Online-offline calibration databases evaluation

Donatella Torretta, Ruth Pordes
Online Systems Department
Fermilab Computing Division

Art Kreymer
Physics Analysis Tools
Fermilab Computing Division

ABSTRACT

We tested and evaluated three databases (ORACLE, mSQL and HEPDB) to see if they could be suitable to be used as online-offline calibration databases by the collaborating DART experiments.

Contents

1 Introduction

High Energy Physics experiments need powerful database systems in which to store information such as detector geometry and calibration constants.

Among the Databases currently available on the market we chose to evaluate three which could be appealing for different and complementary reasons: two of them, Oracle and mSQL, are SQL (Structured Query Language) based databases, while the third one, HEPDB, is based on the ZEBRA system.

Oracle is the widely known and used commercial database developed and licenced by the Oracle Corporation, Redwood City,CA and it is used in significant applications already at Fermilab (ESH tracking, MISCOMP); mSQL is a free database recently developed and distributed by Bond University in Australia; HEPDB has been developed at CERN, Switzerland.

The aim of our tests was to check the performance and reliability of the three selected databases to the insert, retrieve and update a large number of data records. From these tests we can also form a first opinion of the ease to use, support and maintenance issues.

We built an example database as close as possible to a 'real' database used by some experiment and we tried to maintain, when possible, the same structure while using these three databases.

This document reports on the tests done: with the time available to the project, it cannot fully address all the issues related to the use of databases in general and even of the three we chosen to evaluate. However it provides some useful indication of performance and hints to help choose your own database.

For more information on the specific databases, please refer to the documentation listed in the References section at the end of this document.

2 Oracle

2.1 Overview

Oracle is a commercial relational database based on SQL, a high level language with grammar and syntax rules that are basically the rules of spoken English.

Queries and data manipulation can be done both interactively, running SQL*PLUS or remotely, running an application program with embedded SQL statements. If the job has to be done on a regular basis or it needs the procedural processing capacity of a high-level language, the best way to do it is to use embedded SQL in a C, FORTRAN, COBOL or PASCAL program.

Oracle can be invoked within a user program in one of two ways: using the so-called Precompiler or referring to the OCI's (Oracle Call Interface) library. We briefly describe both concepts here but we eventually did all our tests using the Precompilers (PRO*C) method.

2.2 Precompiler

The Oracle Precompiler is a programming tool that allow the user to embed SQL statements in a high-level source program. The Precompiler accepts the source program as input, translates the embedded SQL statements into standard ORACLE runtime calls, and generates a modified source program which can then be compiled, linked and executed in the usual way.

Precompilers are implemented for C and FORTRAN, among other languages.We used the PRO*C precompiler to write C programs to interact with the Oracle database.

2.3 OCI library

The other way to invoke ORACLE within a program is using the OCIs libraries. They are subroutine libraries that allow applications written in C or FORTRAN to access and manipulate data stored in ORACLE database via SQL statements so that it is possible to perform all SQL queries using a high-level program simply by calling different subroutines performing different tasks.

OCIs libraries allow the user to better follow and control what the program is doing, which is more hidden in a Precompiler program ,where all data areas are automatically defined by embedded SQL statements. But, on the other side, Precompilers are easier to use.

2.4 Tests performed

We created a test database to emulate a 'real' high voltage calibration database, i.e. a database suitable to store detector's high voltage parameters, and a set of programs to insert, retrieve and update its records. The basic HV table we used is illustrated in Appendix A and examples of PRO*C programs can be found in Appendix B.

We then ran speed tests in order to evaluate how long it takes to perform such tasks like inserting, selecting and retrieving, and updating a large amount of records.

The timing tests were performed on fsui01, a SunOS running Solaris 5.4 in the **fnalu** cluster.

The results are summarized in Table 1.

TABLE 1. Timing results for Oracle

insert	490 sec/50K records	10 msec/record
update	170 sec/5K records	34 msec/record
select	3 sec/10K records	.3 msec/record

Please note here that:

- the update test was performed in two step:

  first we ‘expired’ the records, changing their expiration date (because we want to keep all the records in the database, since their first insertion, and also keep track of the whole history of the database); this took 78secs, i.e. ~16 msec/record

  then we updated the records with new values and a NULL expiration date, which identifies the records as current; this took 92secs, i.e. ~18msec/record

- also note here that we reached these timings WHEN an INDEX (which provides direct access to rows in a table) was created on the database: with NO INDEX it took ~3 sec to update a record, which means about 4 hours to update the 5K records.

  Rules on how to create an INDEX can be found on the manuals : we just note here that an INDEX will most likely speed up performances if the INDEX is created using a column not affected by the SQL command (insert, update, delete etc) otherwise it may instead slow down the process because Oracle must maintain both the index data as well as the table data.

We also ran subsequent updates of 1000 records each, up to 70K recorsd, to see if the updating time was in anyway affected by the database’s size: we found that, if the database is indexed, the database size does not affect the updating time which resulted to be practically the same (20 msec/record) either when updating 1000 records in a 10K records database as well as updating 1000 records in a 60K records database.

As you would expect though, the updating time is affected by the database’s size if the database is not indexed, since Oracle has to read each record to find the correct one to update: we found that the updating time increased of about 40 seconds for every 1000 records added to the database.

2.5 Data Integrity

Oracle allows you to define and enforce several types of integrity contraints to prevent the entry of invalid information into the database tables. All the informations about these constraints and how to enforce and use them can be found in several Oracle manuals, and in particular in the ‘Data Integrity’ chapter in the ‘Oracle7 Server Concept Manual’. We would like here to add a few words about unique and non-unique indexes.

A UNIQUE index ensures that the table will never contain two rows with the same values in all index columns.

A UNIQUE index (and a NOT NULL constraint on the columns) is automatically created by Oracle when you use a primary key. This means that a primary key:

- does not allow duplicated value in the column
- does not allow NULLS in the colum
- indexes the column (or columns) used in defining the primary key

In addition to the unique index, Oracle also allows the creation of a NON-UNIQUE index. While unique indexes guarantee that no two rows of a table have duplicate values in the column, non-unique indexes do not impose this restriction. And NULLS are allowed on a non-unique index unless the NOT NULL constraints is explicitly set.

2.6 Triggers

A trigger is a stored PL/SQL procedure that is associated with a table and that is automatically executed by Oracle when a specified SQL statement (INSERT,UPDATE oe DELETE) is issued against the table. For instance, you can define a trigger to be fired each time a table is updated to 'copy' the new(old) values in a secondary table which can be more easily and quickly queried.

2.7 Comments

We did not pursue multi-user/data-integrity checks for Oracle, as done for mSQL, with the hope that, being a commercial and heavily used database, such problems would have long since been fixed.

Infact, Oracle is an amazing product: it offers to the users several useful tools to build even very complicated queries, plus data integrity checks and a full support . Tools to build and display reports are also available (SQL*FORMS and SQL*REPORTWRITER). A Tcl inteface (OraTcl) allows to build customized menus and displays. Last, but not least, the manuals, although many, are very well written and the examples are clear and working.

2.8 How to get Oracle

Oracle is used and supported at Fermilab from the Computing Division DBI Department.Details on the number of licences available at Fermilab and their cost can be obtained from them.

3 mSQL

3.1 Overview

mSQL is an SQL based database designed to provide a fast access to stored data with low memory requirement. The name implies (mSQL stands for miniSQL), it offers only a subset of SQL as its query interface but everything it supports is in compliance with the ANSI SQL specification.

The mSQL package includes the database engine, a terminal ‘monitor’ program, a database administration program, a schema viewer and a C language API . The API and the database engine have been designed to work in the client/server environment over a TCP/IP network.

Access to mSQL from script languages is also possible. Currently the following languages are supported: Perl 5, Phyton and Tcl.

3.2 mSQL specification

The mSQL language offers a subset of the features provided by ANSI SQL. It allows a program or user to store, manipulate and retrieve data in table structures. It does not support relational capabilities as views or nested queries, as Oracle does, but it does provide the capability of “joins” between multiple tables.

The available data types are only three: char(len), int and real, which could be a limitation (for example Oracle supports seven primary data types and several versions of these).

mSQL provides a terminal ‘monitor’program that allows a user to interactively submit queries to the database engine: this program is simply called ‘msql’. More informations about interactive queries and more, can be found in ref [4], the mSQL user guide.

3.3 C programming API

The mSQL API library, libmsql.a, is included in the package. It allows a C program to communicate with the database engine. In this way, a user can write a C program to perform complicates queries to the database engine and retrieve and manipulate large amounts of data without interactively connecting to the database.

3.4 Tests performed

We created a calibration database very similar to the one used for the Oracle tests(see Table 2 in Appendix A) and wrote a set of C programs to insert, retrieve and update its records. Examples of code are provided in appendix C1-3.

The results of our tests are summarized in Table 2.

TABLE 2. Timing results for mSQL

insert	73 sec/20K records	3.6 msec/record
update	300 sec/1K records	300 msec/record
select	5 sec/10K records	.5 msec/record

Please note here that the update record was done in the same way we did for Oracle BUT that the mSQL database WAS NOT indexed.

We also performed extensive load and robusteness tests. The results of these tests are described in details in Appendix E.

Briefly, up to 500K records were loaded in a very similar HV database and simultaneous updates/ retrieval of records were ran simulating a multiusers scenario.The updating jobs were scheduled to the server in such a way that 3 to 10 jobs were accessing and updating the database almost simultaneously. At the end of these updates, the database was checked throuroghly and no data corruption was found.

The updating time was indeed found to grow almost linearly with the database size .

