https://doi.org/10.48009/2_iis_2008_323-329

# CREATING CUSTOMIZED DATABASE VIEWS WITH USER-DEFINED NON-CONSISTENCY REQUIREMENTS

**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 their definitions, which are expressed in algebraic operations and selection conditions. In many applications, however, users' requirements for views are not expressible with algebraic operations and selection conditions, and may not be consistent with the current state of the database. Typically, this non-consistency is due to users' requirements 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 methods of expressing non-consistent requirements.*

Keywords: User-defined, Non-consistent, Database views

## 1. INTRODUCTION

Supporting an end-user's information needs for decision-making is a common objective of database management and the World Wide Web. Although the Web has a much shorter history than database management, it has developed many innovative technologies to create web pages that are tailored to users' needs. Web sites may implicitly add customized information to web pages based on the user's online behavior without the user being fully aware, or explicitly allow the user to control the web page's content and/or interface [10]. The former practice is referred to as personalization and the later as customization [11]. Customization requires the user's inputs and the page thus created represents the user's view of the page. The process of creating a customized page typically involves two basic principles: user exclusion and user inclusion. User exclusion is the ability for a user to exclude irrelevant content from the content options offered by the website. User inclusion is the ability of a user to add additional content that is specified or provided by users.

In comparison, database management is lagging behind web technologies in providing customized information. Traditionally, database views have been used to support an end-user's information needs for decision making [4]. A database view is the part of database with which a particular user is interested. It can be defined as any valid query operation over single or multiple base tables or other views to meet a user's need. In implementations, views may exist virtually as a definition or be materialized physically as a separate copy [2].

Traditional views, which we call *consistent views*, reflect the state of the database either at the current time or at a specific point in time as a snapshot view [1, 6, 7]. The word *consistent* in our context means that the view represents the content of the database at a particular point in time. (We note that the word *consistent* in database systems is often used to mean that a database satisfies certain integrity constraints. Our use of this word is different and reflects the user's understanding that a view is derived only from the data in a database at a point in time.) In practice, views returned to the users are not always fully consistent with the user's definitions; they are frequently modified by the system without the knowledge of the user. Database systems may apply *view modification* techniques [5] to change a view's query. They make these modifications in order to control access only to permitted parts of the database or to implement system policies. For example, a user may define a view on a student table to retrieve data for all students with a GPA greater than 3.0: SELECT * FROM Student WHERE GPA > 3.0. The system, however, may restrict the user to access only business school students' records by, in effect, modifying the criteria to GPA > 3.0 And College='Business'. As another example, a customer may define a view on a vendor's product table to receive information about computer games as SELECT * FROM Products WHERE Category='Computer Game'. For marketing purpose, however, the vendor maintains a promotion table that contains products that are currently on promotion and includes the data in this table in every customer's view. To implement this promotion, the view is modified to: {SELECT * FROM Products WHERE Category='Computer Game'} Union {SELECT * FROM Promotion}. The view thus generated is called a *non-consistent view* because there may exist differences between a user's definition of a view and the actual view received by

the user.

In the previous examples the non-consistency between a consistent view and the actual view is controlled by the database system. This research studies cases where the non-consistency is controlled by users. With a consistent view, the content of the view is determined by the view's query and the state of the database and the user has no other means to further control its content. Users, however, may wish to include certain data in a view that is not reflected in the state of the database at some time or exclude certain data in a view that is reflected in the state of the database. For example, a user may wish to include historical data that are not in the current database. Similarly, a user may wish to exclude data in the current database that would otherwise be part of a view in applications that sample a small fraction of the database. Conceptually, this concept uses the two principles of customization - user inclusion and user exclusion - from Web customization to create customized database views. Using our terminology, views that incorporate user inclusion and exclusion in this manner are not consistent with the state of the database and are *non-consistent views*.

The overall purpose of this paper is to explore non-consistent views for meeting user needs. We investigate and define different kinds of non-consistent views and present a mechanism for users to express and incorporate non-consistency requirements into their view definitions.

This paper is organized as follows. Section 2 presents our formal definition of consistent views and introduces the concept of user-defined non-consistent views. Section 3 presents examples that demonstrate situations in which consistent views are not sufficient to meet users' needs thus create the need for non-consistent views. Section 4 formally defines user-defined non-consistent views and examines them in detail. Section 5 concludes the paper and discusses our continuing research.

