ABSTRACT

The role of SAS/SHARE® software in the SAS® System is being expanded to include major database features useful in both single- and multi-user applications. Such features include integrity constraints (the specification of data validation criteria for all data inserted/added to a data set) and an audit trail (the ability to keep track of which users updated what observations and when they did it). The audit trail can optionally be used to reconstruct a data set from an archive copy.

CURRENT ROLE OF SAS/SHARE SOFTWARE

SAS/SHARE software permits multiple users to concurrently access data libraries, SAS files, and observations in SAS data sets for input and update operations. Data can be shared at the member or record (observation) level. Such sharing is generally transparent to the users because SAS/SHARE software manages the access conflicts.

Expanded Role

SAS/SHARE software in Version 5 of the SAS System was planned to be the product for advanced database features. SAS/SHARE was not part of the initial release (Release 6.06) of Version 6 of SAS software, so two new database features, indexing and data set compression, were included in the base product. In Release 6.07 SAS/SHARE software is again available on MVS and CMS, and it is planned for production release on all major platforms in the future. Once again it will become the home for advanced database functions provided by the SAS System. Note that indexing and compression will remain base product features.

When data are shared by many users, maintaining data integrity can become a difficult problem. And it may be hard to keep track of which user modified a particular set of observations. Each application that updates a file may have to be coded to validate data before they are added to the data set.

New functionality will be added to SAS/SHARE software to address these and other problems related to the sharing of data. In its expanded role, SAS/SHARE software will continue to provide shared access to data. In addition, it will use advanced database features to solve problems that arise when data are shared by multiple users.

Although the database features described in this paper will be part of SAS/SHARE software they are not required to be used through a SAS/SHARE server. Applications that do not share data (except in a serial manner) can utilize these new features as well.

Examples of Good Things to Come

The functionality described here is expected to be introduced in a future release of the SAS System. Since this work is still under development, some of the details may change before it becomes available to customers. The descriptions here are not intended to be complete. The goal is to give you an overview of what is planned for the future. Note that this functionality is only available for SAS data sets that are accessed via the default or base engine.

The remainder of this paper focuses on two features that will enhance the integrity of applications that share data: integrity constraints and audit trail. The sharing may be concurrent or serial. In either case it is often important to monitor which users are changing what data and to ensure that the modified data fit within guidelines established by the owner of the data.

INTEGRITY CONSTRAINTS

Developers of SAS applications often add code to validate data values before they are sent to the SAS System for entry into the data set. If several such applications are using the same data set, then this validation code must be replicated and maintained in each application program. Some of these validation requirements can be satisfied by the use of informats. Other validation criteria such as uniqueness and variable-to-variable relationships cannot. A new feature, integrity constraints, offers a powerful tool to make data validation easier.

Integrity constraints are used to preserve the consistency and validity of data that are stored in a SAS data set via the default engine. Data
validation rules called integrity constraints are specified for one or more variables in the data set and are permanently stored in the data set. After the integrity constraints are accepted by the SAS System, each add or update operation to the data set, regardless of whether it comes from a DATA step, an SCL program, or a procedure, is checked. Any modification that does not conform to the integrity constraints will be rejected.

For example, you can limit the valid values for a variable named SEX to 'male' or 'female', or a variable for an employee’s identification number could be limited to contain only unique values between 1000 and 9999.

A given integrity constraint may involve more than one variable. And a given variable may have multiple integrity constraints defined for it. Integrity constraints are named and may be dropped, listed, or replaced.

The industry standard language, Structured Query Language (SQL), supported by the SQL procedure, is used to define integrity constraints. The integrity constraints may be specified prior to adding any data to the data set (or table) so that the data are always in compliance with the constraints. Integrity constraints may also be added to a data set with existing data. Before accepting the new integrity constraint, the SAS System validates the existing data. If any nonconforming data values are found, the integrity constraint is not accepted. The SQL procedure will be able to list the integrity constraints on a data set.

Candidate Integrity Constraint Specifications

A variety of integrity constraint specifications are being considered for implementation in future releases of the SAS System. In what follows, a key is a single variable or a composite variable (multiple variables concatenated together).

- **unique**: requires distinct values for the variable.
- **not null**: prohibits missing values for the variable.
- **primary key**: uniquely identifies a given observation. A primary key is not permitted to have null (missing) values. Only one primary key per data set is allowed.

