Vacuuming in TSQL2*

Christian S. Jensen

Abstract

Updates, including (logical) deletions, to temporal tables that support transaction time result in insertions at the physical level. Despite the continuing decrease in cost of data storage, it is still, for various reasons, not always acceptable that all data be retained forever. Therefore, there is a need for a new mechanism for the vacuuming, i.e., physical deletion, of data when such tables are being managed.

We propose syntax and informal semantics for vacuuming of data from temporal tables in TSQL2 which support transaction time. The mechanism allows—at schema definition time, as well as later, during the life span of a table—for the specification of so-called cut-off points. A cut-off point for a table is a timestamp that evaluates to a time instant. The timestamp may be either absolute or a bound or unbound now-relative timestamp. Conceptually, the cut-off point indicates that all data, current in the table solely before the (current value of the) timestamp, has been physically deleted. Vacuuming based on cut-off points is an example of a more general notion of vacuuming where arbitrary subsets of data may be physically deleted.

1 Introduction

Base tables supporting transaction time are ever-growing because all logical updates, including deletions, transform into insertions at the physical

*This paper also appears as a commentary in R. T. Snodgrass et al.: *The TSQL2 Language Specification*. Correspondence may be directed to C. S. Jensen, at Aalborg University, Datalogi, Fr. Bajers Vej 7E, DK–9220 Aalborg Ø, Denmark, `csj@iesd.auc.dk`.

level. This contrasts snapshot tables where logical and physical deletion coincide. Logical deletion in TSQL2 is covered in a separate commentary. As there is a need for physical deletion capabilities in snapshot tables (and valid-time tables), there is also a need for such capabilities in temporal tables supporting transaction time (i.e., transaction-time tables and bitemporal tables).

This commentary is divided into two parts. In the first, the general notion of vacuuming and associated concepts are introduced and motivated. It is shown that straightforward physical deletion may adversely affect the usefulness of query results and that special attention thus should be devoted to the design of appropriate vacuuming facilities. We then give reasons why disciplined vacuuming is a highly desirable capability. Subsequently, the important concepts of vacuuming are presented by means of a larger example and a survey. In the second part, a specific design of vacuuming capabilities for TSQL2 is presented[1]. The goal has been to design minimal facilities that are easily implemented and yet provide adequate functionality.

2 Motivation and Introduction to the Concept of Vacuuming

Support for transaction time brings with it the potential for accessing any past database state. Physical deletion, by its very nature, limits this potential, and while this may be desirable, care should be taken to avoid an adverse impact on the utility of the data that is retained in the database.

To get an initial understanding of some of the aspects of vacuuming, consider a scenario where we are studying an author's perception of a particular historical phenomenon. Our source is a new printing of the author's one-hundred year old diary. Now consider four possible cases.

1. It is guaranteed that our new printing has the same content as had the original diary.

   In this case, our source is reliable and we can trust its contents.

[1]An earlier design was presented in the TSQL2 tutorial that appeared in *ACM SIGMOD Record,* Vol. 23, No. 3, September, 1994.

2. The publisher removed from the new printing certain parts of the author's original description of the historical phenomenon, but described clearly what was removed.

   In this situation, our source may or may not be of less use to us. If we are not interested in the aspects that were removed, the source is as valuable as in the first case. If we are interested in the aspects that have been removed, the source has lost value, and we will find the omissions unfortunate.

3. We know that the publisher may have removed parts of the author's original description of the historical phenomenon.

   In this case, the source is unreliable and may be of little or no value to us. It may have value, as we know that it may be inaccurate only by omission. Still, without knowledge of the type of omission (e.g., omission of complete paragraphs or omission of words that distort the meaning of individual sentences), much care must be taken to avoid being mislead.

4. Without informing us and in conflict with current practice, the publisher has removed certain parts of the author's description of the historical phenomenon.

   This case is highly problematic. We believe that the source is accurate, but it is not. Thus, the source is now potentially worse than no source at all.

The publisher's removal of material is in principle similar to physical deletion from a temporal database. The first case corresponds to a temporal database system that does not support physical deletion. Here, it is guaranteed that the past states of the database are retained unmodified and never change. In terms of the example, once text is entered into the diary, it remains there, and we know that the diary is complete.

