Müllerstrasse 64, 8004 Zürich, 043 311 59 15

Enter AG

Long-term Preservation of Relational Databases
What needs to be preserved how?

Version: 1.0

Source: http://www.enterag.ch/hartwig/SIARD_Criterion.pdf

15. January 2013

Author: Dr. sc. math. Hartwig Thomas, Enter AG

LONG-TERM PRESERVATION OF RELATIONAL DATABASES

What needs to be preserved how?

Table of Contents

1 INTRODUCTION

The author of this article has defined, designed and implemented a system for the long-term preservation of relational databases for the Swiss Federal Archives[1]. This system SIARD (for Software-Independent Archiving of Relational Databases) consists of a file format definition (SIARD format) and a "reference implementation" (*SIARD Suite*) for converting relational databases to normalized SIARD files and for uploading SIARD files to database systems for research using SQL queries of any desired complexity.

In this connection he has been confronted with questions like the following:

- Which aspects of a relational database need to be preserved?
- How can one judge the quality of the preservation system?
- What are the characteristic attributes of a database?
- Why does your system not restore database triggers?
- Why does your system not check consistency and reject some databases not conforming to our standards of database design?
- How can one check the validity of an archived database file?
- How can one check its integrity?

This article will attempt to answer these questions and many more.

2 THE PROBLEM OF LONG-TERM PRESERVATION

Many people believe, that the limited life-time of storage devices like tape, CD-ROM or solid-state drives is the main problem for the long-term preservation of digital content. In reality, however, this problem is easily solved in a digital world, where perfect copies of any number of bytes are almost for free.

A professional archive will upgrade its storage devices from time to time and copy the archived bits to the new devices. It will store its content redundantly in three or more copies so that preservation is secured should any of the storage locations by physically damaged. It will monitor the identity of those copies regularly and fix any problems, that are detected.

The real problem of long-term preservation is the question of the usability of the archived bits. If the structure of these bits is only understood by proprietary software, it is unlikely that this software will be available after a reasonably long period of time. Typically archived data are accessible to the general public some 50 years after they were archived and should remain so for another 50 years. In this paper "long-term preservation" means preservation for at least 50-100 years.

2.1 Code and Data

Experience of the technical developments in the last 50 year shows that it is very unlikely that today's software will run and be usable on systems (hardware and software) available in 50-100 years from now. Even if you use a perfect DOS emulation program under Windows 8 today, it can hardly guess which printer driver was buried in the Word 2 file from 1988. But even if the emulation were able to do a perfect job, anybody today would find it very hard, if not impossible, to use Word 2 today. Which function key was used for which activity? So we conclude – at least until a durable standard of a digital software system emerges:

Code cannot be archived.

The case for data is slightly better. If we open an ASCII file that was created 50 years ago in an editor, we can still read it. If we play an old CD-Audio, that was bought 30 years ago, it is still played correctly. The reason for this durability of some data is the open standardization of for-

[1] http://www.bar.admin.ch/dienstleistungen/00823/00825/index.html?lang=en

mats. As long as the interpretation of the archived bits is not forgotten, programs can be written to make them available on new hardware in new software environments.

Many institutions are obliged by law to preserve and be able to access their data for long periods of time. They all have experienced many upgrades of their IT environments which needed to make sure that access to the old data remained intact:

Data have a far longer life-expectancy than code.

Therefore there is a chance, that data can be preserved in archives over 50-100 years or longer. But this requires a careful archival strategy. Many of us already have lost plenty of data in the relatively short time of our digital lives.

2.2 Relational Databases

That data live longer than code was recognized relatively early in the history of computers. Around 1970 this led Edgar F. Codd to develop the so-called relational model for databases, which was designed to help separate code from data[1]. Today relational databases are the backbone of almost all government activity. For a short time there was a flurry of so-called object-oriented databases in the 90s of the last century, which cannot be preserved in the long-term, because they mix code and data inextricably. This hype did not last and the vast majority of all databases in use today are relational databases[2].

So the problem arises, how may relational databases be preserved?

3 EMULATION – MIGRATION – NORMALIZATION

Currently the discussion about archival of digital material is dominated by the two strategies Emulation and Migration. We propose to add Normalization as a third option.

Emulation as an archival strategy considers the machine used to access the archived materials like a film projector that must be used for watching archived movies. So, for example, computer games for DOS are preserved by preserving the DOS hardware or a simulation of it. This approach does not take into account that a computer is a general-purpose, universal machine, whereas a projector serves a single special purpose.

Migration as an archival strategy advocates transforming the archived materials into a new format whenever their format is not supported anymore by the archive's IT environment.

Normalization as an archival strategy aims to keep archived material unchanged forever. This is possible, if the material is archived in a “normal” form for the type of objects archived. For example any image format, that is documented in open standards and permits the reconstruction of the color of each spot on a picture can be used for normalization of picture data. Based on this standard, anybody can program a denormalization into image formats suitable for viewing the archived images 50-100 years from now. Similarly any format describing the color of each spot as well as the frequency and intensity of sound at any moment of time of a moving picture can be used for normalizing video content. The same is true for audio content, where the CD audio format is an excellent candidate for normalization because of its widespread use. The only issue in these cases is the choice of resolution which must be high enough, so the information lost by quantization will not be regretted in the future.

The case of text documents, spread sheets and presentations is somewhat more complex, because the exact extent of these types of objects is not universally defined today. In spite of minor disagreement, some promising ISO standards for “office” documents (ODF, OOXML) have been developed in the last decade and show a marked tendency to converge.

