December 2002

END-USERS' NON-CONSISTENT DATABASE VIEWS: ANALYSIS, MAINTENANCE, AND MANAGEMENT

David Chao
San Francisco State University

Robert Nickerson
San Francisco State University

END-USERS' NON-CONSISTENT DATABASE VIEWS: ANALYSIS, MAINTENANCE, AND MANAGEMENT

David Chao
San Francisco State University
dchao@sfsu.edu

Robert C. Nickerson
San Francisco State University
rnick@sfsu.edu

Abstract

Database views typically are maintained to be consistent with the database at a specific point in time. In some applications, however, users may prefer or require views that are not consistent with the database. Typically, this non-consistency is due to users' requirement of excluding qualified records from the views and/or including deleted records in the views. This paper examines the characteristics of non-consistent views and investigates their maintenance and management.

Introduction

Supporting an end-user's information needs for decision making is a major objective of database management. Database views have been used to support an end-user's information needs for decision making. A database view is the part of database in which a particular user is interested. It can be defined as any legal query operation over single or multiple base tables to meet a user's need. Typically, views are maintained to be consistent with the database reflecting the state of the database either at the current time or at a specific point in time as a snapshot view (Adiba and Lindsay, 1980).

Formally, a consistent view CV is the realization of the function CV(A, C, T, D(T)) where A is the sequence of relational algebraic operations needed to select the view, C is the set of logical conditions used with A, T is a point in time called the consistent point, and D(T) is the database state at T. Base table records satisfying the condition C are said to be qualified to the view. A consistent view reflects all the database updates relevant to the view up to the consistent point.

In many applications, however, specifying the operations A and the conditions C is insufficient to fully express the user's criteria for selecting view records. There may exist other criteria that cannot be expressed in A and C. The following examples illustrate this point. Consider a table stored in a database: EMPLOYEES = (Essn, Ename, Address, Sex, Title, HireDate, Salary).

Example 1
In applications that sample a database, typically only a small fraction of a large, qualified population is selected. The following consistent view retrieves all female employees' record:

```
CREATE VIEW FemaleEmp AS
SELECT * FROM Employees WHERE SEX = 'F';
```

If in a study about female employees the analyst needs only 5% of all the qualified records for the study, then the view needed by the analyst is a small subset of the above consistent view. The 5% requirement cannot be expressed in the WHERE clause.

Example 2
Conventional databases model an organization as it changes dynamically with a *snapshot* at a particular time (Snodgrass and Ahn, 1986). The state of a database changes when an update is committed; its past state is not remembered. Views that include past data that no longer exists in the database won't be a consistent view. Many trend analyses require aggregated historical data.

Aggregate functions are applied at different periods of time to gather the needed data. As an example, a consistent view is defined below to compute the current month employee count (assuming the existence of the Now function that returns the current date and the Month function that returns the month of a date), and is to be run on each month's payroll date. This view is not sufficient to perform the trend analysis because it contains only the current month count.

```
CREATE VIEW MonthlyEmpCount AS
SELECT Month(Now()) as Mon, Count(Essn)
FROM Employees;
```

In order to perform the trend analysis this view needs to be computed on the payroll date every month and the results kept in the view. These are views that contain snapshots of the same definition but at different points in time. They include the current view CV and historical data which cannot be specified with the query operations A and criteria C.

Example 3
In some applications an InitialSalary view, with the fields Essn, Ename, and Salary, would be needed to keep track of employees' initial salary. This view is different from the CurrentSalary view defined below which shows employees' current salary:

```
CREATE VIEW CurrentSalary AS
SELECT Essn, Ename, Salary
FROM Employees;
```

The Salary field of an employee's record may have or have not been updated depending on whether the employee's salary has been changed. Both the InitialSalary and CurrentSalary views will contain salary records that have not been updated. For those updated, the CurrentSalary view will contain the latest version and the InitialSalary view will contain the earliest version of the salary. The requirements of keeping the earliest version and not the latest version of salary in the InitialSalary view cannot be specified with the query operations A and criteria C.