As will become clear, it is, however, necessary for a temporal database system to support vacuuming capabilities. Then we would like a database system to behave as in the second case, rather than the third and fourth cases.

It now becomes interesting to investigate trade-offs between specific types of removals and their effect on the utility of the remaining text. For example, if the publisher is allowed to remove individual words such as "not" and "never," the text may become incomprehensible or misleading (it lacks safe interpretations). It may be better to restrict deletions to complete paragraphs or larger textual units. The ease with which removals are explained to the reader is important. Another important issue is exactly how the deletions should be reported to the reader.

It is part of the task of designing vacuuming facilities to choose functionality that is sufficiently powerful, is easily described to the user, and has a minimal effect on the utility of the data that is retained.

Below, we first outline the reasons why physical deletion is necessary and then explore in more detail some problems and solutions [JM90] when designing vacuuming facilities.

Many reasons exist why flexible physical deletion is necessary. In the example, deletion from the original diary may be necessary for these reasons.

- Without reductions, the existing diary is very lengthy and boring. Sections concerning the author's opinions on local politics are of no interest to the general readership.
- While an interesting diary, the publisher has strict page limits that necessitate omissions.
- Certain material in the diary must be omitted to respect confidentiality or national security interests.

In a general database context, there are also good reasons for physical deletion. In many installations, ever-growing tables will eventually outgrow the mass storage devices available (e.g., magnetic disks). In order to guarantee continuous operation, physical deletion capabilities are necessary. It must be possible to delete data that are no longer needed, or when additional space needs to be freed for more important data. Next, the efficiency of query processing generally degrades as tables grow [AS86]. For this reason, means of controlling the table sizes are highly desirable. Finally, many countries have strict laws that require the ability to delete certain records of previous history (while requiring that other records be retained). Customers may demand that no information about them exists in some database. Other laws may require that certain records be kept for a fixed duration of time. For example, information related to personal income tax must, in some countries, be retained by the citizens for five years. Business policies also pose similar requirements.

In correspondence with the first case in the diary scenario, a database systems supporting transaction time and without physical deletion has a very desirable property: any query that is not now-relative [DSJ93] will always return the same answer independently of when it is issued (a now-relative query is simply a query that includes a now-relative timestamp, i.e., a timestamp that is evaluated to a different value when evaluated at different times). When physical deletion is introduced, it should be clear that this property no longer holds.

When designing a vacuuming facility, two different approaches may taken to "minimize the damage."

- The system can attempt to guarantee that unless the contrary is explicitly indicated, query results are not affected by vacuuming [JM90]. With this approach, users are maximally shielded from the fact that vacuuming may occur and may affect query results. Assuming that an attempt is made to not vacuum any useful data, most users need not know that vacuuming occurs at all.

  Query modification techniques [Sto75] and algorithms for testing equivalence among query expressions [ASU79] may be utilized to implement this functionality efficiently.

  What type of notification, then, should be given when a query result may be affected by vacuuming? Again, there are several options. For simplicity, we consider only interactive queries.

  - The system could simply refuse to evaluate the query on the grounds that vacuuming may have affected the result. Since the result could still be useful, this option seems too restrictive.
  - The system could return the result, but also notify the user that the result may be affected by vacuuming. With this design, the user can inspect the descriptions of what has been vacuumed, interpret the query on that basis, and perhaps issue additional queries. For this to be a good option, descriptions of what has been vacuumed should be easily accessible. It seems appropriate for the system to retain, accumulate,

and organize this type of information. That would ensure easy access and completeness.

  - The system could return the result and also provide the user with queries that are "similar" to the original query, but that are not affected by vacuuming. This is the most user-friendly, but also the most complex solution. With this solution, the user may not need to inspect potentially complicated descriptions of what has been vacuumed, but may choose to ignore vacuuming if one of the system-proposed queries are satisfactory.

    Techniques for query specialization and generalization [Cha90, Mot84] that generalize query modification and equivalence preserving transformations may be utilized here.

- The system makes no guarantees and offers minimal assistance. It requires the users to be up-to-date with the vacuuming that has been performed on the database and to always take this information into consideration when formulating and interpreting queries. As was the case above, the system should automatically retain, accumulate, and organize the descriptions of what has been vacuumed.

  Above, users did not need to worry about vacuuming unless explicitly told to by the system. Here, the user must check any query with respect to what has been vacuumed.

