DESIGN AND IMPLEMENTATION

OF AN EXTENDIBLE INTEGRITY SUBSYSTEM.

Eric Simon and Patrick Valduriez

SABRE Project. INRIA
BP.105, 78153 Le Chesnay-cedex, France

ABSTRACT

This paper presents a powerful integrity subsystem, which is implemented in the SABRE database system. The specification language is simple. The enforcement algorithm is general ; in particular, it handles referential dependency and temporal assertions. Specialized strategies efficiently treat each class of assertions. The system automatically manages integrity checkpoints. Also, an efficient method is described for processing assertions involving aggregates. An analysis exhibits the value of the algorithms. It is shown that, in general, this method is better than the query modification method for domain assertions. Measures have also been done for giving the cost added for controlling integrity in comparison with the cost of the request itself.

1. INTRODUCTION

An important aim of a database system is to guarantee database consistency. A database state is consistent if the database satisfies a set of assertions, called semantic integrity assertions. These assertions are specified by the user. Maintaining a consistent database requires various functions such as concurrency control, reliability, protection and semantic integrity control. Semantic integrity control ensures database consistency by rejecting update transactions which would lead to inconsistent states of the database. In other words, the updated database must satisfy the set of integrity assertions.

The main works on this subject focus on formalizing the concept of semantic integrity, such as the model which is based on abstract data types proposed by [BROD78], and formulating and classifying integrity constraints [HAMM75]. Also, several methods for controlling semantic integrity have been proposed.
The main problem encountered is generally its high processing cost. Different solutions have then been given for designing an integrity subsystem by combining optimization strategies. Their purpose is to restrict the number of assertions to test, to simplify them at enforcement time and to improve their processing cost by using the capabilities of the database system.

Relevant solutions based on heuristics or other strategies have been proposed to efficiently handle certain classes of integrity constraints such as domain of an attribute or constraints involving aggregate expressions.

These methods are based on simplification techniques [BLAU81, NICO82, STON75] or on particular tools, such as Hoare logic [GARD79] or entity-relationship model [HAMM78], used for data, transaction and assertion definitions. However, the implemented solutions suffer from a lack of generality since they are restricted to a particular set of assertions. Also, they often are little extendible because of their particular implementation in a system environment. Finally, a few complete cost analysis exists.

This paper deals with the design and implementation aspects of a general and extendible integrity subsystem that integrates specialized strategies for each class of assertions. This subsystem is implemented in the SABRE database system [GARD83], actually running on MULTICS at INRIA. The architecture presented in section 2 locates the place of the integrity subsystem in the SABRE system. The generality of the proposed method is illustrated in section 3 by giving a formulation and a classification of the accepted assertions. Section 4 describes the concepts of the method in comparison with existing methods. The enforcement algorithm on update requests is specified . The new features of the method , discussed in a qualitative analysis are, in particular, the handling of temporal assertions and integrity checkpoints. The last section is a quantitative analysis of the enforcement algorithm, by giving cost functions for the main classes of assertions. A comparison with the query modification method for enforcing domain constraints shows that our method is generally better . Finally, measures performed with the SABRE system exhibit the processing cost of the enforcement relatively to the processing cost of the update itself.

2. SYSTEM ARCHITECTURE

The environment in which the integrity subsystem is included is the SABRE database system [GARD83]. SABRE is actually implemented on MULTICS at INRIA and is being ported on a multi-microprocessor architecture. Several features of the system architecture, such as maintaining a workspace for transactions, will be shown as useful tools for the integrity processing. The SABRE system is organised in a logical machine and a physical machine, as portrayed in figure 1. The logical machine realizes the mapping functions between the user interface and the relational algebra interface, which is the input of the physical machine. The manipulated objects are meta-data such as view predicates or statistic data about relations. The physical machine is a storage and retrieval subsystem which works on underlying data stored on disk units. Some repetitive operations of the physical machine can be optimized by using specialized components while those of the logical machine do not need it.