## 2. CONSISTENT VIEWS

A *consistent view* CV can be formally defined as a realization of the function CV(Q(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, Q(A, C) is the query defined by A and C**,** 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. Given a database state, end-users control a view's contents by specifying the operations A and the conditions C. The resulting CV represents what the database system offers to the users. A virtual CV always reflects the current database state because it is recreated when accessed. A materialized CV needs to be refreshed to reflect the new state. Other than specifying A and C users do not have any other means to further control the content of CV to meet their needs.

In many applications, however, specifying A and C is insufficient to fully express the user's criteria for selecting view records and additional rules may be needed to tailor the view. Incorporating the two principles of customization - user exclusion and user inclusion - and considering CV as the original contents offered by the database system, we let users exclude unwanted records from the CV and include wanted records in the CV. We call this new type of view a *user-defined non-consistent view* (UNCV) because it is not consistent with the state of the database either currently or at a specific previous point in time as defined by the A and the C in the view definition and is defined by the needs of the user. We defer our formal definition of UNCV until after we discuss in more detail the need for non-consistent views.

## 3. THE NEED FOR USER-DEFINED NON-CONSISTENT VIEWS

For a UNCV, the non-consistency is controlled by users. The objective is to allow users to further control a view's contents where it is insufficient to fully express the user's criteria for selecting view records by specifying the A and the C of the view definition alone. The following examples illustrate this point. We use the following table in these examples: Employees (Essn, Ename, Address, Sex, Title, HireDate, Salary). We assume that the target system will be a relational database system such as Oracle, DB2, and SQL Server. Although some of the examples involving historical data could be solved with temporal DBMSs, such systems are rare among commercial applications. Later we discuss temporal databases.

### Views with user exclusion

**Example 1** Some users do not want to include all qualified records in a view. 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 of 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 query operations A and the criteria C in the consistent view.

**Example 2** It is common among professionals such as medical doctors or attorneys to have a policy of not accepting new clients. This practice can be translated to a no new insertion rule in terms of view maintenance. In the previous example of a 5% sample of female employees, if the user wants to study the original sample only, then new female employee records will be excluded from the view even though they are qualified based on the criteria C.

**Views with user inclusion**

**Example 3** Conventional databases model an organization as it changes dynamically with a *snapshot* at a particular time [8]. 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 will not be consistent views. Many applications require recording not only current events but also events that have previously occurred. For example, a view may be needed to identify all employees who are or have previously been managers. The view will contain current managers as well as past managers. The following consistent view retrieves only employees who are currently managers.

```
CREATE VIEW Manager AS
SELECT Essn, Ename, Sex  FROM Employees
WHERE Title='Manager';
```

If a manager changes job title the manager's record will be removed from this view. This view does not meet the user's requirement to keep a manager's record even after the job title changes. In this example, records of past managers are included in the non-consistent view.

**Example 4** Historical data is needed to perform trend analysis. Many trend analyses require aggregated historical data, which could be gathered by applying aggregate functions at different times. As an example, a consistent view is defined below that computes the current month's employee count (assuming the existence of the Now function that returns the current date and the Year and Month functions that return the year and month of a date), and is to be run on each month's payroll date.

```
CREATE VIEW MonthlyEmpCount AS
SELECT Year(Now()) AS Year, Month(Now()) AS
Month, Count(Essn) AS "Employee Count"
FROM Employees;
```

This view by itself is not sufficient to study the monthly change in employee counts because it contains only the current month's count. A view capable of supporting this analysis will include the current count and historical counts of the past months. The historical data included in a view along with the current data is an example of user inclusion.

**Views with user exclusion and inclusion**

**Example 5** Database records are subject to updates. Sometimes the original value of a record may provide much useful information. As an example, an InitialSalary view may be needed to identify employees' initial salaries. This view is different from the CurrentSalary view defined below which shows employees' current salaries:

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

As time passes, employees' salaries may change and new employees may join the company. The CurrentSalary view will contain the latest salary of all employees. The InitialSalary view will contain the initial salary of new employees and retain the initial salary of old employees whose salary has changed. In this example, the latest salary of an employee is excluded from the view and the initial salary of the employee is included in the view.

The above examples illustrate that some views requested by users are different from the consistent views CV(Q(A, C), T, D(T)) . This non-consistency is due to users' requirements in selecting view records that cannot be expressed with the query operations A and conditions C. Such requirements result in non-consistency, and the views thus generated are user-defined non-consistent views (UNCVs). Note that a UNCV shares the same query operations and selection criteria with the consistent view. Because of this characteristic, a UNCV and its matching CV affect the same set of qualified records. The discrepancy between a UNCV and a CV is due to a user's non-consistency requirements in selecting

records.

