DRAFT - ICT Project Guidance

Design:   
Technical – Relational Data Versioning

Version:

0.1

Author:

Sky Sigal, Solution Architecture

## Purpose

This document summarises the need, and the methods to consider, for implementing data versioning within an system’s operational relational database.

## Synopsis

The primary data store type for information systems is remains the relational database. When storage was both physical and expensive, it was a recommended best practice to physically delete records to free space for continued growth. With cloud based virtual and practically infinite storage space that is inexpensive, the emphasis has changed to keeping past versions of record for both auditing and change analysis purposes.  
Implementing Data Version control in a relational database is difficult.

## Contents

[Description 1](#_Toc154927865)

[Synopsis 1](#_Toc154927866)

[Contents 2](#_Toc154927867)

[Purpose 3](#_Toc154927868)

[Background 3](#_Toc154927869)

[Outcomes [Objectives] 3](#_Toc154927870)

[Options [Considered & Selected] 3](#_Toc154927871)

[Constraints 3](#_Toc154927872)

[Assumptions 3](#_Toc154927873)

[Dependencies 3](#_Toc154927874)

[Decisions 3](#_Toc154927875)

[Deliverables/Outputs 3](#_Toc154927876)

[Heading Level 3 3](#_Toc154927877)

[Heading Level 4 4](#_Toc154927878)

[Appendices 5](#_Toc154927879)

[Appendix A - Document Information 5](#_Toc154927880)

[Versions 5](#_Toc154927881)

[Images 5](#_Toc154927882)

[Tables 5](#_Toc154927883)

[References 5](#_Toc154927884)

[Review Distribution 5](#_Toc154927885)

[Audience 5](#_Toc154927886)

[Structure 5](#_Toc154927887)

[Diagrams 6](#_Toc154927888)

[Terms 6](#_Toc154927889)

## Background

…

Context

The following approaches are explored with a common set of tables: Foo, with an single reference value being from a Reference Data called Boo. Both tables have an UUID based ID.

### Single Table, Multiple Key/Value Archiving

A well-known solution[[1]](#footnote-2) is to develop a single table to manage keys and value pairs of record changes (Creations/Adds, Updates, Deletions).

The table is essentially a table of fields to persist the name of the source table, the serialised value of the source table identifiers, the name of a column in the table, the serialisation of the column’s value, an identifier of the system user who made the changes, and when.

**Advantages**

The solution is a well-known approach to enterprise system development and code is readily available on the net to develop the required stored procedure on market leading relational databases.

If a source table’s schema changes, by adding, removing or renaming columns, the archiving table’s schema doesn’t need to change.

The identities of records remain the same during change operations, so referential identity is not impacted.

**Considerations**

Consider placing the archiving table in a separate database schema to keep the primary schemas focused on operational and business entities.

Consider creating a composite index for the table composed of the Table Name, Table Identifier, and Change Date – noting that the Table Identifier may contain strings that are longer than best practice to serialise.

The logic to read records being persisted and creating multiple writes to the archive table can be implemented as an RDMS specific stored procedure in databases that provide stored procedure capabilities. The stored proc approach makes it possible to handle cascading delete scenarios, recording the deletion of the dependent tables first.

Stored procedures make the code less portable, which can make the logic impossible to test using other (lighter) databases.

If code portability is important, the system logic can be executed in the application server by trapping the save event of ORM saves. But while it adds portability, it increases traffic between application server and database server, while also making it much harder to record and restore cascading delete scenarios.

Avoid this approach on systems that are predominantly write operations (e.g.: a school assessment service that writes the learners answer after each question).

As always, we recommend the use of application server generated time based UUIDs rather than DBMS developed integers or UUIDs.

As always, avoid physically deleting records. Physically deleting records complicates considerably the restoration of records, especially if the identifiers are RDMS generated. It also complicates the restoration of cascaded deletes that the ORM was never signalled about.

**Disadvantages**

The approach requires multiple writes – one per source table column – making updating records much slower.

The single table over time becomes tremendously large – basically growing exponentially as it is the number of records changed over the service’s lifespan x the number of attributes in each, not forgetting additional columns in each record to track who & when did the change.

This will impact database and backup storage size requirements as well as database restoration times, potentially impacting Recovery Time Objective (RTO) agreements.

A separate database could be used, but the underlying issue will remain. Namely, the size of the table makes queries necessary for showing a record’s history of records slow down over time.

There is no solution for correctly restoring from archived records the value of fields in a table whose schema was changed since it the archive records were persisted. It may work most times, but not all.

As previous records don’t exist in a production table, end users can’t navigate to them to see the shape of a resource was in the past.

### Single Table, Single XML based Record Archiving

An alternate solution is to store all the field changes as an xml document within a single document and persist that in an archive table.

