Integrity Maintenance in Advanced Data Bases

Ulrich Schiel

Universidade Federal da Paraíba
Departamento de Sistemas e Computação
C.P. 10106 / 58109-970 Campina Grande - PB
e-mail: ulrich@dsc.ufpb.br

INTRODUCTION

The use of databases for advanced applications is a rapidly growing and changing field, due to the continuous incorporation of new technologies and media in current systems. Whereas in the near past Database Management Systems (DBMS) mainly use to store and manage tabular data, now they need to model complex structured objects, multimedia data, semi-structured and unstructured documents. Each of these improvements has its own semantics and complexity.

In order to allow an adequate description of database applications, data models are used to describe the conceptual schema of the database. If new categories of applications need to be incorporated or created, and the data model does not fit well with these applications, the model itself must be expanded. The semantics of the new constructs must be defined and the integrity of objects in the new constructs must be guaranteed.

Since a DBMS is in general not expandable, except for future versions of the same product, there are two alternatives. Move the whole application to another system that is capable to adequately process the new structures, or develop specific routines, probably with its own storage systems, in order to incorporate the new application. Clearly both solutions are unsatisfactory.

The first solution is only applicable if there exists a DBMS that considers the new structures. Even if it exists, moving to the new environment means reimplementation of the application, and this is very traumatic and demands a lot of time and money. The other solution, to expand existing applications by special modules is more straightforward, but creates an unbalanced heterogeneous system, combining a database with a file system. This generates problems of integration and does not allow a unified view of the data of the application.

Despite actual existing DBMS consider many modern database concepts, such as object-orientation and triggers, there are a lot of applications needing more. Concepts of temporal databases, geographic databases, hypertext (Web-) databases are not well attended.
In this chapter we introduce an approach of defining the semantics of a complex data model by means of general (schema-) integrity constraints integrated to the system as rules. This approach allows an easy way to define the semantics of complex data models. The rules systems can, at any time, be expanded in order to incorporate concepts of new applications and can also be used to add application specific integrity constraints. Therefore, data model specific constraints and application specific constraints are treated in a unified manner.

**INTEGRITY IN DATABASES**

Integrity maintenance in a database is achieved with two kinds of integrity constraints: *implicit integrity constraints* and *explicit integrity constraints* [EN 99].

Implicit integrity constraints, also denoted as schema constraints, are constraints defined in the conceptual database schema using the language of a data model, including attribute types, keys, null values, relationship cardinalities, generalizations, and aggregations. If we use a complex semantic data model to describe the conceptual schema, several implicit constraints are built in the schema, which reflects the expressiveness of the underlying data model.

Schema constraint satisfaction can be achieved mainly by three distinct approaches: (1) the DBMS supports the data model completely, and therefore its semantics is embedded in the software of the DBMS; (2) with the mapping of the conceptual schema into an internal schema, supported by the DBMS, the implicit constraints are implemented in the structure of the internal schema and, for features not foreseen in the model of the internal schema, create some controlling procedures; (3) the semantics of the data model is described in form of rules which are able to guarantee full semantic integrity and may be achieved in the DBMS. Since the rules are model dependent, and not application dependent, they are mapped only once to the internal schema. If solution (1) can be applied, it is the most straightforward and efficient solution. If we do not have the adequate DBMS, solutions (2) or (3) may be applied. Solution (3) is better than (1) as (2) with regard to generality and flexibility. Generic since it applies to all data models to be considered, and flexible because we can, at any time, add new rules in order to capture changes in the data model itself.

Instead of the richness of a semantic data model, it remains constraints which can not be expressed in the structure of the conceptual schema. These are the explicit integrity constraints. A constraint language must be used to specify this constraints. Actually, the importance of constraints has been recognized and special databases, containing constraints as ordinal data, so called Constraint Databases are under investigation [RS 97] [KLP00]. For systems using the relational language SQL, the typical language elements for expressing explicit constrains are the ASSERTION and TRIGGER commands. Also expressions of the relational algebra can be used to create predicates as integrity constraints [UW 97].
instance, if $E_1$ and $E_2$ are two relational expressions, we can state a constraint as $E_1 = \emptyset$ or $E_1 \subseteq E_2$.

