Fermilab MTF

Doc Number: MTF-92-0003
Version: 1.3

OVERVIEW OF A DATABASE SYSTEM FOR MAGNET MEASUREMENTS

Bruce C. Brown, Henry D. Glass, Richard Glosson,
Raymond W. Hanft, David J. Harding, James Sim,
Kelley Trombly-Freytag, and Dana G. C. Walbridge
Fermi National Accelerator Laboratory *
P.O. Box 500
Batavia, Illinois 60510

7/8/92

*Operated by the Universities Research Association under contract with the U. S. Department of Energy

Contents

Abstract

The Sybase Database Management System is employed at the Fermilab Magnet Test Facility to store information about the magnet measurement system, measurement subjects, and data collected on measurement subjects. Prescription of measurements, description of hardware and recording of measurement results are encompassed in a common overall schema. The database schema for this is comprised of specialized schema for different magnet measurements plus a set of general schema which services all the measurement hardware systems. This document will describe the general system and will reference documents which define the schema in detail. Documentation and implementation practices for the MTF databases will be described.

1 Introduction

In order to utilize a modern database system at the Magnet Test Facility, a development methodology has been selected and documented, tools created and employed, and an overall data storage schema constructed. This document will introduce the MTF data management system by identifying the database users, identifying special features of SYBASE which are important to this design, describing the overall schema for data storage, describing documentation and implementation practices at MTF, and discussing the tools and utilities which MTF users will utilize. To provide a robust and well documented environment for creating and using databases, certain practices and methods have been adopted. These are discussed below or references are made to other more detailed documents. Some design details are included but most are left to the documentation which is prescribed for each section of the database schema and references to those documents are provided.

1.1 Intended Audience

This document provides an introduction to the MTF database environment and the plan for data storage. All users of this system will find information of interest in this document, but it is written for users included in the user roles of MEASUREMENT-SUPERVISOR, TEST-MANAGER, DEVELOPER, DBA, AND DATA-USER as defined in Section 1.2.

1.2 MTF Database User Roles

The MTF database system is intended for a diverse group of users. User roles specify which background knowledge and use requirements various persons will bring as users. Applications, documentation and training must match these user needs and capabilities. Identified user roles are:

MEASURERS Technicians, Engineers or Physicists who operate the measurement system.

MEASUREMENT-SUPERVISOR Senior measurement person on shift.

TEST-MANAGER Person specifying a measurement requirement. Typically this will be an engineer or physicist who is expert in either a given measurement or a given magnet project.

EQPT-TECH An electronic or mechanical technician who may repair or calibrate equipment used in the measurement system.

DEVELOPER Physicist, Programmer or Engineer who develops a given measurement system.

DBA Database Administrator who manages SYBASE software including installation, backup and validity testing. This person will log in to SYBASE as SA (System Administrator).

DATA-CLERK A worker (technician, clerk, physicist...) who uses data analysis and reporting tools on measurement data to produce archival database data and reports for external use.

DATA-USER An Engineer, Accelerator Physicist or Magnet Physicist who has use for data collected by this system.

The actual role in which a given user acts as he carries out his tasks will determine the database permissions which he will need. Methods for setting these permissions are discussed in Appendix B.

1.3 Relational Database Concepts

Relational databases provide tools to store, manage, manipulate, and access data. The data must be organized into sets of data called tables in which attributes (columns) are stored for instances (rows). Some of the concepts which will be used to describe the MTF system include, in addition to table, the following terms.

COLUMN (ATTRIBUTE) Some type of information which describes the entity to which a table applies. It can be expressed with a specified data type and takes values from a domain of valid values.

ROW (INSTANCE) The values (including possibly NULLS) which describe an example of the entity described by the table.

KEY A column or set of columns used to identify a row.

PRIMARY KEY A selected set of one or more columns which uniquely identifies a row in a table. Each table must have a primary key.

ALTERNATE KEY Another set of one or more columns which (also) uniquely identifies a row in a table. Alternate Keys do not have to exist in all tables.

FOREIGN KEY A column or set of columns in a table whose values are required to match a primary key in some other table.

DATATYPE Defines the data storage method suitable for the domain of values for a column.

NULL A special value which may be permitted in a domain which specifies that that row has an "unknown" or "not applicable" value for this column. This is *not* equivalent to zero or blank.

DEFAULT A value which will be assumed by a column if no other is specified.

RULE A specification which partially or fully limits values of a column to its valid domain.

JOIN A data selection operation in which data rows from two or more tables are considered jointly. Usually, these are restricted to rows which share common values of some columns.

INDEX Additional storage of a key using a method which facilitates retrieval.

SQL Structured Query Language (pronounced sequel) is an ANSI standard [1] data definition and retrieval language.

TransactSQL The Sybase SQL dialect which includes some procedural constructs.

Of these concepts, SYBASE recognizes and supports all except for alternate keys. Common keys are supported by SYBASE but are not provided in the ANSI SQL standard.

Relational data organization has been the subject of intense examination since about 1970. A broad range of reference material is available on relational database ranging from mathematical proofs and ANSI standards documents to textbooks and tutorial introductions. Among all of these, a few with which we are familiar include a "A Visual Introduction to SQL"[2] and "Introduction to SQL" by Oracle. Introductory reference material in the manuals available from Sybase includes Chapter 1 of the "Transact SQL User's Guide"[3] and pages 1-4 and 1-5 plus Chapter 6 of the "Data Workbench User's Guide"[4]. The book "SYBASE and SQL Server"[5] is available at MTF. The Fermilab Library also has a number of books on SQL and relational databases. One of note is the tome by C. J. Date[6]. Selective reading of this book is rewarding.

1.4 MTF Database Schema

A database schema must declare the tables in which data is to be stored and specify the columns (attributes) which are to be described in that table, the relations between columns in that table and other tables in the schema, and permissions, indices, keys and related details of the table specification. Here an introduction of the divisions within the MTF database design is provided. In Section 3 one can find more discussion. Detailed descriptions of the database design are provided in separate design documents referenced in Section 5.1.