Establishing a primary key for a data set ensures that every observation is distinct.

An example of a primary key would be a variable in an employee data set for a person’s employee number.

- **foreign key**: may contain non-unique and null values. The non-null values must match a value in a primary key of some data set, usually a different data set. Multiple foreign keys are permitted for a given data set. Foreign keys are used to enforce referential integrity.

As an example, take an employee data set (EMPLOYEE) with employee number (EMPNUM) as a primary key and a projects data set (PROJECTS) with employee number (EMPNUM) as a foreign key. Referential integrity requires each non-null value in PROJECTS.EMPNUM be a value from EMPLOYEE.EMPNUM.

- a WHERE clause, including variable-to-variable comparisons

**SPECIFYING INTEGRITY CONSTRAINTS FOR AN OUTPUT DATA SET**

The generic SQL procedure syntax for creating a table (data set) is

```sql
PROC SQL;
CREATE TABLE <table-name> (<variable-list>);
```

Integrity constraint specifications are optional and follow the `<variable-list>` in the CREATE TABLE statement:

```sql
CREATE TABLE <table-name> (<variable-list> <ic-list>);
```

Each item in `<ic-list>` has the following form:

```sql
CONSTRAINT <ic-name> <type> (<ic-specification>)
```

where `<ic-name>` is the user-supplied constraint name, `<type>` is CHECK, PRIMARY KEY, FOREIGN KEY, UNIQUE, or NOT NULL and `<ic-specification>` is a variable list or a WHERE clause.
Example Output Data Set with Integrity Constraints

In this example, the integrity constraints are specified along with the definition of the variables prior to adding data values to the data set.

```sql
proc sql;
create table employee
(
    empno num,
    dept num,
    sex char(6),
    constraint c_sex check ( sex in ('male' 'female')),
    constraint empno primary key (empno),
    constraint dept2 check (dept <= 500 and dept >= 100)
);
```

SPECIFYING INTEGRITY CONSTRAINTS FOR AN UPDATE DATA SET

Integrity constraints can be added to an existing data set, or existing integrity constraints can be deleted. When a new integrity constraint is added, the existing data values are verified to conform to the new constraint.

Example Update Data Set with Integrity Constraints

Using the above example for output data sets, this example shows how to add the same constraints to an existing data set.

```sql
proc sql;
alter table employee
    add c_sex check (sex in ('male' 'female'))
    add empno primary key (empno)
    add dept2 check (dept <= 500 and dept >= 100);
```

Later, if the department numbers are expanded to include 600 and 700, the DEPT2 integrity constraint can be changed. No 'replace integrity constraint' function exists, so replacement is achieved by dropping the old version of the constraint and adding the new version.

```sql
proc sql;
alter table employee
    drop check dept2
    add dept2 check (dept <= 700 and dept >= 100);
```

Implementation Issues

Unique and primary key integrity constraints must ensure that duplicate values are not entered into the data set for the specified variables. Determining if a value is unique can be very expensive without some auxiliary data structures to aid the search.

Therefore, an index is created on the variable(s) by the SAS System when a unique or primary key integrity constraint is specified. These indexes are exactly like the user-defined indexes created by the CREATE INDEX command of the DATASETS procedure and the INDEX= DATA step option. This means that the same index can be used for WHERE clause optimization, BY statement processing, and integrity constraint enforcement.

It is important to minimize the number of indexes for a data set. If an index exists for a variable, then it will be used for a primary key or unique constraint if it has the UNIQUE attribute. Otherwise the constraint will be rejected. So, applications that utilize both an index and a unique or a primary key integrity constraint on the same variable must be careful to define a UNIQUE index (or let the integrity constraint do it). Since the NOMISS index attribute does not prevent missing values from being stored in the data set, the integrity constraint does not require this attribute. The integrity constraint code is responsible for enforcing the no missing requirement.

To obtain a list of the indexes on a SAS data set, including the index attributes, use the CONTENTS procedure.

AUDIT TRAIL

In an environment where data are updated by multiple applications and/or multiple users, it is often important to maintain historical information about the various modifications. A primary use of this history is to determine who made the modifications and when they were done. A second use is to recover the updates if the active data set to which they were made is lost. Currently the responsibility for collecting this historical information and solving these two problems falls upon the user's application.

