_1 is issued, the modified IC_1 guarantees that the result will be an empty relation. In many occasions, q_1 results from decomposition of a complex query. The elimination of an empty intermediate relation will save large amounts of time for the component database to process the query and to ship results to the other sites.

6.1.3 Dependency Constraint

While domain constraints restrict the values of attributes, a dependency constraint specifies that a certain relationship between two or more attributes must hold [Car94]. It rests upon the idea of direct implication and indirect

implication. The importance of dependency constraints is magnified in MDDBS. During schema integration, local constraints are used to derive global constraints. Therefore, dependencies exist in most relation-to-relation mapping between GCS and LCS. Consider the following ICs:

$$IC_1: (\text{EMP.AGE} > 40) \Rightarrow (\text{EMP.SALARY} > 40000)$$

$$IC_2: (\text{EMP.SALARY} > 40000) \Rightarrow (\text{EMP.LEVEL} = \text{Administrative})$$

IC_1 is a direct implication, which ensures that if an employee is older than 40, then his salary must be more than 40000. Combining IC_1 and IC_2 yields an indirect implication, which restricts that if an employee is older than 40, then he must be an administrative officer. The implications can be used to remove redundant predicates within a query. For instance, consider a query of *select * from EMP where EMP.SALARY > 40000 and AGE > 50*. In the query, the predicate $SALARY > 40000$ is redundant because if an employee's age is greater than 50, then he must have a salary of greater than 40000. As a result, $SALARY > 40000$ can be removed. Notice that the dependency constraint can also exist in two or more relations. If a set of relations has many of these dependency constraints, the removal of the redundant predicates in a query can greatly reduce the size of intermediate results.

6.1.4 Referential Constraint

Referential constraints refer to a relationship between a pair of attributes in two relations. They are of heavy use in MQO to reduce the number of join operations and redundant joins [HK00, CGK⁺99]. Consider a query involving a join of relations r and s with a join attributes $r.A$ and $s.A$. If a referential constraint of $r.A$ and $s.A$ is known, then a Cartesian product of the r and s does not have to be formed since each tuple in r must correspond to only one tuple in s . It greatly reduces the size of possible set of results.

Besides reducing the solution space, intermediate joins might also be eliminated based on referential constraints [CGK⁺99]. Consider the four relations r_1, r_2, r_3, r_4 at four different sites. Suppose the following query is issued at the global level:

```
select r1.X, r3.Y
from r1, r2, r3, r4
where r1.A = r2.A and (p1)
      r2.B = r3.B and (p2)
      r2.A = r4.A and (p3)
      r2.B = r4.B (p4)
```

The above query is a typical query joining four different relations. Assume the following referential integrity constraints have been defined.

```
r1.A references r2.A
r2.B references r3.B
(r2.A, r2.B) references (r4.A, r4.B)
```

The predicate p_1 joins r_1 and r_2 , p_2 joins r_2 and r_3 , (p_3, p_4) joins r_2 and r_4 . From the user's point of view, the above query correctly joins all necessary relations in order to retrieve the attributes specified in the projection. However, the query is not efficient and can be optimized if the information on referential constraints is available. From the integrity constraints, one will discover that the intermediate join of r_2 is not necessary as r_2 is serving as a bridge for r_1, r_3 , and r_4 . By combining p_1 and p_3 , r_1 and r_4 can be joined directly. Similarly, r_3 and r_4 can be joined directly by integrating p_2 and p_4 . A resulting optimized query is as follow.

```
select r1.X, r3.Y
from r1, r3, r4
where
  r1.A = r4.A and
  r3.B = r4.B
```

6.2 ICs Enforcement and Query Results

As shown in the previous subsections, integrity constraints can reduce the cost of query processing in MDDBS. The underlying assumption is that all integrity constraints have to be completely enforced at all levels of a MDDBS. Otherwise, the query results are not accurate if a not-enforced constraint is used during MQO. Issues and approaches regarding constraint enforcement are covered in earlier sections of this paper. The common approach to maintain integrity constraints is the use of triggers. Triggers are considered highly effective in centralized homogeneous databases. However, in the context of MDDBS, triggers introduce large amounts of overhead and redundant message exchange because there is not a centralized component to control and monitor the interactions and behavior of triggers. Thus, enforcing and maintaining integrity constraints in MDDBS still remain to be one of the open areas for research.