The MTF schema can be divided into three groups of tables. One group describes the test subjects. These are usually magnets[1]. A second general category includes the testing facilities and measurement control and recording systems. This includes test stand and measurement systems, administrative items such as personnel, description of the software and hardware components, configuration and calibration information and specification of measurement requests. The third group of schema contain tables which describe results from measurements.

For reasons described below, we employ a single SYBASE database for storage of production measurement data. We divide the schema into sepa-

[1] Other test subjects may include other accelerator devices, materials such as steel samples, probes or power supplies which we calibrate or other materials for which tests have been prescribed.

rate virtual databases (see below) with the following purposes:

MEASUREMENT SUBJECTS

subjects descriptions of and relations between subjects, series of subjects, and their component parts.

MEASUREMENT SYSTEMS AND TECHNIQUES

admin database and MTF administrative information

facilities information about measurement hardware and software. This includes identification, location, and service history.

logbook measurement instructions (checklists), measurement activity logs and a comments and assessments database

instruments definitions of instruments (combinations of hardware and software) used in measurements

calibrations methods and parameters used to convert raw data to reduced or analyzed data.

ramps prescription of how magnet current is to be varied and the variation evaluated by the measurement system, and which instruments are to be used to control and to measure magnet current.

tot Table of Tables database for database schema description

MEASUREMENT RESULTS

currents magnet current control history and results logging

points results for pointscan measurements with NMR & Hall Probes

flatcoil results from measurements with flatcoil probes

harmonics results from measurements with rotating coil harmonics probes

tevatron information which relates specifically to the measurements taken on Tevatron components with systems available in the 1979-199x period.

lobeta89 measurements which are specific to the project for providing magnets for the Fermilab Tevatron Low Beta upgrade of 1988-199x.

vaxdata results from the VAX-CAMAC Conventional Magnet measurement system 1984-199x.

2 Some MTF Use of SYBASE Concepts

Although the general features which have been designed into this database system can be accomplished in any modern relational database system, we have designed our system to take advantage of crucial features of the SYBASE database system in a manner which minimizes our manpower investment. In this section we will describe these crucial features of SYBASE, especially as they relate to design features of our database system and some features we have developed using these SYBASE characteristics.

SYBASE is a relational database system marketed by Sybase Inc. of Emeryville, CA. SYBASE supports most, though not quite all, features of the SQL ANSI Standard but has extended the standard in several significant ways. Basic use of SYBASE and definition of tables and keys follows the SQL standard. Using SYBASE's 'isql' tool, one can execute scripts to create databases, modify them or query them for data.

2.1 Features of SYBASE

A critical extension by SYBASE[2] to the SQL standard is in the area of data integrity. Rather than waiting for the acceptance of a standard-based solution to data integrity, SYBASE has provided data integrity support through a system of 'triggers'. Triggers are built on a SYBASE concept of stored procedures. Stored procedures are Transact SQL statements which may include control flow statements, and which have been stored in the database system itself. A trigger is a stored procedure which is executed upon insertion, modification or deletion of a row in a table. By writing trigger code one is able to guarantee relationships among database entities based on information stored within the database rather than information stored within applications. In our design, we depend heavily on triggers. For example, before we can utilize a serial number corresponding to a person, a trigger verifies that that serial number has been stored in the database as a valid personnel serial number.

In a relational database table, a primary key (and also an alternate key, if one is specified) is a set of one or more attributes (columns) which uniquely identifies an instance (row) in that table. Some features of SYBASE keys are acknowledged in our design of the Table of Tables application[3], although

[2] This document is based on work with SYBASE Server Ver 4.0.1

[3] A separate document (TM-1707) [7] describes 'tot', an application for recording database schema. This tool has been developed to facilitate creation and documenta-

they appear to have only a little functionality in the present SYBASE implementation. These are features which make entries in the syskeys table. The primary key specification (sp_primarykey) makes an entry in syskeys indicating the existence of the key. The foreign key specification (sp_foreignkey) makes an entry specifying the foreign key and also makes an entry declaring a commonkey. This common key allows SYBASE to provide default JOIN information which facilitates queries but is not fundamental to our uses[4]. The 'VQL' portion of 'dwb' does take advantage of the commonkey features in presenting default JOINs for specified table combinations. The implementation of the critical features of foreign keys in our system is done using triggers generated by our 'tot' application.

2.2 "dataowners"

The general problem of making data available in understandable data sets is facilitated by using meaningful hierarchical structures. The overall hierarchical scheme available for grouping SYBASE tables consists of the server, database, and owner. Its use is limited by the inability to easily share datatype definitions and to make joins and triggers across databases. In addition, some SYBASE administration is easier with a single database. A self-consistent back-up is only guaranteed at the database level. We have therefore chosen to store all the data for this project in a single SYBASE database. We can still divide the data tables into meaningful groups using the data owners for various data tables. Each SYBASE table is owned by a registered SYBASE user and the dataowner is a natural part of the query in SQL.

We establish "dataowners" as special SYBASE "users" (accounts) who own a group of related tables. We use these "dataowners" as *virtual* databases. The SYBASE account (user) name is the same as the *virtual* database name. A single person may have responsibility for one or more dataowner accounts. (At present our database administrator is responsible for all of these dataowners but it is easy to change that with this design.)

Organizational principles for designing this hierarchy are evolving. With

---

tion of our database system. The reader is referred to that document for additional explanation of our notation.