---

[1]For a historical account see: C. J. Date: *The Database Relational Model, A Retrospective Review and Analysis*, Addison Wesley,2000,

http://www.amazon.de/The-Database-Relational-Model-Retrospective/dp/0201612941

[2]For a comprehensive introduction to relational databases see: C. J. Date: *An Introduction to Database Systems*, Eighth Edition, Pearson Education Inc., 2004,

http://www.amazon.de/Introduction-Database-Systems-Chris-Date/dp/0321197844

3.1 Emulation

For digital objects of the type “relational database” emulation means to preserve the database data as well as the database application on the original software platforms – which are possibly “emulated” on future hardware. Preservation boils down to keeping a running environment available for each preserved object. In the recent past this strategy would have necessitated the availability of various emulations and experts in the archive and for the dissemination:

Year	*Submission uses*	*Archive hires*	*Dissemination needs*
1997	Oracle 8	1. DBA	Oracle 8 client
2000	SQL Server 2000	2. DBA	Oracle 8 client and SQL Server 2000 client
2002	MySQL 4	3. DBA	Oracle 8 client and SQL Server 2000 client and MySQL 4 client
2003	Oracle 9		Oracle 8 client and SQL Server 2000 client and MySQL 4 client and Oracle 9 client
2005	SQL Server 2005		Oracle 8 client and SQL Server 2000 client and MySQL 4 client and Oracle 9 client and SQL Server 2005 client
2006	Oracle 10	4. DBA	Oracle 8 client and SQL Server 2000 client and MySQL 4 client and Oracle 9 client and SQL Server 2005 client and Oracle 10 client
2008	Oracle 11		Oracle 8 client and SQL Server 2000 client and MySQL 4 client and Oracle 9 client and SQL Server 2005 client and Oracle 10 client and Oracle 11 client
	SQL Server 2008	5. DBA	Oracle 8 client and SQL Server 2000 client and MySQL 4 client and Oracle 9 client and SQL Server 2005 client and Oracle 10 client and Oracle 11 client and SQL Server 2008 client
	MySQL 5	6. DBA	Oracle 8 client and SQL Server 2000 client and MySQL 4 client and Oracle 9 client and SQL Server 2005 client and Oracle 10 client and Oracle 11 client and SQL Server 2008 client and MySQL 5 client
2012	SQL Server 2012		Oracle 8 client and SQL Server 2000 client and MySQL 4 client and Oracle 9 client and SQL Server 2005 client and Oracle 10 client and Oracle 11 client and SQL Server 2008 client and MySQL 5 client and SQL Server 2012 client

Extrapolate this table until 2060!!

Only emulations of servers and clients are required. But their use requires know-how of the usage of old software for database administrators (DBA) and users of the client software of the archive!

3.2 Migration

For relational databases the migration approach probably would mean to archive the dump format of the database products. The following activities and capabilities would have been needed by archives and dissemination agents in the recent past:

Year	*Submission* *creates*	*Archive* *migrates and hires*	*Dissemination* *restores*
1997	dump to Oracle 8 dump files	store Oracle 8 dump file 1. DBA	restore Oracle 8 dump
2000	dump to SQL Script with SQL Server 2000 types	store SQL Script with SQL Server 2000 types 2. DBA	restore Oracle 8 dump and restore SQL Server 2000 dump
2002	dump to SQL Script with MySQL 4 types	Store SQL Script with MySQL 4 types 3. DBA	restore Oracle 8 dump and restore SQL Server 2000 dump and MySQL 4 dump
2003	dump to Oracle 9 dump files	migrate all Oracle 8 dump files in the archive to Oracle 9 dump file format / store Oracle 9 dump file	restore Oracle 9 dump and restore SQL Server 2000 dump and MySQL 4 dump
2005	dump to SQL Script with SQL Server 2005 types	migrate all SQL Server 2000 dump files to support the new SCHEMA creation	restore Oracle 9 dump and restore SQL Server 2005 dump and MySQL 4 dump
2008	dump to Oracle 10 dump files / dump to SQL Server 2008 dump files / dump to MySQL 5 dump files	migrate all Oracle 9 dump files to Oracle 10 dump file format / migrate all SQL Server 2005 dump files to SQL Server 2008 dump format / migrate all MySQL 4 dump files to MySQL 5 format	restore Oracle 10 dump and restore SQL Server 2008 dump and MySQL 5 dump
2012	dump to SQL Script with SQL Server 2012 types	migrate all SQL Server 2008 dump files to SQL Server 2012 dump format	restore Oracle 10 dump and restore SQL Server 2012 dump and MySQL 5 dump

Extrapolate this table for all supported database systems until 2060!!

The archive must always use the most recent version of the database and must offer migration of older formats for a number of years after each version change.

The number of DBAs in the archive is roughly equal to the number of database systems supported.

A major migration to a different database system takes place, when a database is discontinued (e.g. when Oracle discontinues MySQL).

3.3 Normalization

The normalization strategy for relational databases entails defining a “normal” digital storage format that captures the essence of the data in a relational database independent of the database vendor. We advocate using the SIARD format[1] for this purpose. The data are stored in the archive and are never changed. Only a single application for normalization and denormalization needs to be kept up-to-date.

[1] http://www.bar.admin.ch/dienstleistungen/00823/00825/index.html?lang=en
http://www.ech.ch/vechweb/page?p=dossier&documentNumber=eCH-0165&documentVersion=1.0

Year	*Submission*	*Archive*	*Dissemination*
2008	submit SIARD file conformant to SIARD Format 1.0	store SIARD 1.0 file	upload SIARD 1.0 file to database of choice
2014(?)	submit SIARD file conformant to SIARD Format 2.0	store SIARD 2.0 file	upload SIARD 1.0 or SIARD 2.0 file to database of choice

The various versions of SIARD Format coexist in the database. Each version is based on open standards and is itself an open standard, which can be supported even in the year 2060.