The retrieval time of one record out from the 500K was negligible; the retrieval time for a thousands records from the same size database is of the order of few seconds.

Running over TCP/IP seems to add a 20% more time with respect of running on the node where the database is.

3.5 Comments

The timing performances look very impressive; the fact that the timing performance is database size dependent is indeed a limitation, although the timings found should not be critical for most applications.

mSQL is very easy to use, in terms of code writing. The new release version (1_0_7) seems to have fixed a few problems we experienced with the server in the previous version 1_0_5. Also the fact that you can look at, and modify the code will undoubtely facilitate the product maintenance and help to trace possible bugs.

3.6 How to get mSQL

The current version of mSQL can be found via anonimous ftp from

host: Bond.edu.au(131.244.1.1)

Path: /pub/Minerva/msql

User contributed code can also be found in /pub/Minerva/msql/contrib.A monthly archive of the mailing list is also available in /pub/Minerva/msql/mail-archive.

4 Porting mSQL databases to Oracle and vice-versa

We are testing the possibility of porting databases from mSQL to Oracle and back to mSQL to assure we could do so in case of need.

This can be done using some mSQL and Oracle facilities plus some ‘custom’ made utilities as follows:

- dump the mSQL database content to an ASCII file using the mSQL facility *msqldump*
- read the ASCII file with a C program or with Oracle SQL*LOADER and load the data into an Oracle Table
- dump the Oracle table content to an ASCII file with some custom made SQL*PLUS scripts
- read the ASCII file with a C program and load the data into an mSQL table

5 HEPDB

5.1 Overview

HEPDB has been developed at CERN and it is based on the ZEBRA system and relies heavily on the Random Access I/O package RZ and others (see ref.[5]). RZ files are organized in a hierarchical manner, much like the Unix file system.Thus, data is accessed or stored using a Unix-like **pathname** plus one or more identifiers known as **keys**.

A few general standard keys have been defined and HEPDB assumes that all key vectors consist of at least 10 keys, divided in system and user keys. The system keys should not be touched by the user or application program. The user keys may contain any information. After the so called ‘special user keys’ come a number of key pairs. The number of pair is a database constant and it has to be defined when the database is created. Additional keys can be declared by the user (within an overall limit of 100) and be used later to select the data to be retrieved.

In HEPDB, data is stored in one or more ZEBRA RZ files.Typically, one might use separate directories from the different detector components and, for any given detector, separate subdirectories could be used for the various types of information that are to be stored for that component.

When the data is submitted for inclusion in the database, the pathname of the directory where the data has to reside and the key vector must be supplied.

In a HEPDB database it is possible to store-retrieve integer, real, character data and ASCII text files.

5.2 The database server

Fundamental to the operation of HEPDB is the concept of database server. The normal way of updating a HEPDB database is via a dedicated database server. The requests of changes are not applied directly. Instead they are written into a queue and then sent to the database.

All the informations about how to write and install your own server are in the manual [5]: the server looks pretty robust.

5.3 Tests performed

We wrote a FORTRAN program which updates and retrieves data from an HEPDB database and ran some tests. We had some problems in dealing with character data vectors (like channel names) and we did not succeed in using them.

An example of source code which updates and retrieves integer data to-from HEPDB banks can be found in Appendix D. It contains useful informations about how to correctly define the PAWC common block and how to map the ZEBRA banks.

5.4 Comments

Unfortunately, within the time allotted for this project, we could not perform extensive tests as done for Oracle and mSQL. However, KTev people working on HEPDB are satisfied with the timing performances.

Indeed HEPDB seems to be well suited to the storage and retrieval of detector geometry and calibration information. It provides the user with a selection of FORTRAN callable subroutines for the storage and retrieval of ZEBRA data structures, FORTRAN vectors and text files.

Beeing accessible from a FORTRAN program, makes HEPDB very appealing for updating already existing analysis packages which can easily include HEPDB calls and store data into the calibration database or retrieve data into common blocks and vectors for subsequent analysis.

But you really have to be familiar with the ZEBRA package, terminology and rules, if you want to investigate in a little more detail what it's done also because the HEPDB manual itself is not sufficient in many examples and explanations.

Some concern may also rise about a long term support from CERN.

5.5 How to get HEPDB

HEPDB package and executable come with the CERN library. It is possible to get the manual (hepdb.ps.Z) via anonymous ftp from asisftp.cern.ch in the directory cernlib/pub/ps.dir .

You can also get the manual from the "CN/ASD Group Documentation" WEB Page at the following URL : http://wwwcn.cern.ch/asdoc/Welcome.html .

Acknowlegdment

We wish to tank Matteo Boschini, I.N.F.N. Milan (Italy) for his useful suggestions and examples of ORACLE VMS Precompilers programs; the Computing Division's Database and Information Group, Fermilab, and in particular Shelly Malinski, for their help with Oracle; Joel Butler, Fermilab, for useful discussions about mSQL and Ed Blucher, KTev, for his help with HEPDB.

References

[1] G.Koch," ORACLE, the complete Reference", McGraw-Hill, first (1991) and third (1995) editions.

[2] ORACLE 7, various manuals; in particular "Programmer's Guide to the Oracle Precompilers" and "PRO*C Supplement to the Oracle Precompilers Guide", version 1.5 and higher ; "Oracle 7 Server Utilities User's Guide", "Oracle 7 Server SQL Language Reference Manual", "SQL*Plus User's Guide and Reference".

[3] M.Boschini, "Report on tests of the ORACLE DataBase", internal memo of E831 collaboration, Oct. 1994.

[4] mSQL Reference Manual, available from the tar file.

[5] HEPDB Reference Manual, CERN Long Writeup Q180 and references therein.

Appendix A: HV table(s) used in the tests

A.1 Oracle

A.1.1 HVD1 table

```
create table HVD1 (
chname        CHAR(12) NOT NULL,
chno          NUMBER   NOT NULL,
mf            NUMBER(3),
slot          NUMBER(3),
spigot        NUMBER(6),
type          CHAR(6),
maxv          NUMBER   NOT NULL,
setting       NUMBER,
tolerance     NUMBER,
addr_cre_date CHAR(20),
addr_exp_date CHAR(20),
par_cre_date  CHAR(20),
par_exp_date  CHAR(20)
);
```

A.1.2 Dump of some records from the database using SQL*PLUS interactive commands:

<fsui01> sqlplus

SQL*Plus: Release 3.1.3.7.1 - Production on Wed Aug 16 13:06:06 1995

Enter user-name: dart

Enter password:

Connected to:

Oracle7 Server Release 7.1.6.2.0 - Production Release

With the distributed, replication and parallel query options

PL/SQL Release 2.1.6.2.0 - Production

```
SQL> select count(*) from hvd1;

  COUNT(*)
----------
     53000

SQL> select chname,chno,addr_cre_date,addr_exp_date from hvd1
  2  where chname = 'C3-000020';

CHNAME             CHNO ADDR_CRE_DATE        ADDR_EXP_DATE
------------ ---------- -------------------- --------------------
C3-000020             3 16-AUG-95 11:14:51   16-AUG-95 13:02:35
C3-000020             3 16-AUG-95 11:42:35   16-AUG-95 13:02:35
C3-000020             3 16-AUG-95 13:03:06

SQL> SQL> select chname,chno,addr_cre_date,addr_exp_date from hvd1
  2  where chname like 'C3-%111%';

CHNAME             CHNO ADDR_CRE_DATE        ADDR_EXP_DATE
------------ ---------- -------------------- --------------------
C3-000111            14 16-AUG-95 11:14:51   16-AUG-95 13:02:39
C3-000111            14 16-AUG-95 11:42:36   16-AUG-95 13:02:39
C3-000111            14 16-AUG-95 13:03:08
C3-001110             5 16-AUG-95 11:15:01
C3-001111             6 16-AUG-95 11:15:01
C3-001112             7 16-AUG-95 11:15:01
C3-001113             8 16-AUG-95 11:15:01
C3-001114             9 16-AUG-95 11:15:01
C3-001115            10 16-AUG-95 11:15:01
C3-001116            11 16-AUG-95 11:15:01
C3-001117            12 16-AUG-95 11:15:01

CHNAME             CHNO ADDR_CRE_DATE        ADDR_EXP_DATE
------------ ---------- -------------------- --------------------
C3-001118            13 16-AUG-95 11:15:01
C3-001119            14 16-AUG-95 11:15:01
C3-002111            14 16-AUG-95 11:15:08
C3-003111             6 16-AUG-95 11:15:17
C3-004111            14 16-AUG-95 11:15:32
```

C3-005111 6 16-AUG-95 11:15:39

C3-006111 14 16-AUG-95 11:15:47

C3-007111 6 16-AUG-95 11:15:55

C3-008111 14 16-AUG-95 11:16:03

C3-009111 6 16-AUG-95 11:16:11

21 rows selected.

