ALEPH 88-205
DATACQ 88-38
12th November 1988

A.Belk, D.R.Botterill, J.Harvey

ALEPH Online Database

Abstract

This report summarises the approach taken to the design and implementation of the Aleph online database. It offers a prescription for people wishing to add new information to the database in the same style. It describes what facilities are available for generating and managing the database. An example illustrating how one component of the database has been devised is given in an appendix.

CONTENTS

CHAPTER 1

INTRODUCTION

A large number of parameters are used to describe the various elements of the data acquisition system (eg. the readout and control systems) as well as the various software components used to configure and administer datataking (eg. partitions and runtypes). As these parameters are needed by many parts of the software it is important that they are described in a complete and consistent way. Thus a common approach has been followed for the design and implementation of a database system for containing and accessing these parameters. This approach is described in this document.

CHAPTER 2

OVERVIEW OF ONLINE DATABASE

2.1 DAQ components described in the database

The following is a brief summary of the current contents of the online database. Note that each category is described more completly elsewhere.

- The detector description

  This defines the general structure of the detector in terms of subdetectors, their components and slots. It contains geometry information and other detector specific parameters. It actually consists of a subset of the general Aleph DataBase System (ADBS), which is considered relevant for the online system.

- The partition description

  A partition is a subset of the experiment which can be operated as a self contained datataking system. Partitions are defined in terms of the detector components they contain.

- The run description

  This contains descriptions which can be used to configure the experiment in different ways according to the wishes of the operator. Quantities described include runtypes, tasks and output destinations.

- The readout description

  This contains descriptions of the main readout components, such as Event Builders, Trigger Supervisors etc. and their relationships with the detector components.

- The Fastbus description

  This describes the Fastbus segments and their connectivity as well as all Fastbus devices. The contents of these tables reflect the status of the readout system at any time.

- The slow control description

  This describes the G64 crates, gateways and Utinet local area network. Slow Control devices are also described including their relationships to detector components.

- The graphics description

  This contains a graphical representation of elements of the system (ie. detector, readout and control systems) in terms of the primitives used by a graphics package. Attributes such as default colour, line aspect etc. are also contained in the description.

2.2 Design approach

A data model (entity - relationship) has been used to help ensure the consistency of the database design. This model results in the data being organised in tabular form. Each table contains the description of a group of identical objects; each member of the group occupies one row in the table and each attribute used to describe it occupies a column. Each object can also be further described through its relationship with other types of objects, described in other tables.
An advantage of using this data model is that the unnecessary repetition of information can be avoided and thus the integrity of the data is enhanced. Another advantage is that exactly the same format is used by relational database systems, such as ORACLE.

Diagrams are used to make the design, supplemented by a data dictionary for specifying table contents more precisely. The format of the data dictionary follows that defined by the ADAMO system. An ADAMO tool (VLB) is used to automatically generate the code for making variable declarations from within FORTRAN.

2.3 Implementation Details

As the data structures contained in the database are just tables they can be managed using the ORACLE database system we have running on our central VAX. However, as we were concerned about the overhead when accessing the data through ORACLE, the database has also been implemented using VAX global sections. Each major component of the database has been implemented on a distinct disk based section file. Various tools have been written for copying data between the ORACLE database and the global sections.
As the data has a tabular structure and the language most commonly used on the VAXes is FORTRAN the tables have been declared using the VAX FORTRAN STRUCTURE and RECORD statements.

Normally data acquisition tasks needing access to the database will just map the disk based section corresponding to the component of the database they are interested in. The same copy of the database is available to all machines in the cluster including all VAXstations using the 8700 as a boot node.

ORACLE will normally be used only for exploiting the facilities it offers for managing the database eg. backups, rolling back previous versions, checking integrity etc. ORACLE also provides the possibility of changing the structure (ie. definition) of the tables whilst maintaing all the existing data intact. ORACLE will thus typically be used by the people responsible for managing the various parts of the database.

A UPI interface is being provided for interrogating and managing the contents of the database implemented with global sections. A common style is being followed for the menus. ORACLE can be interrogated using its SQL interface.

2.4 Software Tools available

In order to access data in a global section various files have to be written for each component of the database

- A software routine is needed to map the section file to the variables defined in the program.

- An include file is needed to define the STRUCTURE and RECORD declaration statements
- An option file is needed to link the global section

In principle all these files can be specified automatically from the description of the data given in the data dictionary and therefore several tools have been written to automatically generate them. Tools are also available for automatically generating the ORACLE tables and for transfering data between ORACLE and the global section files.