The above examples illustrate that even with the same query operations and condition the views requested by the users are different from the consistent views due to users' other criteria. We call this type of view a *non-consistent view* NCV. A non-consistent view shares the same query operations and selection criteria with the consistent view. The discrepancy between a NCV and a CV is due to a user's other requirements in selecting records. Non-consistent views usually require support from customized programs. By supporting only the consistent views, database systems do not serve users adequately. This paper shows how to incorporate the users' other criteria of selecting view records into the view definitions. We will analyze and examine the various kinds of non-consistent views and investigates their management.

Analysis of Non-Consistent Views

In this section we first define non-consistent views, then we analyze possible causes for non-consistent view.

Defining Non-Consistent Views

A NCV and its matching CV affect the same set of qualified records. A user can control the discrepancy between a NCV and a CV at the time the view is first created as in the case of the example 1 above. It is also possible that initially the NCV and the CV are equivalent and the discrepancy occurs during the process of view updating as in the case of the example 2 above. Treating modification as the deletion of the old record followed by the insertion of the new record, there are basically two kinds of database updates: insertion and deletion. In maintaining a consistent view, a view deletion occurs when its matching record in the base table is either deleted or becomes unqualified for the view due to modification, and a view insertion occurs when a new qualified record is inserted to the base table or an old base table record becomes qualified for the view due to modification (Chao et al. 1996). A CV becomes non-consistent if user purposely keeps all or some of the view deletions, and/or excludes all or some of the view insertions.

Therefore, a non-consistent view NCV is the realization of the function NCV(A, C, T, D(T), U) where A, C, T, and D(T) are as before, and U is the user's requirement for including and/or excluding view records. Note that the set of records affected by U may be null or non-null at the time when view is created, but eventually the set may become non-null. Records affected by U belong to one of the two sets: UI (user include) and UE (user exclude). UI (user include) is a set of records that are supposed to

be deleted from the view but that the uses wishes to include, and UE (user exclude) is a set of records that are supposed to be included in the view but that the user wishes to exclude. Hence a non-consistent view is a consistent view with user-identified records included or excluded; it is not a *random* view because its inconsistency is controlled and managed.

Algebraically, NCV = CV + UI – UE. Since UE is a subset of CV and UI is a subset of NCV, UI = NCV - CV and UE = CV - NCV. The relationship between NCV and CV is shown in Figure1.

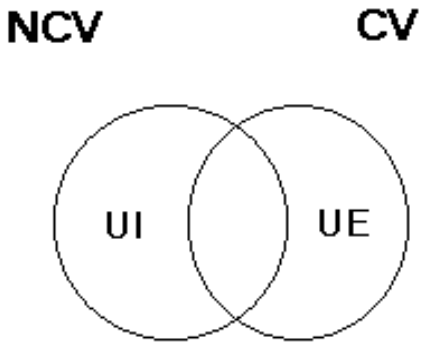

Figure 1. Relationship between NCV and CV

At the time a NCV is created, UI must be null because no updates have been applied to the view records yet. UE may be non-null if the NCV definition initially excludes some qualified records from the view as in the case of example 1; otherwise UE will be null and the NCV is equivalent to the CV. Eventually, in the process of updating the view, UI will become non-null when a user keeps some view deletions, and UE will become non-null if a user rejects some view insertions. Therefore, there are three possible cases for a non-consistent view:

Case 1: UI is null and UE is non-null. In this case NCV is a subset of CV. The example 1 above is one such case. It also happens when the user does not keep view deletions while rejects some view insertions.

Case 2: UI is non-null and UE is null. In this case the user prefers to keep some view deletions while accepting all qualified view records. Hence, CV is a subset of NCV. The example 2 above is one such case.

Case 3: UI is non-null and UE is non-null. In this case the user keeps some view deletions and rejects some view records. The example 3 above is one such case.

We discuss possible causes for each case of NCV below.