SQL> exit

Disconnected from Oracle7 Server Release 7.1.6.2.0 - Production Release

With the distributed, replication and parallel query options

PL/SQL Release 2.1.6.2.0 - Production

<fsui01>

A.2 mSQL

A.2.1 HV table

```
CREATE TABLE hv (
  chname CHAR(32) NOT NULL,
  chno INT,
  mf INT,
  slot INT,
  spigot INT,
  type CHAR(8),
  maxv INT,
  setting INT,
  tolerance INT,
  addr_cre_date INT,
  addr_exp_date INT,
  par_cre_date INT,
  par_exp_date INT
) \g
```

A.2.2 Dump of some records from the database using the msql ‘monitor’ facility:

<da831> msql testd

Welcome to the miniSQL monitor. Type \h for help.

```
mSQL > select * from hv where chname like 'C4%'\g

Query OK.

5000 rows matched.

 +---------------------------------+----------+----------+----------+----------+----------+----------+----------+---
--------+---------------+---------------+--------------+--------------+
 | chname                          | chno     | mf       | slot     | spigot   | type     | maxv     | setting  | to
lerance | addr_cre_date | addr_exp_date | par_cre_date | par_exp_date |
 +---------------------------------+----------+----------+----------+----------+----------+----------+----------+---
--------+---------------+---------------+--------------+--------------+
 | C4-001000                       | 7        | 4        | 14       | 7        |          | 2500     | 2100     |
      | 808505691     | 2000000000    | 808505691    | 2000000000   |
 | C4-000999                       | 6        | 4        | 14       | 6        |          | 2500     | 2100     |
      | 808505691     | 2000000000    | 808505691    | 2000000000   |
 | C4-000998

etc etc

mSQL > select chname,chno,addr_cre_date,addr_exp_date from hv where chname = 'C4-002000'\g

Query OK.

1 rows matched.

 +---------------------------------+----------+---------------+---------------+
 | chname                          | chno     | addr_cre_date | addr_exp_date |
 +---------------------------------+----------+---------------+---------------+
 | C4-002000                       | 15       | 808510618     | 2000000000    |
+---------------------------------+----------+---------------+---------------+


mSQL > \q

Bye!
<da831>
```

Appendix B: examples of Oracle programs

B.1 insert records

```
/*                    ora_insert.pc                                         *
*                                                                           *
*              Author: D.Torretta, M.J. Boschini       *
*                                              *
*  This program is meant to update an Oracle table in a "batch" mode,       *
*  i.e. without interactive Oracle-login.It also evaluates elapsed real     *
*  time between operations.                                                 *
*                                              */

#include <stdio.h>
#include <time.h>
/*                                                                         */
EXEC SQL BEGIN DECLARE SECTION;
  VARCHAR username[8];
  VARCHAR password[10];
  char chname_arr[1000][12];    /* [12] [100 does ONLY 12 records */
  int chno_arr[1000];
  int mf_arr[1000],slot_arr[1000],spigot_arr[1000];
  int setting_arr[1000],maxv_arr[1000],tolerance_arr[1000];
  char hv_chname[12];
  int hv_chno;
  int hv_mf,hv_slot,hv_spigot;
  int hv_setting,hv_maxv,hv_tolerance;
EXEC SQL END DECLARE SECTION;

/* copy in the SQL Communications Area. This command automatically defines all
   data areas needed by the program to login to ORACLE and to pass data to
   and from the database.                                                   */
EXEC SQL INCLUDE sqlca;


main ()
{
  int i,l;
  int nrow;
  int idebug = 0;

  int nrec,nchan,ichan;
  char ins_tmp[500];

  time_t tempo;

  char msg_buf[100];
  int  msg_len;
  int  buf_size=100;
```

```
/*Define username and password and the lengths of these strings. */

  strcpy(username.arr, "torretta");
  username.len= strlen(username.arr);
  strcpy(password.arr, "yyyyyy");
  password.len= strlen(password.arr);

/*Define error exit */

  EXEC SQL WHENEVER SQLERROR GOTO error;
  tempo = time(0);
  printf(" %s \n",ctime(&tempo));

/*Connect to Oracle */

  EXEC SQL CONNECT :username IDENTIFIED BY :password;
  tempo = time(0);
  printf(" %s \n",ctime(&tempo));

  printf("\n Connect to ORACLE as user: %s",username.arr);

  tempo = time(0);
  printf("\n Start Insert %s \n",ctime(&tempo));

/*The following lines define the SQL operation to be performed. */

  nrec  = 50000;
  nchan = 10000;

  for (i=0; i<= nrec-1; i++) {

   ichan=i%nchan;
   strcpy ( hv_chname,"C1-");
   if( i >= 10001)  strcpy ( hv_chname,"C2-");
   if( i >= 20001)  strcpy ( hv_chname,"C3-");
   if( i >= 30001)  strcpy ( hv_chname,"C4-");
   if( i >= 40001)  strcpy ( hv_chname,"C5-");
   sprintf( ins_tmp,"%06d",ichan+1);
   strcat ( hv_chname,ins_tmp);
   if(idebug == 1) printf("\n hv_chname = %s \n",hv_chname);

   hv_chno   = ichan%16;
   hv_mf     = ichan/256+1;
   hv_slot   = (ichan/16)%16;
   hv_spigot = ichan%16;
   hv_maxv   = 2900;
   hv_setting= 2100;
   if( i >= 10001) hv_setting=2200;
   if( i >= 20001) hv_setting=2300;
   if( i >= 30001) hv_setting=2400;
   if( i >= 40001) hv_setting=2500;

   if( idebug == 1){
   printf("\n before SQL INSERT:\n");
   printf("\n chname chno mf slot spigot setting = %s %d %d %d %d %d %d \n",
             hv_chname,hv_chno,hv_mf,hv_slot,hv_spigot,hv_maxv,hv_setting);
   }
   EXEC SQL INSERT INTO HVD1 (CHNAME,CHNO,MF,SLOT,SPIGOT,MAXV,SETTING,
```

```
    ADDR_CRE_DATE,PAR_CRE_DATE)
    VALUES (:hv_chname,:hv_chno,:hv_mf,:hv_slot,:hv_spigot,:hv_maxv,:hv_setting,
            to_char(sysdate,'dd-MON-yy hh24:mi:ss'),
            to_char(sysdate,'dd-MON-yy hh24:mi:ss'));

    if(idebug == 1) printf("\n after SQL INSERT:\n");
    }

   tempo = time(0);
   printf("\n End Insert %s \n",ctime(&tempo));


 /*Now read back some values */

 /*Declare cursor for SQL query */

   EXEC SQL DECLARE hvd_cursor CURSOR FOR
            SELECT CHNAME,CHNO,MF,SLOT,SPIGOT,MAXV,SETTING
        FROM HVD1 WHERE CHNAME LIKE 'C5%';


 /*Open the cursor */

   EXEC SQL OPEN hvd_cursor;

 /*Define branch when data is exhausted */

   EXEC SQL WHENEVER NOT FOUND GOTO done;

 /*Loop to retrieved data from insert statement: ONE FETCH
   will retrieve ALL records inserted*/

 /*  Place data into host variables */

   EXEC SQL FETCH hvd_cursor INTO :chname_arr,:chno_arr,:mf_arr,
                  :slot_arr,:spigot_arr,:maxv_arr,:setting_arr;


 /*Successful execution print count of rows retrieved */

 done:
   printf("\n\n sqlca.sqlerrd[2]= %d \n", sqlca.sqlerrd[2]);
   nrow= sqlca.sqlerrd[2];
   printf("\n %d records processed",nrow);

   if (idebug == 0){
   for (i=0 ;i<nrow ; i++){

     /* Null-terminate the VARCHAR */
     /*chname_arr[i].arr[chname_arr[i].len] = '\0';
     printf("\n %d  %s %d %d %d %d %d %d",i,chname_arr[i].arr,chno_arr[i], */
     printf("\n %d  %s %d %d %d %d %d %d",i,chname_arr[i],chno_arr[i],
            mf_arr[i],slot_arr[i],spigot_arr[i],maxv_arr[i],setting_arr[i]);

    }
   }

 /*Close cursor */
 /*Commit what done */
```

```
  EXEC SQL CLOSE hvd_cursor;
  EXEC SQL COMMIT RELEASE;


/*Here the program makes use of the error handling structures of ORACLE.
  The variable sqlca.sqlerrd is not a typical PRO*C one, but is taken from
  the precompiled version of the same program, thus belongs to ORACLE's
  primitive library interface.                                               */

  printf("\n\n sqlca.sqlerrd[2]= %d \n\n", sqlca.sqlerrd[2]);

  exit(0);

/* Error branch */

error:

  printf("\n\n %.70s \n\n", sqlca.sqlerrm.sqlerrmc);
  sqlglm(msg_buf, &buf_size , &msg_len);
  msg_buf[msg_len] = '\0';
  printf("\n %s \n", msg_buf);
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL ROLLBACK RELEASE;
  exit(1);

}
```