CHAPTER 3

IMPLEMENTATION PROCEDURES

This chapter describes the procedures that have been written for creating and managing the database.

The first stage concerns the creation of the global sections. Before starting, you should have a description of your tables in DDL and an Oracle id/password (this can be obtained from David Botterill). This Oracle ID cannot be used for other purposes as it will be erased by the one of the steps described below. Oracle should be used to keep a backup copy of the global section. The procedure has three major steps:

- Create an empty set of Oracle tables.
- Create routines, .OPT files, and executable programs to create the global section.
- Prepare user programs to access the global section.

As ADAMO, Oracle and Fortran refer to objects by different names it is worthwhile to review some of the terminology used by these three programming environments.

ADAMO	ORACLE	FORTRAN
Entity Set	Table	Array of RECORDs
Attribute,Relationship	Column	Field in Structure statement
Instance of an Entity	Row	One record of the array

Note that the DDL must define the maximum size for each table using the SIZE parameter. This information is used to define the number of RECORDs reserved for the corresponding Fortran array. Initially it is advisable to add some spare columns to avoid rebuilding the section too often.

3.1 Global Section Files

The Aleph DAQ global sections are in files in directories below DISK$COMMON:[ONLINE.ONLDB.component] where component is the name given to each separate part of the database. The structure of the subdirectories below this level and the convention for the logical names used to point to them is as follows:

Logical Name	SubDirectory	Contents
A_xxDB$SRC	[.source]	include file containing declarations .VLB
A_xxDB$DIR	[.nodeb]	files used in linking to global section
A_xxDB_MAK$SRC	[.make.source]	working directory SOURCE files
A_xxDB_MAK$DIR	[.make.nodeb]	work area and utilities EXE files
A_xxDB$GBL	[.dbase]	the global section file

Having created the subdirectory [.component] and made it the default directory, executing the following command file will create the lower sub-directories.

@disk$common:[online.oragen.nodeb]create_global_dir

The command files described in the following sections require the logical names defined above, but do not depend on the directory structure.

3.2 Creating a set of empty Oracle tables starting from DDL

Execute the following command file :
Run @DISK$COMMON:[ONLINE.ORAGEN.NODEB]CREATEDB

Please note that this command file creates a lot of files in the current directory. Also it should only be run on AL0VOL as it needs Oracle.

You must answer the following questions:

```
DDL filename(s) (RET=CSP done)
     Filename of the DDL file. Enter nothing if the ADD file already exists

ADD filename
     Filename of the ADD file, excluding the filetype

Subschemata (RET= ALL)
     Default is to process the whole DDL. Separate schemata names by commas,
no surrounding () is needed.
Include (VLB) filename
     Filename of the VLB file, excluding the filetype
Database name
     The ORACLE user id
Password
     Oracle password
```

Note that there is a known bug in the VLB step which can create a VLB file with lines exceeding column 72 if table names exceed 9 characters. These lines should be split with an editor.

3.2.1 Creating and linking a global section

Before attempting this step please note the following:
Firstly you should already have a VLB file and a set of (possibly empty) Oracle tables. Secondly you must be running on AL0VOL as Oracle is needed. In addition this step must be run after each change of version of VMS or Oracle.

The following command file can now be executed:

$ @DISK$COMMON:[ONLINE.ORAGEN.NODEB]ORAGEN Answer the following questions:

```
Global section name (fb/sc/re/...)?
     This should be a short name for the global section, xx in this
document
Prefix logical name of output directories
     This asks for the string which is represented as A_xxDB and in fact
A_xxDB is the default, where xx is the short name above
File spec of include (VLB) files
     This is the full file specification of the VLB file and should have a
"standard" form  eg  A_xx$SRC:xx.VLB
Database name?
     The ORACLE user id
Password?
     Oracle password
Run compilations in batch (Y/N)?
     Answer Y if you have a lot of tables (>10 depending how
patient you are)
Delete temporary files (Y/N)?
     Answer Y normally
```

This step should create the following files:

Directory	File	Use
A_xxDB$SRC	CREATE_xx_GLOBAL.FOR	used to map section
A_xxDB$DIR	ALL_xx_CLUS.OPT	the linker .OPT file
A_xxDB$DIR	CREATE_xx_GLOBAL.OBJ	OBJ file for mapping section
A_xxDB_MAK$DIR	xx_GBLSEC_LIST.EXE	program to list the global section
A_xxDB_MAK$DIR	xx_GBLSEC_ORA.EXE	program to transfer data between ORACLE and the global section
same as VLB	VLBname.LEN	include file with parameters defining the size of each table