The software for normalization (download from database) and denormalization (upload to database) must be continuously kept up-to-date, but the archived material never needs to be touched.

3.4 Why is Normalization preferable to Migration or Emulation?

Emulation appears to be unmanageable because the number of database administrators (DBAs) as well as the number of client programs increases with every version of every database system. No archive in the world can pay the salaries of so many database administrators.

Migration has proved to be somewhat unstable. Even minor upgrades from Oracle 10gR1 to Oracle10gR2 database dump formats led to serious data loss. All migrated databases must be checked for completeness after each migration. The amount of material to be migrated grows with the amount of material archived. Lossless major migrations from one database system to another are quite difficult. Finally, the number of database systems that need to be supported in an archive is equal to the number of database systems used be the agencies submitting databases.

Normalization has the advantage that nothing is migrated and therefore nothing is lost in migration. Also the archive only needs a single database system as a denormalization platform, where the database content is made available to the users.

4 WHAT NEEDS TO BE ARCHIVED?

The mandate of most public archives on national, district or local level is to preserve the information, not its form. This is lucky, because we do not believe, that present-day code will run on any machine in 50-100 years from now. Thus we cannot preserve the presentation of and interaction with the data in a database except with a few screen shots and in the documentation of the database application.

Relational databases were developed to enable a clear separation of data and code. Most government agencies use large relational databases which actually enforce such a separation. Thus the only viable goal for long-term database preservation today is the archival of the data.

We want to increase the precision of this statement and turn it into a criterion of how to establish that we have preserved the complete relational database. First of all, a relational database is a collection of tables. The data to be preserved reside in the cells of these tables. Traditionally the cells of table could hold data of the types String, Number and Timestamp. So we have preserved the database, if we have preserved the “primary” information of the content of each cell of each table of the database, “meta” information about the table structure and the inter-table referential structure.

Relational databases were never standardized on the bit level. Instead Edgar F. Codd insisted on defining a query language, that would permit to access the data in a non-hierarchical fashion without forcing the user into asking only a subset of the questions one might want to direct at a structured data collection. This query language was based on mathematical logic (predicate calculus), is called SQL today, and pretty much defines, what a relational database is.

Thus we refer to SQL:1999 as the standard that "defines", what a relational database is[1]. We refer to this particular standard although for our purposes it is largely equal to its predecessor SQL-92[2], because it defines conformity to the standard in a more strict way and because it is fairly recent. We have not updated to SQL:2003 because the only changes introduced by that version concerned code (stored procedures), not data.

So if SQL defines what a database is,

the database is preserved completely, if the same SQL SELECT queries executed on the preserved database as on the original database yield the same results.

In the archival context we only are interested in querying the data, not in changing them. So we do not insist, that all SQL statements have the same results but only the SQL SELECT statement.

4.1 Criterion of Complete Preservation

Thus the requirement of complete preservation splits into a requirement for the meta data to record the structure of each table, and a requirement for the primary data to record the content of each cell of each table.

Criterion of Complete Preservation of Meta Data

For meta data this means, that the following is the minimum set of meta data that needs be preserved. On the database level, the meta data of all tables (more technically: of all "base tables") of the database must be preserved. On the table level, the meta data of all columns must be preserved as well as all referential information linking the tables.

Table Meta Data

Name	mandatory
Meta data for all columns	mandatory
Primary Key	optional but desirable
Candidate Keys	optional but desirable
Foreign Keys	optional but very desirable

Column Meta Data

Name	mandatory
Type (SQL:1999 type)	mandatory

The reason, why the key constraints are optional, is that their presence or absence does not change the result of any SELECT query. On the other hand they document the intended data-

[1]Unfortunately most standardizing bodies make the user pay a high price for access to the original standards documents, which are usually worded in very formal legalese. But fortunately there are books available that explain the SQL:1999 standard in comprehensible language:

Jim Melton (Oracle Corporation), Alan R. Simon: *SQL:1999 Understanding Relational Language Components*, Morgan Kaufmann Publishers, 2002

http://www.amazon.de/SQL-Understanding-Relational-Components-Management/dp/1558604561

Jim Melton (Oracle Corporation): *Advanced SQL:1999, Understanding Object-Relational and Other Advaced Features*, Morgan Kaufmann Publishers, 2003

http://www.amazon.de/Advanced-SQL-Understanding-Object-Relational-Management/dp/1558606777

[2]Even today the most excellent introduction into the SQL standard:

C. J. Date, Hugh Darwen: *A Guide to the SQL Standard*, Addison-Wesley, 1997

http://www.amazon.com/Guide-SQL-Standard-4th-Edition/dp/0201964260

base structure, its tables and inter-table references. So they are a very useful addition for researchers of the future.

There are many other SQL:1999 objects, which also do not change the result of any SQL:1999 query, but are useful information helping the interpretation of the meaning of the archived data, like VIEWS, CHECK CONSTRAINTS, TRIGGERS and ROUTINES. Like the key constraints they are of secondary importance to archived data, because they constrain and govern *changes* of the database. But when a database is *archived*, we do not intend to *change* it in the future. That is part of the definition of what it means to archive data. Unlike key constraints, all of these objects can involve *code.* VIEWS are defined by queries often containing function calls into *code*, CHECK CONSTRAINTS – like VIEWS – are expressed by queries that can contain *code* and are meant to prevent *changes* that would make a database inconsistent. TRIGGERS specify, which *code* to execute on *changes*. ROUTINES (stored procedures) are *code*. The more code these objects are based on, the smaller are the chances, that they can be meaningfully interpreted in the future. But they certainly have informational value.

Finally there are some SQL:1999 objects for handling access control (USERS, ROLES, GRANTS). In an archival situation we do not want any access restrictions to persist, but it is informative to know about them.