7 Centralized Hierarchical Modeling of ICs in MDDBS

Among the current literature in MDDBS, there is not a standardized framework that serves as a reference model because of the different needs from different organizations. As a result, many implementation details of the components of a MDDBS are left as an open research area. As mentioned in section 2, the existing information from integrity constraints is not utilized effectively in a MDDBS. In this section, we will present an approach to provide centralized hierarchical modeling of integrity constraints in order to enable effective utilization of the existing information during the core operations, namely reads and updates, on databases.

7.1 The Existing Problem

In a MDDBS, information about the local databases might not be available to the global layer. In particular, local integrity constraints are not completely reflected at the global system. As a result, a global model that precisely describes the relationships of integrity constraints is not available. The lack of such a model disallows effective use of the existing information during global query processing, and causes three undesirable effects: (i) subqueries contain redundant predicates; (ii) subqueries always return an empty relation because of the violation of local integrity constraints; and (iii) updates are inconsistent with the local constraints. The third scenario is even worse if an inconsistent update is a data-intensive operation. As a result, the process of having data to travel to the component databases produces unnecessary overheads, which are redundant transfer and meaningless local processing.

For the problem described above, we have discovered that a centralized hierarchical modeling of the integrity constraints could be a solution. Since requests on the local databases originate at the global system, it is reasonable to have a centralized modeling of integrity constraints at the global level to perform pre-processing of the requests. Our model provides an abstraction that depicts the relationship between two constraints and the inherited properties. More importantly, the centralized hierarchical model will enable effective use of integrity constraints during global query processing.

7.2 Overview of the Centralized Hierarchical Model

As mentioned in section 2.3.2, we consider four categories of integrity constraints – local constraint (LC), derived local constraint (DLC), global constraint (GC), and additional global constraint (AGC). Our model supports only

domain constraints and dependency constraints at the moment. The term integrity constraint that appears in subsequent discussion implicitly refers to domain constraints and dependency constraints.

7.2.1 Major Concepts of the Model

The core idea of our model is to build a centralized model, which resides at the global system, to describe the global and local integrity constraints. The constraints will be modeled hierarchically representing the relationships among them. It will be accomplished by making local information globally available. By having such a model, pre-processing of local databases' operations can be performed at the global layer. We target pre-processing of subqueries involving a single global relation. Fragmentations will also be considered. After a global query is decomposed into local subqueries, each subquery will be checked against the model. We believe that the pre-processing can effectively remove redundant predicates of a subquery, eliminate meaningless shipments of subqueries, and avoid inconsistent local updates.

7.2.2 Structure of the Model

Each integrity constraint will be represented by an IC-node (integrity constraint node). An IC-node stores information that is necessary to perform a verification of predicates. An IC-node can be one of the four categories previously, which are AGC, GC, DLC, and LC. There will also be dummy nodes that serve as bridges among IC-nodes. These dummy nodes primarily store the names of relations and attributes only. After a set of IC-nodes and dummy nodes are established, they will form an IC-tree (integrity constraints tree). An IC-tree provides a hierarchical description of the IC-nodes. There is one IC-tree per global relation, which might be resulted from integration of two or more local relations. The set of IC-trees forms the complete model.

When we refer to the names of attributes or relations globally, we assume that an underlying schema translation providing accurate mapping and resolving conflicts is well established during the integration process.

7.2.3 Usage of the Model

The major usage of the model is for query processing. The two core operations of databases are reads and updates, which will be carried out by select, update, delete, and insert SQL statements. By performing pre-processing at the global level, it can remove redundant predicates in a query, prevent sending a query that returns empty relation, and avoid inconsistent modifications of data. We will demonstrate the use of the model in section 7.5.

Besides global query processing, the model can also be used in several different scenarios. It depicts a visual structure of how ICs are organized. The set of IC-trees also provides another measurement of how close two or more schemata are correlated, which can further derive statistical information for the component databases. Another use of the model is that it will record all constraint violation globally. The constraint violation frequencies and characteristics could give us a better insight into how efficiently the whole system is functioning.

We will only demonstrate the use of IC-Trees in the context of query processing in section 7.5, which provides the major motivation of the model. It is important to note that our model described here targets domain constraints and dependency constraints only. However, we believe that the model can be extended to support other types of constraints. It will require a more complex characterization of the constraints and further exploration of their relationships.