User's Non-Consistency Requirements for NCV

At the initiation of a NCV, a user can specify which qualified records are excluded from the view. Most other types of discrepancy are due to user's requirements about updates. The following discussions concern user's requirements for updates.

User's Requirements for Insertions The default action for a consistent view record insertion is to include it. If the user chooses to exclude all or some of the inserted records, then UE is non-null, and the result may be either a case 1 or case 3 NCV. We will use the phrase *no insertion* to express the user's non-consistency requirement for excluding all view insertions, and the phrase s*elective insertion* to express the user's non-consistency requirement for excluding only part of the insertions. For *selective insertion,* the user should also give the criteria to exclude or include insertions.

The *no insertion* requirement is related to applications that do not accept new records. As an example, many doctor's offices have a policy of not accepting new patients. The s*elective insertion* requirement is related to applications that do not include all qualified records, such as the sampling application in example 1.

User's Requirements for Deletions The default action for a consistent view record deletion is to exclude it. If the user chooses to include all or some of the deleted records in the view then UI is non-null, and the result may be either a case 2 or case 3 NCV. We will use the phrase *no deletion* to express the user's non-consistency requirement for including all view deletions, and the phrase *selective deletion* to express the user's non-consistency requirement for including only part of the deletions. For *selective deletion,* the user should also give the criteria to exclude or include deletions.

User's Requirements for Modifications Treating a modification as the deletion *d* of the before-image followed by the insertion *i* of the after-image, due to possible repeated updates, a modified view record may associate with multiple pairs of updates, {(*d, i*)} where {} denotes repetition. Note that the *d* of the first pair is the original value, and the *i* of the last pair is the latest value. The default action for a view modification is to exclude the before-image and include the after-image. The non-consistency due to modifications occurs when a user includes all or some of the before-images, and/or excludes all or some of the after-images.

If the user chooses to include the before-images in the view then UI is non-null, and the result may be either a case 2 or case 3 NCV. There are two meanings of keeping the before-image. The first is to keep only the original value of a modified record, i.e., the *d* of the first pair of updates. We will use the phrase *keep original* to express the user's non-consistency requirement for including the original before-images. The second is to keep all the before-images of a modified record, i.e., all the *d* in the updates. We will use the phrase *keep modified* to express the user's non-consistency requirement for including all the before-images. Note that *keep modified* implies that a record's update history is kept in the view. A user may choose to keep some of the before-images. One typical practice is to keep only the recent updates. We will use the phrase *keep modified last n* to express the user's non-consistency requirement for including the most recent n before-images in the view. Alternatively, a user may keep before-images that meet certain criteria. We will use the phrase *keep selective modified* to express the user's non-consistency requirement for including only the qualified before-images. For the *selective deletion,* the user should also give the criteria to exclude or include deletions.

If the user chooses to exclude the after-images then UE is non-null, and the result may be either a case 1 or case 3 NCV. Among the pairs of update, $\{(d, i)\}$, the *i* of the jth pair becomes the *d* of the j+1th pair. Hence, the *i* of the last pair is the only true after-image. We will use the phrase *no current* to express the user's non-consistency requirement for excluding the after-images. This situation is where a user does not want to include any updated information in the view.

Table 1 summarizes user's options in managing view updates. As an example, assume an employee's salary has been updated three times, from 4000 to 4500 ,4500 to 5000 and 5000 to 6000. The *keep original* option will keep 4000 in the NCV. The *keep modified last 2* option will keep 4500 and 5000 in the view. The *keep modified* option will keep 4000, 4500, 5000, and 6000 in the view. The *keep modified* option combined with the *no current* option will keep 4000, 4500, and 5000 in the view.