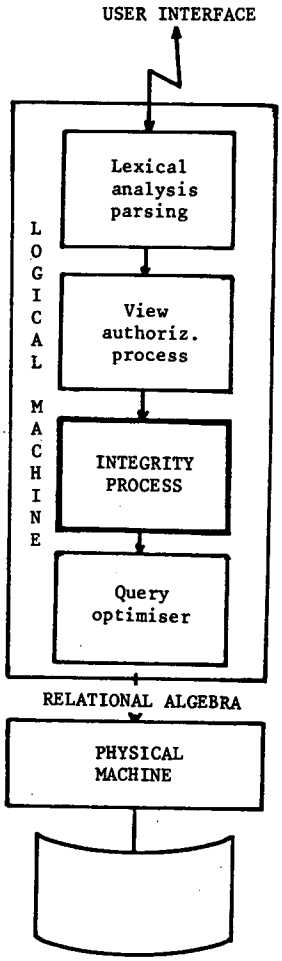

Fig.1 : Architecture of the SABRE system

The logical machine is composed of four main components. The first component is the most external of the system. It performs the lexical analysis and the parsing of user queries , expressed in a SQL like language [ASTR76], and presents the results to the user. The second component manages the views. It mainly maps requests on derived relations into requests on underlying relations, by applying an algorithm based on query modification [STON75]. It also checks authorizations on relations. The next component is the integrity subsystem, kernel of this paper. Its first task is to process definitions of integrity assertions, which are expressed as query qualification predicates. Assertions may be defined at anytime during the lifetime of a relation. Then, the compatibility between the integrity constraint and the current state of the relation is controlled. Assertions are stored in compiled form in a meta-base relation. The other task is to enforce semantic integrity of data when updating a database. The set of tuples violating integrity constraints are given back to the user with some explanations about their rejection. The last component of the logical machine is a query optimiser. It decomposes a multi-relation query into a tree of relational operations. It then restructures the tree in an optimal tree by using statistic informations about relations and indexes. The selected tree is represented by an execution plan, which consists in relational operators and synchronization variables.

The physical machine includes several components whose the main functions are herein remainded. It first realizes the relational operators for querying and updating databases. The operators are optimized by specialized components such as filters or join processors [VALD84]. A multidimensional access method based on predicate trees [GARD84] is also employed to restrict the search to only relevant data. The other basic function is the management of transactions, which includes algorithms for concurrency control [VIEM82], deferred updates and recovery facilities. The last function of the physical machine is to manage a virtual memory where transactions workspaces are maintained. A transaction consists in several requests. The workspace associated with a transaction insures that updates executed by a request i are visible by all requests i+j. The absence of such a strategy implies each request in a transaction to see the database in the same state as at the beginning of the transaction. This is avoided in most systems by an expensive solution that consists in committing each request and managing a "before" image log. The management of workspaces is based on differential files, updates being really committed at the end of a transaction. This strategy is optimized by using a large cache memory for storing temporary relations, resulting from operations on underlying relations.

3. SPECIFICATION OF INTEGRITY ASSERTIONS

An integrity assertion is expressed in the data manipulation language of SABRE, with an ASSERT command. The language is based on tuple

relational calculus. Each assertion is seen as a query qualification which is either true or false for each tuple in the cartesian product of the relations determined by tuple variables. Thus, the specification of an integrity assertion requires the definition of tuple variables and a query qualification referencing these variables. In a similar way to SYSTEM R [ASTR76], two key-words, NEW and OLD, are also introduced in order to express temporal constraints.

The integrity assertions can be classed into three categories depending on the complexity and the cost of their enforcement algorithms. For each of these categories, an enforcement algorithm and cost functions can be defined.

(a) individual assertions : these are mono-relation constraints that can be either mono-variables or multi-variables. The first ones only refer to tuples to be updated (e.g. domain or temporal constraint) while the others express intra-relation dependencies (e.g. functional dependency).

(b) set oriented assertions : these are multi-variable multi-relation constraints. They define structural properties of data represented by semantic links between relations, e.g. referential dependencies [DATE81] or inclusion dependencies expressing a concept of generalization such as "a drinker is a person".

(c) involving aggregates assertions : they need a particular processing because of the cost of evaluation of the aggregates.

Generally, an integrity constraint is seen as an assertion verified by the data. A set of assertions A being associated with a relation R, each update request u on R changing the state D of the database to the state Du must be such that Du satisfies A. Then, A must be chosen as small as possible. A solution proposed by several authors [BLAU81, CREM83, NICO82] is to isolate assertions that are not concerned with a type of update. Certain assertions, defining conditions on the domain of an attribute, only refer to tuples to be added or deleted to the database. Others, like referential dependencies, which specify that the presence of tuples in a relation R1 depends on the presence of tuples in a relation R2, must be validated when adding tuples in R1 or when deleting tuples from R2. More generally, to each assertion of class *a, b or c, one or several* integrity assertions can be associated with a relation and with a given type of manipulation chosen among insert, delete and replace. Thus, to an assertion of referential dependency from R1 to R2 corresponds an integrity assertion for inserting in R1 and an integrity assertion for deleting from R2. This approach is applied in SABRE. Each assertion concerns a relation for a given type of manipulation, which permits to limit the set of assertions to enforce.