When it is possible to make complex vacuuming specifications and when queries are generally not affected by vacuuming, the first approach may be preferable. However, when only simple vacuuming specifications are allowed, the second, simpler approach seems acceptable.

A few additional issues should be mentioned. First, not all deletion specifications are appropriate. A specification stating that data between one and two years old should be vacuumed is an example. Data more than two years old cannot be deleted, and data not yet two years old will eventually become two years old and can therefore not be deleted. In consequence, nothing can be deleted and the specification is at best useless. Next, deletion specifications should not delete data needed by other specifications, integrity constraints, and views.

The actual physical deletion is performed by an asynchronous vacuuming demon according to the specifications. While vacuuming logically has eager semantics, any degree of eagerness or laziness can be adopted for the actual physical removal of base data, and a variety of conditions triggering the demon can be employed. This high degree of flexibility makes it possible to achieve efficient vacuuming implementations.

3 Using Cut-off Points for Vacuuming in TSQL2

Cut-off points provide basic vacuuming. We first present the required language extensions, then discuss the properties of the vacuuming facilities.

3.1 Language Extensions for Vacuuming

We initially consider specification of vacuuming at schema-definition time [RS87]. Then we consider specification of vacuuming after schema-definition time [JM90]. We base the presentation on a sample table, defined as follows on August 1, 1994.

```
CREATE TABLE EmpDep
    (Name CHARACTER ( 30 ) NOT NULL,
    Dept CHARACTER ( 30 ) NOT NULL)
AS TRANSACTION YEAR ( 2 ) TO DAY;
```

We have created a table `EmpDep` with two attributes recording in which departments employees work. This table is declared as a transaction time table. Transaction-time timestamps are to the underlying granularity of `DAY`, with a range of 100 years.

Next, assume that the following updates are performed.

On August 4, 1994:
```
INSERT INTO EmpDep
VALUES ('Jake', 'Ship')
```
On August 9, 1994:
```
INSERT INTO EmpDep
VALUES ('Kate', 'Load')
```

On August 19, 1994:

```
        UPDATE EmpDep
        SET Dep TO 'Load'
        WHERE Emp = 'Jake' AND Dep = 'Ship'
```

On August 22, 1994:

```
        INSERT INTO EmpDep
        VALUES ('Kate', 'Ship')
```

Conceptually, this results in the following instance (seen as of 8/25/94).

Emp	Dep	T
Jake	Ship	$\{8/5/94, 8/6/94, 8/7/94, \ldots, 8/19/94\}$
Jake	Load	$\{8/20/94, 8/21/94, 8/22/94, \ldots, 8/25/94, \mathrm{uc}\}$
Kate	Load	$\{8/10/94, 8/11/94, 8/12/94, \ldots, 8/25/94, \mathrm{uc}\}$
Kate	Ship	$\{8/23/94, 8/24/94, 8/25/94, \mathrm{uc}\}$

In this instance, "uc" is a special symbol that indicates that the information recorded by the tuple is still current, i.e., current "until changed." The symbol allows the system to currectly update the instance with additional time values as time advances [JS92, JS93]. With this example available, we proceed by considering a sample specification of a cut-off point (assume that `EmpDep` has not yet been created).

```
CREATE TABLE EmpDep
    (Name CHARACTER ( 30 ) NOT NULL,
    Dept CHARACTER ( 30 ) NOT NULL)
AS TRANSACTION YEAR ( 2 ) TO DAY
VACUUM NOBIND(DATE 'now - 7 days');
```

A cut-off point of `NOBIND(DATE 'now - 7 days')` has been specified, meaning that only data current within the most recent seven days is available to queries.

Vacuuming is a logical notion which is independent of the particular representation chosen for the temporal table. Logically, facts with transaction times that overlap a cut-off point lose those transaction times that are before the cut-off point, and facts that have transaction times completely before the cut-off point are removed. Logically, the table will contain no transaction times before the cut-off point. Thus, with the above specification in effect, on August 29, 1994, the sample table will have the following contents.