3.2.2 Creating the Global section

To create the global section in A_xxDB$gbl
$ Run xx_GLBSEC_ORA

This program asks for the Oracle id and password, and then one of the following keywords:

- "toglobal" - to copy data from Oracle to the global section
- "tooracle" - to copy data from the global section to Oracle
- "compare" - to compare the global section with Oracle

These commands can be abbreviated to unique strings ie. tog,too,and c

3.2.3 Listing the Global section

The contents of the tables in your global section can be listed as follows:
$ RUN xx_GLBSEC_LIST

It asks for a table name. "ALL" can be specified to list the contents of all tables. If table names are abbreviated, the first matching name is used.

3.2.4 Preparing programs that access the global section

The main program should call for each global section
iret=CREATE_xx_GLOBAL('WRITE')

to map with read/write access, or alternatively

iret=CREATE_xx_GLOBAL(' ')

to map with read access only. The return value is a VMS condition code and should be checked for success.

If the global section is changed the following routine can be called to to force the changes to be written to disk:

iret=UPDATE_xx_GLOBAL()

Note that the contents are automatically written when your program terminates, ie. when the global section is no longer in use, so that it is not normally necesary to call this routine unless the information is needed by a program running in another machine

The global section is created by the first program that is run containing the call to CREATE_xx_GLOBAL. This program should have write access of course. There is no provision for interlocking updates of the section by different programs nor for maintaining the same section in different VAXes of the cluster. NB. Software has been written to perform the required synchronisation across the cluster but has not been released yet.

3.2.5 To link a program

The following must be included in the link commands: once per program:
A_gbl$DIR:create_global,gblsec_open,gbl_errors

and for each global section

A_xxDB$DIR:ALL_xx_CLUS/opt

These programs need SYSGBL privilege.

This completes the steps when first creating a global section.

3.3 DDL modification whilst preserving existing data

If you need to modify your DDL and you have no data in the global section or in Oracle to preserve, you may repeat all of the above steps.
If you wish to preserve your data, you should do the following

1) copy the data into Oracle using the old "xx_GLBSEC_ORA" program

2) modify your DDL and run VLB to produce a new VLB file, but do not run @CREATEDB

3) use SQLP to add any new columns or tables to Oracle.

4) rerun @ORAGEN

5) delete the old gblsec file A_xxDB$GBL:xx_GBLSEC.GBL

6) run the new xx_GLBSEC_ORA to create a new global section file and copy data from Oracle into it.

The ORACLE commands for performing step 3) are as follows:

Firstly define the Oracle symbols by

$ @disk$oracle:[oracle]orauser

and then login to ORACLE :

$ sqlp oracleid/oracle_password

The SQL prompt (SQL>) will appear.

To add columns type

SQL> ALTER TABLE table ADD(column spec);

where "table" is the table name

"column" is the column name

"spec" is related to the ADAMO type

ADAMO	SQLP spec
INTE	NUMBER
REAL	NUMBER
CHAn/CHnn	CHAR(nn)
LOGI	CHAR(1)

Note that new ADAMO relationships imply new columns. If in doubt, you can acquire a second Oracle ID, fill it with the new tables and compare with the old tables.

The SQLP command "DESC table" can be used to print a description of a "table"

To add a table:

SQL> CREATE TABLE table(

column spec,

column spec, etc

column spec) SPACE ALEPH;

You must include a column "ID" spec "NUMBER". (This could all be automated if there was sufficient demand...)

Note that old columns or tables that are not in the new DDL will be ignored by the new xx_GBLSEC_ORA program.

3.4 Implementation Conventions

1. The first column of every table is the ADAMO ID. The convention for DAQ global sections is that:

   table(i).ID <= 0 means the row is not in use

   table(i).ID = i means the row is in use

   Other values must be avoided. This convention is compatible with, but more restrictive than ADAMO.
2. Relationships create a column in one table which should contain the ID of the row in the other table. The column name is the other table unless the DDL specified a BY construct. If the relation is to one of a set of tables, there is another row with the other table name.
3. The COMMON block contains two words, COL and ROW, to be compatible with the representation of tables in BOS banks. COL is the size of a row in 32 bit words, and ROW is the number of rows. Both words are initialised by xx_glbsec_ora when transfering from Oracle to the global section.