The shape of the table is similar to the previous approach, with a serialisation of the source table name, a serialisation of the source table identifiers, an identifier of the user who made the changes, the operation type, and when.

Much of the advantages, considerations and disadvantages are the same as the previous approach, bar the points listed below.

**Advantages**

The table size grows less rapidly than the previous approach.

**Consideration**

Consider placing the archive table in its own dedicated schema to let operational database schema stay focused on system and business concerns.

Consider using XML over JSON for the ability to leverage XSLT to develop pages, noting that each record type may require its own XSLT to render records.

Not all RDMS systems can manage XML so the approach may not be portable – possibly impacting testing approaches later.

**Disadvantages**

If the data is persisted in the operational database, it will impact operational and backup space requirements as well as the time to make data restoration, that may impact DTO.

The data could be persisted in a separate database, but the general issue remains. Queries, Backups, and restorations will constantly increase over time.

The structure of the XML file is not managed by the database schema, so queries can’t be developed to select the records for a list in the same operation as looking up the values to include the reference data values the FKs are pointing to.

An option to address this could be to make queries so that the XML file contains the both the original FK value and the reference data value they point to at the time of the data change operation. However, this approach cannot be done generic/universal manner as databases don’t support interfaces or other approaches to automatically determine the name of the column to include (“Text”, “Title”, “DisplayText”, etc.).

As previous records don’t exist in a production table, end users can’t navigate to them to see the shape of a resource was in the past.

### Single Table, Single document Record in an External Store

The above approach can be applied, using an external, non-relational, cloud storage solution.

**Advantages**

Database size is not affected by number of archiving records created, so the time required to save and restore data is not negatively affected.

Cloud storage solutions provide backups as a service, so novel backup and restoration solutions are not required to be developed.

Saving to another datastore does not eat up precious RDMS I/O bandwidth.

**Considerations**

.

**Disadvantages**

If done by stored proc, the database is required to a) have access to the cloud b) persist integration credential to the cloud datastore. Both aspects increase system security failure risks.

If not done as a stored proc, cascading deletes are not managed.

### Multiple Archiving Table (Per Table) Archiving

An approach is to duplicate tables as archive tables, removing any instances of referential integrity on FK constraints.

The Archive table will contain additional metadata columns to capture the identifier of the User who made the change, the change operation type (Create, Update, Delete), and the date and time of the operation.

**Advantages**

FK integrity between operational entities remains straightforward, requiring no change.

**Considerations**

Consider using a separate schema (e.g., ‘Archives’), otherwise the operational schema becomes overly complex with twice as many tables, with all the archive tables having no referential integrity so they just hang there in any schema diagram.

Both the Id and any FK are required to have their constraints removed. For one, there may be multiple entries with the same Id – due to the auditing of different operations on the same records. For another, the FKs would be pointing to records in source reference data tables, not their archive related table.

Again, this can be done in the RDMS, using Stored Procedures, or within the Application itself, with similar advantages & disadvantages as previously discussed. Stored Procedures are less portable, so immediate impacts -- aside from potential licensing costs – is that the same database type (rather than a lighter and faster one) is required for testing. However, they are able to be notified and correctly handle cascading deletes.

as ORMs need to capture Update and Delete operations and clone existing values into the archive tables before letting the change operation be completed, updating the current values. Handling cascading deletes on the application server would be difficult.

If the interface wanted to show a history of previous changes, then the data use system logic to query the archive table, and make a query for each Foo record displayed, obtaining Bar values as they were at the time the record existed in Foo.

The complexity increases linearly as the number of FK attributes within Foo.

The complexity also increases again if Bar has FKs to other tables.

The complexity of the operational database schema increases linearly by the number of tables that are archived if a “Shadow” schema is not used for the Archive records.

**Disadvantages**

Each deployment requires both the operational schema and archiving schema to be updated at the same time. This is difficult to coordinate under the best of circumstances, even less likely to be error free in a context with multiple deployment streams, or 3rd party plugins.

As previous records don’t exist in a production table, end users can’t navigate to them to see the shape of a resource was in the past.

## Audit Records

An alternate approach is to record the values of the record, in a XML or json file in a separate datastore.

To show record values as they were when the record was changed or deleted, the system must first retrieve the existing referenced data to record the values as well as the references.

With the original values embedded in the record, the system can show previous records as they were at the time of the operation.

As the number of referenced columns increases, the number of queries that need to be completed first, before the saving of the changed record, increases linearly.

If the record is metadata for media stored elsewhere.

Rolling back is done by updating the record with the values from the XML.

If the referenced data Bar has changed in the meantime, a decision has to be made by the authorised user to roll back the reference data as well.