We now illustrate the specification of integrity assertions by means of examples using the following toy database :

```
WINE (WINE#, VINEYARD, DEGREE)
PARTY ( DATE, CITY, WINE#, DRINKER, QUANTITY)
```

(a) specification of a domain constraint:
The degree of a wine is between 11 and 14.

```
V = WINE ;
ASSERT on V when insert
      V.DEGREE ≥ 11 and V.DEGREE ≤ 14 ;
```

(b) specification of a temporal constraint
The degree of a wine can only decrease.

```
V = WINE ;
ASSERT on V when replace
     NEW.DEGREE ≤ OLD.DEGREE ;
```

(c) referential dependency
A wine drunk at a party must exist.

```
V = WINE ;
P = PARTY ;
ASSERT on P when insert
   P.WINE# = V.WINE# ;
ASSERT on V when delete
   COUNT (V.WINE# where V.WINE#=P.WINE#) = 0;
```

(d) specification of a domain constraint :
Only those tuples in PARTY may be deleted whose drunk quantity is zero.

```
V = WINE ;
ASSERT on V when delete
   V.QUANTITY = 0 ;
```

4. ENFORCEMENT ALGORITHM

4.1 Principles

An integrity control method can be expressed by answering four questions: what, where, when and how to enforce.

(a) what to enforce

The type of accepted assertions and the choice of the set of assertions to enforce at each update have been specified in section 3.

(b) where to enforce

Two basic methods permit to reject inconsistent updates. The first method is based on detection of inconsistencies like in [ASTR 76]. The update u is executed and the database state D is changed to a state Du. The enforcement algorithm verifies that all relevant assertions hold in that state. If the state Du is inconsistent, the database must meet again a consistent state D°u or D. This approach leads to undo updates by using a log mechanism. The second method is based on prevention of inconsistencies, like in [STON75]. An update is executed only if it changes the database state to a consistent state. This approach is less expansive since it is seldom necessary to undo an update. Our method is based on prevention and guarantees that it is never necessary to undo. It is compared with the method of INGRES in section 4.3.

(c) when to enforce

Theoretically, integrity constraints should be enforced at transaction end, since the transaction is the basic unit of consistency, i.e an atomic unit of concurrency, security and

integrity. A transaction consists in several requests. The taxonomy of integrity assertions exhibits that certain types of assertions could be enforced when the update request is processed (e.g. individual assertions) while others must be checked at transaction end (e.g set oriented assertions). In the latter case, a deeper analysis permits to determine possible enforcement points in the transaction. The method is based on automatic checkpoints, which avoids a long transaction to be completely rejected. At each integrity checkpoint, the set oriented assertions are enforced. In case of inconsistency, the transaction is undone until its most recent chekpoint.

The integrity checkpoints are automatically determined within the transaction by the following algorithm. Let us consider G(R,A) a directed graph, where R is a set of relations of a database and A is a set of arcs representing set oriented dependencies between relations. G is stored in a compiled form in a relation of the metabase and is useful for other purposes. The detection of directed subgraphs permits to determine integrity checkpoints. G' is constructed by the successive requests in the transaction by associating to each request on relation x a set of arcs (x,y) where $y \in R$. In particular we can have x=y. If , for n successive requests, the graph $G'(R',A')$, where $R' \subset R$ and $A' = (x,y)$ associated with each request¶, is such that:

$\forall (x,y) \in G' \ \exists \ z \in R \ / (y,z) \in G'$,

then the end of the request n is an integrity checkpoint.

If the schema of the database is acyclic and if the requests in a transaction are ordered in dependency order (and deletions before insertions) then all the assertions involved by a given request may be enforced before processing the next one.

(d) how to enforce

The enforcement algorithm is the subject of section 4.2.