Criterion of Complete Preservation of Primary Data

The most important question, however, when we want to archive a database, is the question of whether we have preserved all *primary* data. Directed by the goal that any SELECT query should yield the same result, whether executed on the original data or the preserved data, we arrive at the following criterion for the completeness of the preserved data.

If we normalize the database in our archive format and denormalize it to the original system as a restored database, then for each table TO with columns CO1, CO2, … CON of the original database there must be a corresponding table TR with columns CR1, CR2, … CRN of the restored database, such that the following three SQL queries result in the same number:

```
SELECT COUNT(*) FROM TO - - number of records of TO,
SELECT COUNT(*) FROM TR - - number of records of TR,
SELECT COUNT(*) FROM TO,TR
WHERE
  TO.CO1 = TR.CR1 AND
  TO.CO2 = TR.CR2 AND
  ...
  TO.CON = TR.CRN - - number of records from TO and TR with identical cell contents.
```

Thus the answer to the question of whether the cell content is faithfully preserved is based on the meaning of the equal sign (=) in SQL (SQL:1999 to be precise). This shows, that the preservation is complete, even if the types of columns CO1 and CR1 are not identical but comparable (e.g. CHAR(5) and VARCHAR(8)) and all (e.g. string) values are the same. This is of considerable use, because today's concrete database systems have introduced a wild flurry of types – many inconsistent with the SQL standard. This is largely a consequence of worries about disk space of earlier times, where it made sense to distinguish longer and shorter strings or numbers. In more modern standards, like the XML standard, this concern has disappeared and all strings have one single type *xs:string* independent of their length.

The above simple version of the Criterion of Complete Preservation fails, when database tables are allowed to have identical records (unfortunately permitted by the SQL:1999 standard, but necessitating the absence of primary and candidate keys for the table), when some columns are so-called large objects, or when the database contains complex data types like ARRAY or user-defined data types (UDTs). Then we are forced to replace the criterion by more complex expressions, but the basic idea remains the same.

Criterion of Complete Preservation – Repeated Records

If a table T with columns C1, C2, … CN can contain repeated records, the expression

```
SELECT C1, C2, … CN, COUNT(*) AS REPS FROM T GROUP BY C1, C2, … CN
```

results in a table with the same records, each record repeated only once, and with an additional last field REPS indicating how many times the record was repeated in T.

Therefore one can replace TO and TR in the queries above by such SELECT expressions:

```
SELECT COUNT(*) FROM
  (SELECT CO1, CO2, ... CON, COUNT(*) AS REPS FROM TO GROUP BY CO1, CO2, ... CON) VO
- - number of distinct records of TO = number of records of VO
SELECT COUNT(*) FROM
  (SELECT CR1, CR2, ... CRN, COUNT(*) AS REPS FROM TR GROUP BY CR1, CR2, ... CRN) VR
- - number of distinct records of TR = number of records of VR
SELECT COUNT(*) FROM
  (SELECT CO1, CO2, ... CON, COUNT(*) AS REPS FROM TO GROUP BY CO1, CO2, ... CON) VO,
  (SELECT CR1, CR2, ... CRN, COUNT(*) AS REPS FROM TR GROUP BY CR1, CR2, ... CRN) VR
WHERE
  VO.CO1 = VR.CR1 AND
  VO.CO2 = VR.VR2 AND
  ...
  VO.CON = VR.CRN AND
  VO.REPS = VR.REPS
- - number of records of VO and VR with identical cell contents and identical number of
repetitions in TO and TR.
```

Criterion of Complete Preservation – Arrays and user-defined datatypes (UDTs)

The complex data types introduced by SQL:1999 are very similar to collections of columns.

So, for a column COA of array type of VO of length M with corresponding column CRA of VR, replace the condition of cell equality above by

```
VO.COA(1) = VR.CRA(1) AND
VO.COA(2) = VR.CRA(2) AND
…
VO.COA(M) = VR.CRA(M)
```

Similarly for a user-defined datatype (UDT) column COU with members M1, M2, … MK of VO with corresponding column CRU, replace the condition of cell equality above by

```
VO.COU.M1 = VR.CRU.M1 AND
VO.COU.M2 = VR.CRU.M2 AND
...
VO.COU.MK = VR.CRU.MK
```

Criterion of Complete Preservation – Large Objects

Even in 1999 there was some concern about storage capacity still around. Therefore the SQL standard does not permit an equality condition on large objects (large character strings CLOBs or large binary strings BLOBs). So, in order to extend and apply the Criterion of Complete Preservation to this case, one must implement two FUNCTIONs *clob_equal* and *blob_equal*, which return TRUE, if two large objects are equal, and FALSE if they are not. With the help of these functions, the condition of cell equality for a column COB of type BLOB must be replaced by

```
blob_equal(VO.COB, VR.CRB)
```

and the condition of cell equality for a column COC of type CLOB must be replaced by

```
clob_equal(VO.COC, VR.CRC)
```

Thus, although the details are somewhat convoluted, it is very easy, in principle, to determine whether all primary data have been preserved completely. The two criteria of complete preservation answer the question of what the “characteristic attributes” of a database are. They are the basis of every quality control of the preservation activity.

5 SIARD FORMAT REQUIREMENTS

The normalization format for relational databases was called SIARD for “Software-Independent Archiving of Relational Databases”. A prototype[1] was developed by S. Heuscher, S. Jaermann, P. Keller-Marxer and F. Moehle as part of the ARELDA (ARchival of ELectronically stored DAta) project of the Swiss Federal Archives in the years 2002-2005. The current version of SIARD Format and *SIARD Suite*[2] were developed by H. Thomas from Enter AG for the Swiss Federal Archives in the years 2007-2008. Their use is offered free of charge to any interested party by the Swiss Federal Archives. It has since been expanded and maintained by Enter AG for the Swiss Federal Archives. In 2013 it was homologized as E-Government Standard eCH-00165[3].