The most complete language for expressing constraints in object-oriented databases is OCL - Object Constraint Language [UML97]. OCL is part of the UML - Unified Modeling Language specification. It is a very rich formal language able to specify implicit constraints of applications modeled with the UML language. OCL allows the statement of invariants on classes and types, describe pre- and post-conditions on operations, guards, and so on. An OCL expression can refer to types, classes, properties, and datatypes. For instance, the constraint that "Married people are of age $\geq 18$" is attached to the class Person as

```plaintext
Person
  self.wife ->. notEmpty implies self.wife.age >= 18 and
  self.husband ->. notEmpty implies self.husband.age >= 18
```

OCL is a pure expression or declarative language. This means that if an OCL expression is evaluated it always returns a value and has no direct side-effect in the system. The user must program his side-effects based on the value returned. Another restriction is that it only acts at the application level. It is not possible to state generic constraints about classes, generalisations, associations, or other constructs of a data model. Even at the application level, there is no explicit reference to UML stereotypes such as generalisation, aggregation and composition.

This chapter presents a formalism, which enables the specification and enforcement of both implicit and explicit integrity constraints. Considering the intension/extension dimension of the ANSI/SPARC DBMS architecture proposal [ANSI86], which divides a database description into four levels (Metadata Model, Data Model, Application Model and Application Data), the formalism is intended to allow the specification of the semantics at the Data Model level as well as at the Application Model level.

Implicit constraints are located at the Data Model level and explicit constraints are located at the Conceptual schema/Application Model level. Using this approach it is also easy to extend the data model itself, in order to incorporate new technologies and concepts into the model. This update is done adding new rules at the Data Model level. The new concepts are defined at the Metadata Model level and their semantics established by the creation of new implicit constraints at the Data Model level.

The formalism described in this chapter uses a rules model, based on the well-known ECA-Rules (event-condition-action) [Da 88]. Two kinds of rules are distinguished: Dynamic Axioms and Side Effects. Dynamic Axioms (DAs) inhibit operations that violate the semantic integrity and Side Effects (SEs) react on potential integrity violations and trigger actions to recompose the integrity. A side effect of an implicit integrity constraint is called a System Side-Effect (SSE), whereas application dependent side effects are called User Side-Effects (USE).
Despite the formalism can be used or adapted for any data model we will use a relatively complex object-oriented data model, in order to prove its generality.

The data model we consider is based on the notions of class as a collection of objects. Relationships are defined between classes with two directions and for each direction a min and a max cardinality is associated. Between classes there can hold the well-known abstractions of Generalization/specialization, aggregation and grouping. Generalization and aggregation has been defined first in [SS 77] as an extension of the relational data model. It has been considered in the most semantic data models, such as SDM [HM 81], THM [Sch84] and ACM/PCM [BS 84]. Also object-oriented models, such as ODMG [Ca 01] and UML [JBR00] use generalization and aggregation. The abstraction we call grouping is also contained in SDM, THM and ACM/PCM (here called association), and is discussed in [Od-98] as power type. In the UML world [P-J9*] calls it composition. The main difference between aggregation and grouping is that the first is heteromeous with a fixed number of components and grouping is homeomeous with a variable number of components [P-J9*]. Details of these concepts will become clear during the text.

The system presented in this chapter is based on the data model TOM [Sch91], which is richer than ODMG and UML, whereas UML has some concepts not included in TOM. Therefore, its use for ODMG based applications, the exceeding DAs and SSEs must be excluded and for UML an adaption must be made.

ELEMENTARY OPERATIONS AND STRUCTURAL PREDICATES

We define some primitive operations and predicates over objects, relationships and classes. The notation for operations follows the notion of message in object-oriented systems, and has the form: <receptor> <message (parameters)>

OPERATIONS

- $C$ create $(e)$: creates object $e$ in class $C$.
- $e$ delete $(C)$: deletes object $e$ from class $C$.
- $e_1$ establish $(r, e_2)$: establishes relationship $r$ between objects $e_1$ and $e_2$.
- $e_1$ remove $(r, e_2)$: removes relationship $r$ between objects $e_1$ and $e_2$.