Table 1. A User's Options in Including and Excluding View Updates

		Non-Consistent Requirements Applied	
Update	**Default**	**To All Updates**	**To Some Updates**
Insertion	Include all	No insertion	Some insertion
Deletion	Exclude all	No deletion	Selective deletion
Modification:			
Before-image	Exclude all	Keep original/ Keep modified	Keep selective modified/ Keep modified last n
After-image	Include all	No current	NA

Implementation and Maintenance of Non-Consistent Views

Conventional database views can be implemented either as a virtual view or as a materialized view. Virtual views exist only as a definition, and are generated only when they are accessed. Hence, virtual views require a full regeneration and are more expensive to maintain if they are accessed frequently. Also, users have no control of the consistent point because virtual views are always consistent with the current database. Materialized views exist physically as a separate copy, and can be maintained efficiently by a differential refresh scheme. A differential refresh scheme keeps track of updates that have occurred at the base tables since the view was created or last refreshed, and applies those updates to the views to keep them up-to-date. This differential refresh process can be performed immediately as the update occurs at the base tables, or by a delayed update scheme. The immediate view update is typically initiated by the database system. With immediate update materialized views are always consistent with the current database and users have no control of the consistent point. The delayed update scheme first batches updates in a differential file and later applies the differential file to update the view. Unlike the immediate update, the delayed update process can be initiated by the database system, or be initiated in response to user's request. An advantage of the user-initiated delayed update is users can control the consistent point by refreshing the view to a new consistent point other than the current time.

Similarly, non-consistent views can be implemented as virtual views or materialized views, maintained by full refresh or differential refresh. In the following, we discuss the major issues in maintaining NCV without elaborating the technical details.

Implementing NCV as Virtual View To implement a NCV as a virtual view, a database system must be able to keep track of all qualified insertions and deletions since the view was defined so that it can generate UE and UI based on the user's non-

consistency requirement. When the NCV is requested a full generation is performed to create the CV, and the NCV is created by:

NCV = CV – UE + UI

Implementing NCV as Materialized View A materialized view can be more efficiently maintained by a differential refresh scheme. Treating modification as a pair of deletion and insertion, a CV can be maintained by the following differential refresh process:

New CV = Old CV - Deletions + Additions

where Deletions and Additions are the net changes since the last time the view was refreshed. Maintaining a NCV is an extension of maintaining a CV. To maintain a non-consistent view, the maintenance scheme, in addition to generating the Deletions and Additions, must be able to generate UI and UE since the NCV was last refreshed according to the non-consistent clause. Algebraically, a NCV can be refreshed differentially by the following expression:

New NCV = Old NCV - (Deletions - UI) + (Additions - UE)

Commands for Managing Non-Consistent Views

Defining views and refreshing views are two major tasks in managing views. In this section we propose SQL-like commands to define and refresh NCVs. The CREATE NON-CONSISTENT VIEW command below allows users to define (1) the type of NCV to create, (2) the query operations and criteria in identifying qualified view records, (3) the user's non-consistency requirements, and (4) the user's requirements in maintaining the view. The command consists of four parts indicated by numbers which corresponding to the command's four functions.

(1) CREATE NON-CONSISTENT [MATERIALIZED] VIEW *viewname*
(2) AS *query operations*
WHERE *criteria*
(3) NON-CONSISTENCY REQUIREMENT
[AT INITIATION: *criteria*]
[AT UPDATE:
[(NO INSERTION|SELECTIVE INSERTION: *criteria*),]
[(NO DELETION|SELECTIVE DELETION: *criteria*),]
[{(KEEP ORIGINAL|KEEP MODIFIED|KEEP MODIFIED LAST *n* |KEEP SELECTIVE MODIFIED: *criteria*),}]
[NO CURRENT]
(4) [REFRESH (AUTOMATIC|ON DEMAND)]

In the syntax, words in italic are user-entered values; clauses enclosed in the brackets are optional; clauses separated by the vertical bars are possible choices for non-consistency requirements; clauses enclosed in the braces indicates more than one clause may be selected.