The decision that a normalization format was needed for relational databases ,and the concept of what needs to be preserved presented above, led to the following requirements for the new normalization format to be defined.

5.1 The SIARD format *must* store a complete database in a single file.

Experience has shown that approaches to database archival that store the tables individually, experience a lot of problems with referential integrity. Storing the whole database in a folder structure with individual files, resulted in problems with the authenticity of the stored data, because archive employees are wont to “correct” errors in the primary data. It is also very desirable to be able to transmit or copy a database from one place to the other “as a whole”.

5.2 The SIARD format *must* treat all data accessible to an archival user using SQL as a complete database.

Obviously a normalization program cannot access any data that are not accessible to the database user employed for archival. So a complete SIARD file cannot store *more* than what is accessible to the archival user. Many applications access more than one database schema. So from the perspective of the application, everything accessible to it is a unit. It is desirable to be able to store the whole unit as a database. This is possible, if the archival user has access to the same data as the database application. Therefore a complete SIARD file should not store *less*, than what is accessible to the archival user.

By creating an archival user and assigning a judiciously selected number of privileges to it, the scope of the archived database can be fine-tuned at the moment of normalization.

5.3 The SIARD format *must* be based on open standards.

As mentioned above, this requirement must be fulfilled by any storage format for long-term preservation. In a niche situation like the archival of databases, the format will not be preserved because of the sheer number of datasets stored in this format, like e.g. the CD-Audio format. Thus it is even more imperative that the standard be open and freely accessible for all concerned parties.

5.4 The SIARD format *must* store all primary data completely.

What complete storage of primary data means has been defined above. Any SELECT query on the archived table data must yield the same result as on the tables of the original database.

5.5 The SIARD format *must* contain technical meta data about all tables and columns.

The meta data must permit identifying the archived tables and columns and their types.

[1] http://arxiv.org/abs/cs/0408054

[2] http://www.bar.admin.ch/dienstleistungen/00823/00825/index.html?lang=en

[3] http://www.ech.ch/vechweb/page?p=dossier&documentNumber=eCH-0165&documentVersion=1.0

5.6 The SIARD format *should* contain all meta data about database keys.

The database keys (primary keys, candidate keys, foreign keys) are not absolutely required in the archive. The Criterion of Complete Preservation can be fulfilled without any keys. No result of a SELECT query is changed by the absence of a key. Keys are constraints to keep a database from changing to an inconsistent state. Archived databases, however, are never changed. Nevertheless, information about database keys greatly helps interpretation and understanding of the database structure. So they should be preserved whenever they are available.

5.7 The SIARD format *may* contain meta data about all other SQL:1999 objects.

The SQL:1999 standard defines what a database is. For archival purposes it is very similar to SQL-92 but requires more strict compliance. The more recent standard SQL:2003 has added requirements for stored procedures (code), which are of lesser significance for archival purposes. It is imaginable, that new versions of the SIARD format will be needed when the SQL standard changes significantly.

5.8 The SIARD format *must not* contain any information specific to an institution.

When confronted with the SIARD format, many archivists confuse the SIARD format with the format of an archive-specific Archival Information Package (AIP) according to the OAIS[1] standard. Thus they would like to add the meta data referring to the tectonics of their institution to the SIARD file. But a SIARD file is much more similar to an MS Word file, which is *contained* in an AIP, together with additional context information of the archive. It should be possible to transmit and integrate the same database file in different archival institutions.

5.9 The SIARD format *should* enable integrity control of the primary data.

Whereas an archive may have a need to add explanatory meta data to a SIARD file, its primary data, which guarantee the completeness (see the Criterion of Complete Preservation above), must never be changed. A file format in itself cannot control its own integrity. But it can simplify integrity control by computing a digest (check sum) over the primary data, which can be stored in a catalog and compared to a computation of the digest on the primary data at a later time.

6 SIARD FILE STRUCTURE

Similar to many other modern file formats (MS Office Open XML, Open Document Format) a SIARD file is a ZIP file containing XML files and – possibly – some text and binary files representing large objects. Due to the fact, that databases often are larger than 4 GB, the 64-bit extension of the ZIP standard is used. The ZIP file is only used as a container and stores the content uncompressed – leaving the compression concern to lower-level hardware drivers. All XML files store their data as UTF-8 characters. Thus the SIARD file structure is based on these international standards:

- ZIP (64-bit Version > 4.5) for the container,
- SQL:1999 for the structure,
- XML for the data,
- UTF-8 for the characters.

The complete definition of the SIARD format can be found elsewhere[2]. Here we only give a brief overview as a basis for the following discussion.

6.1 Folders *header* and *content*

A SIARD file consists of two folders *header* and *content* on the highest level.

The *header* folder contains the file *metadata.xml*, which describes the complete database by listing all database objects in SQL:1999 compliant format that could be retrieved.

[1] http://en.wikipedia.org/wiki/Open_Archival_Information_System

[2] http://www.bar.admin.ch/dienstleistungen/00823/00825/index.html?lang=en

http://www.ech.ch/vechweb/page?p=dossier&documentNumber=eCH-0165&documentVersion=1.0

The *content* folder contains a folder for each database schema archived. All folder and file names in the SIARD file are short and restricted ASCII ('0'-'9','A'-'Z','a'-'z') for maximum compatibility with any operating system. So the schema folder names are usually *schema0*, *schema1*, *schema2*, … The map from the real schema name in the database to the folder name is established in the *metadata.xml*.