B.2 update records

```
/*                     ora_update.pc                                          *
*                                                                             *
*              Author: D.Torretta,M.J. Boschini        *
*                                             *
*  This program is meant to update an Oracle table in a "batch" mode,         *
*  i.e. without interactive Oracle-login.It also evaluates elapsed real       *
*  time between operations.                                                   *
*                                             */

#include <stdio.h>
#include <time.h>

EXEC SQL BEGIN DECLARE SECTION;
  VARCHAR username[8];
  VARCHAR password[10];
  char chname_arr[1000][12];    /* [12] [100 does ONLY 12 records */
  int chno_arr[1000];
  int mf_arr[1000],slot_arr[1000],spigot_arr[1000];
  int setting_arr[1000],maxv_arr[1000],tolerance_arr[1000];
  char addr_cre_date_arr[1000][20];
  char addr_exp_date_arr[1000][20];
  char par_cre_date_arr[1000][20];
  char par_exp_date_arr[1000][20];
  char hv_chname[12];
  int hv_chno;
```

```
  int hv_mf,hv_slot,hv_spigot;
  int hv_setting,hv_maxv,hv_tolerance;
EXEC SQL END DECLARE SECTION;

/* copy in the SQL Communications Area. This command automatically defines all
   data areas needed by the program to login to ORACLE and to pass data to
   and from the database.                                                     */
EXEC SQL INCLUDE sqlca;

main ()
{
  int i;
  int l,nrow;
  int idebug = 0;

  int nrec,nchan,ichan;
  char ins_tmp[500];

  time_t tempo;

  char msg_buf[100];
  int  msg_len;
  int  buf_size=100;

/*Define username and password and the lengths of these strings. */

  strcpy(username.arr, "torretta");
  username.len= strlen(username.arr);
  strcpy(password.arr, "yyyyyy");
  password.len= strlen(password.arr);

/*Define error exit */

  EXEC SQL WHENEVER SQLERROR GOTO error;
  tempo = time(0);
  printf(" %s \n",ctime(&tempo));

/*Connect to Oracle */

  EXEC SQL CONNECT :username IDENTIFIED BY :password;
  tempo = time(0);
  printf(" %s \n",ctime(&tempo));

  printf("\n Connect to ORACLE as user: %s",username.arr);

/*The following lines define the SQL operation to be performed. */

  nrec  = 5000;
  nchan = 10000;

/*  first EXPIRE records; ie. UPDATE the nrec records setting EXPIRATION DATE  */

  tempo = time(0);
  printf("\n Start Expire %s \n",ctime(&tempo));

  for (i=0; i<= nrec-1; i++) {

   ichan=i%nchan;
   strcpy ( hv_chname,"C5-");
```

```
    sprintf( ins_tmp,"%06d",ichan+1);
    strcat ( hv_chname,ins_tmp);
    if(idebug == 1) printf("\n hv_chname = %s \n",hv_chname);

    if( idebug == 1){
    printf("\n before SQL UPDATE :\n");
    printf("\n chname chno mf slot spigot setting = %s %d %d %d %d %d %d \n",
               hv_chname,hv_chno,hv_mf,hv_slot,hv_spigot,hv_maxv,hv_setting);
    }
    EXEC SQL UPDATE HVD1 SET
             ADDR_EXP_DATE= to_char(sysdate,'dd-MON-yy hh24:mi:ss'),
             PAR_EXP_DATE=to_char(sysdate,'dd-MON-yy hh24:mi:ss')
             WHERE CHNAME = :hv_chname;

    /*
    IMPORTANT: if you are going to manipulate a large number of records and you
    DO NOT COMMIT every reasonable amount, you may fill out the Rollback segment and
    your job will crash.
    To be safe, we did COMMIT every 100 records */

    if( (i/100)*100 == i){
     printf("\n COMMIT called after %d records :\n",i);
     EXEC SQL COMMIT ;
     }
    }

 /* example of update for all values:
    EXEC SQL UPDATE HVD SET CHNAME=:hv_chname,CHNO=:hv_chno,

 MF=:hv_mf,SLOT=:hv_slot,SPIGOT=:hv_spigot,MAXV=:hv_maxv,SETTING=:hv_setting,
             ADDR_CRE_DATE= to_char(sysdate,'dd-MON-yy hh24:mi:ss'),
             PAR_CRE_DATE=to_char(sysdate,'dd-MON-yy hh24:mi:ss')
             WHERE CHNAME LIKE :hv_chname;
    example with fixed values :
    EXEC SQL UPDATE HVD SET CHNAME='C1-000777',CHNO=77,
             MF=2,SLOT=2,SPIGOT=777,MAXV=2700,SETTING=2077,
             ADDR_CRE_DATE= to_char(sysdate,'dd-MON-yy hh24:mi:ss'),
             PAR_CRE_DATE=to_char(sysdate,'dd-MON-yy hh24:mi:ss')
             WHERE CHNAME LIKE 'C1-00088%';
 */



   EXEC SQL COMMIT ;
   tempo = time(0);
   printf("\n End Expire %s \n",ctime(&tempo));

 /*  second: UPDATE records: ie. INSERT again the records with new values and
                             NO EXPIRATION DATE  */

   tempo = time(0);
   printf("\n Start Insert %s \n",ctime(&tempo));

   for (i=0; i<= nrec-1; i++) {

    ichan=i%nchan;
    strcpy ( hv_chname,"C5-");
    sprintf( ins_tmp,"%06d",ichan+1);
    strcat ( hv_chname,ins_tmp);
```

```
    if(idebug == 1) printf("\n hv_chname = %s \n",hv_chname);

    hv_chno   = ichan%16;
    hv_mf     = ichan/256+1;
    hv_slot   = (ichan/16)%16;
    hv_spigot = ichan%16;
    hv_maxv   = 4444;
    hv_setting= 2444;

    EXEC SQL INSERT INTO HVD1 (CHNAME,CHNO,MF,SLOT,SPIGOT,MAXV,SETTING,
    ADDR_CRE_DATE,PAR_CRE_DATE)
    VALUES (:hv_chname,:hv_chno,:hv_mf,:hv_slot,:hv_spigot,:hv_maxv,:hv_setting,
            to_char(sysdate,'dd-MON-yy hh24:mi:ss'),
            to_char(sysdate,'dd-MON-yy hh24:mi:ss'));

    /* again, we COMMIT every 100 records */
    if( (i/100)* 100 == i){
     printf("\n COMMIT called after %d records :\n",i);
     EXEC SQL COMMIT ;
     }

    }


   EXEC SQL COMMIT ;
   tempo = time(0);
   printf("\n End Insert %s \n",ctime(&tempo));

 /*Now read back some values */

 /*Declare cursor for SQL query */

   EXEC SQL DECLARE hvd_cursor CURSOR FOR
            SELECT CHNAME,CHNO,MF,SLOT,SPIGOT,MAXV,SETTING,
            ADDR_CRE_DATE,PAR_CRE_DATE
        FROM HVD1 WHERE CHNAME LIKE 'C5%';


 /*Open the cursor */

   EXEC SQL OPEN hvd_cursor;

 /*Define branch when data is exhausted */

   EXEC SQL WHENEVER NOT FOUND GOTO done;

 /*Loop to retrieved data from insert statement: each SQL FETCH call will retrieve
   as many records as the host variables dimension: here, it will retrieve 1000 records.
   If you don't know how many records your FETCH is going to return, you can FETCH the
   records in batches with a loop  like:
     for ( i=0; ; i++) {
     EXEC SQL FETCH hvd_cursor INTO aa,bb,cc ;
     print here the rows returned ;
     }
   See ora_select.pc for more details */

 /*  Place data into host variables */
     EXEC SQL FETCH hvd_cursor INTO :chname_arr,:chno_arr,:mf_arr,
                    :slot_arr,:spigot_arr,:maxv_arr,:setting_arr,
```

```
                   :addr_cre_date_arr,:addr_exp_date_arr;

/*Successful execution print count of rows retrieved */

done:
  printf("\n\n sqlca.sqlerrd[2]= %d \n", sqlca.sqlerrd[2]);
  nrow= sqlca.sqlerrd[2];
  printf("\n %d records processed",nrow);

  if (idebug == 0){
  for (i=0 ;i<nrow ; i++){

    /* Null-terminate the VARCHAR
    chname_arr[i].arr[chname_arr[i].len] = '\0'; */
    printf("\n %d %s %d %d %d %d %d %d %s %s",i,chname_arr[i],chno_arr[i],
           mf_arr[i],slot_arr[i],spigot_arr[i],maxv_arr[i],setting_arr[i],
           addr_cre_date_arr[i],addr_exp_date_arr[i]);

   }
  }

/*Close cursor */
/*Commit what done */

  EXEC SQL CLOSE hvd_cursor;
  EXEC SQL COMMIT RELEASE;


/*Here the program makes use of the error handling structures of ORACLE.
  The variable sqlca.sqlerrd is not a typical PRO*C one, but is taken from
  the precompiled version of the same program, thus belongs to ORACLE's
  primitive library interface.                                              */

  printf("\n\n sqlca.sqlerrd[2]= %d \n\n", sqlca.sqlerrd[2]);

  exit(0);

/* Error branch */

error:

  printf("\n\n %.70s \n\n", sqlca.sqlerrm.sqlerrmc);
  sqlglm(msg_buf, &buf_size , &msg_len);
  msg_buf[msg_len] = '\0';
  printf("\n %s \n", msg_buf);
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL ROLLBACK RELEASE;
  exit(1);

}
```