Emp	Dep	T
Jake	Load	$\{8/22/94, 8/23/94, 8/24/94, \ldots, 8/29/94, \mathrm{uc}\}$
Kate	Load	$\{8/22/94, 8/23/94, 8/24/94, \ldots, 8/29/94, \mathrm{uc}\}$
Kate	Ship	$\{8/23/94, 8/24/94, 8/25/94, \ldots, 8/29/94, \mathrm{uc}\}$

While unbound now-relative cut-off points generally seem more useful for schema definitions than bound now-relative and absolute cut-off points, bound now-relative and absolute cut-off points may also be used. For example, `VACUUM DATE 'August 3, 1994'` in a schema definition issued on August 1, 1994, will result in a table that does not retain any data until August 3, 1994.

The default `VACUUM` clause is `VACUUM DATE 'now'`, a bound now-relative cut-off point. Then, if part of a schema definition issued on August 1, 1994, `now` will evaluate to that date meaning that the table will not contain data current before the time it is created. Thus, if no `VACUUM` clause is present, no vacuuming is done. Specifying an bound now-relative or absolute cut-off point that is before the time bound to `DATE 'now'` generates a warning and is otherwise ignored. Specifying a `VACUUM` clause for other tables than transaction time and bitemporal tables results in an error.

It follows from the above that any transaction time and bitemporal table has associated exactly one cut-off point. This time is recorded in an appropriate system table.

In a system with vacuuming, it is possible that the same query yields different results when issued before and after vacuuming. However, when vacuuming is specified only at schema definition time, the complication is completely avoided. This is so because the user has never had access to data that was later deleted. Note that this proposal does support the specification of vacuuming after schema definition time.

Vacuuming may be implemented by query modification and should pose no performance problems. When computing a query, the system simply replaces all references to tables that have been vacuumed by expressions that compute what is left according to the cut-off points. For example, consider a query that retrieves the salary record of Jake from the EmpDep table. Assume that the table was created on August

1, 1992, and that its cut-off point is April 1, 1993. Then the original query is simply modified to a new query that retrieves the part of Jake's salary record that was current during and after April 1, 1993.

With this approach, the system can perform the actual physical deletion when this is most convenient without affecting the semantics of the vacuuming specifications. In the extreme, no deletions need be performed at all. Often doing no physical deletions defeats the purpose. All data related to facts current only before the cut-off point or data related alone to those parts of timestamps of facts that are before the cutoff point may be deleted. What can be physically deleted clearly depends on the physical representation chosen for the table.

We next consider the specification of vacuuming after schema-definition time. This is done using the `ALTER` clause.

As an example, assume that the following vacuuming specification takes effect on August 31, 1994.

```
ALTER TABLE EmpDep
VACUUM NOBIND(DATE 'now - 7 days')
```

The meaning follows the same rules as defined above. From August 31, 1994, this specification will have the same effect as had the earlier now-relative specification at schema-definition time.

With the exception that this type of vacuuming may be specified any time after the table is defined, the functionality is the same as before.

Subsequent vacuuming replaces existing vacuuming specifications and must be at least as restrictive. Thus, it is not immediately possible to relax vacuuming of `EmpDep` to retain the facts pertaining to the two most recent weeks, i.e., to issue

```
ALTER TABLE EmpDep
VACUUM NOBIND(DATE 'now - 14 days')
```

This specification cannot be honored because data pertaining to the period between one and two weeks ago may already have been deleted. Instead, it is possible to issue this vacuuming specification.

```
ALTER TABLE EmpDep
VACUUM DATE 'now - 7 days'
```

In this specification `now` is bound when the specification is issued. So at the time it is issued, this specification is exactly as restrictive as the specification it replaces and is thus allowed. Note also that the specification, as time passes, allows more data than than did the earlier, unbound now-relative specification. Consequently, seven days later it possible to issue a new specification as follows.

```
ALTER TABLE EmpDep
VACUUM NOBIND(DATE 'now - 14 days')
```

This way, an unbound now-relative vacuuming specification may be replaced by a less restrictive, unbound now-relative specification.

When transaction time support is added to a table after it initial creation, i.e., using an `ALTER` ... `ADD TRANSACTION` clause, the default cut-off point is set to the time when the `ALTER` clause takes effect.