A user can control the discrepancy between a UNCV and a CV by including or excluding qualified view records. This process can be done at the time the view is created and/or at updates during the life span of the view. In example 1, 95% of qualified records are excluded from the view when it is created; whereas in example 2, a "no new record" rule affects the non-consistency after the view is created. It is also possible that initially the UNCV and the CV are equivalent and the discrepancy occurs during the process of view updating. In example 4, initially both the UNCV and CV have the first month count, but after a period of time the UNCV will keep the previous month counts while the CV will not. Supporting UNCVs will allow database management systems to provide customized information for users.

Our concept of non-consistent views may be realized in some instances by designing the database in a way that will allow the required data to appear in a traditional, consistent view. Users, however, often cannot identify all their data needs when a database schema is designed. At some point, after a database has been in use for a period of time, a user may find a need for certain data that cannot be obtained from the existing database design. Although the database could be redesigned and reconstructed to reflect this change in the user's requirements, doing so may be time consuming and expensive, and could create integrity or other problems when migrating the current database to the new design. Thus a different way of meeting the user's needs is called for.

## 4. ANALYSIS OF USER-DEFINED NON-CONSISTENT VIEWS

In this section we first formally define user-defined non-consistent views. Then we present ways to express user's non-consistency requirements.

### 4.1 Defining User-defined Non-Consistent Views

A user-defined non-consistent view, UNCV, is the realization of the function UNCV(Q(A, C), T, D(T), U) where Q(A, C), T, D(T) are the same as for consistent views, and U is the set of user's non-consistency requirements for including and/or excluding view records. The non-consistency requirements U may apply to records at the time the view is created and/or to qualified updates after the view is created. 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 in the base table or an old base table record becomes qualified for the view due to modification [3].

Records affected by U belong to one of two sets: UI (user include) and UE (user exclude). Let SI (system include) denote the set of records that are included in a consistent view. Let SE (system exclude) denote the set of records that are deleted from a consistent view over time because of view maintenance. Then UI is the subset of SE that the user wishes to include in a non-consistent view, and UE is a subset of SI that are included in a consistent view but that the user wishes to exclude from a non-consistent view. Therefore, records in the UI and UE are all qualified for Q(A, C). These requirements eliminate cases where non-consistency is introduced by adding non-qualified records to the view.

Algebraically, UNCV = CV + UI – UE. Since UE is a subset of CV and UI is a subset of UNCV, UI = UNCV - CV and UE = CV - UNCV. The relationship between UNCV and CV is shown in Figure 1.

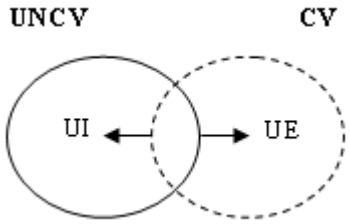


**Figure 1: Relationship between UNCV and CV.**

There are three possible cases for a non-consistent view:

*Case 1: UI is null and UE is non-null.* In this case UNCV is a subset of CV. Example 1, where the user requires only 5% of the qualified records, is one such case. This case also occurs when the user does not keep view deletions while rejecting some view insertions, as in example 2.

*Case 2: UI is non-null and UE is null.* In this case the user wants to keep some view deletions while accepting all qualified view records. Hence, CV is a subset of UNCV. Examples 3 and 4 are such cases. In

example 3, those employees who had been managers are kept in the view regardless of their current title. In example 4, the previous months' counts are kept in the view.

*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. Example 5 is one such case. When salary changes the initial salary is kept in the view while the new salary is excluded from the view.

### 4.2 User's Non-Consistency Requirements for UNCV

We now discuss possible user's non-consistency requirements for exclusion and inclusion. The non-consistency requirements are not logical conditions expressible with the condition C, and are not meant to modify the condition C, otherwise they would be equivalent to view modifications. Note that this discussion is not meant to be complete but only to illustrate typical non-consistency requirements.

**Non-consistency requirements for user exclusion**

Non-consistency requirements for user exclusion exclude unwanted qualified view records. These requirements can be applied at the time the view is initiated and/or at the time new qualified records appear in the database after the view is initiated. Examples of non-consistency requirements at view initiation include random selection with an X% chance of a qualified record being selected, random selection of n qualified records, and limiting the view to contain at most n qualified records. These requirements may be expressed in phrases like: RANDOM SELECT x%, RANDOM SELECT n RECORDS, and VIEW CONTAINS AT MOST n RECORDS, respectively.