4.2 Enforcement algorithm

The enforcement algorithm is executed on an update request. First, a retrieval request whose predicate qualifies tuples to update is triggered. The result of the request is either a temporary relation that contains tuples to be inserted (TR1) or a temporary relation that contains tuples to be deleted (TR2) or both TR1 and TR2 if it is a replacement. A replace command is treated as a deletion of tuples present in TR2 followed by an insertion of tuples present in TR1. A set of assertions is then checked among this or these temporary relations by generating a retrieval request with a qualification composed of one or more inverted predicates of the constraints. The number of constraints added in the qualification depends on the management of errors. Our error mechanism allows to give back rejected tuples with specific messages for each violated constraint. The generated request returns a temporary relation TR3 containing rejected tuples. If TR3 is empty then the update is validated. A validated update becomes visible by the sucessive requests, using the workspace which is managed in the physical machine. If, at some integrity checkpoint i, a set oriented assertion does not hold, the transaction is back out to the point i-1 and all temporary relations associated with requests between i and i-1 are simply deleted. The validated tuples in temporary relations are really updated in the database at transaction end, using a two step commit protocol.

The method has common points with methods based on assertion simplification [BERN81,NICO82]. Only tuples that are to be updated are checked, since the database in a stable state is consistent and simplified assertions are generated.

4.3 Properties of the algorithm

4.3.1. Generality and Extendibility

All algebraic formulas can be expressed as integrity constraints. With a method based on query modification, integrity constraints concern the new values added or modified in the database. Algorithms consist in embedding conditions of constraints in the extended qualification of the update. Therefore, tuples are deleted from a relation with no integrity checking. Thus, it is not possible to express individual assertions "when delete" (such as assertion (d) in section 3) or to check referential dependencies on delete statement. In fact, tuples which are subject to update are not isolated; request execution is not separated from integrity control. Also, this removes the capability of specifying that an assertion must be tested after duplicate elimination [STON75]. Inconsistencies can then be introduced in the database when testing assertions involving aggregates without detection by the integrity subsystem. By isolating the tuples to be updated in temporary relations, our method allows detecting duplicates before testing assertions. Furthermore, the distinction between new tuples and old tuples makes efficient the control of temporal constraints. The request execution is separated from integrity control. Unlike INGRES, we can control referential dependencies by formulating constraints "on delete" and "on append".

The specification of our integrity subsystem implies the modularity of the enforcement algorithm. This algorithm permits to adapt the control strategy. In particular, it is possible to define specialized algorithms to each class of constraint, (individual, set oriented or involving aggregates). Thus, the method is extendible.

4.3.2. Qualitative analysis

We now analyze the design and algorithmic aspects of the integrity control method.

First, the possibility of expressing all the assertions for a given type of update significantly reduces the number of assertions to test at enforcement time. Otherwise, all assertions involving the updated relation must be enforced and some of them are not relevant (e.g referential dependencies) . One of the main advantages of the query modication method is that constraints are only tested on new tuples

concerned by the update. By isolating tuples concerned by updates in temporary relations and expressing assertions for a given type of update, our method provides the same advantage and furthermore permits to test assertions on "old" tuples . It is then possible to simplify the constraint evaluation. See for example the referential dependency of section 3, such a constraint would not be evaluated on both relations WINE and PARTY. If there is a deletion on WINE, the range of tuple variable V will be modified to specify the temporary relation containing the tuples to delete. The size of temporary relations is small relatively to permanent relations. Thus, the gain is important in comparison with other methods in wich such constraints are evaluated at transaction end on the whole modified database.

The extendibility of the method allows specializing the enforcement algorithm according to each class of constraints.

First, individual assertions are checked during request processing, before the request is validated for domain constraints and just after the validation of the request for the other individual constraints (multi-variable mono relation constraints). The method is adapted in case of a replace command and is very simple. The enforcement is done in two steps: checking for the delete assertions and then for the replace and insert assertions.

Secondly, the control of set oriented assertions is efficiently helped by integrity checkpoints. A partially enforced transaction can then be commited.

Finally, assertions involving aggregates are enforced during request processing by modifying them and maintaining aggregate values. The specification of an assertion involving aggregates generates redundant information maintained in the database for the knowledge of the current values of the aggregates. The assertion is then modified. Let us consider the following example:

```
V = WINE ;
ASSERT on V when insert
     AVG ( V.DEGREE ) < 12 ;
```

Two aggregates values are stored in a relation: the sum of degrees, SUMDEGREE and the number of wines, NBWINES. The qualification becomes:

$$\frac{\text{SUM (V.DEGREE) + SUMDEGREE}}{\text{CONST + NBWINE}} < 12$$

where CONST will be the number of tuples to insert. V specifies tuples to be inserted. Such a modified constraint can be tested during request processing. The aggregate is always evaluated on the temporary relation which is, in general, much smaller than the permanent relation .