3.2 Properties of the Proposal

Vacuuming is specified on a per table basis. For each table, exactly one cut-off point is recorded in the system catalog. Only data current after the cut-off point is available to queries. Both absolute and now-relative cut-off points may be specified. A now-relative cut-off point is of the format `DATE 'now` $\pm$ *span*`'` where *span* is some span, i.e., duration of time. (In SQL terminology, a span is an `INTERVAL`.) Now-relative spans may be left unbound using the `NOBIND` function as follows: `NOBIND(DATE 'now` $\pm$ *span*`')`. In this case, the value of the cut-off point changes as time advances.

The cut-off point of a table is initially set to the time when the table was created. A vacuuming specification may change a cut-off point to a time that is either not before the current cut-off point or is after the current time, at the time the specification takes effect.

For example, with the current cut-off point being `NOBIND(DATE 'now - 14 days')`, an `ALTER` statement that takes effect on August 31, 1994, may change the cut-off point to `DATE 'August 17, 1994'` or to this: `NOBIND(DATE 'now - 7 days')`. Assuming that the present time is August 31, 1994, a current cut-off point of `DATE 'September 15'` may be replaced by a new cut-off point, `DATE 'September 1, 1994'`

Vacuuming of a table may be specified at any time during the lifespan of the table.

In summary, the flexibility with respect to what can be vacuumed is restricted to a practical minimum. Vacuuming is specified solely in terms of the transaction times of tuples, and it is possible only to vacuum data currently earlier than a chosen cut-off point. It is not possible to vacuum, e.g., all information for a particular employee or group of employees, just as it is not possible to vacuum, e.g., salary information for all employees not currently employed.

Yet, with these restrictions, vacuuming is still useful. In addition, this limited functionality of specifications provides simplicity to other aspects of vacuuming.

It is easy to keep record (in the data dictionary) of the vacuuming that has been specified for a table. For each table, exactly one cut-off point needs to be recorded at any time. By the same token, it is easy for users to comprehend what vacuuming has been specified for a table.

As another consequence, it is easy to (re-)specify vacuuming for tables. To prepare a correct vacuuming specification, it is necessary only to pick a cut-off point that is either not before the current cut-off point or is after the current time.

More importantly, we have been able to exploit the fact that, with the restrictions we have imposed, the effect of vacuuming on the results of queries is also relatively straightforward.

Without going into detail, a general, user-friendly system would be designed so that users need not worry at all about what has or has not been vacuumed. The system would be capable of detecting when vacuuming may have made a difference to the result of a query and of communicating this to the user in a convenient and effective manner. For an interactive user, the system could inform the user about what vacuuming specifications may have affected the result of the query, or it could provide the user with a list of queries similar to the original query and which are not affected by the vacuuming.

By allowing only cut-off points, we have been able to choose a simpler design. Specifically, we require users to know about the cut-off points (as stored in the system tables) for all tables and to then be able to decide whether or not (and how) queries may be affected by vacuuming.

Finally, efficient implementation appears unproblematic. By a simple application of query modification techniques that database systems generally already support, it is possible to implement vacuuming specifications in a simple and efficient manner.

Briefly, the system uses the cut-off points to modify queries on vacuumed tables so that data that has been specified as vacuumed are excluded from consideration when computing the result. This is similar to the type of query modification that is employed when views are used in queries. To modify a query that uses a view, the view name is simply replaced by its definition. To modify a query on a vacuumed table, the name of the vacuumed table is simply replaced by the query expression that yields the part of the table that has not been vacuumed, i.e., the part of the table that is after the most recently specified cut-off point. Since vacuuming expressions are very simple, this added query modification step is also very simple. This query modification strategy ensures that vacuuming specifications are given the right semantics, and it also allows the system to do the actual deletion of data asynchronously, e.g., when system load is low.

Acknowledgements

Discussions with Richard Snodgrass helped greatly in making various design decisions leading to a minimal, SQL-compatible design. I am also very thankful for enlightening discussions with the additional members of the TempIS group, Curtis Dyreson, Michael Soo, Nick Kline, and Michael Böhlen.

This work was supported in part by the Danish Natural Science Research Council under grants 11–1089–1 SE and 11–0061–1 SE.

References

[AS86] I. Ahn and R. T. Snodgrass. Performance Evaluation of a Temporal Database Management System. In *Proceedings of the 1986 ACM SIGMOD Conference*, pages 96–107, Washington, DC, May, 1986.