7.3 Characteristics of ICs

An important idea of the model is how to characterize both local and global ICs. It relies heavily on the notion of semantic equivalence, which is discussed in earlier sections. With some analysis based on semantic similarities, we can characterize and organize the ICs in a more meaningful way that is more than just using the name of the attributes involved in an IC. Our model rests upon the idea that two semantically similar predicates have a relationship of strong and weak. The relative strength of an IC, and thereof a predicate of query, constitutes the construction and operation of the model.

7.3.1 Implication of ICs and Predicates

Implication closure [HK00] is the concept behind implication of two predicates, or ICs. The three types of implications, trivial, direct, and indirect, are described in sections 6.1.2 and 6.1.3. An implication may exist between two similar predicates. For instance, SALARY > 30000 implies SALARY > 20000 is a trivial implication. Direct implication exists between two or more predicates that describe different attributes. For instance, SALARY > 30000 implies AGE > 30 is a direct implication. An indirect implication is formed by two direct implications. Trivial implication plays an important role for domain constraint whereas direct and indirect implications are more meaningful to dependency constraints.

7.3.2 Relative Strength of ICs

We introduce the notion of strong IC and weak IC. Strong and weak are relative measurements and depend on the context. The strength of an IC is only meaningful to the same class of ICs, which means the strength of ICs will produce a partial ordering of ICs. We define strong and weak based on implication. It is important to notice that the two definitions also apply to predicate-predicate and predicate-IC measurement.

Definition 7.1 IC_1 is stronger than IC_2 if IC_1 implies IC_2 .

Definition 7.2 IC_1 is weaker than IC_2 if IC_2 implies IC_1 .

For instance, if the IC is SALARY > 30000 and a predicate of SALARY = 50000 is compared. We say that the predicate is stronger than the IC.

7.3.3 Satisfaction of an IC

With implication and the strength of ICs being described, we now will define the most important notion of the model.

Definition 7.3 A set of predicates p satisfies an IC I if p implies I .

Notice that we use ‘a set of predicates’ instead of ‘a predicate’ in the definition. It is due to the fact that when we deal with dependency constraints, it always involves two or more predicates. Consider the following examples.

Example 1 IC_1 : SAL > 30000; p = SAL = 50000

With the above constraint and predicate, p satisfies IC_1 .

Example 2 IC_2 : (SAL > 30000) \Rightarrow (AGE > 30); p_1 : SAL = 50000; p_2 : AGE = 40

With the above dependency constraint and predicates, the set of predicates $\{p_1, p_2\}$ satisfy IC_2 . With some pre-processing, a dependency constraint can be expressed as disjunction of simple predicates using logical transformation. Given P and Q are propositions, $P \Rightarrow Q$ is equivalent to $\neg P \vee Q$. Thus, IC_2 described above will become (SAL \leq 30000) \vee (AGE > 30). It means that when the predicates satisfy SAL > 30000 but with AGE \leq 30, IC_2 is violated.

7.4 Construction of the Model

In this section, we will present the details of the structure of the model and our approach to model local and global integrity constraints.

7.4.1 IC-Node Structure

An important part of the design of the model is to determine what information needs to be stored. The building blocks of the model are the set of IC-nodes. Recall from section 7.2 that an IC-node is a structure to represent an IC. At this point, we decide to have the following information stored in an IC-node:

- Attribute(s) involved in the IC.
- Name of the relation in which the IC resides.
- The constraint
- Violation frequency

The name of attributes, name of relation, and the constraint are essential for an IC-node. It gives us all information that is necessary to determine whether a given predicate or set of predicates satisfies the IC. The violation frequency is additional information that might be of use when we want to derive further characteristics about the databases. An IC-node can be realized and stored by a normal tuple as suggested in [Car94]. A major advantage is that it does not require any additional functions other than a relational database to store the IC-nodes.

7.4.2 IC-Tree Structure

An IC-tree is made up of a set of IC-nodes. Recall that there is one IC-tree per global relation. There are two types of nodes in an IC-tree - dummy nodes and IC-nodes. The IC-nodes are what is being described in the previous section. A dummy node serves as a bridge to connect two or more IC-nodes. The most important notion of an IC-tree is the relative strength of ICs. The goal of the model is to have the strongest ICs residing at the bottom of the tree whereas the weaker ICs are located above the stronger ones. A typical IC-tree for a global relation has five levels:

- A dummy node that stores the name of the relation as the root of an IC-tree.
- Below the root are the IC-nodes for AGCs.
- Below the AGCs are the GCs
- Below the GCs are the LCs.
- Below the LCs are the DLCs.