The implementation of integrity control in a system maintaining workspaces for transaction leads to never undo validated requests. The gain is very high in comparison with a method based on detection and significant in comparison with other methods based on prevention (e.g AIM in [CREM83]).

5. ANALYSIS

5.1. Analysis criteria

The evaluation costs of the enforcement algorithm for various classes of constraints, pointed out in section 5.2, requires the precise definition of analysis criteria. The cost evaluation is based on the I/O number, that we consider as the basic and most critic performance parameter. An I/O operation is a page transfer between a disk unit and the cache memory. The following notations are needed to estimate the proposed algorithm:

n : number of pages of an operand relation R,
RS : selectivity factor of a restrict operation over R, defined as: size of Restrict(R)/n,
JS : selectivity factor of a join operation, over R1 of n1 pages and R2 of n2 pages, defined as:

size of (R1 join R2)/(n1 * n2).

The following architecture dependent parameters are also needed:

q : number of page frames in cache memory used for a complex operation (e.g join),
Tio : time for input or output a page,
Fr : probability of a page fault when accessing a page in the cache memory,
Fw : probability of a page fault when writing a page.

In SABRE, Fr and Fw are maintained low by the cache manager of the physical machine. The I/O time of a read operation is given by Tr where Tr= Fr * Tio. Similarly, the I/O time of a write operation is Tw, where Tw = Fw * Tio.

5.2. Evaluation

The I/O number of the enforcement algorithm is evaluated for two distinct types of assertions, that we consider as representative. These are domain constraints and referential constraints. The I/O number for reading assertions is supposed to be the same for each type of assertion and thus, is omitted. Furthermore, the method for enforcing domain assertions is compared with the well known query modification method.

5.2.1. Domain assertions

Domain assertions are enforced on insert, delete or replace requests. We distinguish two types of inserts. The simplest one is the insertion of tuples given by the user in a temporary relation. The other one is the calculated insertion of tuples already existing in the database and specified by a query qualification. Enforcing a domain assertion on replace or delete is equivalent to a calculated insertion.

The I/O cost of a simple insertion is incurred in reading the temporary relation of n pages to insert and in writing the relation resulting from a restrict operation using the assertion qualification of selectivity factor RS. Thus, the I/O number of a simple insertion is :

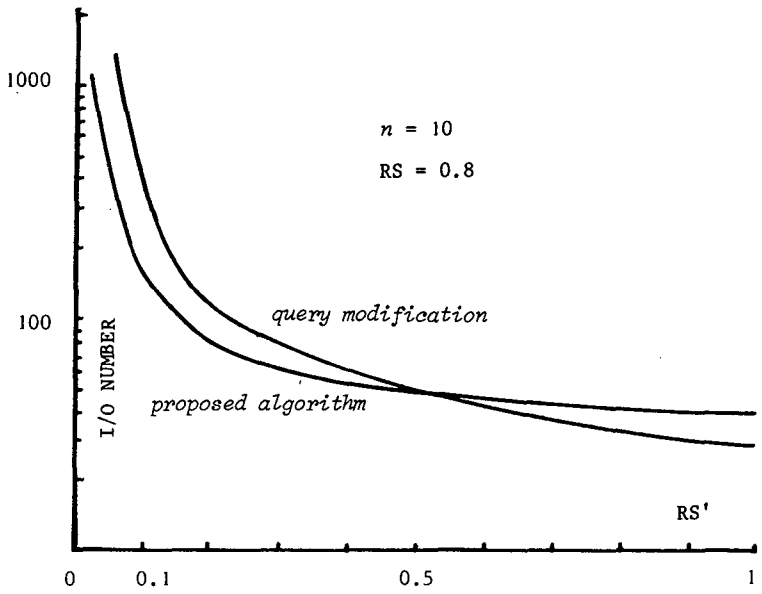

Fig 2. I/O number versus restrict selectivity factor

n * Tr + n * RS * Tw

A calculated insertion is done in two steps. First, a temporary relation is constructed by selecting a subset of the database by a retrieval. This temporary relation contains tuples to insert as in the case of a simple insertion. Secondly, as a result of the execution of the enforcement algorithm, a temporary relation contains rejected tuples. For simplicity, we consider a calculated insertion where the query qualification is a restrict operation over a permanent relation S with a selectivity factor noted RS'. We suppose that the number of pages of relation S is n/RS'. The result of the restrict operation returns n pages. RS is the selectivity factor of the assertion qualification. The I/O cost is given by the cost of the restrict operation , noted C1, and the cost of inserting the restricted relation, noted C2. C2 is the cost of a simple insertion. Then,the cost of the calculated insertion is C1 + C2, where :

