

AIM - AN INTEGRITY MONITOR FOR THE DATABASE SYSTEM INGRES

A. B. Cremers and G. Domann

Informatik VI, Universität Dortmund, FRG

Abstract: Entirely programmed in EQUOL, AIM consists of a batch component for periodic integrity control and an interactive component which provides an instantaneous semantic control mechanism at the user interface. For certain simple updates AIM implements a prevention strategy, whereas, in particular for transactions, a detection and recovery strategy is observed. Both static and dynamic integrity constraints may be specified on the basis of QUEL qualifications. Integrity constraints are stored in a separate integrity constraint base which consists of ordinary UNIX files. The AIM concept and the embedded approach by Stonebraker are largely complementary.

1. Introduction

The design goal for AIM is to extend the INGRES user interface with a nontrivial semantic control mechanism. Integrity constraints are partitioned into two categories: conditions that are to be checked periodically and others that are to be examined with every relevant update request. Each category is stored in a separate UNIX file which serves as an **integrity base** for an AIM batch resp. interactive component.

In terms of the integrity subsystem model by Hammer and McLeod [4], AIM may be briefly characterized as follows:

Descriptive language: We have developed a simple syntax for integrity constraints. The qualification part of a constraint is to be specified in terms of the database language QUEL of INGRES. Besides a QUEL predicate, every integrity constraint contains a definition of the objects concerned as well as the information required for identifying and categorizing the constraint.

Translation: AIM incorporates a parser for this syntax; the translation of QUEL predicates is done by INGRES.

Integrity constraint checker: This component is implemented by a set of procedures which identify the relevant data objects for each constraint to be checked. The violation of a given integrity constraint is detected by way of submitting a query whose qualification is the negation of the QUEL predicate stated in the constraint. The component distinguishes between periodic and instantaneous control.

Action component: The actions of this component are triggered by the integrity constraint checker. In many cases the output of the constraint along with the violating

tuples will be required. In addition, the interactive subsystem is capable of resetting violating updates. (The present version of AIM, however, does not yet provide a facility for automatically resetting compound update requests, i.e. transactions.)

Validity checker: A subsequent version of AIM will incorporate a set of routines for checking the data-independent validity of the integrity constraints defined.

AIM has been successfully implemented by G. Domann on a DEC PDP 11/60 computer system running under the UNIX operating system. The present paper describes the design and implementation of AIM, operational September 1982 under UNIX, Version 6, and INGRES [7], Version 6.1/10.

2. Scope and format of integrity constraints

The normal mode of integrity control is **postoperative**, that is, all relevant constraints are validated after the effect of an update request (APPEND, DELETE, REPLACE) has been computed. For efficient retrieval of relevant constraints, every integrity constraint is uniquely named and contains identification of all data objects (relations) to which it refers. Old tuples and values, as well as newly inserted data together with their keys ought to be saved in an appropriate way so as to enable the system to recover upon violation of a constraint. Unless specified otherwise, the decision as to which constraints have to be validated is based on the data objects alone, and not on the kind of update operation to be performed. Typical instances are conditions based on the mean value of a given numeric domain, or foreign key conditions.

Consider the (quite standard) example of a database consisting of the following relations. Primary key in each relation is the attribute "number:"

```
city(number, name, distance)
dept(number, name, floor, room, manager)
employee(number, name, salary, manager, startdate)
item(number, name, dept, price, qoh, supplier)
pers_data(number, city, fam_status, children, birth-
            date, taxclass)
supplier(number, name, city)
```

AIM requires a nonempty list of tuple variable declarations, e.g.:

"For no sales item the quantity-on-hand exceeds 100 unless the supplier of that item is located more than 2,500 miles away:"

```
I536 item : i, supplier : s, city : c ;
    i.qoh < 100 or i.supplier != s.number or
    s.city != c.number or c.distance > 2500
```

The use of aggregate functions in constraints is illustrated by the following example:

"The average salary of employees with the same manager must be less than 30000 \$"

```
I392 employee : e; avg(e.salary by e.manager) > 30000.
```

There may be more than one tuple variable declared for a given relation, e.g. in: "All departments have different names:"

```
I465 dept : d1, dept : d2; d1.name != d2.name or
    d1.number = d2.number
```

To show a foreign key condition, let the constraint be that every city value in the supplier relation agree with a number entry in the city relation:

```
I608 supplier : s, city : c; count(s.city) =
    count(s.city where s.city = c.number).
```

ON-Conditions

Conditions that refer to single values or to dependencies within a tuple (validity constraints) may be controlled **before** a given update request takes place: tuples other than the ones to be changed are not affected. AIM allows for an important class of **preoperative** constraints by providing a facility to specify the kind of update for which such a constraint is to be validated. The AIM syntactical element for this facility is termed **ON-condition**. Since ON-conditions are being introduced for the sake of economy, and their validation requires temporary relations, it seems reasonable to restrict them to validity constraints.

A constraint "ON DELETE," of course, refers to the tuples **before** execution of the update request: Just for the tuples that qualify, the deletion will take effect. For instance, consider the constraint that only those sales items may be deleted whose quantity-on-hand is zero:

```
I105D item : i; i.qoh = 0.
```

The ON-condition is specified immediately after the constraint name. To show a constraint "ON APPEND" consider the case that a new employee joins the enterprise:

```
I134A employee : e; e.startdate = 1983.
```

In the case of APPEND, the qualification refers to the tuples to be inserted. Similarly, for REPLACE, the qualification pertains to the new values: the update request is to be rejected unless the new values satisfy the constraint.

All constraints are immediate in the sense that they are supposed to hold after every elementary update (A, D, R). AIM, however, also accommodates transactions, and for transactions all constraints, except for those with ON-conditions, change into delayed assertions. Relevant delayed assertions are checked upon the end of a transaction.

For integrity violations AIM implements the following maintenance strategy: Either the update request does not take effect (ON-conditions, even inside transactions) or all changes implied by the violating request are undone. In the second case, no further change is allowed on the objects affected until the undo is complete. Clearly, the primary keys of these objects must be uniquely identifiable. For this purpose, AIM provides the option of an automatic check of the primary key condition. We stress that the present version of AIM does not provide for triggered undo in the case of a transaction: Delayed assertions are "soft" assertions. Stronger measures depend on whether an ON-condition has been specified (immediate assertion) and, in the case of elementary updates, on whether the automatic primary key checking option has been switched on.

3. Comparison with an embedded approach

In the embedded approach by Stonebraker [8], defined integrity constraints are stored in a catalogue, in proximity to the description of the data objects to which they refer. Constraints are implemented by means of **query modification**:

Example: Consider the constraint

```
RANGE OF e IS employee
INTEGRITY CONSTRAINT IS
    e.salary > 12000
```

The INGRES update request

```
RANGE OF e IS employee
REPLACE e (salary = e.salary - 1500)
    where e.name = "Jones"
```

is transformed into

```
RANGE OF e IS employee
REPLACE e (salary = e.salary - 1500)
    where e.name = "Jones"
    AND e.salary - 1500 > 12000.
```

The **automatic** integration of the constraint in the request guarantees the **prevention** of certain integrity violations. In order to obtain all tuples that satisfy the original qualification but whose alteration would violate the constraint, a second query would have to be dispatched which outputs those tuples in an error relation:

```
RETRIEVE INTO Rerror (e.all)
    where e.name = "Jones"
    AND NOT (e.salary - 1500 > 12000)
```

The fact that integrity control is based on the **new** values may, however, entail serious problems, e.g. in connection with incomplete updates, duplicate-respecting aggregates, deletion, state transition and interrelational constraints; periodic and delayed modes of integrity control are ruled out [3].

In comparison, AIM largely mitigates those problems while preserving some of the advantages of a subsystem designed by the embedded approach.