There are two special operations for the insertion and removal of elements of groups. The effect is similar to the creation and deletion of objects, only that they acts on group objects and not on classes:
- $g$ gr-insert $(o)$: inserts $o$ as a new element of the group $g$.
- $g$ gr-delete $(o)$: eliminates $o$ of the group $g$. 
In order to verify facts in the database, several predicates are defined. We divide them as basic predicates, hierarchies

BASIC PREDICATES

- \textit{in}(\textit{e}, \textit{C}): \textit{e} is an instance of class \textit{C}.
- \textit{is-rel} (\textit{e}_1, r, \textit{e}_2): \textit{e}_1 and \textit{e}_2 are related by \textit{r}.
- \textit{min}_r, \textit{max}_r.
  Each relationship \textit{r} has associated with it a cardinality expressed by a pair \((\textit{min}_r, \textit{max}_r)\), which means that each instance of the starting class is associated at least to \textit{min}_r and at most to \textit{max}_r instances of the target class.

HIERARCHIES

- **Generalization/Specialization**
  
  Means the creation of more specific subclasses of a given generic class. We can apply several specializations, characterized by several \textit{roles}. Each role is given by a predicate \textit{p(e)} that establishes to which subclass a specific object can be associated. For instance, we can have \textit{sex(PERSON)} = MALE, FEMALE and \textit{age(PERSON)} = YOUNG, MIDDLE, OLD. For a given person \textit{p} the predicate \textit{ageYOUNG(p)} is true if \textit{p} is a young person.

  - \textit{is-a} (\textit{C}_1, \textit{C}_2, \textit{p}): class \textit{C}_1 is a subclass of \textit{C}_2 by role \textit{p}.
  - \textit{p}_G(e)
    The specialization role is a disjunctive predicate \(p_G(e) = p_{c1}(e) \lor \ldots \lor p_{cn}(e)\) where \textit{e} is an object of a generic class \textit{G} and \(p_{ci}(e)\) is true iff \textit{e} is an instance of the subclass \textit{C}_i of \textit{G}.
  - \textit{disjunctive} (\textit{D}, \textit{C}_1,\ldots,\textit{C}_n)
    \textit{D} is a disjunctive generalization of \textit{C}_1,\ldots,\textit{C}_n, i.e., each instance of \textit{D} can be contained in only one subclass \textit{C}_i of \textit{D}.
  - \textit{covering} (\textit{D}, \textit{C}_1,\ldots,\textit{C}_n)
    \textit{G} is a covering generalization of \textit{C}_1,\ldots,\textit{C}_n, i.e., each instance of \textit{D} must occur in at least one subclass \textit{C}_i of \textit{D}.

- **Aggregation**

  Means the creation of new object as combination of several distinct parts.

  - \textit{is-part} (\textit{e}_1, \textit{e}_2): \textit{e}_1 is one component of the aggregate \textit{e}_2.
- **aggregation**\( (A, C_1,\ldots,C_n) \): class \( A \) is an aggregation of classes \( C_1,\ldots,C_n \).
- **aggregation-r**\( (A, C_1, C_2, r) \)
  
  Class \( A \) is an aggregation of classes \( C_1 \) e \( C_2 \) by relationship \( r \) iff the related objects are just the elements of the aggregated class \( A \).
- **agg-inherit**\( (A, C_1,\ldots,C_n, r) \)
  
  \( A \) is an aggregation of \( C_1,\ldots,C_n \) and the component classes inherit the relationship \( r \).
- **agg-comp-inherit**\( (A, C_1,\ldots,C_n, r, \sigma) \)
  
  \( A \) is an aggregation of \( C_1,\ldots,C_n \) and the inheritance is determined by a function \( \sigma \).

- **Grouping**

  Several objects of a class are grouped together to form higher order objects.