[4]We believe that SYBASE may add functionality which provides additional validity checking based on these items so we choose to make these entries now using 'tot'. For now, they provide mostly a documentation function although applications can be written which take advantage of them if one should choose.

exactly two levels of organization (dataowners and tables) we will want to balance the number of dataowners *vs.* the number of tables per dataowner to make any given table as easy to find as possible. We currently organize our dataowners so that the tables design can be illustrated in one or two pages of pictures (we created them in a standard format using the idraw program). We can group our existing virtual databases (dataowners) into the groups described above as follows:

subjects: subjects

systems: admin, calibrations, facilities, instruments, logbook, ramps, tot

results: currents, points, flatcoil, harmonics, tevatron, vaxdata

2.3 MTF data types

In addition to standard data types, SYBASE supports user defined datatypes in which one is permitted to specify special cases of the standard data types for which one can then provide meaningful names. MTF variable-length character data types have lengths which match certain templates. This reduces the choices one considers during design and helps clarify the uses of various data. A user defined datatype, sntype, is utilized as the data type for serial numbers. At present, MTF serials are implemented using an integer. We can, however, replace the integer serial number with an alternate one, *e.g.* one which uses a letter followed by an integer string to represent the serial number. By changing the serial number generator code and recreating the database from its definition, one could replace the serial number system. Table 1 shows the datatypes defined in the datatypes table for 'tot' [7]. The most common SYBASE standard types are included to facilitate their use in tables which are to be described in 'tot' (where validity checking demands that specified data types are entered in the datatypes table). The datetimestamp type is special in that it has a default to the current datetime.

2.4 Valid Tables

In designing some of our more complex data schema, a new application of the SYBASE stored procedure system has occured to us. In some cases which we identified, it was not possible to specify relational integrity triggers within each table to guarantee validity. For some of these cases, reasonable

Table 1: MTF SYBASE Datatypes

MTF DATATYPE	Underlying SYBASE DATATYPE	PURPOSE OR USE
User Defined Datatypes		
sntype	int	serial number for primary key
aliasstr	varchar(12)	brief name of item
namestr	varchar(30)	normal length name of item
titlestr	varchar(80)	full title of item
descstr	varchar(255)	maximum length description
datetimestamp	datetime	default to current datetime
SYBASE datatypes also currently in 'tot'		
bit	bit	single bit logical variable
datetime	datetime	date with 1/300 second least count
float	float	IEEE double precision real number
int	int	4 byte signed integer
smallint	smallint	2 byte signed integer
tinyint	tinyint	1 byte unsigned integer (0-255)
char	char	fixed length character string
varchar	varchar	variable length character string

triggers would have been too restrictive in the required order of data entry. For other cases, the relations were such that no possible data entry order would have allowed all known validity relationships to be checked. However, it was observed that a new table (a valid table) could be created whose only contents were the serial number of the object whose validity is to be checked and a date (valid_dt). The object to be validated will have its own serial number and usually will have information in one or more related tables which must have several valid relations among them.

When an application wishes to use an item whose validity is checked in this way, it will find the serial for that item and then examine the valid table for an entry with that serial number. If it fails to find it, it will execute a validity stored procedure. This procedure will examine all of the relationships among the various entries associated with this item. If it succeeds, it will store the serial number and a current datetime in the valid table and return a success status. If it fails, it will remove the serial number and date-time from the validity table (if present) and return a failure status. One can examine the validate_calib_set procedure and calib_set_valid table in the calibrations database for an example of this strategy.

3 Introducing the MTF Virtual Databases

The fundamental organization of SYBASE databases is determined by the SYBASE software design. The client-server organization places all data under the control of the SYBASE server. All production data for MTF activities are held in a single SYBASE server. As described above, the tables of data are then organized by dataowners. This section will describe this organization (See Figure 1).

3.1 MTF SYBASE Database Servers and databases

To support the creation of various database and data acquisition tools as well as the database design effort, several SYBASE servers have been implemented. Each server can serve independent needs. At present we have defined the following servers:

MTF-PRODUCTION Server for permanent data and definitions.

MTF-DEVELOP Server for user intensive development such as development of database schema and database applications.

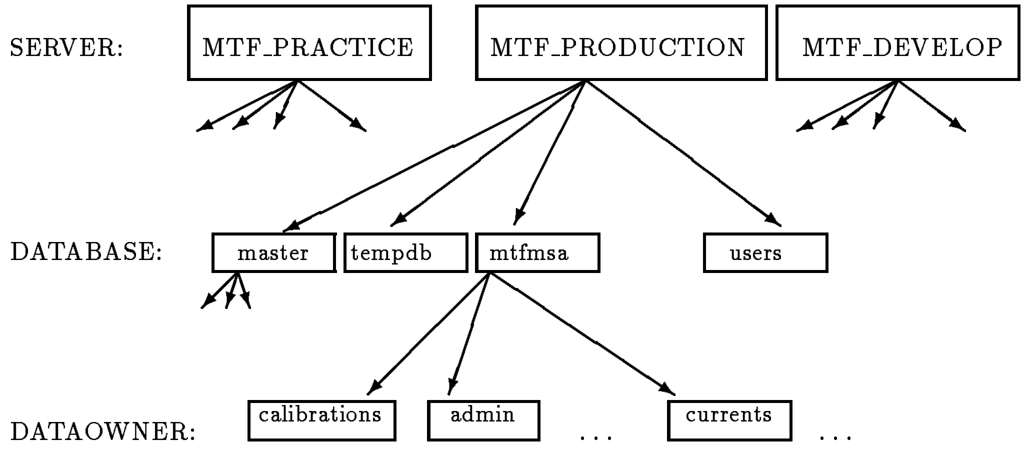

Figure 1: Overview of MTF Data Storage System. Dataowners are used to organize tables within a database. All production measurement data will be contained in the mtfmsa SYBASE database on the MTF-PRODUCTION SYBASE server.

MTF-PRACTICE Server for data intensive development such as data acquisition and data transfer application development.

Access to these Servers is selected with the environment variable DSQUERY or with a command line parameter for most SYBASE applications.

Under each SYBASE server, many databases are placed. The database administrator (DBA) has control of these databases and can assign that control suitably. Management issues such as concurrent control of backup and logging files make it convenient to store all of the data which we are describing in this document in one SYBASE database. This database has the name 'mtfmsa'. Independent copies of it will exist on the three servers. Other databases exist in each MTF server, but they will not be addressed in this document.

This document provides a view of the database structures implemented at a given time. If you wish a current view of the structure of the design, you may query the database tables in which our Table of Tables application stores this information. These are in the 'tot' virtual database and the tables of interest include the dataowners table, the tables table and the columns table. These can be queried directly or via the 'tot' application.