A parent-child relationship indicates the relative strength of two ICs with the stronger one as the child. This means that GCs are stronger than AGCs, LCs are stronger than GCs, and DLCs are stronger than LCs. We treat fragmentation predicates as DLCs, which are stronger than their corresponding parent LCs. After the set of IC-trees is built, a super-root will serve as the root of the complete model and connects all the IC-trees. A generic representation of the complete centralized hierarchical model is shown in figure 5.

7.4.3 Example Construction of an IC-Tree

We will construct an example IC-tree for a global relation, and the example IC-tree will be used in section 7.5 when we demonstrate the uses of an IC-tree. We will only use a very simple example. In reality, the IC-tree could be more complex than what we show. We define two local schemata and a global schema below.

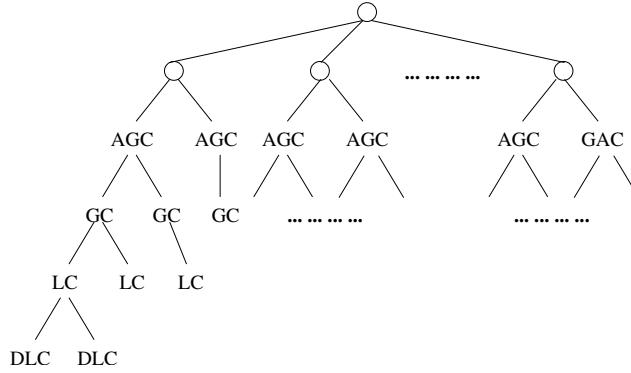


Figure 5: Structure of the complete model.

Local DB1: EMPLOYEE (EMPID, DEPTID, AGE, SALARY, WORKLEVEL, TITLE)
 LC1: SALARY > 30000
 LC2: AGE > 18
 LC3: (SALARY > 40000) ⇒ (WORKLEVEL > 7)

Fragments: DB1.EMPLOYEE₁ (EMPID, DEPTID, AGE, SALARY, WORKLEVEL, TITLE)
 DLC1: SALARY < 40000
 DB1.EMPLOYEE₂ (EMPID, DEPTID, AGE, SALARY, WORKLEVEL, TITLE)
 DLC1: SALARY ≥ 40000

The relation EMPLOYEE in local DB1 is horizontally fragmented into EMPLOYEE₁ and EMPLOYEE₂. The location of the fragments is not of our concern. Notice that the ICs on SALARY of the fragments are considered DLCs because the predicates are used for fragmentation and are derived from their parent-relation; whereas other LCs are directly inherited from the parent-relation without any change.

Local DB2: EMP (EID, DID, AGE, SAL, YEARS, TI)
 LC1: SAL > 50000
 LC2: AGE > 25

We assume that DB1.EMPLOYEE and DB2.EMP are semantically equivalent relations, and all the attributes are semantically equivalent. Below we define the integration of the two local relations to g-EMP. Our convention is to add 'g-' in front of the global relations and attributes. Figure 6 shows the corresponding IC-tree for g-EMP.

Global Schema: g-EMP (g-EID, g-DID, g-AGE, g-SAL, g-YEARS, g-TITLE)
 GC1: g-SAL > 20000
 GC2: g-AGE > 16
 GC3: (g-SAL > 40000) ⇒ (g-YEARS > 5)
 AGC1: g-SAL > 0
 AGC2: g-AGE > 0

7.5 Usage of Hierarchical Modeling of ICs

The major use of the model is global query processing. We will use the example schema as described in section 7.4.3 and figure 6. We do not demonstrate the use for dependency constraints here since it is very similar to the