B.3

```
/*                     ora_select.pc                                    *
*                                                                       *
*              Author: D.Torretta (Matteo J. Boschini)       *
*                                            *
*  This program is meant to retrieve records from an Oracle table in a        *
*  "batch" mode,i.e. without interactive Oracle-login. It also evaluates      *
*  elapsed real time between operations.                                *
*                                            */

#include <stdio.h>
#include <time.h>
/*                                 */
EXEC SQL BEGIN DECLARE SECTION;
  VARCHAR username[8];
  VARCHAR password[10];
  char hv_chname_arr[1000][12];    /* [12] [100 does ONLY 12 records */
  int hv_chno_arr[1000];
  int hv_mf_arr[1000],hv_slot_arr[1000],hv_spigot_arr[1000];
  int hv_setting_arr[1000],hv_maxv_arr[1000],hv_tolerance_arr[1000];
  char hv_chname[12];
  int hv_chno;
  int hv_mf,hv_slot,hv_spigot;
  int hv_setting,hv_maxv,hv_tolerance;
EXEC SQL END DECLARE SECTION;

/* copy in the SQL Communications Area. This command automatically defines all
   data areas needed by the program to login to ORACLE and to pass data to
   and from the database.                                               */
EXEC SQL INCLUDE sqlca;

void print_rows();

FILE *fd;
int ifile;

main (argc,argv)
int argc; char *argv[];
{
  int i,l;
  int idebug = 0;
  int nrows=0;
  int num_ret=0;

  char filename[256];

  time_t tempo;

  char msg_buf[100];
  int  msg_len;
  int  buf_size=100;

if ( argc <2  ){
    printf ( " Dumping records on screen \n");
    ifile = 0;
   }
```

```
if ( argc >= 2 ) {
    sscanf ( argv[1] , "%s" , &filename ) ;
    printf ( " Dumping records in %s\n", filename ) ;

    if((fd = fopen(filename,"w"))==NULL)
    {
      printf("Cannot open file\n");
      exit(-1);
    }
    ifile = 1;
   }

/*Define username and password and the lengths of these strings. */

  strcpy(username.arr, "torretta");
  username.len= strlen(username.arr);
  strcpy(password.arr, "yyyyyyy");
  password.len= strlen(password.arr);

/*Define error exit */

  EXEC SQL WHENEVER SQLERROR GOTO error;
  tempo = time(0);
  printf(" %s \n",ctime(&tempo));

/*Connect to Oracle */

  EXEC SQL CONNECT :username IDENTIFIED BY :password;
  tempo = time(0);
  printf(" %s \n",ctime(&tempo));

  printf("\n Connect to ORACLE as user: %s",username.arr);

/*  Declare a cursor for the SELECT query */

  EXEC SQL DECLARE sel_cursor CURSOR FOR
           SELECT CHNAME,CHNO,MF,SLOT,SPIGOT,SETTING
           FROM HVD1
           WHERE CHNAME LIKE 'C5%';

           /* other examples of WHERE clause:
           WHERE CHNAME = 'C5-000555';
           WHERE SETTING = 2200;
           ****  NO WHERE clause set means SELECT ALL records from the DB ***
           */


/* Open cursor */

   EXEC SQL OPEN sel_cursor;

/* exit if the last row has been fetched */

   EXEC SQL WHENEVER NOT FOUND GOTO done;

   tempo = time(0);
   printf("\n Start Fetching records %s \n",ctime(&tempo));

/* fetch and process data in loop */
```

```
   for ( i=0 ; ;i++ ) {

/*  Place data into host variables */

    printf("\n *****  FETCH *****\n");
    EXEC SQL FETCH sel_cursor INTO :hv_chname_arr,:hv_chno_arr,:hv_mf_arr,
    :hv_slot_arr,:hv_spigot_arr,:hv_setting_arr;

    print_rows(sqlca.sqlerrd[2] - num_ret);
    num_ret = sqlca.sqlerrd[2] ;

    }



/*  Successful execution print count of rows retrieved */

done:
    tempo = time(0);
    printf("\n End Fetching records %s \n",ctime(&tempo));

    /*Print remaining rows from last FETCH if any */
    if((sqlca.sqlerrd[2] - num_ret)>0)
      print_rows(sqlca.sqlerrd[2] - num_ret);

    printf("\n %d rows(records) processed\n",sqlca.sqlerrd[2]);
    printf("\n sqlca.sqlerrd[2], sqlca.sqlerrd[3] %d %d\n",sqlca.sqlerrd[2],
                     sqlca.sqlerrd[3]);
    nrows = nrows + sqlca.sqlerrd[2];
    printf("\n %d rows(records) processed\n",nrows);


    printf("\n");
    tempo = time(0);
    printf(" %s \n",ctime(&tempo));

    fclose(fd);

/*  Close cursor */
/*  Close ORACLE session */

  EXEC SQL CLOSE sel_cursor;
  EXEC SQL COMMIT RELEASE;

  exit(0);

/* Error branch */

error:

  printf("\n\n %.70s \n\n", sqlca.sqlerrm.sqlerrmc);
  sqlglm(msg_buf, &buf_size , &msg_len);
  msg_buf[msg_len] = '\0';
  printf("\n %s \n", msg_buf);
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL ROLLBACK RELEASE;
  exit(1);
```

```
}


void print_rows(n)
{
  int i,l;

  if( ifile == 0) printf("\n *** Print %d row_fetched ***\n", n);
  if( ifile == 1) fprintf(fd,"\n *** Print %d row_fetched ***\n", n);

/*  dump results into a file */

  for(i=0;i<n ;i+=3)
   {
    if( ifile == 0) printf("\n", n);
    if( ifile == 1) fprintf(fd,"\n", n);
    for(l=0;l<3;l++){
     if(ifile == 0)
     printf(" %s %4d %7d",hv_chname_arr[i+l],hv_chno_arr[i+l],hv_setting_arr[i+l]);
     if(ifile == 1)
     fprintf(fd," %s %4d%
7d",hv_chname_arr[i+l],hv_chno_arr[i+l],hv_setting_arr[i+l]);
    }
   }

  return;

}
```

Appendix C: examples of mSQL programs

C.1 insert records

```
/*  LOAD   records [faketime]                        *
 *                                                   *
 *  FIELDS chname, chan, mainframe, slot, spigot,    *
 *         hvmax, setting,                           *
 *         addr_cre_date, addr_exp_date              *
 *         par_cre_date, par_exp_date                *
 *  Database testd                                   *
 *  Table    hv                                      */


#include <stdio.h>
#include <string.h>
```

```
#include <time.h>

#include “msql.h”

char *host      = “da831” ;
char *dbName    = “testd” ;

int sock   ;
int status ;

int mrec   ;            /* records to load , from command line, default 2000  */

int nchan = 10000 ;     /* number of channels in fake detector */
int ichan         ;     /* detector channel, derived from loop index */

int istart           ;  /* start  time , unix Epoch from time() */
int iexp = 2000000000;  /* expiry time , 50 years in the future */

time_t tempo;

char inbuff [ 500 ] = “INSERT INTO hv
(chname,chno,mf,slot,spigot,maxv,setting,addr_cre_date,addr_exp_date,par_cre_date,p
ar_exp_date) VALUES (\0” ;
char ins_tmp[ 500 ]  ;
char buffer [ 500 ]  ;

/* loop variables */
int i, j ;

                   /*    m a i n    */

main ( argc, argv )  int   argc   ; char *argv[] ;
{
/* interconnection with the mSQL engine which is running on host  */

mrec = nchan ;
if ( argc >= 2 ) {
    sscanf ( argv[1] , “%i” , &mrec ) ;
    printf ( “ Doing %i records \n”, mrec ) ;}

istart = time(0) ;
if ( argc >= 3 ) {
    sscanf ( argv[2] , “%i” , &istart ) ;
    printf ( “ Faking start time as %i\n”, istart ) ;}

/*  sock = msqlConnect( host )  ; */
sock = msqlConnect( NULL )  ;

if ( sock == -1 ) {
    printf( “ the error message is \”%s\”\n”, msqlErrMsg )  ;
    exit( -1 )  ; }

/* select the data base *dbName */
  status = msqlSelectDB( sock, dbName )  ;
  if ( status == -1 )
    { printf( “ the error message is \”%s\”\n”, msqlErrMsg )  ;
      exit( -2 )  ;   }

        /*   m a i n    l o o p   */
```

```
tempo  = time(0) ;
printf ( " Start Insert %i\n", istart ) ;
printf ( "              %s\n", ctime(&tempo)) ;

for ( i = 0 ; i <= mrec-1 ; i++ )

{
    /* chname chno frame slot spigot hvmax setting
       addr_cre_date addr_exp_date par_cre_date par_exp_date */
    strcpy(  buffer, inbuff )  ;
    ichan = i%nchan ;
    if ( i<= 10000)
    sprintf( ins_tmp, "'C4-%06d',%d,%d,%d,%d,%d,%d,%d,%d,%d,%d)" ,
        i%nchan+1, ichan%16, ichan/256+1, (ichan/16)%16, ichan%16, 2500, 2100,
        istart, iexp, istart, iexp) ;
    if ( i>= 10001)
    sprintf( ins_tmp, "'C5-%06d',%d,%d,%d,%d,%d,%d,%d,%d,%d,%d)" ,
        i%nchan+1, ichan%16, ichan/256+1, (ichan/16)%16, ichan%16, 2500, 2100,
        istart, iexp, istart, iexp) ;
    strcat( buffer, ins_tmp )  ;

/*    if( i<10 )  printf( "the query string is >%s<\n", ins_tmp )  ;
      if( i<10 )  printf( "the whole buffer string is >\n%s<\n", buffer )  ; */

    /* do the SQL */
    status = msqlQuery( sock, buffer )  ;
    if ( status == -1 )
        {
        printf( " the error message is \"%s\"\n", msqlErrMsg )  ;
        exit( -3 )  ;
     }
}

tempo  = time(0) ;
printf ( " End Insert %i\n", istart ) ;
printf ( "            %s\n", ctime(&tempo)) ;

/* close  the connection to the mSQL engine */
msqlClose( sock )  ;
}
```