3.2 Virtual databases for MTF Measurement Subjects

3.2.1 subjects

The MTF testing systems is planned on the basis of testing multiple subjects (usually magnets) in a series with the same design features for all things in the series. The subjects virtual database contains the lists of test subjects. Separate tables contain attributes of the series and specific attributes of the subjects. A hierarchical structure provides for relating each object to the components which comprise it.

3.3 Virtual databases for MTF Measurement Systems and Techniques

3.3.1 admin

The admin virtual database contains tables which are used throughout the MTF database system. The serial number system is here, as is the documents table. Personnel-related information including the tables: personnel, user_accounts, groups, sybase_permissions, and group_sybase_permissions are stored here.

3.3.2 facilities

Information on the hardware, software, and real estate which has been obtained for the MTF measurement system is described by tables in the facilities virtual database. Manufacturer and products tables are shared between software and hardware. Specific hardware devices are described by a devices and device_parts table. Software versions, installations, and documentations each have a table. Locations for measurements (stands) and material storage are recorded in the locations table. Electronic systems for measurement are described by tables da_systems_classes, measurement_stations and da_systems. Some purchase order information which will help identify facilities is also stored here.

3.3.3 logbook

The logbook provides a prescription (checklist) for measurements to be carried out and a description (activity_log) of what measurements actually have been carried out and in what sequence. Comments and assessments on the various aspects of the operation can be recorded and attached to all items in the database.

The hierarchical checklist is created using the checkitems and checklist_contents table with auxiliary information attached in the installed_sublist_versions table. Applications and application_types tables describe what applications are available to carry out specified checkitems. The activities_log is a table with the auxiliary activity_data table to record details. The activities and activity_status tables provide generic labels for identifying activities. The assessments and comments tables may utilize associated connections tables to represent a useful many-to-many relation between these features and other entities stored in this database.

3.3.4 instruments

Within the MTF database an instrument is the labeled collection of hardware devices and software required to perform a control or measurement operation. We have constructed virtual, conceptual and installed instruments for measurement prescription. Tables which describe these entities and their relations are in this database.

3.3.5 calibrations

The functions for converting data associated with inputs, conversions, or outputs for a given device and their associated numerical parameters are stored in the calibrations database. In addition, tables are provided in which analysis of calibration data can be recorded and sets of useful calibration data identified for use in data acquisition and analysis programs.

3.3.6 ramps

Specification of the required time variation (ramp) of the magnet current and requirements for verifying that time variation are stored in the ramps virtual database. Both the types of information (ramp types and verification procedures) and the detailed parameters are provided. The choice of which items are executed is carried out using the logbook checkitems and checklist contents tables. The rampitems table is an extension of the checkitems table which shares a common primary key.

3.3.7 tot

Information required to specify database schema in the MTF Table of Tables database design tool are stored in this virtual database. In addition to information required to build the database, descriptive information is stored on the dataowners, tables and columns. Of general interest is the information stored in the columns table which includes the desired label, units and description for each column in the database.

3.4 Virtual databases for MTF Measurement Results

3.4.1 currents

The measured results on magnet excitation currents are stored in the currents virtual database. This information includes the detailed measurements associated with specific magnetic measurements and also results which allow determination of the magnetic history associated with each measurement.

3.4.2 points

The MTF POINTSCAN measurement uses NMR and Hall Probes to record magnetic fields at points within the magnetic field. The points virtual database is designed to record data collected with this set of equipment.

3.4.3 flatcoil

The MTF FLATCOIL measurement uses various coils attached to rigid coil forms or constructed from stretched loops of wire to record magnetic flux changes. Data from this system is recorded in the flatcoil virtual database.

3.4.4 harmonics

The MTF HARMONICS measurement uses various coils attached to rigid cylinders and measures flux changes associated with rotating those probes in static or slowly varying magnetic fields. The harmonics virtual database records HARMONICS measurements.

3.4.5 tevatron

Measurements of properties of magnets created for the Fermilab Tevatron superconducting accelerator have been measured at MTF in the period from 1979 to the present. Measurements on the cryogenic test stands using PDP11 computers (and more recently some VAX-based measurements) are stored in this virtual database.

3.4.6 lobeta89

Some measurement of magnets associated with the 1989 Low Beta upgrade to the Fermilab Tevatron are stored in this database.

3.4.7 vaxdata

This virtual database is used to store measurements from the MTF VAX HARMONICS and VAX FLATCOIL measurements. These measurements have been used on a variety of copper and iron magnets from 1984 to the present.

4 MTF Database: Broadly Used Features

The following features of our database system are described more fully (and the descriptions will be kept current) in the documentation for the admin virtual database. However, their significance in understanding our overall system is such that their description is included in this document as an introduction.

4.1 The Serial Number System

Serial numbers are utilized as the primary key in a large number of our tables. This provides several advantages, but the concept got started as a way to allow our comments to be quite general. Let us describe the development of the serial number concept and then the specifics of its application[5].

[5] The serial number system is included in the admin virtual database and the latest version of the document "Design of the **admin** Virtual Database" provides the most current information on this design.

4.1.1 Comments and the serial number system

In designing our system, it was realized that comments apply to a wide variety of things. But we preferred to store them in a single place. They may not even apply to just a single item but may be properly directed, as a comment, to more than one item. For example, a device under test had some hardware which was used to test it. An unusual test result may deserve a comment which should be attached to the hardware devices (and instrument styles) and also to the test results. In designing a flexible and precise comment scheme, we considered the possibility of attaching a single serial number column in each table which one could then reference in making one's comment. In order to make comments and the connection between the comments and the serial numbers straightforward, we store the serial numbers in a table in which we also record the table for which it was created.

As we began implementing such a system, we found the serial number concept was of substantial value in several other aspects of our database schema. By inserting into each row (of some table) a serial number as a unique key, one can always use that as the primary key for the table, and refer to it as the foreign key which one utilizes in other tables. This not only simplified the design of the tables but made possible the design of an automatic trigger writing system within our Tables of Tables tool to provide referential integrity.