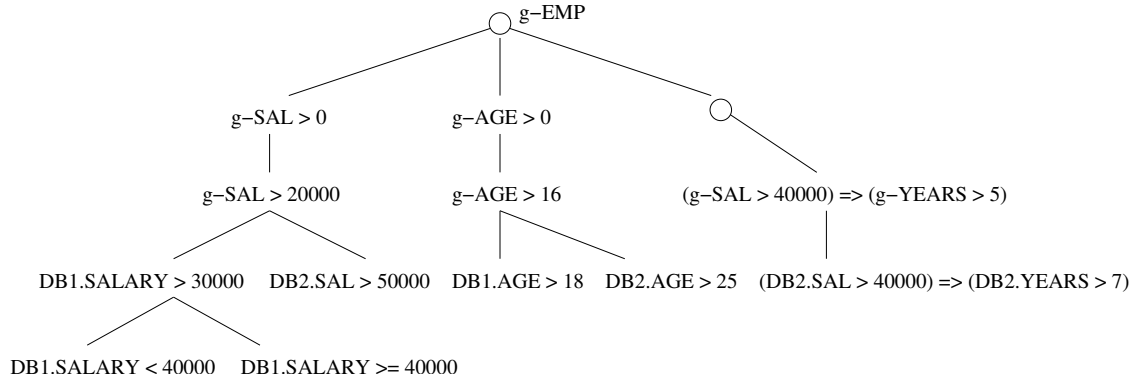


Figure 6: IC-tree for g-EMP.

use of domain constraint. Recall that during global query processing, we want to remove redundant predicates, determine whether to ship the subquery request to local databases, and avoid inconsistent updates. Our major focus is to deal with query involving a single relation and simple predicates. We assume the decomposition is being performed by the global query decomposer.

7.5.1 Read-only query

We demonstrate two scenarios that involve removal of a redundant predicate and eliminating a subquery that results in empty relation.

Removing Redundant Predicates. Consider the following subquery:

```
select * from DB2.EMP
where AGE > 30 and YEARS = 5
```

Denote p_1 as $AGE > 30$ and p_2 as $YEARS = 5$. The set of predicates will be checked against the IC-tree for g-EMP in figure 6. The predicate p_2 has no corresponding IC-branch, so p_2 automatically passes the check. When checking p_1 against the IC-tree, it starts the traversal from the g-EMP node, which is the root. Then it goes down to the middle branch. It satisfies $g-AGE > 0$ and $g-AGE > 16$, which are AGC and GC, respectively. Then it takes the right branch and arrives at $AGE > 25$, which is the leaf of its IC-branch. It is obvious that p_1 and $AGE > 25$ has a trivial implication relationship, which means p_1 is redundant and can be removed.

Eliminating Request for Empty Relation. Consider the following subquery:

```
select * from DB1.EMPLOYEE
where AGE > 30 and SALARY > 50000
```

Denote p_1 as $AGE > 30$ and p_2 as $SALARY > 50000$. The subquery is issued against a local relation at DB1, which was further fragmented into EMPLOYEE1 and EMPLOYEE2. The predicate p_1 will be successfully validated since p_1 satisfies all IC-nodes of its corresponding branch and will arrive at $AGE > 18$. The predicate p_2 will take the left-most branches all the way down to $SALARY > 30000$, which is the local constraint of DB1.EMPLOYEE. Predicate p_2 only satisfies $SALARY \geq 40000$, which is the fragmentation predicate for EMPLOYEE2. Since the IC-node contains information such as relation name and location, the traversal will notify the global system that the subquery will result in an empty relation when it is issued against EMPLOYEE1.

7.5.2 Updates Query

Avoiding Inconsistent Updates. Generally, updates refer to both insert and update statements. We will demonstrate this by an insert statement below. Consider the following query:

```
insert into DB1.EMPLOYEE
values (1001, 20, 30, 25000, 10, 'Manager')
```

There are three local ICs on EMPLOYEE at local DB1. The query generates two predicates that have an associated IC-branch in the IC-tree. The two predicates are SALARY = 25000 and AGE = 30, which will be denoted as p_1 and p_2 , respectively. Predicate p_2 satisfies its corresponding IC-branch and gets the AGE > 18 IC-node; but, p_1 will be rejected at SALARY > 30000 since it does not satisfy the IC. Therefore, the data will not be sent to the local database to perform the insertion. Imagine that a process fetches thousands of tuples from another relation and tries to insert the data to DB1.EMPLOYEE, in which most tuples have a SALARY of less than 30000. Without pre-processing at the global level, the huge amount of data transfer will produce redundant traffic.

7.6 Tradeoffs of the Model

There are several tradeoffs to have such a model globally. The most obvious one is that it requires more processing time at the global level. A high volume of queries might cause the validation process against the IC-trees to be a bottleneck. A simple technique could solve this problem by first performing a check on the query to determine whether it has predicates involving any ICs. It will eliminate unnecessary waits by the queries that are not involved with the ICs.

The model also requires interaction between component databases and the global system. Component databases have to be willing to share their information on local constraints. Furthermore, modifications of local ICs have to be reflected accurately at the global level. Because of the high degree of autonomy, local databases are not obligated to inform about their changes, which might cause difficulties during implementation of the model.