C.2 update records

```
/* update  records [faketime]                                *
 *                                                           *
 *  FIELDS chname, chan, mainframe, slot, spigot,            *
 *         hvmax, setting,                                   *
 *         addr_cre_date, addr_exp_date                      *
 *         par_cre_date, par_exp_date                        *
 *  Database testd                                           *
 *  Table    hv                                              */
```

```
#include <stdio.h>
#include <string.h>
#include <time.h>

#include "msql.h"

char *host      = "da831" ;
char *dbName    = "testd"   ;

int sock   ;
int status ;

int mrec    ;          /* records to load, from command line, default is nchan */

int nchan = 2000 ;     /* channel count   */
int ichan ;            /* current channel */

int istart           ;  /* start  time , unix Epoch from time() */
int iexp = 2000000000;  /* expiry time , 50 years in the future */

time_t tempo;

char upbuff[ 500 ] = {
    "UPDATE hv SET \0" }  ;
/*    "UPDATE hv SET addr_exp_date=\0" }  ; */
char inbuff [ 500 ] = "INSERT INTO hv
(chname,chno,mf,slot,spigot,maxv,setting,addr_cre_date,addr_exp_date,par_cre_date,p
ar_exp_date) VALUES (\0" ;

char ins_tmp[ 500 ]  ;
char buffer[ 500 ]  ;

/* loop variables */
int i, j ;

                   /*    m a i n    */

main ( argc, argv )  int   argc   ; char *argv[] ;
{
/* interconnection with the mSQL engine which is running on host  */

mrec = nchan ;
if ( argc >= 2 ) {
    sscanf ( argv[1] , "%i" , &mrec ) ;
    printf ( " Doing %i records \n", mrec ) ;}

istart = time(0) ;
if ( argc >= 3 ) {
    sscanf ( argv[2] , "%i" , &istart ) ;
    printf ( " Faking start time as %i\n", istart ) ;}

/*  sock = msqlConnect( host )  ; */
    sock = msqlConnect( NULL )  ;

if ( sock == -1 ) {
    printf( " the error message is \"%s\"\n", msqlErrMsg )  ;
    exit( -1 )  ; }

/* select the data base *dbName */
```

```
status = msqlSelectDB( sock, dbName )  ;
if ( status == -1 )
    { printf( " the error message is \"%s\"\n", msqlErrMsg )  ;
      exit( -2 )  ;   }

        /*   b i g    l o o p   */

tempo = time(0) ;
printf ( " Start Expire %i\n", istart ) ;
printf ( "              %s\n", ctime(&tempo)) ;

for ( i = 0 ; i <= mrec-1 ; i++ )

{
        /* Expire the old records  */

    ichan = i%nchan           ;

    strcpy( buffer, upbuff )  ;
    sprintf( ins_tmp, "addr_exp_date=%d,par_exp_date=%d WHERE chname='C5-%06d' and
addr_exp_date=%d\n" ,
        istart, istart, ichan+1, iexp ) ;
    strcat( buffer, ins_tmp )  ;
/*    printf( "the query string is >%s<\n", buffer )  ; */

    status = msqlQuery( sock, buffer )  ;

/*    printf( " sql status is  %d %d \n", status, i )  ;*/
    if ( status == -1 )
        {
        printf( " the error message is \"%s\"\n", msqlErrMsg )  ;
        exit( -3 )  ;
     }

    tempo = time(0) ;
    if((i/1000)*1000 == i) printf(" Expired %d records %s\n", i, ctime(&tempo) ) ;

}

tempo = time(0) ;
printf ( " End Expire %i\n", istart ) ;
printf ( "            %s\n", ctime(&tempo)) ;


tempo = time(0) ;
printf ( " Start Insert %i\n", istart ) ;
printf ( "              %s\n", ctime(&tempo)) ;

for ( i = 0 ; i <= mrec-1 ; i++ )

{
        /* Insert the new records  */

    /* chname chno frame slot spigot hvmax setting
       addr_cre_date addr_exp_date par_cre_date par_exp_date */

    strcpy(  buffer, inbuff )  ;
    ichan = i%nchan ;
    sprintf( ins_tmp, "'C5-%06d',%d,%d,%d,%d,%d,%d,%d,%d,%d,%d)" ,
```

```
        i%nchan+1, ichan%16, ichan/256+1, (ichan/16)%16, ichan%16, 2700, 2300,
        istart, iexp, istart, iexp) ;
    strcat( buffer, ins_tmp )  ;

/*    printf( "the query string is >%s<\n", buffer )  ; */

/* do the SQL query */
    status = msqlQuery( sock, buffer )  ;

/*    printf( " sql status is  %d %d \n", status, i )  ; */

    if ( status == -1 )
        {
        printf( " the error message is \"%s\"\n", msqlErrMsg )  ;
        exit( -3 )  ;
     }
}

tempo = time(0) ;
printf ( " End Insert %i\n", istart ) ;
printf ( "            %s\n", ctime(&tempo)) ;

 /* close  the connection to the mSQL engine */
msqlClose( sock )  ;
}
```

C.3 select records

```
/* select and print rows from testd             */

#include <stdio.h>
#include <string.h>
#include <time.h>

#include "msql.h"

char *host      = "da831" ;
char *dbName    = "testd" ;

int sock   ;
int status ;

m_result *result ;
m_row    row    ;
m_field  *field  ;

int nrows   ;
int nfields ;

int mrec    ;
int idate, iexp ;

int length  ;

char buffer[ 500 ]  ;
```

```
/* loop variables */
int i, j ;

time_t tempo;
                   /*    m a i n    */

main ( argc, argv )  int   argc   ; char *argv[] ;
{
/* interconnection with the mSQL engine which is running on host  */

if ( argc == 2 )
{
    strcpy ( buffer , argv[1] ) ;
/*    printf ( " Command is >%s<\n", argv[1] ) ;*/
}
else
{ printf ( " Need a Select command !\n" ) ; exit (-2) ; }

/*  sock = msqlConnect( host )  ; */
sock = msqlConnect( NULL )  ;

if ( sock == -1 ) {
    printf( " the error message is \"%s\"\n", msqlErrMsg )  ;
    exit( -1 )  ; }

/* select the data base *dbName */
  status = msqlSelectDB( sock, dbName )  ;
  if ( status == -1 )
    { printf( " the error message is \"%s\"\n", msqlErrMsg )  ;
      exit( -2 )  ;   }

idate = time(0) ;  iexp = 2000000000 ;

        /*   b i g    l o o p   */

tempo=time(0);
printf( " Start Select %s\n", ctime(&tempo));

status = msqlQuery( sock, buffer )  ;
/*printf( " sql status is  %d %d \n", status, i )  ;*/
if ( status == -1 )
    {
    printf( " the error message is \"%s\"\n", msqlErrMsg )  ;
    exit( -3 )  ;
    }
tempo=time(0);
printf( " End Select %s\n", ctime(&tempo));

result  = msqlStoreResult() ;
nrows   = msqlNumRows   ( result ) ;
nfields = msqlNumFields ( result ) ;
/*
*/
printf ( "Rows and Fields are %d , %d \n",nrows,nfields ) ;

for ( i = 0 ; i<nrows ; i++ )
{   row = msqlFetchRow ( result ) ;
    printf (" %12s%12s%12s%12s%12s%12s\n",
```

```
    row[0],row[1],row[2],row[3],row[4],row[5] ) ;
    /*for ( j = 0 ; j<nfields ; j++ )
    {   printf ( " %d %d field is >%s<\n" , i , j , row[j] ) ;
    } */
}

/* close  the connection to the mSQL engine */
msqlClose( sock )  ;
}
```

Appendix D: examples of HEPDB programs

D.1 create directory structure(s)

```
      PROGRAM DIR_HV
c|    ================
c|
c|    To Create a directory structure in a HEPDB database
c|
      PARAMETER    (NWPAW=100000)
      COMMON/PAWC/ PAW(NWPAW)
      PARAMETER    (NKEYS=1)
      PARAMETER    (MAXOBJ=1000)
      PARAMETER    (NODIRS=5)
      CHARACTER*40 DITAG(NODIRS)
      CHARACTER*8  CHTAG(NKEYS)
      CHARACTER*10 CHFOR
      CHARACTER*4  CHTOP
      CHARACTER*80 CHFILE
      CHARACTER*40 DNAME
c|
c|    Initialise directory names
c|
      DATA DITAG/'/C1/CHNAME',                    !CHANNEL NAME
     &           '/C1/CHNO',                      !CHANNEL NUMBER
     &           '/C1/MFR',                       !MFR
     &           '/C1/SLOT',                      !SLOT
     &           '/C1/SPIGOT'/                    !SPIGOT
      CALL CDPAW(NWPAW,NHBOOK,IDIV,'USR-DIV',5000,50000,'ZPHU',IRC)
      LUNCD  = 1
      LUNFZ  = 2
      CALL CDPREF(10,'HV',CHTOP,CHFILE,IRC)
      IF(IRC.GT.4) THEN
         PRINT *,'EGOPEN. STOPPING DUE TO FATAL ERROR FROM CDPREF'
         STOP 16
      ENDIF
      LRECL  = 0
      CALL CDOPEN(LUNCD,LUNFZ,CHTOP,CHFILE,LRECL,IDIV,' ',IRC)
c|
c|    Define key types and tags
c|
      CHFOR = 'HH'
      CHTAG(1) = 'TYPE'
      IPREC=0
      DELTA=0.0
c|
c|    Loop to create directories
C
      DO 99 IDX=1,NODIRS
```

```
          DNAME='//CDHV/HVDET'//DITAG(IDX)
          PRINT *,'CREATING DIRECTORY `,DNAME,' ...'
          CALL CDMDIR(DNAME,NKEYS,CHFOR,CHTAG,MAXOBJ,IPREC,DELTA
     +                ,'CP',IRC)
          PRINT *,'IRC AFTER CDMDIR =',IRC
99    CONTINUE
c|
c|    Terminate
c|
      CALL CDEND(` `,'A',IRC)
      END
```