$$C1 = (n/RS')*Tr + n*Tw$$
$$C2 = n*Tr + n*RS*Tw$$

We now compare our method with the query modification method [STON75], since it is generally considered as the best known method for handling domain constraints. The query modification method acts in two steps. The first step adds the assertion qualification to the restrict qualification by an AND operator and selects tuples satisfying the modified qualification. The second step is needed for retrieving tuples that do not match the integrity assertion to give them back to the user. A qualification with the restrict qualification AND the inverted assertion qualification of selectivity factor (1 - RS) is created for selecting those tuples. The I/O cost of the method is the sum of the I/O costs of the two selections, noted C1 and C2 :

$$C1 = (n/RS')*Tr + (n*RS*RS')*Tw$$
$$C2 = (n/RS')*Tr + (n/RS')*RS*(1 - RS)*Tw$$

Figure 2 illustrates the I/O number of the two methods versus varying restrict selectivity factors RS'. The parameters are fixed as follows : n = 10 , RS = 0.8 and Tr is supposed equal to Tw. Other calculations have been done for weaker RS and other values and similar results have been found. Our method is better as RS becomes smaller. When RS' becomes greater than 0.5, then the query modification is better. Remark that this value is high since it selects half of a relation. Generally, such a RS' is very small. The difference is due to the fact that the query modification method requires two selections on the permanent relation while our algorithm requires a single selection on the permanent relation and another selection on a smaller relation. When RS' is high, our second selection has a higher cost. Similar calculations can be done with a query involving a join operation instead of a restrict operation. The same principle is applied and the query modification method requires two joins, one for selecting relevant tuples and one for selecting rejected tuples. The same observation can be seen according to the selectivity factor of the join. Generally, selectivity factors are weak and our method is better than the query modification method.

5.2.2. Referential dependency assertions

Enforcing a referential assertion requires joining the temporary relation noted R1, that contains tuples to update, with the permanent relation noted R2, involved in the assertion. In SABRE, several join algorithms are implemented, each having an application domain [VALD84]. In case of referential assertion, the temporary relation is almost always small and fits entirely in cache memory. If the number of distinct values of the attributes involved in the assertion is small then the semi-join algorithm is the best one. Otherwise, the nested loop join algorithm is chosen. We suppose that this latter algorithm is applied since it is more general. The I/O cost of the algorithm is the cost of reading R1 and R2 and writing the result. R1 (of n1 pages) is read one time. R2 (of n2 pages) is read n1/(q - 2) times by using (q - 1) page frames for input pages, ((q - 2) for R1 and one for R2), and one for output the result. The result is of size n1*n2*JS pages where JS is the selectivity factor of the join corresponding to the inverted assertion. Thus, the I/O number for enforcing a referential assertion is

$$n1*Tr + n1/(q - 2)*n2*Tr + n1*n2*JS*Tw$$

The formula exhibits the value of having n1 small for decreasing the number of readings of R2. Isolating tuples to update in a temporary relation makes n1 small.

5.3. Measurements

Real measures have been done with the integrity subsystem of SABRE implemented on MULTICS on a toy database. Future measures using sophisticated benchmarks will be realised. Relations are relatively small (n = 5 pages of 4Kbytes), which is sufficient for obtaining relevant observations. These measures are usefull for knowing the added cost of integrity control in comparison with the request itself for different types of assertions. Therefore, execution times of requests with and without integrity control are measured. The time for reading assertions is taken into account. Selectivity factors of assertions are high (RS = 0.8) since a lot of tuples match the assertions. For measurements, the number of page frames in cache memory, q, is 4. Also, the I/O times Tr and Tw are approximately 4 and 14ms.

Figure 3 illustrates the execution time of a simple insert request with a domain constraint versus the size of the temporary relation to insert. Essentially, integrity control adds the reading of assertions. The increasing difference between the two curves is due to the increasing probability of page faults on writing.

Figure 4 depicts the execution time of a calculated insert with a domain constraint versus the selectivity of the retrieval request triggered for finding qualified tuples. The retrieval is supposed to be a join whose selectivity factor is JS. The time of integrity control increases like a restrict operation on a relation whose size is proportional to JS. Therefore, the cost added to the request increases as JS approaches 1.

