SURVEY ON DATABASE INTEGRITY CONSTRAINTS IN VARIOUS OBJECT ORIENTED DATABASE MANAGEMENT SYSTEMS

Vaishali J. Dindoliwala\(^1\) and Rustom D. Morena\(^2\)

\(^1\)Assistant Professor, C. B. Patel Computer College, Bharthana, Vesu, Surat, Gujarat, India
vaishali_1331@yahoo.co.in\(^1\)

\(^2\)Professor, Department Of Computer Science, Veer Narmad South Gujarat University, Surat, Gujarat, India
rdmorena@rediffmail.com\(^2\)

Abstract: The accuracy, consistency and integrity of data in Object Oriented Database Management Systems (OODBMSs) are extremely important for developers and users. Checking the integrity constraints like domain integrity, entity integrity and referential integrity in OODBMSs is a fundamental problem in database design. Existing OODBMSs lack the capability for automated verification and enforcement of integrity constraints that appear amongst attributes in association, composition and inheritance relationships. In this paper, we have surveyed OODBMSs like db4O, Objectivity/DB, ObjectStore, GemStone and O2 based on constraint mechanism provided by them and made a comparative survey on them.

Keywords: OODBMS, Integrity, Relationships, Constraints, Domain Integrity, Entity Integrity, Referential Integrity

1. INTRODUCTION

The term integrity constraint covers both consistency and validity that is data is well organized in accordance with the requirements of a data model and all invalid data is excluded from the database. The proper handling of various integrity constraints like domain integrity, entity integrity and referential integrity constraints is essential for any database. For that OODBMSs may specify rules for the objects stored in a database. These rules may be used to limit the type of data that can be stored in object attribute which ensures the accuracy and reliability of the data in the objects. If there is any violation between the constraint and the data action, the action is aborted.

In OODBMSs, the objects may have various relationships with other objects stored in the database. These relationships are mainly inheritance, association, aggregation or composition. So the constraint defined for an object of a class should be handled when that object is related with the other objects. Constraint allows defining certain data requirements that the data in the database needs to meet [17].

In this paper, we have studied constraints and their management during the execution of various operations by different OODBMSs like db4O, Objectivity/DB, ObjectStore, GemStone and O2 and made comparison based on various integrity constraints provided by them.

2. RELATIONSHIPS IN OBJECT ORIENTED DATABASES

Integrity constraints describe the restrictions placed on an object as a result of the relationships in which that object participates. To implement this, constraints must be translated to enforcement rules which define how the constraints will be maintained. Since these constraints are properties of the relationships between objects, the enforcement rules should be implemented as part of relationship constructs. [2]

Checking the integrity constraints in OODBMSs is a fundamental problem in database design [15]. It is not an easy task to identify violations and check integrity constraints in them. This is due to the fact that detecting all constraints that appear as a result of relationship between classes [1]. In OODBMSs, classes are organized into a hierarchy of classes. That is there may be IS-A relationship and HAS-A relationship among various classes. IS-A relationship is implemented through inheritance in object oriented programming languages, while HAS-A relationship is implemented through Association, Aggregation and Composition. Every operation executed on OODBMSs is somehow connected to the objects stored in the database. The possible operations that can be performed on objects are creating objects, deleting existing objects, updating existing objects and retrieving objects. While executing any of these operations for an object, constraints should be maintained.

Relationships between objects in memory are maintained by object references. Once the objects have been stored in an object database, the object references are no longer available, so the objects maintain their relationships by storing the Object Ids (OIDs) of any related objects. [3]

A. Constraints in Inheritance

Inheritance is an IS-A relationship [3]. One big advantage of object-oriented programming is the ability to extend the
behavior of an existing class while reusing the code written for the original class. When a class is extended, the new class inherits all the public members of the class being extended. Also methods may be overridden in a deriving class. It is a relationship between super classes and subclasses [1]. A super class may have some domain integrity constraints defined for attributes of a class. These constraints are checked automatically while performing operations like creating, deleting or updating objects of that class. When this class is inherited in another class then performing any operations on that sub class objects, the constraints defined for super class attributes should not be violated.