4.1.2 The Serial Number Table and the Serial Generator

The serial table is a sequential enumeration of serial numbers and tables in which they are employed with these columns: serial (sntype), target_table_name (namestr) and issued_dt (datetimestamp). By saving the location (table) in which each serial number is used, one can refer only to the serial number and use the serials table to find the useful information associated with that serial number. When creating and registering a serial number for a new row in a table, a new serial number must be created and the serial number and table in which it is to be used must be stored. When one inserts data into this table one must update the value stored in the current_serial table where the present value of the serial number is stored as 'lastval'. One might note that since we have not provided a separate data owners column in this table, implicit in our design is the assumption that table names are unique across the domain of data owners for which this serial server will be used. Currently we implement serial numbers as

SYBASE integers. Since we label them in all cases as type 'sntype', we are able at some later date to replace the integer type with some alternative data type with a minimal impact on the general database system. A new serial number generator would be required.

The serial creation trigger code for an arbitrary table is created when 'tot' observes that the primary key is of type sntype and NULL is allowed. This code is made part of the insert and update trigger for that table. It is executed on insert for the table when the primary key supplied is NULL. When the trigger executes it uses a serial generator stored procedure[6]. which increments 'admin.current_serial.lastval' and saves the new value which it stores along with the target table name (where insertion trigger was called) in 'admin.serials'. The trigger then inserts the new serial as the primary key in the target table.

We have not established true guidelines of the desirability of ever deleting a row from the serials table. But it is clear that triggers must include the delete trigger such that a row cannot be deleted from this table while the serial number is stored in any other table. Note that the serial numbers are used not only in the primary table in which they are key but also in other tables in which they are foreign keys.

4.2 The Comments System

The comments system consists of three tables: the comments (header) table, the comment_contents table and a comments_connections table. The combination comments and comments_contents is sufficient to record arbitrarily long ASCII text as a comment. The comment_line_number determines the order of the comments_contents rows within a comment. We have chosen to implement comment_text with a 255 byte variable length data type within SYBASE rather than the text data type because the text data type employs, as a very minimum, two kilobytes of text. We believe that this is wasteful given our expectation that most comments will consist of 20-80 characters. An existing comments application allows up to 255 characters to be recorded in a single comment field; the database design is sufficient for an arbitrarily long comment field to be included in a single comment within the comment database.

The comments_connections table then allows one to specify a relationship between any comment and any possible subject or subjects within our

[6]Following an examination of performance requirements and performance provided by SYBASE on our SUN 4/260 we selected, for our serial server, a trigger-based system.

database system. One can even make comments about comments, or comments about comment connections. To relate the comments to a given object one merely must list the comment serial number and serial number of the object to which it is to be related. One can use the serials table to discover what table the serial is created for and discover thereby its meaning.

4.3 Assessments

Not all data has the same worth. Data deserves various levels of acceptance. Rather than ask measurement personnel to make absolute decisions to keep or discard data, we ask that they identify the worth they see in that data.[7] We expect to provide a way for a measurer to evaluate data as it is presented and analyzed and to record his assessment of the data as part of an analyzed (or reduced) data header row. The acceptable values will be stored in the assessments table.

The assessment table contains a list of valid assessments. One can use these in two alternative ways. One way uses the assessments_connections table. Here a measurer or analyzer will place the serial number of an assessment with the serial number of a data set. The structure of that data set and thereby the number of items which can be described by an assessment depends on the data structures for that particular measurement style. When looking at the data, one will need independent knowledge that the assessment is provided in the assessments_connections table (or one will discover it there by a query).

An alternative way which is applied to some data sets is to mark the assessment of data in an assessment column within the data set. Attribute validity can be supplied by a trigger which allows only valid assessments. If we are to maintain a general policy that data rows allow no revisions, we must carry out the assessment only once and that prior to recording the data. Alternatively, we can permit updates to the assessment in a sufficiently restricted environment for some data sets. The way of providing assessments keeps them with the data set where a user will naturally discover their existence.

[7] We found that much time can occasionally be lost if measurers are allowed to discard data. Knowing that it is an irretrievable activity, the decision process can become unduly slow. The alternative of never letting raw data be discarded has proven to be much more satisfactory.

4.4 Parameter Tables

One feature which is used in many MTF virtual databases is called a parameter table. These tables are specified to permit one to store information in a separate table as if it were stored in an additional column of a 'parent' table. In this fashion, one can utilize fewer tables when storing related but slightly different information about entities. Information which is always specified or usually specified (but NULL allowed) are stored in columns of the table. Attributes which are needed for a limited portion of the entries in the table are not stored in that table. Instead, they are labeled and placed in a separate table. A variety of table types are specified to permit storage of a variety of data types.

This practice is implemented more easily in MTF databases due to the use of serial numbers as primary keys. One can store the primary key to almost all MTF tables in one column with a single data type. To complete a parameter table, one provides a column for storing a label. This label serves the same function as a column name in the case where one wishes to store the information in the parent table. Finally, the table must have a column to store the data. Since this must be of a single data type, separate parameter tables are needed for each supported data type. Existing tables support parameters of type serial_number, integer, float, and ascii. An additional type of parameter table stores arrays of values. A further explanation of parameter tables exists in "Design of the **admin** Virtual Database" as well as in the design documents for each database where they are defined. This design feature is also discussed in the appendix of this document, Section A.5.

An additional type of parameter table is provided in some databases to store the labels and datatypes of required parameters which are associated with various objects. These are stored in association with the tables which specify the types of these objects. An example of this is the 'instr_param_labels' table in the instruments virtual database.

4.5 Does a Table Need a Serial Number Primary Key?

With these features well understood, a designer may wish to choose whether to generate a serial number as a primary key for a table by deciding:

- Will there be a need to comment on this entity?
- Will parameters be attached to this entity?

- Will there be a one-to-many or many-to-many relation which ties this table to another?