After view initiation, the default action for a view record insertion is to include it. The user may choose to exclude some or all of the new insertions. We will use the phrase NO INSERTION to express the user's non-consistency requirement for excluding all view insertions, and the phrase SELECTIVE INSERTION to express the user's non-consistency requirement for excluding only some of the insertions. For SELECTIVE INSERTION, a phrase such as RANDOM SELECT x% can be used to select an insertion with x% chance and a phrase such as VIEW CONTAINS AT MOST n RECORDS can be used to limit the number of view records. Users may have other requirements. For example, a user may want to accept up to n new records either sequentially as the records appear or randomly. As another example, a user may want to accept new records only if they satisfying other specified conditions. These requirements may be expressed in phrases such as ACCEPT n INSERTIONS, RANDOM ACCEPT n INSERTIONS, ACCEPT INSERTION IF *condition*.

**Non-consistency requirements for user inclusion**

Records in UI are view deletions due to base table deletions or modifications. For a database that does not keep historical data UI will be null at view initiation. The non-consistency requirements apply to view deletions after the view initiation.

***User's requirements for modifications*** A view record may be modified repetitively creating many historical versions of the record. These versions can be viewed as a series. Let Rn denote the current version of the record and R0 denote the earliest version. Then R0, R1, …, Rn-2, Rn-1, Rn are all the versions of the record. A consistent view will contain Rn only. A non-consistency requirement may include all or some of the earlier versions. We will use the phrase ORIGINAL to identify R0, FIRST *i* to identify R0 thru Ri-1, LAST *i* to identify Rn-i thru Rn-1, and CURRENT to identify Rn.

A modification can also be viewed 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. The default action for a view modification is to exclude the before-image and include the after-image. A non-consistency requirement may apply to keep the before-image. We will use the phrase BEFORE-IMAGE to identify the before-image of the last pair of (*d, i*), that is Rn-1. Note that a non-consistency requirement can also be applied to the after-image to exclude it from the view.

In summary, we will use a phrase KEEP ORIGINAL to express the user's non-consistency requirement for including R0; KEEP MODIFIED ALL to express the user's non-consistency requirement for including R0 thru Rn-1; KEEP MODIFIED LAST i to express the user's non-consistency requirement for including Rn-i thru Rn-1; KEEP MODIFIED FIRST i to express the user's non-consistency requirement for including R0 thru Ri-1; KEEP MODIFIED BEFORE-IMAGE to express the user's non-consistency requirement for including Rn-1; NO CURRENT to express the user's non-consistency requirement for excluding Rn. Alternatively, a user may selectively keep R0 thru

Rn-1 that meet certain criteria. We will use the phrase KEEP SELECTIVE MODIFIED IF *condition* to express the user's non-consistency requirement for conditionally including R0 thru Rn-1.

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 UNCV. The KEEP MODIFIED LAST 2 option will keep 4500 and 5000 in the view. The KEEP MODIFIED ALL option will keep 4000, 4500, 5000, and 6000 in the view. The KEEP ORIGINAL option combined with the NO CURRENT option will keep the 4000 in the view.

***User's requirements for deletions*** The default action for a consistent view record deletion is to exclude it. We will use the phrase NO DELETION to express the user's non-consistency requirement for including all view deletions in the view, and the phrase SELECTIVE DELETION IF *condition* to express the user's non-consistency requirement for including only selective view deletions.

### 4.3. Expressing Non-Consistency Requirements with Commands

This section incorporates the phrases defined in Section 4.2 in expressing the non-consistency requirements into commands that may be used with a view definition command. The commands are presented below.

NON-CONSISTENCY REQUIREMENT

[AT INITIATION: RANDOM SELECT *x* % | RANDOM SELECT *n* RECORDS | VIEW CONTAINS AT MOST *n* RECORDS]

[ON INSERTION: NO INSERTION | SELECTIVE INSERTION RANDOM SELECT *x* % | VIEW CONTAINS AT MOST *n* RECORDS | ACCEPT *n* INSERTIONS | RANDOM ACCEPT *n* INSERTIONS | ACCEPT INSERTION IF *criteria*]

[ON MODIFICATION: {KEEP ORIGINAL | KEEP MODIFIED ALL | KEEP MODIFIED LAST *n* | KEEP MODIFIED FIRST *n* | KEEP SELECTIVE MODIFIED IF *criteria* | KEEP MODIFIED BEFORE-IMAGE} | NO CURRENT]

[ON DELETION: NO DELETION | SELECTIVE DELETION IF *criteria*]

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

To illustrate the use of these phrases, we present commands to define the non-consistency requirements of the examples of Section 2.