D.2 put/get records into/from database

```
C-------------------------------------------------------------------------
     PROGRAM HEPDB_MAIN
C----------------------------------------------------------------------
C!
C!    Author   :- D. TORRETTA      2-JUNE-1995
C!
c|
c|    Opens a  HEPDB database, fills and retrieves banks
c|
c|
     PARAMETER    (NWPAW=400000)
         COMMON/PAWC/ NWP,IXPAWC,IHDIV,IXHIGZ,IXKU,FENC(5),LMAIN
     &   ,PAW(399989)
      PARAMETER    (NKEYS=1)
      PARAMETER    (MAXOBJ=1000)
      PARAMETER    (NODIRS=5)
      CHARACTER*40 DITAG(NODIRS)
      CHARACTER*8  CHTAG(NKEYS)
      CHARACTER*10 CHFOR
      CHARACTER*4  CHTOP
      CHARACTER*80 CHFILE
      CHARACTER*40 DNAME
      CHARACTER*4  BANKNAME(NODIRS)
      CHARACTER*4  BANK
c
     PARAMETER    (NARRAY=20)
     INTEGER RUN1,RUN2,SPILL1,SPILL2
     INTEGER ISOURCE
     INTEGER ARRAYIN(NARRAY)
C    REAL ARRAY(NARRAY)
     INTEGER ARRAY(NARRAY)
C
     CHARACTER*8    ARRAYTMP
     INTEGER ARR_CHNA(NARRAY)     ! CHANGE TO CHAR NEXT
     INTEGER ARR_CHNO(NARRAY)
     INTEGER ARR_MFRM(NARRAY)
     INTEGER ARR_SLOT(NARRAY)
     INTEGER ARR_SPIG(NARRAY)
     CHARACTER*6    CHNAME(NARRAY)
```

```
C
      COMMON /USRLNK/IDIV,JADDR
C
c-----------------------------------------------------------------
c|
c|     Initialise directory names
c|
      DATA DITAG/'C1/CHNAME',                    !CHANNEL NAME
     &            'C1/CHNO',                        !CHANNEL NUMBER
     &            'C1/MFR',                         !MFR
     &            'C1/SLOT',                        !SLOT
     &            'C1/SPIGOT'/                      !SPIGOT

      DATA BANKNAME/'CHNA','CHNO','MFRM','SLOT','SPIG'/

C     DATA  CHNAME/
C       +  'C1-000001'*10,'C1-000002'*10,'C1-000003'*10,
C       +  'C1-000004'*10,'C1-000005'*10,
C       +  'C1-000006'*10,'C1-000007'*10,'C1-000008'*10,
C       +  'C1-000009'*10,'C1-000010'*10/
c
c-----------------------------------------------------------------
c
C       IPUT=1    ! =1 to test banks filling session
C       IGET=0    ! =1 to test banks retrieving session
c
        PRINT *,'PLEASE ENTER IPUT AND IGET VALUES:           '
        PRINT *,'       1= ACTIVATE SELECTION, 0=NO ACTION    '
        READ(5,*) IPUT,IGET
        IF(IPUT.NE.0) PRINT *,' OPTION FILL BANKS SELECTED '
        IF(IGET.NE.0) PRINT *,' OPTION RETRIEVE BANKS SELECTED '

c!
c!      Initialize and open a database
c!
        CALL CDPAW(NWPAW,NHBOOK,IDIV,'USR-DIV',5000,50000,'ZPHU',IRC)
        LUNCD  = 1
        LUNFZ  = 2
        CALL CDPREF(10,'HV',CHTOP,CHFILE,IRC)
        PRINT *,'IRC AFTER CDPREF =',IRC
        IF(IRC.GT.4) THEN
           PRINT *,'EGOPEN. STOPPING DUE TO FATAL ERROR FROM CDPREF'
           STOP 16
        ENDIF
        LRECL  = 0
        CALL CDOPEN(LUNCD,LUNFZ,CHTOP,CHFILE,LRECL,IDIV,' ',IRC)
        PRINT *,'IRC AFTER CDOPEN =',IRC
c!
c!      zeroes ARRAYIN
c!
        DO I=1,NARRAY
         ARRAYIN(I)= 0
        ENDDO
c!
c|
c|      Fill input vectors
C!
        DO I=1,NARRAY
         ARR_CHNA(I)  = 999
```

```
        ARR_CHNO(I)  = I
        ARR_MFRM(I)  = 2
        ARR_SLOT(I)  = 4
        ARR_SPIG(I)  = I
       ENDDO
c!
       IF(IPUT .NE. 1) GO TO 222    ! SKIP FILLING SESSION
c|
c|     Set some constants
C!
       ISOURCE =0
       RUN1 = 250
       RUN2 = 350
       SPILL1= 2000
       SPILL2= 3000

       CHOPT = 'I'
       N = NARRAY
c|
c|     Loop to fill directories
C!
       DO 10 IDX=1,NODIRS
          DNAME= '//CDHV/HVDET/'//DITAG(IDX)
      BANK = BANKNAME(IDX)
          PRINT *,'FILLING BANK ',BANK,' ...'
C
          IF( BANK.EQ.'CHNA') THEN
          CALL UCOPY(ARR_CHNA,ARRAYIN,N)
          ELSEIF( BANK.EQ.'CHNO') THEN
          CALL UCOPY(ARR_CHNO,ARRAYIN,N)
          ELSEIF( BANK.EQ.'MFRM') THEN
          CALL UCOPY(ARR_MFRM,ARRAYIN,N)
          ELSEIF( BANK.EQ.'SLOT') THEN
          CALL UCOPY(ARR_SLOT,ARRAYIN,N)
          ELSEIF( BANK.EQ.'SPIG') THEN
          CALL UCOPY(ARR_SPIG,ARRAYIN,N)
          ENDIF
C
      CALL PUTDB(BANK,ARRAYIN,N,RUN1,RUN2,SPILL1,SPILL2,
     &              ISOURCE,CHOPT,IERR)
          PRINT *,'IERR AFTER PUTDB =',IERR

10     CONTINUE

222    CONTINUE

       IF( IGET.NE.1) GO TO 999
c|
c|     Set some constants
C!
       IRUN=300
       ISPILL=2500

       CHOPT = 'I'
       N=NARRAY
c!
c!     Loop to retrieve  banks
c!
       DO 20 IDX=1,NODIRS
```

```
           DNAME='//CDHV/HVDET/'//DITAG(IDX)
        BANK = BANKNAME(IDX)
           PRINT *,'RETRIEVING BANK `,BANK,' ...'
        CALL GETDB(BANK,IRUN,ISPILL,ARRAY,N,RUN1,RUN2,
     &              SPILL1,SPILL2,BANKLENGTH,CHOPT,IERR)
           PRINT *,'IERR AFTER GETDB =',IERR
           PRINT *,'RUN1, RUN2, SPILL1, SPILL2 `,RUN1,RUN2,SPILL1,SPILL2

20      CONTINUE

c|
c|      Terminate
c|

 999    CONTINUE

        CALL CDEND(` `,'A',IRC)
        END

C-----------------------------------------------------------------------
        SUBROUTINE GETDB(BANK,IRUN,ISPILL,ARRAY,N,RUN1,RUN2,
     &     SPILL1,SPILL2,  BANKLENGTH,CHOPT,IERR)
C-----------------------------------------------------------------------
C!  - Get BANK for IRUN,ISPILL from database
C!
C!   Author   :- E. Blucher            14-APR-1995
C!               D. Torretta            2-JUN-1995
C!
C!   Inputs:
C!        - BANK: 4 letter bank name
C!        - IRUN: run number
C!        - ISPILL: spill number
C!        - N: number of constants
C!        - CHOPT: R = real, I= integer
C!   Outputs:
C!        - ARRAY: array of constants
C!        - RUN1,RUN2: run validity range of constants
C!        - spill1,spill2: spill validity range of constants
C!        - BANKLENGTH: Length of bank found in database
C!        - IERR: = 0 if successful completion
C!
C!  NOTE: The first BANKLENGTH elements of ARRAY are filled if
C!        BANKLENGTH < N to allow for future extensions of banks.
C!        Currently, if BANKLENGTH > N, the job is terminated to prevent
C!        overwriting memory.
C?
C!======================================================================
        CHARACTER*4 BANK
        CHARACTER*1 CHOPT
        CHARACTER*2 CDOPT
        INTEGER RUN1,RUN2,SPILL1,SPILL2
        INTEGER IRUN,ISPILL
        INTEGER IERR
C        REAL ARRAY(*)
        INTEGER ARRAY(*)
        INTEGER N
        INTEGER BANKLENGTH
        CHARACTER*40 PATH
        PARAMETER(IONLINE=0)
```

```
      PARAMETER(IOFFLINE=1)
      PARAMETER(NHDBKEYS=15)
      INTEGER IMASK(NHDBKEYS)
      INTEGER HDBKEYS(NHDBKEYS)
      INTEGER ISELECT(2)
      COMMON/PAWC/ NWP,IXPAWC,IHDIV,IXHIGZ,IXKU,FENC(5),LMAIN
     &  ,PAW(399989)
      DIMENSION IQ(2),Q(2),LQ(8000)
      EQUIVALENCE(LQ(1),LMAIN),(IQ(1),LQ(9)),(Q(1),IQ(1))
      COMMON/USRLNK/IDIV,JADDR
C
      CHARACTER*4 BANKNAME
C
      IDEBUG = 1
C
C  Find correct path (replace with alias)
C
      CALL GTPATH(BANK,PATH)
C
C  Convert character bank name to hollerith
C
      CALL UCTOH(BANK,HDBKEYS(15),4,4)
C
C  Set up imask
C
      DO I=1,NHDBKEYS
        IMASK(I)=0
      END DO
C
C--- imask(10) is online/offline flag
C--- imask(15) is bank name
C
      IMASK(10)=1
      IMASK(15)=1
C
C  Set run and spill number
C
      ISELECT(1)=IRUN
      ISELECT(2)=ISPILL
C
C
      PRINT *,'**** IN GETDB ****'
C
C  First, look for offline constants
C
      HDBKEYS(10)=IOFFLINE
C
      CALL CDUSEM(PATH,JADDR,ISELECT,IMASK,HDBKEYS,' `,IRC)
C
C  If offline constants not found, look for online constants
C
      IF(IRC.EQ.33)THEN
        HDBKEYS(10)=IONLINE
        CALL CDUSEM(PATH,JADDR,ISELECT,IMASK,HDBKEYS,' `,IRC)
      ENDIF
C
      IF(IRC.EQ.0)THEN
        BANKLENGTH=IQ(LQ(JADDR-1)-1)
        PRINT *,'BANK= `,BANK,'   BANKLENGTH= `,BANKLENGTH,
```

```
     &          `   JADDR= `, JADDR
        IF(BANKLENGTH.GT.N)THEN
C!          CALL KTTERM(`_GETDB_ Banklength > array dimension')
          PRINT *,'_GETDB_ Banklength > array dimension'
        ENDIF
        CDOPT='G'//CHOPT
        CALL CDVECT(` `,ARRAY,BANKLENGTH,LQ(JADDR-1),CDOPT,IRC)
        RUN1=IQ(JADDR+11)
        RUN2=IQ(JADDR+12)
        SPILL1=IQ(JADDR+13)
        SPILL2=IQ(JADDR+14)
      ENDIF

     IF (IDEBUG .EQ. 1) THEN
         PRINT *,'>>CONTENT OF IQ FOR BANK = `,BANK,' FROM JADDR `,JADDR
         PRINT *, ( IQ(JADDR+I),I=1,100)
         IF(BANK.EQ.'CHNA') THEN
         PRINT *,'>>CONTENT OF IQ FOR BANK = `,BANK,' FROM JADDR - 50 `
      PRINT *, ( IQ(JADDR - 51 +I),I=1,100)
      ENDIF
         PRINT *,'>>LQ(JADDR-1) = `,LQ(JADDR-1),
     &           `    IQ(LQ(JADDR-1)-1) = `,IQ(LQ(JADDR-1)-1)
         PRINT *,'>>CONTENT OF IQ FOR BANK = `,BANK,' FROM LQ(JADDR-1)'
         PRINT *, ( IQ(LQ(JADDR-1) + I),I=1,100)
         PRINT *,'>>CONTENT OF ARRAY FOR BANK = `,BANK
         PRINT *, (ARRAY(I),I=1,BANKLENGTH)
     ENDIF
c!
c!   zeroes the array vector, to make sure that we really print
c!   what's being filled for each bakn
c!
      DO I=1,N
       ARRAY(I)= 0
      ENDDO
c!
      CALL MZDROP(IDIV,JADDR,' `)
      IERR=IRC
  999 RETURN
      END
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
      SUBROUTINE PUTDB(BANK,ARRAY,N,RUN1,RUN2,
     &     SPILL1,SPILL2,ISOURCE,CHOPT,IERR)
C-----------------------------------------------------------------------
C!  - Put BANK for (RUN1,RUN2), (SPILL1,SPILL2) in database
C!
C!   Author   :- E. Blucher            14-APR-1995
C!               D. Torretta            2-JUN-1995
C!
C!   Inputs:
C!        - BANK: 4 letter bank name
C!        - ARRAY: array of constants
C!        - N: number of constants
C!        - RUN1,RUN2: run validity range of constants
C!        - SPILL1,SPILL2: spill validity range of constants
C!        - ISOURCE: source of constants (0=online,1=offline)
C!        - CHOPT: R = real, I= integer
C!   Outputs:
C!        - IERR: = 0 if successful completion
```

```
C?
C!====================================================================
      CHARACTER*4 BANK
      CHARACTER*1 CHOPT
      CHARACTER*2 CDOPT
      INTEGER RUN1,RUN2,SPILL1,SPILL2
      INTEGER ISOURCE
      INTEGER IERR
C      REAL ARRAY(*)
      INTEGER ARRAY(*)
      INTEGER N
      CHARACTER*40 PATH
      PARAMETER(NHDBKEYS=15)
      INTEGER IMASK(NHDBKEYS)
      INTEGER HDBKEYS(NHDBKEYS)
      COMMON/PAWC/ NWP,IXPAWC,IHDIV,IXHIGZ,IXKU,FENC(5),LMAIN
     &    ,PAW(399989)
      DIMENSION IQ(2),Q(2),LQ(8000)
      EQUIVALENCE(LQ(1),LMAIN),(IQ(1),LQ(9)),(Q(1),IQ(1))
      COMMON/USRLNK/IDIV,JADDR
C
C  Find correct path (replace with alias)
C
      CALL GTPATH(BANK,PATH)
C
C  Convert character bank name to hollerith
C
      CALL UCTOH(BANK,HDBKEYS(15),4,4)
      HDBKEYS(9)=1    !version of constants
      HDBKEYS(10)=ISOURCE
      HDBKEYS(11)=RUN1
      HDBKEYS(12)=RUN2
      HDBKEYS(13)=SPILL1
      HDBKEYS(14)=SPILL2
C
      CDOPT='P'//CHOPT
      CALL CDVECT(' ',ARRAY,N,JADDR,CDOPT,IRC)
      CALL UCTOH(BANK,HDBKEYS(15),4,4)
      CALL CDSTOR(PATH,JADDR,LDUMI,IDIV,HDBKEYS,' ',IRC)
      IF(IRC.NE.0)THEN
C!        CALL KTTERM('_PUTDBR_ Error in CDSTOR')
        PRINT *,'_PUTDBR_ Error in CDSTOR'
      ENDIF
      CALL MZDROP(IDIV,JADDR,' ')
      IERR=IRC
  999 RETURN
      END

C-------------------------------------------------------------------
      SUBROUTINE GTPATH(BANK,PATH)
C-------------------------------------------------------------------
C!  - Get correct PATH for BANK in database.
C!
C!   Author   :- E. Blucher            20-APR-1995
C!               D. Torretta            2-JUN-1995
C!
C!   Inputs:
C!       - BANK
C!
```

```
C!   Outputs:
C!        - PATH
C!======================================================================
C
      CHARACTER*4 BANK
      CHARACTER*40 PATH
      IF(BANK.EQ.'CHNA')THEN
        PATH='//CDHV/HVDET/C1/CHNAME'
      ELSEIF(BANK.EQ.'CHNO')THEN
        PATH='//CDHV/HVDET/C1/CHNO'
      ELSEIF(BANK.EQ.'MFRM')THEN
        PATH='//CDHV/HVDET/C1/MFR'
      ELSEIF(BANK.EQ.'SLOT')THEN
        PATH='//CDHV/HVDET/C1/SLOT'
      ELSEIF(BANK.EQ.'SPIG')THEN
        PATH='//CDHV/HVDET/C1/SPIGOT'
      ENDIF
      RETURN
      END
```