(1) Integrity control in AIM, with the exception of constraints ON DELETE, is **not** based on query modification. The search for tuples violating a constraint is specified in terms of a RETRIEVE query whose

qualification is the inverted predicate. Since this task is entirely delegated to INGRES, complex constraints are handled, from the subsystem point of view, as easily as simple ones.

- (2) An update causing a violation is either aborted (violation of an ON-condition) or its effect undone. For the only form of a modified qualification (constraints ON DELETE) AIM provides the option for the user to either proceed with the modified qualification or to abort in order to avoid an undesirable partial deletion.
- (3) Due to the separation of request execution and integrity control there is no difficulty with aggregate functions. Due to the implementation of the primary key checking option AIM restricts insertions to a single tuple per APPEND. However, a single REPLACE may change several tuples as long as primary keys are not affected.
- (4) The AIM feature ON DELETE closes a serious gap of the integrated approach. This is probably the most important case of a preoperative constraint.
- (5) All constraints for which no ON-condition has been specified are validated after execution of the request. Therefore, AIM is capable of detecting the violation of e.g. foreign key conditions due to deletions. Transactions can be handled by means of delaying the point of control. Periodic control also becomes possible, an important point in view of complex constraints.
- (6) The present version of AIM does not accommodate state transition constraints but this feature is not hard to incorporate.
- (7) Constraints may be arbitrarily complex, in principle. Their validation may encompass sets of tuples from several relations. Implementation-dependent restrictions on lengths, number of items etc. are rather liberal and accommodate most practical examples.

We conclude the discussion by an evaluation of the AIM design in the light of advantages conceivable for the integrated approach.

Simple constraints: Query modification is presumably more efficient for very simple conditions, whereas for more demanding constraints the AIM mechanism is clearly preferable.

Storage: The decision to implement the integrity base by means of UNIX files entails ease of change. Different integrity bases for several user views may be supported.

Adequacy of effort: With ON-conditions just the tuples to be changed have to be checked. Compared to Stonebraker's proposal, there is greater flexibility in that constraints may be specified e.g. ON APPEND but not ON REPLACE. On the other hand it seems desirable to adopt Stonebraker's more flexible unit of control for a more advanced version of AIM.

Subsystem security: A separate integrity base, of course, requires separate security measures. For the present version of AIM we have not taken any steps to increase subsystem security beyond the standard protection provided by the system.

Extendability: Conceptual separation favors change. It is not only easy to extend the integrity base, the modular design of the AIM software also supports quick changes in control strategy. For this purpose it is decisive that the monitor be independent of the extant INGRES processes. As a further consequence, AIM is easily adaptable to different versions of the database system. Finally, AIM could perfectly well coexist with an embedded integrity control based on query modification.

4. Structure of the integrity monitor

As mentioned in the Introduction, AIM consists of a batch component and an interactive component. The latter builds on the first, and requires more discussion.

4.1. Batch component

The purpose of this program is to detect violations of integrity constraints and, in that case, to identify the inconsistent parts of the database. The constraints to be checked by the batch component are stored in a separate file. Typically these are either low priority or more demanding constraints which are to be checked in greater intervals. If the batch program is to be executed not only by user activation but also automatically in periodic intervals, a simple UNIX command procedure takes care of this. AIM provides the option for the user to execute the batch component in dialogue mode. In this mode the user may watch the output on the screen and, for every constraint in the file, has the choice of either requesting the constraint to be checked or skipping it (or switching the monitor off). In view of highly complex constraints this option seems to be very useful. If the mode of activation is batch then all constraints in the file are checked and results written in an output file. In this mode it is possible for UNIX to process the component in the background while the user may execute other actions in the foreground.

4.2. Interactive component

The interactive component uses a protocol file in which to log all database requests, the changes performed, the integrity constraints selected, all detected violations, and the recovery measures taken. If necessary, the protocol file together with the last system checkpoint allows the reconstruction of a consistent database. However, in all cases, except for transactions, AIM will be capable of maintaining a consistent database on its own power. We mention that, in analogy to the key check, the protocol option may be switched off by the user. (Yet, normally we would expect the option to be in effect.)