Figure 5 describes the execution time of a simple insert request with a referential constraint versus the size of the temporary

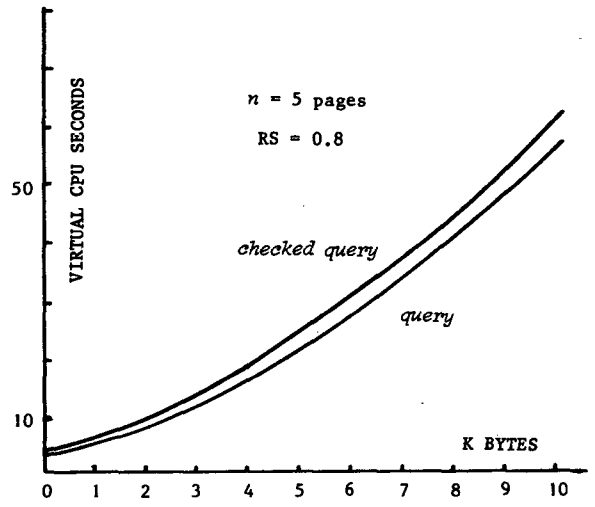

Fig 3. Execution time of a simple insert request with a domain constraint versus the size of the temporary relation to insert.

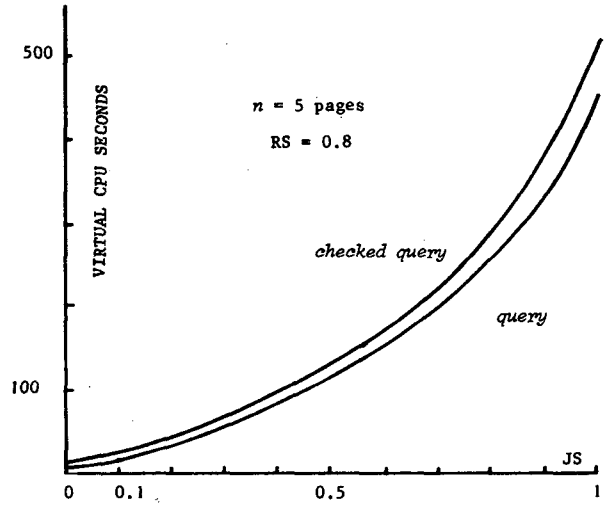

Fig 4. Execution time of a calculated insert with a domain constraint versus the selectivity of the qualification request.

relation to insert. Integrity control costs a join between the temporary relation and the permanent relation (of 5 pages). When the size of the temporary relation increases, the cost of integrity control becomes very high. In this case, the ratio execution time of integrity control versus request is the highest

Figure 6 shows the execution time of a calculated update (insert or delete) with a referential constraint versus the selectivity factor JS of the retrieval request, supposed to be a join. As JS approaches 1, the time of integrity control increases like a join operation on a relation whose size is proportional to JS. When JS = 1, the size of the temporary relation is the biggest (n = 25 pages) and the added cost is that of a join of n*n pages with a selectivity factor 1 -RS = 0.2. The total cost of the request is bounded by one and half times the cost of the request itself.

6. CONCLUSION

This paper has described the integrity subsystem implemented in the SABRE database system. This subsystem allows for an instantaneous enforcement of all temporal or static assertions. The method is preventive and exploits assertion simplification strategies for each class of assertions (individual, set oriented or involving aggregates). The integrity control of user transactions is greatly improved by an algorithm which automatically manages integrity checkpoints. An analysis shows the general superiority of our method in comparison to the well known query modification method. Real measures of the system exhibit the cost of integrity control relative to the cost of the update itself. It is clearly seen that referential dependency assertions give the highest cost.

The method here presented can handle a lot of types of assertions and can be still extended for managing dynamic assertions. In particular, using triggers or equation constraints to introduce a dynamic control of consistency is compatible with our approach. Equation assertions can improve database system capabilities in enhanced computing environments while triggers can increase the efficiency of integrity control for complex constraints. Finally, our integrity subsystem is modular and can be seen as a basic tool for integrating more capabilities in order to improve the usability of databases.