6.2 Schema Folders

Each schema folder contains a table folder (*table0*, *table1*, *table2*, ...) for each table in that database schema.

6.3 Table Folders

Each table folder contains an XML Schema Definition for the table data, an XML file representing the table data, and – possibly – folders (*lob0*, *lob1*, *lob2*, ...) for each large object column.

The XML Schema Definition maps the SQL types to XML types. The table XML file holds the actual primary data with a *row* tag for each column and a column tag (*c1*, *c2*, *c3*, …) for each column. The row and column tags have been chosen as short as possible. The size of a SIARD file should be dominated by the primary data content and not by the tags.

6.4 Large Object Folders

The large object folders contain a file for each large object of that column with a length larger than some minimum length. Every large object value is either “in-lined” in the table XML like any non-large value or referenced as a file in the corresponding LOB folder.

The reason for this somewhat arbitrary “in-lining” is the experience, that many LOBs are NULL or very short and that very many small LOB files in the LOB folder are difficult to handle on some operating systems.

6.5 Integrity of the Data, Formerly Called Authenticity

The SIARD format permits storing a message digest (check sum) over the contents of the *content* folder in the *metadata.xml*. By recalculating this digest and comparing it with the one stored in the meta data at the moment of creation of the SIARD file, one can detect if the primary data have been tampered with after the download. Metadata are not part of this integrity check because archives must have the possibility to enhance them any time.

Of course it is easy to change the primary data, recompute the message digest and store the changed message digest in the *metadata.xml*. The SIARD format does not pretend to establish a cryptographic barrier against fraud. However, if the check sum proves integrity at the moment of ingest, then it can be stored in a separate database (e.g. a database acting as the “Data Management” unit of the OAIS[1]) of the institution and can be used at the moment of access for proving the integrity of the primary data.

6.6 Validity

A SIARD file is valid, if it conforms to the standards mentioned above:

1. It must be a valid ZIP 64 file.
2. The XML files must conform to the respective XSDs and all be coded in UTF-8.
3. The *metadata.xml* and the *table<n>.xsd* files must describe a database structure that conforms to SQL:1999. Specifically all constraints (column types, database keys) mentioned in the meta data must be met by the data.

Unfortunately the last requirement is forgotten by some validation systems that focus too narrowly on XSD conformance.

The ZIP conformance can be checked individually by unzipping it with PKZIP[2] or Zip64File[3].

---

[1] http://en.wikipedia.org/wiki/Open_Archival_Information_System

[2] http://www.pkware.com/software/pkzip/

[3] http://sourceforge.net/projects/zip64file

The XSD compliance can be checked with any XML tool, utilizing the *metadata.xsd* of the SIARD format definition.

The SQL conformance can be checked by attempting to upload the SIARD file to a database system which is reasonably conformant to the SQL:1999 standard (e.g. SQL Server).

A very robust practical validation is achieved by opening the SIARD file using *SIARD Suite*, which checks ZIP conformance and XSD conformance of all XSDs, and test-uploading it to a dissemination database of the archive (e.g. a SQL Server instance) which checks conformance to SQL constraints.

6.7 The Problem of Binary Large Objects

Binary objects – whether they are large objects or not – present a special problem. They may contain anything (possibly code) and thus not be suitable for archival. If they are unimportant, storing some bits that cannot be understood on dissemination, may not hurt much. If the whole database consists of binary objects, it is probably an object-oriented database and therefore not suitable for long-term preservation, because code and data are inextricably mixed in proprietary formats.

Depending on the rules of acceptable archive formats of an institution, one may want to restrict Binary Large Objects (BLOBs) to formats (e.g. image file formats) accepted for archival by the institution.

The same problem must be faced, when archiving documents (PDF, DOCX, ODT, …) which may contain binary image files.

7 THE MAPPING OF SQL TO XML

The file format described thus far, makes the assumption, that every SQL type can be mapped to an XML type in a valid manner. As real database systems support a very wide – often non-standard – variety of database types and we should like to be able to archive any relational database that is presented to the archive, we must show that no information is lost in the transformation. We will discuss this for the most typical database types using the Criterion of Complete Preservation derived above.

When databases were standardized, limitation of maximum storage space of a cell was a major concern, because variable length data could not be handled easily and thus every cell used the maximum amount of storage allocated to the column. This led to the proliferation of database types. When XML was standardized these concerns had vanished. It was clear that data between tags were allowed to take any amount of space. Thus many different SQL types can be mapped to few XML types without loss of information.

7.1 Correctness of Character String Preservation

All character string types (except large objects) can be checked for equality with “=” in an SQL query. Cell content 'Hello' in a column of type CHAR(5) is equal to cell content 'Hello' in a column of CHARACTER VARYING(255). Therefore the database is preserved correctly even if types are not identical, as long as the values are identical. Thus the XML type *xs:string* can be used for storing any character-string database type.

7.2 Correctness of Preservation of Dates and Times

All date and time database types can be compared using SQL equality. '01-21-2013' as DATE is equal to '01-21-2013 00:00:00.000000' as TIMESTAMP(6). So these values are preserved correctly even if their types differ as long as their values are the same.

We use *xs:date*, *xs:time* and *xs:dateTime* for storing all dates and times in a database.

Although XML permits storing these values with a time zone and a day-light savings offset, *SIARD Suite* stores all times in UTC (Universal Time closely corresponding to Greenwich Mean Time GMT). At the moment of downloading *SIARD Suite* maps local time to universal time. On uploading it maps universal time to local time. It is possible to prevent these conversions using a command-line switch setting the time zone to UTC. Otherwise the local time zone is retrieved from the system in which the *SIARD Suite* runs.