Our description follows the menu technique offered by the interactive component. After giving the names of the database and integrity base to be accessed, the user may choose between the following modes of operation:
read / write / transaction / quit.

The transaction mode is independent of read and write modes: As part of a transaction arbitrarily many read and write requests (i.e. read/write mode alternations) are possible. The selection of the next INGRES command and the transmission of the parameters is the same inside and outside of transactions. The user remains in the mode selected until he requests a change of mode instead of the execution of a further INGRES command. A transaction

request is finished by changing the mode.

In **read mode** the commands **HELP**, **PRINT**, and **RETRIEVE** are possible. The parameters for the former two (i.e. relation names) are obtained in dialogue with the user. Upon **HELP**, AIM outputs the dictionary information stored for the relation specified (name, owner, type, storage structure, size etc.). Upon **PRINT**, the current instance of the relation is displayed. Inputs for **RETRIEVE** are given in the same way as required by the ordinary INGRES dialogue monitor. In this way the full bandwidth of the **RETRIEVE** command is made available without excessive dialogue overhead: There is no buffering of single parameters and the transmission of the complete command to INGRES is simplified. The original string is transmitted through a **UNIX pipe**.

In **write mode** ("modify"), this efficient way of transmission has not been possible due to the fact that the use of temporary relations for the validation of preoperative constraints (ON-condition) necessitates several consecutive changes of relation names and tuple variables. Therefore, the transmission of the INGRES commands for a write request is programmed in **EQUEL**, where the necessary changes can be accommodated by program variables. The INGRES commands enabled in write mode are **APPEND**, **DELETE**, and **REPLACE**. Every command has its own control procedure in AIM. When a constraint has been determined to be relevant it is validated, by means of a triggered **RETRIEVE** with the predicate inverted, against either the changed database or (ON **APPEND** resp. **REPLACE**) a temporary relation of new tuples.

In **transaction mode**, there are the following differences with respect to the processing of update commands: As soon as an update has taken effect in the database, it is considered to be complete. That is, for **APPEND** and **REPLACE** only the ON-conditions are checked, for **DELETE** query modification is employed. Instead of the immediate subsequent control of postoperative constraints, the names of the relations modified are stored in a list. Only after the end of transactions, the relevant postoperative constraints are checked. The present version of AIM does not support automatic recovery from violations of postoperative constraints caused by a transaction. Clearly, keeping a copy of all relations affected is too much inefficient. For the time being, AIM in this case relies on the support provided by the log subsystem.

We note that the load modules of AIM require some 17 KBytes and 35 KBytes for the batch and interactive components, resp.

References

1. Eswaran, K.P.; Chamberlin, D.D.: Functional Specification of a Subsystem for Data Base Integrity, Proc. VLDB, Framingham, 1975, 48-68.
2. Gardarin, G.; Melkanoff, M.: Proving Consistency of Database Transactions, Proc. VLDB, Rio de Janeiro, 1979, 291-298.
3. Härder, Theo: Implementierung von Datenbanksystemen, Carl Hanser Verlag, München, Wien, 1978.
4. Hammer, M.M.; McLeod, D.J.: Semantic Integrity in a Relational Data Base System, Proc. VLDB, Framingham, 1975, 25-47.
5. Hong, Y.C.; Stanley, Y.W.: Associative Hardware and Software Techniques for Integrity Control, ACM TODS 6, 1981, 416-440.
6. Melo, Rubens, N.: Monitoring Integrity Constraints in a CODASYL - like DBMS, Proc. VLDB, Rio de Janeiro, 1979, 209-218.
7. Stonebraker, Michael; Wong, Eugene; Kreps, Peter; Held, Gerald: The Design and Implementation of INGRES, ACM TODS, 1976, 189-222.
8. Stonebraker, Michael; Implementation of Integrity Constraints and Views by Query Modification, ACM SIGMOD, San José, 1975, 65-78.