5 Design and Implementation Guidance

The development group at MTF has developed a strategy for developing database schema. This is laid out in the document "Database Verification Methodology and Documentation" by Kelley Trombly-Freytag[8]. Following that methodology, we design virtual databases. Some design and implementation details are usefully described here.[8]

5.1 Documentation

As prescribed in the methodology, we prepare a document which identifies why we have designed our databases in a given way. This supplements the specification of what we have designed which is stored in the 'tot' design capture system and the graphic representations of our designs which are recorded in pictures using the 'idraw' graphic drawing package. Finally, we must also store procedure code which implements special SYBASE triggers and other critical SYBASE procedures. This documentation is stored and managed within our MTF Source Code Management System. A convention has been adopted which will apply to most of our database development. For a virtual database vdbxxx we will create

- A document entitled "Design of the **vdbxxx** Virtual Database" which we will develop as a LaTeX file called vdbxxx_dsgn_rpt.tex which will be stored in our code management system documents/specs directory.
- A graphic representation of the table structure (with column names, variable types/sizes, key information and foreign key relations shown) will be created and called vdbxxx_1.idr (and perhaps vdbxxx_2.idr) and it will be stored in the documents/idraw directory.
- The actual code which creates tables, rules, triggers, etc. which is created by 'tot' from the information stored in the 'tot' tables will be maintained by the DBA.

[8] To obtain a read-only copy of any documentation stored in the MTF code management system one must execute the procedure sread.sh with the full file name as a parameter. The code management system description is available by obtaining (with sread.sh) the file code_management.mem (which is an ASCII file).

- The special triggers which must be written for some tables and other procedure code will be stored in the code management system in the database/triggers directory.

- An SQL report which documents the 'tot' entries for a given virtual database is obtained by entering the command 'get_virtual_db'.

This information is utilized in the design review mechanism specified in [8] and is then available for all users.

5.2 Implementation

To develop systems and applications using this database system, one will utilize the following steps:

1. Develop the database schema using the steps outlined in the methodology document described above.

2. Using the data (in 'tot') and documentation from the previous step, the DEVELOPER will work with the DBA to implement the database on a suitable server (see also section 5 of TM-1707):

   (a) Using 'tot', create scripts for table definition, foreign key creation, insert and update trigger creation, and delete trigger creation.

   (b) For tables which require special triggers, examine the requirements specified in the design documents and create code to be integrated with the 'tot' trigger code to implement the special triggers.

   (c) Combine the special triggers as suitable with the 'tot' output to create trigger scripts for each table.

   (d) Execute the required scripts to create the virtual database.

3. Develop applications to fill the tables created. In some cases a general use tool such as DWB (SYBASE Data Work Bench) will suffice for some tables.

4. Work with DBA to move suitable data and specified applications to production environment.

The applications will be stored within the MTF Code Management System.

6 Tools for Accessing MTF Databases

The requirements to access the MTF database depend strongly on the type of user involved (see Section 1.2). This discussion will focus on the DATA-USERs since this group has the broadest needs. In addition, the access which will be provided for most specialized users is not fully defined (June 1992). We will be providing specific applications for data entry and reports for data retrieval which cover the most important and frequently used tables. These will include data entry applications, data analysis applications and reports of validity and completion for both technical and management requirements. In addition, there are several general purpose applications which can be utilized to access MTF data tables. Some discussion of each of these will follow. Contact MTF personnel to learn how to access any of these tools. Also note that the data types and units and a description for each column can be obtained by querying the 'tot.columns' table.

6.1 SYBASE isql

The SYBASE isql (Interactive Structured Query Language) program allows a users to interact with the database using SQL commands in all ways which are consistent with his privileges. In particular, a user can query the database interactively, can write scripts which he can then execute, and can use isql in UNIX shell scripts and in conjunction with other UNIX utilities such as grep.

6.2 SYBASE dwb

The SYBASE Data Work Bench program provides a facility for interacting with the SYBASE server. Queries using SQL can be entered directly or with the assistance of VQL (Visual Query Language)[9] *Ad hoc* queries can be formulated and formatted. The SYBASE Report Workbench[9] is available on MTF computers as part of dwb. It is a tool for accessing databases to create or execute formatted reports. Such reports can also be run directly or can be embedded in a UNIX shell script (or VMS DCL procedures) to permit their easy execution.

[9] The Visual Query Language is a forms based tool for constructing SQL statements and providing for their execution and also for formatting the results. It provides a useful introductory tool for interacting with the SYBASE database.

6.3 20/20 Spreadsheet and the 20/20 Database Connection

The 20/20 Spreadsheet program (available from Computer Associates, Inc.) is a spreadsheet program available on many computer platforms including VAX/VMS and SunOS UNIX. The MTF group has purchased a license for multiple (currently 3) users to access 20/20 on MTF computers. In addition, we have obtained the 20/20 Connection software with access to our SYBASE server. Using this software, one can access the SYBASE data and obtain query results directly in the spreadsheet software where it can be used for calculations and results can be obtained and formatted for display. 20/20 includes simple graphics as well as calculational capabilities.

6.4 Other Tools at MTF

In addition to the above tools for direct interaction with SYBASE data, we expect to employ several other general purpose tools. These currently include DataViews for graphics display, GNUPLOT for graphing of simple results, and PAW (Physics Analysis Workstation) for analysis and display. Simple bridges can be provided to each from our SYBASE database.

6.5 Access from other SYBASE clients

MTF currently has SYBASE client software on Sun SPARC and DEC VAX workstations. We will use these for our access to SYBASE. Other DATA-USER's may wish to access SYBASE from their own work area. SYBASE clients at other locations will be able to access this data through their own client software.

6.6 Display terminal and printer issues

The terminal environment required to access MTF database depends upon the specific application which is employed. SYBASE applications including Data WorkBench, 'tot', and 20/20 all include both X-Windows support and terminal support for at least a vt100 class terminal. The user is asked to consult documentation or MTF personnel to confirm the display requirements for any given application. Printer output options will always include POSTSCRIPT. When possible, other choices, such as Talaris or Printronix printers will be permitted so that user flexibility is maintained.

A Some Generalities on Database Design

In this section we will explicitly discuss some of the sorts of data structures we find ourselves utilizing repeatedly. The term entity is used in this context to represent a 'thing' about which information (attributes) needs to be stored. Different entities require different data structures to store the complete information about them. Some discussion of typical structures may make current and future designs more obvious. Suggestions for attributes which should be included in designs are presented.