A new procedure, AUDIT, can be used to place the burden of recording such information upon the SAS System. The AUDIT procedure will manage creating, deleting, and accessing a log file called
the audit trail. The audit trail will contain data items relating to all data set modifications such as who performed the modifications and when they occurred. The AUDIT procedure will also make the audit trail data available in the form of a SAS data set for browsing and analysis via standard SAS procedures. It is important to understand that 'modifications' does not include any operation that replaces the data set. Data set replacement will be discussed shortly.

An audit trail is a separate file, like an index file, associated with a data set. It has the same name as its parent data set but has a different member type. A maximum of one audit trail per data set is permitted, and a given audit trail contains data for only one data set. The audit trail is not actually a SAS data set, but it can logically be viewed as one. It has all the variables of its parent data set, plus some user-visible variables for keeping track of the modification information and some system-defined variables.

Requesting the AUDIT procedure to create an audit trail is called arming. Once the audit trail is armed, the SAS System automatically writes a record to the audit trail for every modification made to its parent data set.

The audit trail has two logging modes. By default, a history of which users modified what data is maintained. Optionally, the audit trail may contain data to permit recovery of a lost data set. The default mode is called RECOVERY = NO, and the recovery mode is called RECOVERY = YES.

**Recovery=No Mode**

In this default mode, the audit trail receives a record for each modification to the data set. Each record contains user-visible variables to record information about the update. These variables are

- **_AUDOPC_** opcode for type of update
  - A add an observation
  - D delete an observation
  - R read - before update data record image
  - W write - after update data record image
  - I create an index
  - X drop an index
  - N update a variable's name or label

- **_AUDRD_** unique identifier of updated observation
- **_AUDTT_** date/time of update
- **_AUDPRC_** name of procedure requesting update
- **_AUDUSR_** user/session id requesting update
- **_AUDVAR_** to-be-determined user-assigned value

The AUDIT procedure is used to extract data from the audit trail into a SAS data set. The new data set will have all the variables from the parent data set plus the user-visible variables described above. An optional where clause, on any of the variables in the new data set, can be used to subset the data during this extraction operation.

In this mode, two other options are valid.

**SUSPEND** temporarily terminates the logging of records to the audit trail. The audit trail remains associated with the parent data set.

**RESUME** restarts record logging activities. The new records are added after the last record written prior to the SUSPEND operation.

**Recovery=Yes Mode**

This optional mode is designed to support reconstructing the current data set if the current data set is lost due to a catastrophic failure such as a disk crash. It can also be used to remove a series of undesirable updates. As described above for RECOVERY = NO, information sufficient to recreate each update operation is written to the audit trail. The audit trail information can be applied via the AUDIT procedure to a previously saved copy of the data set, restoring it to the point just before the failure. This recovery process is called rollforward.

All the features described above in RECOVERY = NO mode, except for the SUSPEND and RESUME options, are also available in RECOVERY = YES mode.

Here is the general rollforward scenario:

- The AUDIT trail is armed by using the AUDIT procedure. Proc AUDIT creates the AUDIT trail and archives the data set. Optionally, the user can archive the data set.
• Updates are made to the data set by one or more applications.
• The disk containing the data set crashes.
• Proc AUDIT, or the user, restores the archive copy of the data set.
• Proc AUDIT applies (rollforward) the requested updates from the audit trail to the archived data set, recreating the lost data set.

During the apply operation, the ROLLFORWARD option allows a specification of a date/time value to limit the updates to be applied to the archived data set. The apply operation begins with the first update on the audit trail and continues through the last update by default or through the last update specified via the date/time value.

Note that rollforward implies that the audit trail is on a different device from the data set in order for it to survive a disk crash. Device separation is not required but strongly recommended if protection from device failure is desired.

Data Set Replacement with an Audit Trail

When a data set is replaced, the logging of updates to the audit trail is terminated for the old (replaced) data set. An empty audit trail is automatically created for the new (output) data set. The optional ARCH NAME related-member-list option, like the AGE option in the DATASETS procedure, controls the name and number of audit trail generations. The name of the newest generation is always the first name in the related-member-list, the name of the next generation is the second name in the list, and so forth. As the number of generations exceeds the number of names in the list, the oldest file is deleted and the last name is reused.