B. Constraints in Association

Associations can help us to make design decisions about the structure of our data. We can make decisions about not only the classes that are needed to contain the data but also about which classes are needed to share the data with other classes. It supports data sharing between classes. It is HAS-A type relationship between objects which is implemented by using object references [3]. Association may be Unidirectional or Bidirectional. In Unidirectional Association, one object is associated with another object but the reverse is not true just like one way communication while in bidirectional association, one object is related with other objects and the reverse is also true just like two way communication. It can be in the form of 1:1, 1:M or M:N [1]. In various object oriented programming languages, implementation of association may vary. In C++, association is represented by using pointers while in C# associations between objects are implemented using object references where the reference type is the class name of the associated object. Referential integrity constraints or entity integrity constraints among these objects should be preserved by the OODBMSs.

I. Constraints in Aggregation: Aggregation is a relationship in which one object is formed from other objects. It captures the whole-parts relationship between objects [18]. It is a specialize form of association where all object have their own lifecycle but there is an ownership like parent and child. In an aggregation relationship, the owner object (parent) holds a reference to its owned objects (child). The owned objects may be single objects or collections. Here, both the objects can survive individually which means ending one entity will not affect the other entity. In relational databases, this type of relationship is implemented using foreign keys [3]. In OODBMSs, this relationship among objects or referential integrity constraints should also be maintained.

II. Constraints in Composition: Composition is similar to aggregation but there is a stronger relationship between owner and owned objects that is the owned object cannot exist on its own [3]. As opposed to Aggregation, composition has two additional constraints as below:

- **They are non-shareable:** This is the case when one class can only be the part of one and only one other class.

- **They are existence-dependent:** This is the case when one class can only exist with the existence of another class. That is composition implies a relationship where the child cannot exist independent of the parent.

The OODBMSs should provide these types of referential integrity constraints among objects. Apart from this, they should also provide Cascade Update and Cascade Delete like mechanisms which are available in relational databases. So that when a parent object is deleted, all its child objects will also be deleted. If there is any change in parent object, it should also be reflected in its child objects automatically.

3. Literature Review

Most of the current OODBMSs lack the capability of an ad-hoc declarative specification of enforcing and maintaining integrity constraints which appear as a result of relationships among the objects. They do not have adequate support for certain types of constraints especially the ones defined in a class composition and inheritance hierarchies. The integrity constraints must be maintained in the backward direction along the class composition and inheritance hierarchies as well as in the forward direction [11].

In this section, we have reviewed some of the Object Oriented models in which constraint management is integrated.

M. Doherty, J. Peckham and V. Wolfe [2] have designed a model called SORAC (Semantic Objects, Relationships and Constraints) which combines the object-oriented model with relationship semantics through which object interaction is expressed and implemented. It also provides a data definition language called OIL (Object Interaction Language) which supports the semantics of arbitrary objects and active relationships. It provides monitors which implement the active behavior of relationships. Monitors operate by reacting to the messages received by the related objects and thus confirm to object-oriented encapsulation. A relationship object monitors the messages received by the participating objects and takes appropriate actions to maintain the integrity of the relationship. A relationship object can enforce an assertion on a participating object by causing it to reject a message which would violate the assertion. The SORAC database management system is implemented using the Ontos object database management system on a SUN Sparc architecture.

H. Oakasha, S. Conrad and G. Saake [4] presented the idea of consistency management in object oriented databases. When an object is created, constraints of this object are retrieved from the database metadata known as constraint catalog and the relationship between these constraints and the object is established. Constraint catalog is a repository where information about all constraints is stored. They have designed the classes IC, Shell and Kernel. The class IC stores complete specification of an integrity constraint defined for
In order to achieve efficient checking of constraints, they have introduced Shell and Kernel. The class Shell holds information that is necessary for efficient consistency maintenance. The shells are sharable among all objects of the database. So user can’t disable or enable constraints for specific objects. To avoid this limitation, they have introduced the concept of Kernel class. The class Kernel stores information that is local to objects and which is common among interrelated objects. They have also developed methods for consistency maintenance. Of course, it requires certain overhead to adequately maintain integrity constraints, shell and kernel in order to keep them consistent.