The general discussion of database design within most texts is phrased in terms of the concept of data normalization. The concept involved in data normalization concerns defining entity storage so that information is stored only once. Data storage design is often described in terms of levels of normal form design: first normal form, second normal form, ... fifth normal form. In most cases MTF databases have achieved third normal form design. Issues of entity and relational integrity are important in how these designs are structured since correctly understanding the entities avoids mixing data from related but distinct entities in ways which make data upkeep (maintenance) difficult.

A.1 Entities Described in One Table

When an example of (instance of) an entity can be well described by providing a single set of values to a set of attributes, a single table is a sufficient structure. Columns are defined for each attribute to be specified. A data type specifies the kind of data which can be stored. If NULL is to be permitted, that must be specified. (It must be realized that this demands *extra* storage space.) Defaults may provide for a value to be stored if none is supplied. Rules or foreign key triggers may be used to restrict the values to a chosen domain. The following questions usually need to be considered in designing the columns:

- Is a date or datetime useful in understanding this data?
- Is it meaningful to associate with this data a responsible person?
- Does this table have use for a generated serial number as its primary key? (See Section 4.5).

A.2 Master-Detail Design and Fixed Hierarchies

A very common structure is used for entities which have a variable number of similar parts. An example is a shipping invoice. Here the entity is the shipment. To describe it we have a 'master' table in which the existence of the shipment is recorded along with information such as the shipping and receiving firms, the date and other information which pertains to the entire shipment. But since the shipment involves a variable number of items, a relational design for storing the information on the shipment contents utilizes a 'detail' table in which each item which is shipped is listed. The detail table must store the key to the master table plus the identity of each item which is included in the shipment. Will the detail table list the quantity of each item? This would be typical, but it is also possible to describe the contents with an entry for each individual item.

Within the MTF database design, the 'admin.purchase_orders' and 'admin.purchase_order_items' tables are typical master-detail designs. The full structure of the facilities database is more complex but 'facilities.manufacturers' and 'facilities.products' provide a simple master-detail structure. A three level hierarchical structure which is simply two nested master-detail lists is illustrated in 'tot.dataowners', 'tot.tables' and 'tot.columns'. The structure of measurement data storage is expected to usually consist of a set of three (or more) related fixed hierarchical structures with labels of sequences, runs, and points for the levels of detail.

A.3 Hierarchical lists

The need to construct a structure for an arbitrary hierarchical relationship is solved with a somewhat different construct in our databases. In Table 2 we illustrate such a hierarchical problem with a job list in outline form. We represent this in just two tables. The first table (items) lists the items which appear on the lists, whether they be jobs to accomplish or the name of another list. Each item must appear in this table. The second table (list_contents) describes the master-detail relation between items in the first list. A row in the list_contents must define what list it is a part of (master) and which item in the list it declares (detail). If order (sequence) is significant then a column in the list_contents table must declare (number) it. We use this for both work list problems and the more familiar 'parts list expansion' problem. A work list example is logbook.checkitem and logbook.checklist_contents, in which order usually is significant. A parts list

Today's Job List

1. Clean House
   (a) Mop Hallway
   (b) Vacuum Den
   (c) Clean Kitchen
      i. Wash dishes
      ii. Mop Floor
      iii. Wash Windows
      iv. Sort Junk Drawer
   (d) Vacuum Basement Stairs
2. Wash Car
3. Do Shopping
   (a) Buy Groceries
   (b) Pick Up Cleaning
   (c) Order Flowers
4. Mow Lawn
5. Read Newspaper

Table 2: An Outline of a Job List

expansion example is the subjects.subjects and subjects.subject_components tables where no meaningful ordering is required. Note that there is no restriction implied that an entry in the items table appears only once in a list_contents table. In the example of a job list, it is quite possible that one needs to repeat a cleaning job again.

A.4 Many to Many Relations – Connection Tables

Some critical information can be provided by simply connecting two sets of data. This can in general be done in a relational database by entering in a row of a single table the primary keys from rows in two other tables. Examples of this sort are quite common in our database design. They are frequently referred to as many-to-many relations in the database design literature. The MTF database frequently identifies them as connection tables. We have discussed above the case of the comments_connection table in which one identifies a comment and a target to which that comment applies. This sort of connection is made much more convenient in our designs by utilizing serial numbers as primary keys, since it permits a connection to be specified with just two columns.

The concept of a one-to-many relation is also easily handled in the relational database design. The master-detail design is an example. Other uses of the one-to-many design may not actually fit the master-detail construct when one considers the contents of the tables, but the data structure may be the same.

A.5 Explicit Columns *vs.* Parameter Table Data Representations

When large volumes of data must be represented in a database, the design of tables will proceed to specify precise data representations and normalization arguments will provide guidance in clear specification of data storage. Each attribute will be assigned to a column in a table. Performance issues may cause some data to be de-normalized but storage space limits will demand efficient storage design.

We face a different problem. Small to modest volumes of data of a wide variety of types must be stored in the MTF database for measurement control. One, a few, or at most dozens of items will be of a given type. An entity description which matches each item to its own storage description will require a laborious effort in database, application and management design effort. By providing a more generalized data storage schema, the design and maintenance tool costs can be spread across a wider variety of entities. We have designed this method of data storage by storing both labels and data within our tables so that a given table can store data with a wide variety of uses.

A.5.1 Storing generalized entities in generalized columns

Consider the problem of generalizing a table which is designed to describe a given device. Suppose that 1 column identifies the device (primary or alternate key) and 3 additional columns provide properties of that device. Suppose that this table is to describe nuts for machine screws. We may label the columns as part number (key) and material, size, and thread pitch. If we wish to store in this table also the screws then additional columns will be needed: length, head style. But, we will have to allow NULL to be stored in these for nuts. And we will need a column which specifies screw or nut. And what about the changes when we add a washer! We will need more NULLS and perhaps we will want to change the label on some columns. This is getting harder.

A.5.2 Storing generalized entities in parameter tables

We have chosen to use parameter tables to provide generalized data storage.[10] Let us consider parameter tables of type I (PT-I) and parameter tables of type II (PT-II). Together they will provide generalized data storage for a variety of uses with the ability to provide many attribute and relational integrity support features. Assume that each entity will have a master table with a key which identifies the item and as many other columns as are useful for this general entity.