If the ARCHNAME option has been specified, then the audit trail is automatically archived (renamed) by the SAS System to the next generation; if RECOVERY=YES, the output data set is automatically archived (copied) by the SAS System to the next corresponding data set generation.

The current generation consists of three files: the current data set with updates, its audit trail, and the archived copy of the current data set immediately following the replace operation. The audit trail's updates applied to the archived data set will re-create the current data set. Subsequent generations consist of two files: an audit trail and the data set to which the updates are to be applied. So, each replacement operation will create two additional files until it has reached the maximum number of generations as specified by the ARCHNAME option.

Here is an example:

```
step: create data set BONDS
file: BONDS.DATA (generation 1)

step: arm audit trail with RECOVERY=YES and
      ARCHNAME=BND01-BND02
      - create BONDS.AUDIT
      - copy BONDS.DATA to BND01.DATA
files: BONDS.DATA (unchanged, generation 1)
      BONDS.AUDIT (empty)
      BND01.DATA (archive of BONDS.DATA, generation 1)

step: perform some updates
files: BONDS.DATA (updated)
      BONDS.AUDIT (updates to make
                   BONDS.DATA from BND01.DATA)
      BND01.DATA (unchanged)

step: first replace (create generation 2)
      - replace BONDS.DATA
      - rename BND01.DATA to BND02.DATA
      - copy BONDS.DATA to BND01.DATA
      - rename BONDS.AUDIT to BND02.AUDIT
      - create BONDS.AUDIT
files: BONDS.DATA (newly replaced, generation 2)
      BONDS.AUDIT (empty)
      BND01.DATA (archive of BONDS.DATA, generation 2)
      BND02.AUDIT (updates to make pre-
                   replacement BND01.DATA from
                   BND02.DATA)
      BND02.DATA (archive of BONDS.DATA, generation 1)

step: perform some updates
files: BONDS.DATA (updated, generation 2)
      BONDS.AUDIT (updates to make BONDS.DATA
                   from BND01.DATA)
      BND01.DATA (unchanged)
      BND02.AUDIT (unchanged)
      BND02.DATA (unchanged)

step: second replace (create generation 3)
      - replace BONDS.DATA
      - delete BND02.AUDIT
      - delete BND02.DATA
```
rename BND01.DATA to BND02.DATA
- copy BONDS.DATA to BND01.DATA
- rename BONDS.AUDIT to BND02.AUDIT
- create BONDS.AUDIT

files: BONDS.DATA (newly replaced, generation 3)
       BONDS.AUDIT (empty)
       BND01.DATA (archive of BONDS.DATA, generation 3)
       BND02.AUDIT (updates to make pre-replacement BND01.DATA from BND02.DATA)
       BND02.DATA (archive of BONDS.DATA, generation 2)

Note that the second replace operation creates the third generation of the data set. Since the ARCHNAME option specifies that only two generations are to be archived, the oldest data set and its audit trail are deleted, thus limiting the amount of disk space but destroying historical information.

It is important to understand that modifications made during the replacement operation are not recorded on any audit trail. This means that a rollforward operation will not apply updates across replacement boundaries. In order to reconstruct a data set from an archived data set that is more than one generation old, you must also have the programs that perform the replacement operations.

If the ARCHNAME option is not specified, the existing audit trail data will be lost, and a new audit trail will be created for the output data set. This means even if RECOVERY=YES the existing audit trail will be lost. In this case there will be only one pair of files after the replace step: the output data set and its empty audit trail.

**Copying a Data Set with an Audit Trail**

Copying a data set with an audit trail will have no impact on the "from" data set. The "to" data set will have no audit trail associated with it. The AUDIT procedure can be used to arm an audit trail on the new copy.

**CONCLUSION**

Many applications that share data must include provisions to validate data inserted into the database. They may also need to keep track of the various users making the updates. In future releases, the SAS System will provide features to handle these responsibilities for your applications. The two enhancements discussed here, integrity constraints and audit trail, will be a part of SAS/SHARE software, and are suitable for both concurrent-shared data access and serial-shared data access.

SAS and SAS/SHARE are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. © indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.