B. Zaqaibeh, H. Ibrahim, A. Mamat and N. Sulaiman [5] have proposed run time model for enforcing integrity constraints in object oriented databases for attributes that are derived from composition and inheritance hierarchies. The Assertion Model of ICs (AMIC) can represent ICs and their relationships over the composition, association, and inheritance hierarchies. It consists of four components: Compile-Time Model (CTM), Run-Time Model (RTM), Object Meta Data (OMD) and Detection Method (DM). The CTM enforces and maintains structural integrity constrains. The RTM is responsible for enforcing logical integrity constraints, verifying inserting, updating, and deleting objects, checking constraint domains and maintaining the unaccepted user request. The simplified form of the constraint is evaluated before an object is inserted to the database. All constraints and domains are verified, optimized and collected in the OMD. Also, the attributes and their relationships are stored in the OMD. When inserting a new object, all constraints in OMD that are related to that object must be checked to verify the new data state. Each object carries a class members and a reference to the OMD. A reference supervises the connection between the object and its related constraints in the OMD. The DM is an overloaded method that can access and modify the OMD. It is designed for constraint validation checking purpose. The AMIC enforces the data integrity whenever a request for inserting, deleting or updating objects occurs.

D. Weber, S. Leone and M. Norrie [6] focus is to manage constraints for semantically rich data models in order to complement object databases specifically for db4O with an extended data quality management facility. They have presented an approach, framework and implementation of a constraint-based approach to data quality that extends and complements object databases with support for constraint-based data quality management which can handle complex constraints like association also. The constraints are managed within their framework and validated against the database based on an event-condition-action (ECA) paradigm. They have classified two types of constraints - hard constraints and soft constraints. Hard constraints such as integrity constraints are necessary conditions that data has to fulfill to be valid. If a hard constraint is violated, data quality is deficient while soft constraints are constraints that are validated in a more relaxed way compared to hard constraints. They could also be seen as recommendations since they are not just valid or invalid but rather increase or decrease data quality. Violations of soft constraints do not lead to errors but the user is instead informed of possible data quality deficiencies. In their approach, the constraint definitions can have different structures like type of the constraint, the name which identifies the kind of constraint, the details which provide information about the individual attributes and conditions, the list of database events upon which the

validation should take place, the validation type that is the standard validator or a custom and the message which will be used for the execution or text message.

V. Benzaken and X. Schaefer [7] have proposed an efficient technique to statically manage integrity constraints in object-oriented database programming languages. The technique deals with integrity constraints in object-oriented database programming languages that provide integrated facilities, as the extended version of O2. This technique focuses on a compile-time checking of constraints.

4. CONSTRAINT MANAGEMENT IN EXISTING OODBMSs

This section provides constraint management in various OODBMSs such as ObjectStore, db4O, Objectivity/DB, GemStone and O2. We have also studied how constraints are handled while performing various operations on objects having relationships with other objects.

A. ObjectStore

For managing domain integrity constraints for collections, ObjectStore provides allow_duplicates, allow-nulls and signal_duplicates facilities which can be set by the user. For collections, without allow_duplicates and signal_duplicates behaviour, inserting something that is already an element of the collection leaves the collection unchanged. For collections with signal_duplicates behaviour, inserting a duplicate raises err_coll_duplicate_insertion. For collections with allow_duplicates behaviour, each insertion increases the collection’s size by one and increases by one the count of the inserted element in the collection. If you insert a null pointer into a collection without allow-nulls behaviour, the exception err_coll_nulls is raised. For dictionaries with signal_dup_keys behaviour, if an attempt is made to insert something with the same key as an element already present, err_index_duplicate_key is raised.

Referential integrity in ObjectStore is implemented with dual pointers between a pair of objects that the DBMS engine keeps mutually consistent [12]. ObjectStore allows you to declare two data members as inverses of one another, so they stay in sync with each other according to the semantics of binary relationships. ObjectStore allows you to model binary relationships with pointer-valued or collection-of-pointer-valued data members that maintain the referential integrity of their inverse data members. You implement this inverse maintenance by defining an embedded relationship class, which encapsulates the pointer or collection of pointers. So that it can intercept updates to the encapsulated value and perform the necessary inverse maintenance tasks. While performing updating object, an update to one side of the relationship always triggers a corresponding update to the other side. While performing deletion, by default, deleting an object that participates in a relationship automatically updates the other side of the relationship so that there are no dangling pointers to the deleted objects [13] [19] [15] [4].