Part 1 allows users to specify the type of view to create. Note that many researches consider view materialization a system performance problem that is transparent to users (Labrinidis and Roussopoulos, 2000). We allow users to explicitly specify materialization because it allows users to control the consistent point. Part 2 allows users to specify query operations and criteria to identify qualified records. Part 3 Allows users to specify the non-consistency requirements at the view initiation and at update. The keywords we used to specify the non-consistency requirements have been described earlier. Note that for partial inclusion/ exclusion users must also specify the criteria to include or exclude updates. Without specifying the non-consistent requirements, the default actions of processing updates are assumed. Part 4 allows users to specify the refresh method. AUTOMATIC indicates the view's refresh is handled by the database system without user's interference whereas ON DEMAND indicates the view is refreshed only at user's request. Without the optional REFRESH clause, the default refresh method for a view is AUTOMATIC. If a view is refreshed ON DEMAND, the user must issue the following REFRESH command to update the view:

REFRESH VIEW viewname
[AS OF *consistent point*]

With the optional AS OF clause users can specify the exact new consistent point for the view. Otherwise, the view is refreshed to the current time.

To illustrate, we will use the CREATE command to define the three NCVs examples in section 1.

Example 1
The following command will create a view containing 5% of female employees :

```
CREATE NON-CONSISTENT VIEW FemaleEmpSample
AS      SELECT * FROM Employees
        WHERE SEX = 'F'
NON-CONSISTENCY REQUIREMENT
        AT INITIATION: random 5%
        AT UPDATE:    SELECTIVE INSERTION: random 5%
REFRESH AUTOMATIC
```

With this view, assuming the system implemented a random selection criteria, about 5% of female employees will be included at the time view is created, and any new female employee will also have 5% chance to be included; deletions and modifications will be processed by default actions.

Example 2
The following command will create an aggregate view that contains current month and the last 5 months' count of employees:

```
CREATE NON-CONSISTENT MATERIALIZED VIEW SixMonthEmpCount
AS      SELECT Month(Now()) as Mon, Count(Essn)
        FROM Employees
NON-CONSISTENCY REQUIREMENT
        AT UPDATE:    KEEP MODIFIED LAST 5
REFRESH ON DEMAND
```

At the initiation, the view will contain only the count of that month. During the second month the user issues a refresh command. Since there is only one record in this aggregate view, the new count is actually an update of the old count, and due to the KEEP MODIFIED LAST 5 request, the old count is kept. This process will continue every month, and eventually the last 5 counts will be kept in the view with the current count.

Example 3
The following command will create a view containing current employees' initial salary since the view was created:

```
CREATE NON-CONSISTENT VIEW EmpInitialSalary
AS      SELECT Essn, Ename, Salary
        FROM Employees
NON-CONSISTENCY REQUIREMENT
        AT UPDATE:    KEEP ORIGINAL, NO CURRENT
```

This view initially is a consistent view. A new employee's record will be added to the view as usual, and an old employee's record will be deleted when the employee leaves the company. When an employee's salary is changed, due to the KEEP ORIGINAL, NO CURRENT request, the original salary record is kept and the updated salary is excluded from the view.

Conclusion

This paper defined NCVs and analyzed the causes of non-consistency. We incorporate a user's requirements of view maintenance in the view definition while creating views that fit a user's information needs. In a decision support environment, NCVs can be a useful tool to support end-users. We proposed SQL-like commands to define and refresh NCVs. This paper ignored the detailed

technical implementation of the NCV maintenance scheme. The regular view maintenance schemes need to be changed to maintain NCVs. We continue our research in developing NCV maintenance schemes.

References

Adiba, M. and Lindsay, B. “Database snapshots,” Proceedings of the 6th International Conference on Very Large Data Bases, 1980, pp. 86-91.

Chao, D., Diehr, G., and Saharia, A. "Maintaining Join-based Remote Snapshots Using Relevant Logging," Proceedings of the Workshop on Materialized Views, ACM-SIGMOD Conference, June 1996.

Labrinidis, A. and Roussopoulos, N. “Webview Materialization,” ACM SIGMOD International Conference on Management of Data, May 14-19, 2000

Snodgrass, R., and Ahn, I. "Temporal Databases," IEEE Computer, Sept. 1986, pp. 35-42.