APPENDIX A

EXAMPLE : RUN DESCRIPTION

A.1 DDL

```
SUBSCHEMA Runs

      :      'A system for storing run control information'

 AUTHOR   'J.Harvey'
 REVIEWER '  '
 VERSION  '1.4'
 DATE     '07/03/88'

 DEFINE ATTRIBUTE

      Date        = INTE [0,99999999] ;
      Time        = INTE [0,99999999] ;
      Status      = CH16 ;
      Name        = CHA8 ;

 END ATTRIBUTE

 DEFINE ESET

 RunType
    : 'Identifies the type of run'

        SIZE 100,100

        = ( Name                                  : 'Name of runtype',
            FirstSD          = INTE               : 'Pointer to first subdetector',
            FirstTask        = INTE               : 'Pointer to first task')
        ;

 SubDet
    : 'Contains list of subdetectors which can be associated with other
           parts of the database by name'

        SIZE 15,15

        = ( Name                                  : 'Name of subdetector',
            Spy              = LOGI               : 'Spy channel flag',
            FirstRT          = INTE               : 'Pointer to first runtype',
            FirstTask        = INTE               : 'Pointer to first task',
            FirstTT          = INTE               : 'Pointer to first trigtype')
        ;

 SDRT
    : 'Links subdetectors with runtypes'

        SIZE 250,250

        = ( NextRT           = INTE               : 'Down link to next runtype',
            PrevRT           = INTE               : 'Up link to previous runtype',
            NextSD           = INTE               : 'Down link to next subdetector',
            PrevSD           = INTE               : 'Up link to prev subdetector' )
        ;

 Task
        : 'All DAQ tasks to be synchronised by Run controller'

        SIZE 100,100

        = ( Name                                  : 'Name of task',
            FirstSD          = INTE               : 'Pointer to first subdetector',
            FirstRT          = INTE               : 'Pointer to first runtype')
        ;
```

```
SDTask
   : 'Links subdetectors with tasks'

       SIZE 250,250

       = ( NextTask         = INTE            : 'Down link to next task',
           PrevTask         = INTE            : 'Up link to previous task',
           NextSD           = INTE            : 'Down link to next subdetector',
           PrevSD           = INTE            : 'Up link to prev subdetector' )
       ;
RTTask
   : 'Links runtypes with tasks'

       SIZE 250,250

       = ( NextTask         = INTE            : 'Down link to next task',
           PrevTask         = INTE            : 'Up link to previous task',
           NextRT           = INTE            : 'Down link to next runtype',
           PrevRT           = INTE            : 'Up link to prev runtype',
           SpyEnab          = LOGI            : 'Spy datastream',
           DefStream        = CHA8            : 'Default datastream main/spy')
       ;
TrigType
   : 'Identifies the type of trigger'

       SIZE 50,50

       = ( Name                               : 'Name of runtype',
           FirstSD       = INTE               : 'Pointer to first subdetector')
       ;

SDTT
   : 'Links subdetectors with trigger types'

       SIZE 100,100

       = ( NextTT           = INTE            : 'Down link to next trigtype',
           PrevTT           = INTE            : 'Up link to previous trigtype',
           NextSD           = INTE            : 'Down link to next subdetector',
           PrevSD           = INTE            : 'Up link to prev subdetector' )
       ;

END ESET
```

```
DEFINE RSET

( SDRT [1,1] -> [0,*] SubDet )
        : ' Used to establish many to many rel between SDs and RTs'
            ;

( SDRT [1,1] -> [0,*] RunType )
        : ' Used to establish many to many rel between SDs and RTs'
            ;

( RTTask [1,1] -> [0,*] RunType )
        : ' Used to establish many to many rel between RTs and tasks'
            ;

( RTTask [1,1] -> [0,*] Task )
        : ' Used to establish many to many rel between RTs and tasks'
            ;

( RTTask [0,1] -> [0,1] SubDet
        BY SpyChan )
        : ' Used to identify which SD spy channel task is associated with'
            ;

( SDTask [1,1] -> [0,*] SubDet )
        : ' Used to establish many to many rel between SDs and tasks'
            ;

( SDTask [1,1] -> [0,*] Task )
        : ' Used to establish many to many rel between SDs and tasks'
            ;

( SubDet [0,1] -> [1,1] Task
        BY ControlTask )
        : ' Used to identify which task is the control task for each SD'
            ;

( SDTT [1,1] -> [0,*] SubDet )
        : ' Used to establish many to many rel between SDs and TTs'
            ;

( SDTT [1,1] -> [0,*] TrigType )
        : ' Used to establish many to many rel between SDs and TTs'
            ;

( TrigType [0,1] -> [0,*] Task )
        : ' Used to establish many to many rel between trigtypes and tasks'
            ;

END RSET

END SUBSCHEMA
```