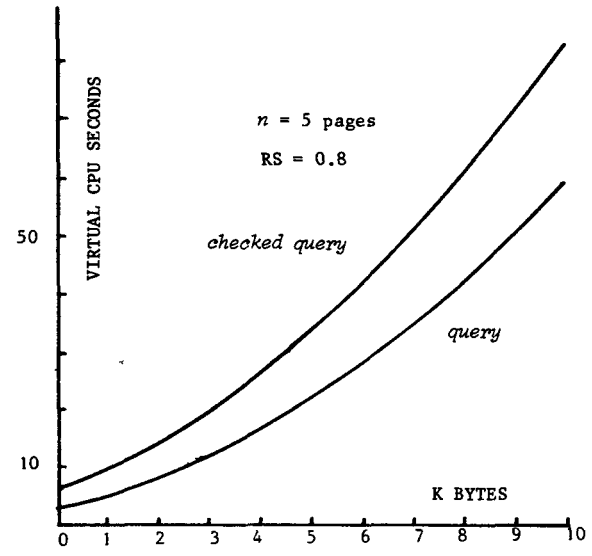

Fig 5. Execution time of a simple insert request with a referential constraint versus the size of a temporary relation to insert.

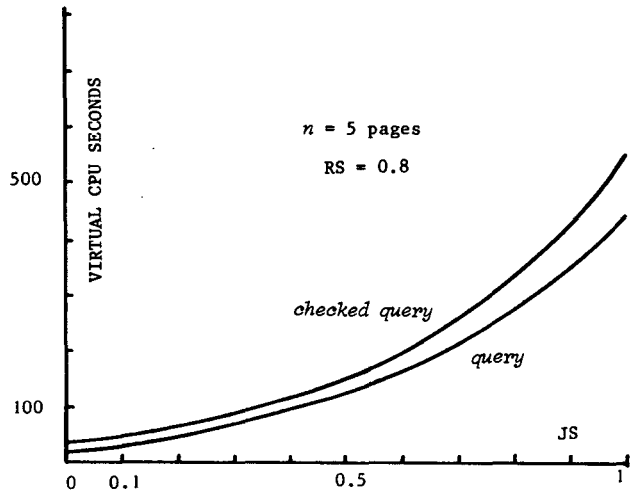

Fig 6. Execution time of a calculated insert with a referential constraint versus the selectivity factor of the qualification request.

REFERENCES :

ASTR76 M. M. Astrahan & al. "System R : Relational approach to database management" Publ. ACM-TODS Vol. 1, June 1976.

BERN82 P. Bernstein,B. Blaustein "Fast methods for testing quantified relational calculus assertions" Proc.ACM-SIGMOD Conf.,Orlando Florida June 1982.

BLAU81 B. Blaustein "Enforcing database assertions.Techniques and applications" Ph.D Harvard University,August 1981.

BROD78 M. Brodie "Specification and verification of database semantic integrity" Ph.D Toronto University,March 1978.

CREM83 A. Cremers,G. Domann "AIM:An integrity monitor for the database system INGRES" Proc. VLDB Conf.,Florence Nov.1983.

DATE81 C.J. Date "Referential integrity" Proc. 7th VLDB Conf.,Cannes Sept.1983

GARD79 G. Gardarin,M. melkanoff "Proving consistency of data base transactions" Proc.5th VLDB Conf.,Tokyo Sept.1979.

GARD83 G. Gardarin, P. Bernadat, N. Temmerman, P. Valduriez, Y. Viemont "Design of a Multiprocessor Relational Database System" IFIP° 9th World Computer Congress, Paris, September 1983.

GARD84 G. Gardarin, P. Valduriez, Y. Viemont "Predicate Trees" To appear in Proc. of the Computer Engineering Conference, IEEE Ed., Los Angeles, April 1984.

HAMM75 H. Hammer,D.J Mc Leod "Semantic integrity in a relational database system" Proc.1st VLDB ,Framingham 1975.

HAMM78 M.Hammer,S.K. Sarin "Efficient monitoring of data base assertions" Proc. ACM-SIGMOD Conf.,Austin June 1978.

NICO82 J.M Nicolas "Logic for improving Integrity checking in relational data bases" Acta Informatica,July 1982.

SIMO84 E. Simon, P. Valduriez "Design and implementation of an extendible integrity subsystem" Research Report No 271, INRIA-Rocquencourt, Feb 1984.

STON75 M. Stonebraker "Implementation of integrity constraints and views by query modification" Proc. ACM-SIGMOD Conf.,San Jose,1975.

VALD84 P. Valduriez, G. Gardarin "Join and Semi-Join Algorithms for Multiprocessor Database Machine" To appear in ACM-TODS, Vol. 9, No. 1, March 1984.

VIEM82 Y. Viemont, G. Gardarin "A Distributed Concurrency Control Algorithm Based on Transaction Commit Ordering" 12th Int. Conf. on FTCS, Los Angeles, June 1982.