  - **is-elem**\( (e_1, e_2) \): \( e_1 \) is an element of the group object \( e_2 \).
  - **is-elem**\( (C, G, p) \) \( G \) is a class of groups of elements of \( C \) and group containment is governed by predicate \( p \), i.e. \( p(e, g) \) holds iff \( e \) is an element of \( g \).
  - **disjoint-gr**\( (C, D) \)
    
    \( D \) is a grouping of \( C \) and each instance of \( C \) occurs in at most one group of \( D \).
  - **covering-gr**\( (C, D) \)
    
    \( D \) is a grouping of \( C \) and each instance of \( C \) occurs in at least one group of \( D \).

**DYNAMIC AXIOMS AND SIDE-EFFECTS**

The semantic integrity of an application is maintained by two kinds of rules: Dynamic Axioms and Side Effects. The dynamic axioms are integrity restrictions that avoid that the user realizes updates that are not consistent with the conceptual schema, whereas the side effects execute auxiliary operations in order to recompose the integrity of the database. A few examples of **System-Side-Effects**, which guarantee the structural components of the data model, are presented. The application designer can add application dependent rules, called **User-Side-Effects**. All rules are of the form ON <event> IF <condition> DO <action>.

**1. Dynamic Axioms**

The action **abort** in a rule characterizes an integrity rule and avoids the execution of the operation on the event part.

\( DA1 \Rightarrow \) An **establish** may not violate the maximal cardinality
ON \(x\) establish \((r, y)\) IF \(\#\{<x,z> / is-rel(x,r,z)\} = \text{max}_r\) DO abort

DA2 \(\Rightarrow\) A remove may not violate the minimal cardinality

ON \(x\) remove \((r, y)\) IF \(\#\{<x,z> / is-rel(x,r,z)\} = \text{min}_r\) DO abort

DA3 \(\Rightarrow\) The insertion of a new instance in a subclass must respect the role

ON \(C\) create \((x)\) IF \(is-a (C, D, p) \land not p_c(x)\) DO Abort

DA4 \(\Rightarrow\) In a covering generalization an object cannot remain only in the generic class

ON \(x\) delete \((C)\) IF \(is-a (C, D, p) \land\)

\[\text{covering}(D, C_1...,C_n) \land (C_i <> C \Rightarrow not \ x \ in \ C_i)\]

DO Abort

DA5 \(\Rightarrow\) Group elements must satisfy the grouping predicate

ON \(g\) gr-insert \((x)\) IF \(is-elem (C, G, p)\)

\[g \ in \ G \land not p(x,g)\]

DO Abort

DA6 \(\Rightarrow\) For a covering grouping, a new instance of the element class must be in a group

ON \(C\) create \((x)\) IF gr-covering \((C, G)\)

\[g \ in \ G \land not x \ in \ g\]

DO Abort

2. System Side Effects

For system side effects the action part of the rule, given in the DO clause, contains a primitive operation necessary to maintain the integrity of the database.

- **Generalization**

SSE1 \(\Rightarrow\) An instance of a subclass must be also in the superclass

ON \(C\) create \((x)\) IF \(is-a (C, D) \land not x \ in \ D\) DO \(D\) create \((x)\)

SSE2 \(\Rightarrow\) A create may not violate the disjoint generalization

ON \(C\) create \((x)\) IF \(is-a (C, D) \land\)

\[\text{disjoint}(D, C_1...,C_n) \land (C <> C_i \Rightarrow x \ in \ C_i)\]

DO \(x\) delete \((C_i)\)

SSE3 \(\Rightarrow\) A delete may not violate the covering generalization
ON $x$ delete ($C$) IF is-a ($C$, $D$) $\land$ covering ($D$, $C_1$,...,$C_m$) $\land$ ($C_i <> C$ $\Rightarrow$ not $x$ in $C_i$) DO $x$ delete ($D$)

SSE4 $\Rightarrow$ A new instance of a generic class must be inserted in the compatible subclasses

ON $D$ create ($x$) IF is-a ($C$, $D$) $\land$ $p_c(x)$ DO $D$ create ($x$)

SSE5 $\Rightarrow$ An object deleted from a generic class must be deleted from all its subclasses

ON $x$ delete ($D$) IF is-a ($C$, $D$) $\land$ $x$ in $C$ DO $x$ delete ($C$)

SSE6 $\Rightarrow$ If, as consequence of the change in a relationship, an object cannot remain in its subclass, it must be removed to the compatible new subclass