A.2 VLB

The declaration code generated from the DDL by the VLB tool for this Subschema is as follows

```
      STRUCTURE / Runs_RTTask /

        INTEGER                    ID
        INTEGER                    NextTask
        INTEGER                    PrevTask
        INTEGER                    NextRT
        INTEGER                    PrevRT
        LOGICAL                    SpyEnab
        CHARACTER*8                DefStream
        INTEGER                    RunType
        INTEGER                    SpyChan
        INTEGER                    Task

      END STRUCTURE

      RECORD   / Runs_RTTask /  RTTask  ( 250 )

      INTEGER RTTask_COL, RTTask_ROW
      COMMON   / COM_RTTask / RTTask_COL, RTTask_ROW, RTTask
C
C-----------------------------------------------------------------------
C
      STRUCTURE / Runs_RunType /

        INTEGER                    ID
        CHARACTER*8                Name
        INTEGER                    FirstSD
        INTEGER                    FirstTask

      END STRUCTURE

      RECORD   / Runs_RunType /  RunType  ( 100 )

      INTEGER RunType_COL, RunType_ROW
      COMMON   / COM_RunType / RunType_COL, RunType_ROW, RunType
C
C-----------------------------------------------------------------------
C
      STRUCTURE / Runs_SDRT /

        INTEGER                    ID
        INTEGER                    NextRT
        INTEGER                    PrevRT
        INTEGER                    NextSD
        INTEGER                    PrevSD
        INTEGER                    RunType
        INTEGER                    SubDet

      END STRUCTURE

      RECORD   / Runs_SDRT /  SDRT  ( 250 )

      INTEGER SDRT_COL, SDRT_ROW
      COMMON   / COM_SDRT / SDRT_COL, SDRT_ROW, SDRT
```

```
C
C-----------------------------------------------------------------------
C
      STRUCTURE / Runs_SDTT /

        INTEGER                    ID
        INTEGER                    NextTT
        INTEGER                    PrevTT
        INTEGER                    NextSD
        INTEGER                    PrevSD
        INTEGER                    SubDet
        INTEGER                    TrigType

      END STRUCTURE

      RECORD  / Runs_SDTT /  SDTT ( 100 )

      INTEGER SDTT_COL, SDTT_ROW
      COMMON  / COM_SDTT / SDTT_COL, SDTT_ROW, SDTT
C
C-----------------------------------------------------------------------
C
      STRUCTURE / Runs_SDTask /

        INTEGER                    ID
        INTEGER                    NextTask
        INTEGER                    PrevTask
        INTEGER                    NextSD
        INTEGER                    PrevSD
        INTEGER                    SubDet
        INTEGER                    Task

      END STRUCTURE

      RECORD  / Runs_SDTask /  SDTask ( 250 )

      INTEGER SDTask_COL, SDTask_ROW
      COMMON  / COM_SDTask / SDTask_COL, SDTask_ROW, SDTask
C
C-----------------------------------------------------------------------
C
      STRUCTURE / Runs_SubDet /

        INTEGER                    ID
        CHARACTER*8                Name
        LOGICAL                    Spy
        INTEGER                    FirstRT
        INTEGER                    FirstTask
        INTEGER                    FirstTT
        INTEGER                    ControlTask

      END STRUCTURE

      RECORD  / Runs_SubDet /  SubDet ( 15 )

      INTEGER SubDet_COL, SubDet_ROW
      COMMON  / COM_SubDet / SubDet_COL, SubDet_ROW, SubDet
C
C-----------------------------------------------------------------------
C
      STRUCTURE / Runs_Task /

        INTEGER                    ID
        CHARACTER*8                Name
        INTEGER                    FirstSD
        INTEGER                    FirstRT

      END STRUCTURE

      RECORD  / Runs_Task /  Task ( 100 )

      INTEGER Task_COL, Task_ROW
      COMMON  / COM_Task / Task_COL, Task_ROW, Task
```

```
C
C-----------------------------------------------------------------------
C
      STRUCTURE / Runs_TrigType /

        INTEGER                   ID
        CHARACTER*8               Name
        INTEGER                   FirstSD
        INTEGER                   Task

      END STRUCTURE

      RECORD  / Runs_TrigType /  TrigType ( 50 )

      INTEGER TrigType_COL, TrigType_ROW
      COMMON  / COM_TrigType / TrigType_COL, TrigType_ROW, TrigType
C
C-----------------------------------------------------------------------
C
```