8 Conclusion

In this paper, we have provided a report in studying integrity constraints in MDBS. The study is focused on the issues and approaches in specifying, maintaining, and using integrity constraints in MDBS. We have presented the motivation of MDBS in section 1. Sections 2 and 3 presented the overview of MDBS and integrity constraints in general. Section 4 described the issues of semantic heterogeneity in the context of integrity constraints as well as their solutions. Section 5 discussed integrity constraints during schema integration in MDBS. Section 6 provided a brief overview of the role of integrity constraints in MDBS global query processing. In Section 7, we have proposed a possible approach to represent the global and local integrity constraints as a centralized hierarchical model.

References

- [BE96] Orman Bukhres and Ahmed Elmagarmid. *Object-Oriented Multidatabase Systems: A Solution for Advanced Applications*. Prentice Hall, New Jersey, 1996.

- [BGMS92] Yuri Breitbart, Hector Garcia-Molina, and Abraham Silberschatz. Overview of multidatabase transaction management. *VLDB Journal: Very Large Data Bases*, 1(2):181–293, 1992.
- [Bob96] Angelo R. Bobak. *Distributed and Multi-Database Systems*. Artech House Publishers, San Francisco, California, USA, 1996.
- [BT99] S. Balko and C. Turker. Integration of aggregation constraints. *Proc. 2nd Int. Workshop on Engineering Federated Information Systems, EFIS*, 1999.
- [Car94] J. Cardiff. Semantic query optimization in heterogeneous DBMSs. In Hesham El-Rewini and Bruce D. Shriver, editors, *Proceedings of the 27th Annual Hawaii International Conference on System Sciences. Volume 2 : Software Technology*, pages 273–282, Los Alamitos, CA, USA, January 1994. IEEE Computer Society Press.
- [CGK⁺99] Qi Cheng, Jarek Gryz, Fred Koo, T. Y. Cliff Leung, Linqi Liu, Xiaoyan Qian, and K. Bernhard Schiefer. Implementation of two semantic query optimization techniques in db2 universal database. In Malcolm P. Atkinson, Maria E. Orłowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie, editors, *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 687–698. Morgan Kaufmann, 1999.
- [EN00] Rames Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems*. Addison-Wesley, Menlo Park, California, 3rd edition, 2000.
- [ERS99] Ahmed Elmagarmid, Marek Rusinkiewicz, and Amit Sheth. *Management of heterogeneous and autonomous database systems*. Morgan Kaufmann, Boston, London, 1999.
- [GETE88] M.S. Gamal-Eldin, G. Thomas, and R. Elmasri. Integrating relational databases with support for updates. In *Proc. Int'l. Symp. on Databases in Parallel and Distributed Systems*, page 202, Austin, TX, December 1988.
- [GGGM98] Parke Godfrey, John Grant, Jarek Gryz, and Jack Minker. Integrity constraints: Semantics and applications. In *Logics for Databases and Information Systems*, pages 265–306, 1998.
- [GTE89] S. GamalEldin, G. Thomas, and R. Elmasri. Local and global constraints in database integration. In *Proc. 22th Hawaii Int. Conf. on System Sciences*, volume 2, pages 604–611, January 1989.
- [HK00] Chun-Nan Hsu and Craig A. Knoblock. Semantic query optimization for query plans of heterogeneous multidatabase systems. *IEEE Transactions on Knowledge and Data Engineering*, 12(6):959–978, 2000.
- [LNE89] J. Larson, S. Navathe, and R. Elmasri. A theory of attribute equivalence in databases with applications to schema integration. *IEEE Transactions on Software Engineering*, April 1989.
- [LOG93] H. Lu, B. C. Ooi, and C. H. Goh. Multidatabase query optimization: Issues and solutions. In *Proceedings of Third International Workshop on Research Issues in Data Engineering: Interoperability in Multidatabase Systems*, pages 137–143, 1993.
- [Mar91] Victor M. Markowitz. An architecture for identifying objects in multidatabases. *Interoperability in Multidatabase Systems, 1991. IMS '91 Proceedings*, pages 294–301, April 1991.

- [MT93] D. Mukhopadhyay and G. Thomas. Practical approaches to maintaining referential integrity in multidatabase systems. *Proceedings RIDE-IMS '92.*, pages 42–49, April 1993.
- [SL90] Amit P. Sheth and James A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys (CSUR)*, 22(3):183–236, 1990.