The integrity control problem in ObjectStore is illegal pointers. ObjectStore can detect two kinds of illegal pointers - pointers from persistent memory to transient memory and cross-database pointers from a segment that is not in allow_external_pointers mode. ObjectStore provides facilities that automatically detect such pointers upon transaction commit. One can control whether ObjectStore always detects illegal pointers with the method void
set_check_illegal_pointers (os_boolean) of os_segment macro. If you pass true to this method, check_illegal_pointers mode is enabled for the specified segment.

B. db4O

With db4O, constraints can’t be applied by the database and they need to be defined in the application’s object model. The constraints like domain integrity are not applied by the db4O. They need to be defined in the application’s object model using appropriate methods for the objects. The domain constraints can be implemented by the object itself. For example, to implement NOT NULL constraint, exceptions can be caught when a field is set to null value or one can enforce some rules with the declarations in the appropriate setters of the objects.

In case of referential integrity constraints, if one object is directly related to another object via some relationships and one wants to ensure that the other object exists then one can define a constructor with the necessary objects passed in.

The entity integrity constraint can be implemented in db4O by assigning unique object ID (OID) to every object stored in the database which does not depend on any attribute value. Apart from this, there is no other mechanism of ensuring entity integrity constraint. For example, if one wants to prevent objects with duplicate values in particular fields being stored in the database, he has to first query the database for matching objects. If query returns a non-null result then one can take appropriate action that is whether to update the new object or discard it.

db4O provides CascadeOnDelete mechanisms. By default, CascadeOnDelete is not set. That means when an object is deleted, all its associated objects will remain in the database. If CascadeOnDelete is set, db4o resolves all objects referenced by the object to delete and subsequently deletes them. But db4o is not concerned about multiple object references. That is all objects referenced by the object to delete, are deleted regardless of whether they are referenced by any other objects in the database.

While updating the object, db4O gives the choice. That is if one wants to update associated objects also, he can do so. Here, Set method is used for updation. By default, only the object explicitly passed to the Set method is updated. The way this is controlled is using the concept of update depth. It controls how deep an object graph will be traversed on update. The default update depth for all objects is 1 that means only primitive and string members will be updated. One can change the behaviour by setting CascadeOnUpdate. With this option set, the object to update and its associated objects are updated correctly by storing just the object to update. For example, there are Customer and Address classes. Customer class has object of Address class as its member. The customer object is going to be updated. With CascadeOnUpdate option set, the Customer and Address objects are updated correctly by storing just the Customer object explicitly.

C. Objectivity/DB

Relationships are maintained automatically by Objectivity/DB. Objectivity/DB supports unidirectional and bidirectional relationships, relationship cardinality and the use of relationships to create composite objects. When one object is linked by an association to another object, an application can traverse the association link from the source object to find the destination object. When the association between two objects is bidirectional, an application can rely on the database to maintain referential integrity. In bidirectional associations, Objectivity/DB provides enough information to maintain referential integrity. When a destination object is deleted, all bidirectional association links referencing that object are also deleted which reduces the possibility of dangling object identifiers. [9]. In contrast, it is not possible to ensure that a unidirectional association link references a valid destination object. Unidirectional associations, however, require somewhat less overhead and offer better performance than bidirectional associations. [20]

D. GemStone

GemStone Smalltalk allows you to constrain instance variables to hold only certain kinds of objects. The keyword constraints: in a class creation message takes an argument that specifies the classes the instance variable will accept. It ensures that the variable will contain either nil or instances of the specified class or that class’s subclasses. When constraining an instance variable to be a kind of array, it guarantees that it will always be an array, an instance of some subclass of array or nil. In inheritance, a subclass class inherits instance variables and any constraints on them from its superclass. One can make inherited constraints more restrictive in the subclass by naming the inherited instance variables in the argument to constraints: in the creation statement [10]. In GemStone/S 64 Bit, object constraints are no longer enforced by the virtual machine, although some Smalltalk methods may still enforce constraints. GemStone/S 64 Bit provides some new methods without the keyword constraints: for providing subclass constraints [16]. Apart from this, there is no other constraint management in GemStone.