A PT-I parameter table stores information about an entity as one row per attribute. Even with a key for the entity table implemented as a serial number (one column) we will need to provide two more columns in our parameter table. One is a label column while the other is the value of the parameter (attribute) to be stored. This table can now store a carefully labeled value for any number of properties of the entity, restricted only to properties which can be represented by the same data type. We have implemented parameter tables for data of type float (real), integer, ASCII,

[10] This topic was previously addressed in Section 4.4 but we will attempt to address it from a different perspective here.

and sntype (serial number). The appropriate primary key for this table is the entity key column plus the label.

A.5.3 Storing arrays of generalized entities in parameter tables

To aggregate groups of values which together describe an attribute one need only add an order column to a parameter table definition. This table will have a primary key of the serial number of the entity, the label and the order. Values with different labels but the same order number can now be associated. A program which queries the database can use these related parameters to fill arrays or structures. These tables are called 'table parameter' tables.

A.5.4 Providing attribute integrity for generalized entities in parameter tables

To permit one to provide attribute integrity for the labels in a PT-I parameter table, one constructs a PT-II parameter table. Here one lists the valid labels for parameters in PT-I tables associated with some entity table. This table will have columns for a foreign key to the entity table, a valid labels column and columns to declare where values with this label are stored. (This is equivalent to declaring the data type of the attribute with that label.) In the MTF Database, this construct usually has one additional level of abstraction. The entities are in classes. The PT-II tables identify valid labels for classes of entities. The serial number of the class provides a foreign key in the PT-II table. In choosing a label for an attribute of an entity in a PT-I table, one verifies the class of the entity and queries the PT-II table for valid labels for entities of that class.

Such a structure is completely flexible, is compact as a way of storing modest quantities of information but provides no guidance to the report writer as to how many columns he might want to write in series, write together or other features. Users of this data will find that formatted reports will be particularly helpful in this regard.

B MTF Use of SYBASE Groups & Privileges

A part of providing entity integrity in a database consists of protecting the data by providing that only specific groups of users have authorization for specific actions. Within SYBASE, permissions are defined which apply to a domain of objects. A simplified view of this is shown here.

DEFINED PERMISSION	APPLICABLE OBJECTS
select	table, view, column
insert	table, column
update	table, column
delete	table
execute	procedure
create	table, procedure, view
create	default, rule, database

At MTF these permissions are granted to users only as members of groups. Each user account is assigned to precisely one group.

Within the MTF admin virtual database, tables are defined to provide ways to document groups and assign permissions to the groups. This structure could be used by 'tot' to assign permissions as code is created. Although we expect to provide this in the future, we currently have only coded within the 'tot' build applications a set of hardcoded permissions which are assigned according to the following list. This has been organized to provide a strictly hierarchical arrangement in which each successive group has more privileges than the previous one. This is currently convenient, but not fundamental.

The groups which we currently have defined and which are used by the code in 'tot' include:

public

data_clerk

measurer

test_manager

developer

administrator.

We currently assign the permissions for objects created by 'tot' build scripts as follows:

PERMISSION	OBJECTS	GROUPS
select	table	public
insert, update	table	data_clerk, measurer, test_manager
delete	table	administrator, developer (variable)

We described earlier the generic categories of users which will want to employ the MTF databases. Each SYBASE account will apply to a specific individual in a specific role and he (she) will be placed in a specific group according to that role by the DBA, but to further clarify the overall plan, the expected group assignment for each role will be delineated here:

role	**group**
MEASURERS	measurer
MEASUREMENT-SUPERVISORS	measurer or test_manager
TEST-MANAGER	test_manager
EQPT-TECH	data_clerk
DEVELOPER	developer
DBA	administrator
DATA-CLERK	data_clerk
DATA-USER	public

C MTF Database Documents

The following documents have been created to describe the MTF database effort at this time:

MTF-92-0003 Overview of a Database System for Magnet Measurements

MTF-92-0004 Design of the **admin** Virtual Database

MTF-92-0005 Design of the **calibration** Virtual Database

MTF-92-0006 Design of the **facilities** Virtual Database

MTF-92-0007 Design of the **instruments** Virtual Database

MTF-92-0008 Design of the **logbook** Virtual Database

MTF-92-0009 Design of the **ramps** Virtual Database

MTF-92-0010 Design of the **subjects** Virtual Database

MTF-92-0011 Design of the **currents** Virtual Database

References

[1] American National Standards Institute. *Database Language SQL.* Document ANSI X3.135-1986. Also available as International Standards Organization Document ISO/TC97/SC21/WG3 N117.

[2] J. Harvey Trimble, Jr and David Chappell. *A Visual Introduction to SQL.* John Wiley and Sons, New York, 1989.

[3] Marcy Darnovksy and July Bowman. *Transact-SQL User's Guide.* Sybase, Inc., Emeryville, CA, May 1989. Release 4.0.

[4] Judy Bowman and Karen Paulsell. *Data Workbench User's Guide.* Sybase, Inc., Emeryville, CA, May 1989. Release 4.0.

[5] D. McGoveran and C. J. Date. *SYBASE and SQL Server.* Addison-Wesley Publishing Company, Menlo Park, CA, 1992.

[6] C. J. Date. *An Introduction to Database Systems.* Addison-Wesley Publishing Company, Menlo Park, CA, fifth edition, 1990.

[7] Bruce C. Brown, Karen Coulter, Henry D. Glass, Richard Glosson, Raymond W. Hanft, David J. Harding, Kelley Trombly-Freytag, Dana G. C. Walbridge, and David B. Wallis. Table of Tables – A Database Design Tool for SYBASE. TM 1707, Fermilab, January 1991.

[8] Kelley Trombly-Freytag. Database Verification Methodology and Documentation. Unpublished MTF document, 1991.

[9] Dee Elling and Bob McDonald. *Report Workbench User's Guide.* Sybase, Inc., Emeryville, CA, August 1990. Release 4.1.