<table>
<thead>
<tr>
<th>Aggregation</th>
</tr>
</thead>
</table>

SSE7 $\Rightarrow$ An aggregate cannot lose one of its components

ON $x$ delete ($C$) IF is-part ($C$, $D$) $\land$ aggregate ($y$, $x_1$,...,$x_n$) $\land$ $y$ in $D$$\land$$x$ $\in$ {$x_1$,...,$x_n$} DO $y$ delete ($D$)

SSE8 $\Rightarrow$ For an aggregation by relationship, the removal of a relationship eliminates the corresponding aggregate

ON $x$ remove ($r$, $y$) IF aggregate-$r$ ($D$, $C_1$, $C_2$, $r$) $\land$ $x$ in $C_1$ $\land$ $y$ in $C_2$ DO <$x$, $y$> delete ($D$)

SSE9 $\Rightarrow$ For an aggregation by relationship, the establishing of a relationship creates the corresponding aggregate

ON $r$ establish ($<$x$, y$>) IF aggregate-$r$ ($D$, $C_1$, $C_2$, $r$) $\land$ $x$ in $C_1$ $\land$ $y$ in $C_2$ DO $D$ create ($<$x$, y$>)

SSE10 $\Rightarrow$ The creation of an aggregate creates also its parts

ON $D$ create ($y$) IF aggregate ($D$, $C_1$,...,$C_m$) is-part ($x_i$, $y$) DO $C_i$ create ($x_i$)
SSE11 ⇒ The creation of an aggregate defined by a relationship, also the relationship must be established

\[
\text{ON } D \text{ create } (y) \quad \text{IF} \quad \text{aggregate-r} (D, C_1, C_2, r) \quad \text{DO } C_1 \text{ create } (x_1) \land \\
\text{is-rel} (C_1, r, C_2) \quad C_2 \text{ create } (x_2) \land \\
\text{is-part} (x_1, y) \land \text{is-part} (x_2, y) \quad x_1 \text{ establish} (<r, x_2>)
\]

**Grouping**

SSE12 ⇒ If \( p(x, g) \) holds and \( x \) is inserted in the element class, it must also be added to \( g \)

\[
\text{ON } C \text{ create}(x) \quad \text{IF} \quad \text{is-elem}(C, G) \quad \text{DO } g \text{ gr-insert}(x) \\
g \text{ in } G \land p(x, g)
\]

SSE13 ⇒ In a covering grouping, if an object is eliminated from a group, it must also be eliminated from the elements class

\[
\text{ON } x \text{ gr-delete}(g) \quad \text{IF} \quad \text{gr-covering}(C, G) \land x \text{ in } C \quad \text{DO } x \text{ delete}(C)
\]

SSE14 ⇒ If an object is deleted from the elements class, it must also be removed from the groups

\[
\text{ON } x \text{ delete}(C) \quad \text{IF} \quad \text{is-elem}(C, G, p) \land x \text{ in } g \quad \text{DO } x \text{ gr-delete}(g)
\]

SSE15 ⇒ The elements of a new group must be on the element class

\[
\text{ON } G \text{ create}(g) \quad \text{IF} \quad \text{is-elem}(C, G) \land \\
P(x, g) \land \text{not } x \text{ in } C \quad \text{DO } C \text{ create}(x)
\]

SSE16 ⇒ In a covering grouping, the removal of a group removes also its elements from the element class

\[
\text{ON } g \text{ delete}(G) \quad \text{IF} \quad \text{is-elem}(C, G) \land \\
\text{gr-covering}(C, G) \land x \text{ in } g \quad \exists h (x \text{ in } h \land h \text{ in } G \land h < > g)
\]

SSE17 ⇒ In a disjoint grouping, a new element of a group must be eliminated from other groups

\[
\text{ON } g \text{ gr-insert}(x) \quad \text{IF} \quad \text{disjoint}(C, G) \quad \exists h (x \text{ in } h \land h < > g) \quad \text{DO } x \text{ gr-delete}(h)
\]

**Relationships**

SSE18 ⇒ The delete of an object must remove all its relationships

\[
\text{ON } x \text{ delete}(C) \quad \text{IF} \quad \text{is-rel} (x, r, y) \quad \text{DO } x \text{ remove} (r, y)
\]
EXTENSION

In this section we show how the system can be expanded in order to include new functionalities.