7.3 Correctness of Integer Preservation

All integer types can be checked for equality using SQL. 4 as SMALLINT equals 4 as INT. Therefore we can use *xs:integer* to store any integer type.

7.4 Correctness of Preservation of Decimal Types

All decimal types can be checked for equality using SQL. 0.5 as DECIMAL(1) equals 0.5 as DECIMAL(5). Therefore we can use *xs:decimal* to store any decimal type.

7.5 Correctness of Preservation of Floating Point Types

Although comparing floating point numbers in numerical applications is dangerous because of rounding errors, it is permitted for database archival because no arithmetical operations are performed. The 4-byte and 8-byte floating point numbers are ISO standards which are the basis of the database as well as the XML types. Conversion of the binary ISO format to the decimal XML format may introduce a rounding error but preserves equality as well as weak ordering. All floating point types can be checked for equality in SQL. 0.3 as FLOAT(3) equals 0.3 as REAL(5). Therefore they can all be stored either as *xs:float* (4 byte ISO floating point) or *xs:double* (8 byte ISO floating point) with limited loss of precision due to rounding.

7.6 Correctness and Preservation of Identifiers

Real-life database systems make different rules concerning which identifiers are valid. Most of them support only a subset of the permissible identifiers in SQL:1999. Some database systems exclude identifiers that are longer than 14 characters. Most database systems have different sets of reserved keywords and forbid identifiers that are equal to one of them. Some database system have different scope rules than SQL and thus prohibit using the same name of an SQL object in a different schema.

SIARD Suite always stores the identifier name found in the original system in the SIARD file.

However, it might encounter difficulties on uploading a SIARD file to a different system due to the difficulties mentioned above. Therefore a disambiguation mechanism is used for creating valid identifiers in the target system. A forbidden column name VALUE might be replaced by VALUE0 in a system, where VALUE is a reserved keyword. All identifiers longer than 14 characters may be shortened to 14 characters in a system which limits identifier length. If the truncation process maps two different identifiers to the same value IDENTIFIER, then *SIARD Suite* will use IDENTIFIER1 for one of them and IDENTIFIER2 for the other when uploading the database.

In spite of this difference between the original and the restored database the Criterion of Complete Preservation is still fulfilled, because the values of the primary data do not depend on the names of tables or columns. All SELECT queries still yield the same result, as long as it is known, which identifiers in the SIARD file were replaced by which disambiguated names.

8 SQL OBJECTS OTHER THAN TABLES

In a SIARD file only cell values of so-called base tables are considered to be primary data. All other objects are only described in the meta data.

8.1 Keys

As already mentioned before, the primary keys, candidate keys and foreign keys of a database should be documented in the meta data. They only refer to schemas, tables and columns and thus only describe the primary data more closely. Although, strictly speaking, their absence will not change the outcome of any SELECT query, they are a great help for understanding the database structure. Also their presence permits more close control of the consistency of the database. Thus, downloading a live database where records are entered or deleted between the lengthy archival of two tables might introduce relational inconsistencies, which would be detected with the help of keys. (For this reason only snap shots of a database should be archived, that do not change during archival.) If it is desired that they should be excluded when *SIARD Suite* is used for archiving, one will have to drop all constraints before downloading the database.

Whenever possible, *SIARD Suite* attempts to enable the key constraints in databases uploaded onto a dissemination server. This may fail because some databases only permit primary keys as target for foreign key references, whereas others also allow candidate keys. Even if the constraints cannot be all enabled on upload, the database can be queried without data loss. Therefore in this case the upload is successful, although accompanied by some warnings.

8.2 Views

Views are SELECT queries, that are stored under a name. Unfortunately they may contain code (function calls). Real-life database systems also support many non-standard SQL constructs, which may appear here. Therefore it is improbable, that the same query can be executed on a different database system, than that upon which the database is uploaded.

The text of the query may nevertheless be of interest and help understanding the use of the database, as the SQL dialects at least resemble each other a lot. Therefore it is useful if views are documented in the meta data. Unfortunately many database systems do not enable access to the query text. *SIARD Suite* stores the query text if it can find it. Otherwise it at least stores the name and the column names and types of the query result and lets the user add the query text manually. On upload to a dissemination database, views stored in the SIARD file are not created because of the differences in SQL dialects, which make a general upload almost impossible. The user of the dissemination database will have to create them manually based on the information in the SIARD file if they are important.

8.3 Constraints and Triggers

Check constraints are similar to the WHERE condition of a SELECT query. They can contain code (function calls) and are often formulated in an SQL dialect which does not conform to the SQL standard. They are used rarely and some database systems even do not support them. As with all constraints their value is only informational. Archived databases do not change. Constraints serve to prevent inconsistent changes of a database. So no primary data are lost when they are not preserved.

In the SIARD Format a meta data object is defined for check constraints, which are objects specified in the SQL:1999 standard.

SIARD Suite displays the check constraint data, when they are present in the SIARD file, but does not create them on download or upload them.

A trigger is code that is executed when a certain database change occurs. Again this is immaterial for archived databases because no change ever occurs. Also the code is most likely not executable on the dissemination system. Therefore a trigger only serves an informational purpose.

In the SIARD Format a meta data object is defined for triggers, which are objects specified in the SQL:1999 standard.

SIARD Suite displays the trigger, when they are present in the SIARD file, but does not create them on download or upload them.

8.4 Routines

Routine is the word used in the SQL:1999 standard for stored procedures. These are all code and thus certainly cannot be objects of long-term preservation. Also in real-life databases they come in all kinds of programming languages (PL/SQL, JAVA, Visual Basic).

SIARD Suite usually stores at least the signature (routine name and parameter types and names). For some database systems, the body of the routine can be stored too. In all other cases, it is possible to enhance the routine meta data by copying the body manually.