[ASU79] A. V. Aho, Y. Sagiv, and J. D. Ullman. Equivalences among Relational Expressions. *SIAM Journal of Computing*, 8(2):218–246, May 1979.

[Cha90] S. Chaudhuri. Generalization as a Framework for Query Modification. In *Proceedings of the Sixth International Conference on Data Engineering*, pages 138–145, February 1990.

[DS92] C. E. Dyreson and R. T. Snodgrass. Time-stamp Semantics and Representation. Technical Report TR 92-16a, Department of Computer Science, University of Arizona, July 1992.

[DSJ93] C. E. Dyreson, R. T. Snodgrass, and C. S. Jensen. On the Semantics of "now" in Temporal Databases. TempIS Technical Report 42, Department of Computer Science, University of Arizona, April 1993.

[JM90] C. S. Jensen and L. Mark. A Framework for Vacuuming Temporal Databases. Technical Report CS-TR-2516/UMIACS-TR-90-105, Department of Computer Science, University of Maryland, College Park, MD, August 1990.

[JS92] C. S. Jensen and R. T. Snodgrass. Proposal of a Data Model for the Temporal Structured Query Language. TempIS Technical Report 37, Department of Computer Science, University of Arizona, Tucson, AZ, July 1992.

[JS93] C. S. Jensen and R. T. Snodgrass. The TSQL2 Data Model. Commentary, TSQL2 Design Committee, 1993.

[Mel90] J. Melton (ed.). *Solicitation of Comments: Database Language SQL2*. American National Standards Institute, Washington, DC, July 1990.

[Mot84] A. Motro. Query Generalization: A Technique for Handling Query Failure. In *Proceedings of the First International Workshop on Expert Database Systems*, pages 314–325, October 1984.

[RS87] L. Rowe and M. Stonebraker. The Postgres Papers. Technical Report UCB/ERL M86/85, University of California, Berkeley, CA, June 1987.

[Sto75] M. Stonebraker. Implementation of Integrity Constraints and Views by Query Modification. Memorandum, ERL-M514, Electronics Research Laboratory, College of Engineering, University of California, Berkeley 94720, March 1975.

A Modified Language Syntax

The organization of this section follows that of the SQL-92 standard. The syntax is listed under corresponding section numbers in the SQL-92 document. All new or modified syntax rules are marked with a bullet ("•") on the left side of the production.

Where appropriate, we provide disambiguating rules to describe additional syntactic and semantic restrictions. We assume that the reader is familiar with the SQL-92 standard, and that a copy of the standard is available for reference.

A.1 Section 5.2 <token> and <separator>

One reserved word was added.

```
<reserved word> ::=
•         | VACUUM
```

A.2 Section 11.3 <table definition>

The production for the non-terminal <table definition> was previously augmented with an additional, optional clause. Here, that clause is extended with a <vacuuming definition>.

```
<table definition> ::=
            CREATE [ { GLOBAL | LOCAL } TEMPORARY ] TABLE <table-name>
               <table elements>
                  [ <temporal definition> ]
•                 [ <vacuuming definition> ]
                       [ ON COMMIT { DELETE | PRESERVE } ROWS ]
```

One production is added.

<vacuuming definition> ::=
- `VACUUM` <datetime value expression>

Additional general rules:

1. The <vacuuming definition> is only allowed when the table supports transaction time.
2. If <vacuuming definition> is not specified, `VACUUM TIMESTAMP CURRENT_TIMESTAMP` is assumed (the default).

A.3 Section 11.10 <alter table statement>

The <alter table action> is augmented with the ability of changing vacuuming definitions.

<alter table action> ::=
- | <alter vacuuming definition>

<alter vacuuming definition> ::=
- `VACUUM` <datetime value expression>

Additional syntax rules:

1. For the <cast valid definition>, T shall be a valid-time or bitemporal table.

Additional general rules:

1. The <datetime value expression> must, when the <alter table statement> is issued, evaluate to a time value that is either not before the current cut-off point or is after the current time.
2. When an <alter table statement> with an <add transaction time> clause, but with no <alter vacuuming definition>, is applied to a table that does not support transaction time, the time the <alter table statement> takes effect is used as the cut-off point of the altered table.

A.4 Section 21.3.8 TABLES base table

```
ALTER TABLE TABLES ADD COLUMN
       VACUUM_CUT-OFF    TIMESTAMP
```