Suppose we want to expand the existing data model with two new concepts. First, we would be able to model applications who needs valid-time temporal objects [TCG+93]. In order to consider Temporal Databases, the following additions are needed.

Classes and relationships may be defined as temporal, by the predicates:

- $t\text{-Class}(C', C) \Rightarrow \text{aggregation}(C', C, I)$
  A temporal class $C'$ is obtained aggregating a ordinal class $C$ with the class of time intervals $I$.

- $t\text{-Rel}(r', C_1, C_2) \Rightarrow \text{aggregation}(r, C_1, C_2, I)$
  A temporal relationship $r'$ between two classes $C_1 \in C_2$ aggregates these two classes with $I$

Now we define a dynamic axiom that avoids the creation of objects with valid time intersecting the valid time of this object in the database.

DA7 $\Rightarrow$ In a temporal class it is not allowed to create instances with valid-time intersecting with its existing time

ON $C' \text{ create} (<x, (t_{1}, t_{2})>)$ IF $t\text{-Class} (C', C, I) \land <x, (t'_{1}, t'_{2})> \text{ in } C' \land \text{overlap } ((t_{1}, t_{2}), (t'_{1}, t'_{2}))$ DO Abort

As side effects we consider the following. The first one allows the user to define an object without considering if it is temporal or not. He just creates the object and the side effect attaches the default valid time.

SSE20 $\Rightarrow$ If a temporal object is created, its creation time must be established

ON $C \text{ create}(x)$ IF $t\text{-Class} (C', C) \land t = \text{now}$ DO $C' \text{ create}(<x, (t, \text{now})>)$

SSE21 $\Rightarrow$ In a temporal class, a deleted object is moved to the past

ON $x \text{ delete}(C)$ IF $t\text{-Class}(C', C) \land t = \text{now}$ DO $<x, (t_{0}, \text{now})> \text{ delete}(C') \land C' \text{ create} (<x, (t_{0}, t)>)$
On the creation of a temporal relationship, the time must also be created

$$\text{ON } x \ \text{establish}(<r, y>) \ \text{IF } t-\text{Rel}(r, C_1, C_2) \land t = \text{now} \ \text{DO } r \ \text{create}(<x, y, (t, \text{now}>)$$

The removal of a temporal relationship must be moved to the past

$$\text{ON } <x, r, y> \ \text{remove} \ \text{IF } t-\text{Rel}(r, C_1, C_2) \land t = \text{now} \ \text{DO } <x, y, (t_0, \text{now})> \ \text{delete} \ <x, y, (t_0, \text{now})> \ \text{in } R' \ \text{DO } R' \ \text{create}(<x, y, (t_0, t)>)$$

Suppose now that the user has an application with classes EMPLOYEE, SALARY and STATUS and with the relationships EMPLOYEEhas-salarySALARY and EMPLOYEEhas-statusSTATUS. We want to state a User Side Effect (USE) which guarantees that, ever when the salary of an employee becomes greater that $100,000 his status must be fixed to be '4'. Therefore we have:

USE1: ON e establish(has-salary, s) \text{IF } s > 100,000 \land e\.has-status=s_0 \land e\.has-status=s_0=e\.establish(has-status, '4') \land s_0 \neq '4'

4. Conclusion

In this chapter a system for the enforcement of implicit and explicit integrity constraints for a complex object oriented data model has been presented. This system is flexible enough to be used for extensible data models, also known as open data models, which allows the incorporation of new capabilities in the data model, in order to facilitate the modelling of new application categories.

The idea of system side effects has first been developed together with the Temporal Hierarchic Model - THM [Sch83], later improved with object capabilities [Sch91]. A running prototype has been implemented together with a mapping module, which transforms an Object Schema to a relational one [daSil00]. According to the relational schema generated by this mapping, the rules are converted to adequate triggers of the DBMS and are executed on the database.

References


[KLP00] *


[RS 97] *