Routines are a very minor concern in the context of long-term database preservation. They are not uploaded to a dissemination database by *SIARD Suite*.

8.5 Permissions

SQL:1999 objects like users, roles, privileges also carry some information about how the database was used. They can be documented in the SIARD file. They constrain access and

changes to parts of a database for some users. They must never be enabled on upload to a dissemination system, because an archive almost certainly will need to apply a different kind of access control to its data.

8.6 Applications

Database applications are not part of the SQL:1999 standard. However, almost all databases are accessed by their users through the user interface of an application. For example MS Access files contain Forms, Reports and Visual Basic code, which present database table data to the user and extract some specialized reports from them. As mentioned in the chapter Code and Data, they are not suited for long-term preservation.

What *can* be preserved, however, are some sample screen shots (image files) and sample reports (document files), that document, how people accessed the table data. It is very useful, to store such documents together with a manual and a data dictionary along with the database in an Archive Information Package to help future researchers understand the meaning of the data.

9 SIARD FORMAT AND *SIARD SUITE*

For an archive, the all-important question is the choice of the normalized archive format. As soon as documents in this format accumulate in the archive, it becomes very difficult for the archive to extract itself from that commitment. In the case of long-term preservation of databases we consider the choice of the SIARD format reasonable. No other archival format of databases is an open, durable standard, which has been developed with the normalization strategy as a basis. It is to be hoped that many archives adopt this format and thus join the community in which databases in this format can be exchanged.

The archives must commit to a common format because they are a niche market. No single archive can afford to define and maintain their own standard. Only through wide adoption of a common format is the exchangeability of data between archives guaranteed

If all archives start defining their own derivation of the SIARD format – with different sets of meta data for example – then again database documents cannot be exchanged between archives. The users of the SIARD format must agree to stick to the relatively few versions developed by the standard committee (currently: Swiss Federal Archives and the Swiss E-Government Standards association eCH[1], but a more international standardization process could and should be adopted). Requests for features of future standard can be submitted to the committee and must be approved by it.

In other types of business, the vendor lock-in is usually through the software. In the case of archives with a normalization strategy, the software is secondary. *SIARD Suite* is a “reference implementation” which is used in real-life archiving solutions. This software for normalizing and denormalizing is very versatile, supports many database systems and does not depend on the operating system. However, for special purposes (e.g. direct conversion of some dump format to a SIARD file with high performance) other programs can make use of the format. The only condition is that they conform to the format.

Whereas SIARD files are never migrated in the archive, *SIARD Suite* as a program performing the normalization and denormalization of real-life database systems must be maintained and upgraded with the arrival of new versions of database systems and operating environments.

9.1 Guiding Principle for *SIARD Suite*: All Databases can be Archived

The goal of an archive is to document real government activity, not to censor the data it receives. Of course, it is desirable that government agencies proceed in an orderly fashion. But if politics fail to enforce it, the archive is usually not in a position to remedy that at the time when the documents are to be archived.

So on the one hand, an archive must attempt to limit the data formats it accepts as narrowly as possible. This goal is well served by normalization. On the other hand, it must be able to include all data – or at least the large majority – that are handed to it. This goal can only be achieved, if the normalization process attempts to archive any document of the given type.

[1] http://www.ech.ch/vechweb/page

Therefore *SIARD Suite* attempts to normalize any database into the SIARD format, if the data are available in one of the database types it supports[1]. This may result in some migration transformation for databases that do not conform to the SQL standard very well. E.g. MySQL databases on LINUX will have their identifiers stored in lower-case and thus queries, that worked in the original database, may fail, if they do not quote the identifiers. MS Access databases store the query of a view in a very rudimentary format, which is combined into something like an SQL query by *SIARD Suite.* ...

10 HOW TO ACHIEVE PRESERVATION

Long-term preservation has succeeded when the data can be disseminated to users in 50-100 years *and be understood by them*. The SIARD format makes sure that the data in the cells can be understood technically. It is of no help, however, when their meaning must be understood. Imagine a historian studying a database of a large bank in 50 years from now. If she does not realize, that a certain innocent-looking column called *I* with values 0 or 1 means “invalid transaction” or “deleted entry”, then she will be confronted with a very misleading picture of the current financial crisis.

So the agencies downloading databases for archival *must* accompany the data with as much documentation of their *meaning* as possible. That is the reason why every meta data object in the SIARD format has a field called description, where information about its meaning can and should be entered. *SIARD Suite* downloads database table and column comments from the original database and stores them in this field whenever possible.

Large databases often have several hundreds of tables with hundreds of columns each. So documenting the meaning of each column requires a sizeable effort. Fortunately many serious developers of database applications have documented the meaning of each field in a so-called data dictionary. Often this documentation can even be uploaded as table or column comments to the database before archival.

SIARD Suite helps handling this explanatory effort by permitting the merger of the field description of a previous download with a new download. So, if a database only changed a little, most of the explanation needs not be repeated.

SIARD Suite does not force the user to add a description to every column of every table before archival is possible. For many small databases (MS Access) the effort is not worth it. Also it makes no sense whatsoever to insist on a comment that the CITY table is a table listing cities and the COUNTRY table holds country names. Finally, the database may be described very well in accompanying manual and data dictionaries; it makes more sense, to refer to them, rather than enter text in 10'000 description fields.

When the download of a database using SIARD Suite has succeeded and the data are stored in SIARD format, the technical part of archival is over. The archivists work, however, has just started. It consists of ensuring the preservation of meaning.

Zürich, den 15. January 2013 Dr. sc. math. Hartwig Thomas

[1] at the time of writing: MS Access, Oracle, SQL Server, and MySQL