A means of not having to roll back Reference Data is to use a ValidFromUTC/ValidUntilUTC attributes, such that when Foo is reverted back with data from the XML record, the reference data value is returned to pointing to the value that exists at the time.

## Valid From/To

Each table can be given a ValidFrom/Until columns. The To column is nullable. Both are indexed.

**Considerations**

The Query used to retrieve the list of Reference Data to display in dropdowns has to be add Constraints to only return current values (where ValidUntil is NULL).

The ID must not be UNIQUE to permit multiple instances of a record, with a different validuntil datetime be recorded.

A unique key has to be created from the combination of the non nullable ID and the nullable ValidUntil.

See: [Entity Framework Core: Foreign key linked with a non-primary key – Felipe Gavilán (gavilan.blog)](https://gavilan.blog/2019/04/14/entity-framework-core-foreign-key-linked-with-a-non-primary-key/) and [Keys - EF Core | Microsoft Learn](https://learn.microsoft.com/en-us/ef/core/modeling/keys?tabs=data-annotations#alternate-keys)

**Disadvantages**

If you update a Reference Data item, you have to update things in a transaction. Specifically, you have to Update the existing Record with a filled ValidTo, then add the new record, with a null ValidUntil date. If you didn’t add a transaction, you run the risk of a another record being read, having a dependency on a record that no longer exists because the ValidUntil has been filled. And if you tried ot save the record first, then update the old, it would fail due to two Records having the same Id and null values.

**Disadvantages**

There is more complexity in setting up AlternateKeys (in EF.Core) – but it should become easier.

The ID and ValidUntil field must be UNIQUE, allowing only one version of each record to have a Null value.

Keeping previous records permit users to navigate to previous copies of the record.

It \*does\* mean that Foo table needs to have two fields to reference the FK.

Stord Procs Approach

Recommended Approach

Recommendations

While acknowledging the use of Stored Procedures does diminish portability, consider using a stored proc on the server side as it can also be configured to capture cascading data changes (Deletes of Referenced data) that an ORM would not be made privy to.

To diminish the number of operations required to persist an audit record in a database, persist the values in a single record, using a commonly used format (e.g., XML or JSON).

To keep the operational schema focused on system and business entities, use a separate schema for the temp audit table(s).

To avoid the database having to have access and credentials to an external cloud based datastore, use a database table to store records temporarily -- until a system-initiated background task periodically moves them to cloud storage.

By moving the records to a datastore external to the operational database backup space and restoration time are not negatively affected over time.

The cloud-based storage is not required to store all changes in a single datastore -- a datastore could be developed for each table.

This might make rendering historical records faster.

This may also facilitate the case when database table names are changed, as datastore names may be both faster and easier to update than the table name value within thousands of records.

Acknowledging that FK values point to Reference Data that may have been changed, affecting the rendering of previous copies of records, do not attempt to persist the values of reference data entities that FKs point to (it can’t be done at the database level in a general way as it can’t guess the name of the reference data column to include, or how many attributes to include). The best that can be done is to query these values when rendering history records.

The solution could also be implemented in the application logic, in the ORM save event, to permit some portability and dynamic testing of the general approach -- acknowledging that a) the logic is similar but not exactly the same, b) cascaded deletes will not be captured.

Rendering historical records is not perfect, but it also probably doesn’t need to look like the original record, just a summary record, with changed values under it, needing a different rendering solution.

Appendices

Appendix A - Document Information

### Versions

* 1. Initial Draft

### Images

[Figure 1: TODO Image 2](#_Toc144995112)

### Tables

[Table 1: TODO Table 3](#_Toc145048484)

[Table 2: TODO Table 2 3](#_Toc145048485)

### References

**There are no sources in the current document.**

### Review Distribution

The document was distributed for review as below:

|  |  |
| --- | --- |
| Identity | Notes |
|  |  |
|  |  |
|  |  |

### Audience

The document is technical in nature, but parts are expected to be read and/or validated by a non-technical audience.

### Structure

Where possible, the document structure is guided by either ISO-\* standards or best practice.

### Diagrams

Diagrams are developed for a wide audience. Unless specifically for a technical audience, where the use of industry standard diagram types (ArchiMate, UML, C4), is appropriate, diagrams are developed as simple “box & line” monochrome diagrams.

### Terms

Refer to the project’s Glossary.

##### IT

: acronym for Information, using Technology to automate and facilitate its management.

##### ICT

: acronym for Information & Communication Technology, the domain of defining Information elements and using technology to automate their communication between entities. IT is a subset of ICT.

1. [Jon Galloway - Adding simple trigger-based auditing to your SQL Server database (asp-blogs.azurewebsites.net)](https://asp-blogs.azurewebsites.net/jongalloway/adding-simple-trigger-based-auditing-to-your-sql-server-database) [↑](#footnote-ref-2)