E. O2

O2 is also a database system and an object-oriented system. As a database system, it provides support for accessing and updating large amounts of persistent, reliable and shared data and as an object-oriented system, it supports features such as object identity, complex objects, inheritance, encapsulation, overriding and run-time binding of methods to objects. In the O2 system, all forms of referential integrity must be explicitly handled by the application programmer since it does not have any constraint management. [11]

5. DISCUSSION

Integrity constraints establish a set of rules that define what will be the permissible states of individual objects in the database and thus what will be the permissible states of the overall database. For example, in any database management
system, the data type specification is in the database schema is a kind of domain integrity constraint. Whenever an update operation is performed in the database whether it is to insert, delete or change of some attribute value for an instance then that instance is constrained to be correct according to the schema definition. This is implicit integrity constraint of a data model which is included in the schema. Apart from this, some constraints are not implicit and needs to be specified explicitly outside the database schema. In most of the OODBMSs, not all the constraints like domain integrity, entity integrity and referential constraints are maintained. Mostly all OODBMSs provide application oriented management of integrity constraints like defining default values for the attributes of an object within a constructor of a class. So whenever an object is created, attribute values will be checked and appropriate action can be taken. But there is a lack of centralized management of integrity constraints by the OODBMSs. Also existing OODBMSs do not have satisfactory support for user defined constraints defined for a class composition or inheritance hierarchy. In existing OODBMSs, integrity constraints are just maintained by rolling back a transaction, disallowing a transaction or modifying operations that produces inconsistent database state.

Table 1 summarizes and compares the constraint management provided by various OODBMSs like db4O, Objectivity/DB, ObjectStore, GemStone and O2.

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Constraints</th>
<th>db4O</th>
<th>Objectivity/DB</th>
<th>ObjectStore</th>
<th>GemStone</th>
<th>O2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Domain Integrity</td>
<td>Supports only data type and unique key constraint for class attributes. Apart from this no other domain integrity constraint is provided. No user defined constraint is supported.</td>
<td>No user defined constraint is supported.</td>
<td>No user defined constraint is supported.</td>
<td>No user defined constraint is supported.</td>
<td>No user defined constraint is supported.</td>
</tr>
<tr>
<td>2</td>
<td>Entity Integrity</td>
<td>Supported by means of OIDs.</td>
<td>Supported by means of OIDs.</td>
<td>Supported by means of OIDs.</td>
<td>Supported by means of OIDs.</td>
<td>Supported by means of OIDs.</td>
</tr>
<tr>
<td>3</td>
<td>Referential Integrity</td>
<td>Not supported at database level. Provides CascadeUpdate and CascadeDelete features as in relational databases. But while deleting objects, it does not check whether object is referenced anywhere else.</td>
<td>Not Supported at database level. Provides CascadeUpdate and CascadeDelete features.</td>
<td>Supported by means of dual pointers between a pair of objects. Provides CascadeUpdate and CascadeDelete features.</td>
<td>Not Supported at database level.</td>
<td>Not Supported at database level.</td>
</tr>
</tbody>
</table>

6. CONCLUSION

The aim of OODBMSs is to manage complex and highly interrelated information. The accuracy, consistency and integrity of data in OODBMSs are important for both developers and users. The detection of violations of integrity constraints in the OODBMSs is not an easy task because there is an association and inheritance of hierarchies exists among the classes. So, existing OODBMSs lack maintenance of integrity constraints and cannot support user-defined constraints. db4O, Objectivity/DB, ObjectStore, GemStone and O2 do not support user defined constraints. Most of these OODBMSs do not support entity integrity and referential integrity constraints also. The aim of constraint management is to restrict invalid database states to those that are considered as legal. So, constraint management should be a part of OODBMSs itself and not a part of application code.
7. REFERENCES