- Example 1: Select only 5% of the qualified records when the view is created and select 5% of the new records afterward:
  AT INITIATION: RANDOM SELECT 5 %
  ON INSERTION: SELECTIVE INSERTION RANDOM SELECT 5 %

- Example 2: Implement the no new client policy:
  ON INSERTION: NO INSERTION

- Example 3: Keep managers who change titles:
  ON DELETION: NO DELETION

- Example 4: Keep previous months' employee counts:
  ON DELETION: NO DELETION

- Example 5: Keep employees' initial salary and not the current salary:
  ON MODIFICATION: KEEP ORIGINAL, NO CURRENT

### 4.4 UNCV and Temporal Databases

A temporal DBMS keeps the transaction time and/or valid time of a record. The transaction time records the starting and ending time a record is current in a temporal database, and the valid time supports the retroactive updates that allows a user to view the true historical data as of now [8]. Many query languages such as TSQL2 [9] have been developed to query temporal databases. With a temporal database, the D(T) of UNCV(Q(A, C), T, D(T), U) will be the state of the database at time T including its historical data, and Q(A, C) will be a temporal query if historical data is needed. Therefore, if the historical data of a view, such as the InitialSalary view of example 5, can be expressed by a temporal query then the UI will be null and we can only define Case 1 UNCV.

While a temporal query uses conditions involving transaction time and/or valid time to retrieve historical data, we treat a record's update history as a series of individually identifiable versions and offer an alternative way to include historical data in a view without referring to the time. Also, UNCV emphasizes customizing the maintenance of a volatile view for future updates.

## 5. CONCLUSION

This paper defined non-consistent views. The non-consistency between a consistent view and a non-consistent view is due to a user's requirements to include or exclude qualified view records in view maintenance. The objective is to create customized views by giving users additional control of a view's content beyond a conventional view definition. We present a few such requirements to illustrate the concept.

UNCVs can be supported in many ways. One way is by dedicated procedures such as database triggers that monitor database updates and perform appropriate view maintenance. With a relational database system, they can be supported by adding transaction time stamps to a base table to store records' update history or by using a dedicated update log to store historical data. They can also be supported by a temporal database that has historical data. We continue to research on issues related to the UNCV implementation.

UNCVs can be used to support end-user information needs for decision making in situations where consistent views are inadequate. By including deleted records or excluding qualified records users can customize the information they receive to their specific situation. Organizations benefit from better decisions made with more relevant information.

## REFERENCES


1. Adiba, M., Lindsay, B., Database snapshots. Proceedings of the 6th International Conference on Very Large Data Bases (1980) 86-91.
2. Blakeley, J. A, Larson, P. A., Tompa, F. W., Effiently Updating Materialized Views, Proceedings.of the 1986 ACM SIGMOD International Conference on Management of Data (1986) 61-71.
3. Chao, D., Diehr, G., Saharia, A., A Differential Refresh Scheme for Remote End-User's Views, Journal of International Information Management, Spring 1996, Volume 5, Number 1 (1996) 75-85.
4. Connolly, T., Begg, C., Database Systems: A Practical Approach to Design, Implementation and Management, Addison Wesley, 2004.
5. Mannino, M., Database Design, Application Development, and Administration, 2/e , Irwin McGraw-Hill, 2003.
6. Microsoft, How to Enable the Snapshot Transaction Isolation Level in SQL Server 2005 Analysis Services. From: http://support.microsoft.com/kb/919160, 2006
7. Oracle, Oracle Database Data Warehousing Guide, 10g Release 2. From: http://download-east.oracle.com/docs/cd/B19306_01/server.102/b14223/title.htm, 2005
8. Snodgrass, R., Ahn, I., Temporal Databases, IEEE Computer, Sept. (1986) 35-42.
9. Snodgrass, R., et al., TSQL2 Language Specification. SIGMOD Record 23(1): 65-86(1994) Electronic Edition
10. Wu, D., Im, I., Tremaine, M., Instone, K., Turoff, M., A Framework for Classifying Personalization Scheme Used on e-Commerce Websites, Proceedings of the 36th Hawaii International Conference on System Sciences (2003)
11. Zo, H., Personalization vs. Customization: Which is More Effective in E-Services?, Proceedings of the Ninth Americas Conference on Information Systems, Tampa, FL (2003) 251-256.