Appendix E: E831 mSQL Database Tests

29 August 1995

E831 MSQL Database Tests

We are certifying the MSQL Database for use in the E831/FOCUS online system, as a replacement for the VMS-specific Datatrieve system used in E687.

We have tested the reliability and performance of MSQL both within the expected, and well beyond the expected limits of usage.

The largest existing E687 database files contained about 10,000 records. In normal usage only one process was likely to be actively accessing any database. Typical thousand row updates required several hours to complete.

We have tested up to 500,000 records with 3 to 8 simultaneous updates. Thousand row updates comparable to those in E687 complete in under 2 minutes.

We have tested MSQL Version 1.0.7, under IRIX 5.3 . Test are run on a 64 Megabyte Indy R4600PC-133 (dc831.fnal.gov), using a local server and local disk.

Our tests measure:

1. Speed of loads, updates and selects
2. Integrity of data

We measure these under several conditions:

1. Various table sizes (1K, 20K, 100K, 200K, 500K)
2. Single client and multiple client

These tests use rows of one CHAR(16) string and five INT's. The data fields are

- name CHAR(16)
- main INT
- slot INT
- chan INT
- start INT
- exp INT

'name' is a channel name, set to X:nnnn for this test. 'start' and 'exp' are start and expirations time for the row. 'exp' is set to 2000000000 for unexpired rows.

Each record is updated in two steps:

1. Update field ‘exp’in the unexpired record matching each channel:

   UPDATE data SET exp=... WHERE name=’...’ AND exp=2000000000

2. Write a new unexpired record

   INSERT INTO data VALUES (...)

Multiple Client Reliability

We apply 20 sets of updates to each preloaded table, each set updating 1000 records. The update sets are timed so that at least 3 will be active simultaneously. As many as 8 update sets run simultaneously for the smaller tables.

An ASCII dump of rows modfied and inserted by each update set was compared to a dump from a small reference table built with a single client. The dumps are found to be identical in all cases.

Performance

We appear to be CPU limited in these tests. The Unix ‘top’ utility shows that 80% to 95% of the available CPU capacity is being used by the client and server during all substantial updates. Typically, the server uses 80% and the client 15%.

MSQL presently supports only Primary Keys, which must be unique. Our E831 tables do not have any unique fields which could be used as Primary Keys. Therefore we are measuring the raw, unindexed speed of the MSQL database.

We display in the chart below the times required to perform each of the measured tasks. The times reported are elapsed times, measured with the ‘time’ shell command. The system is otherwise idle during these tests. The tasks are performed by ‘load’, ‘update’ and ‘select’ test programs written in C.

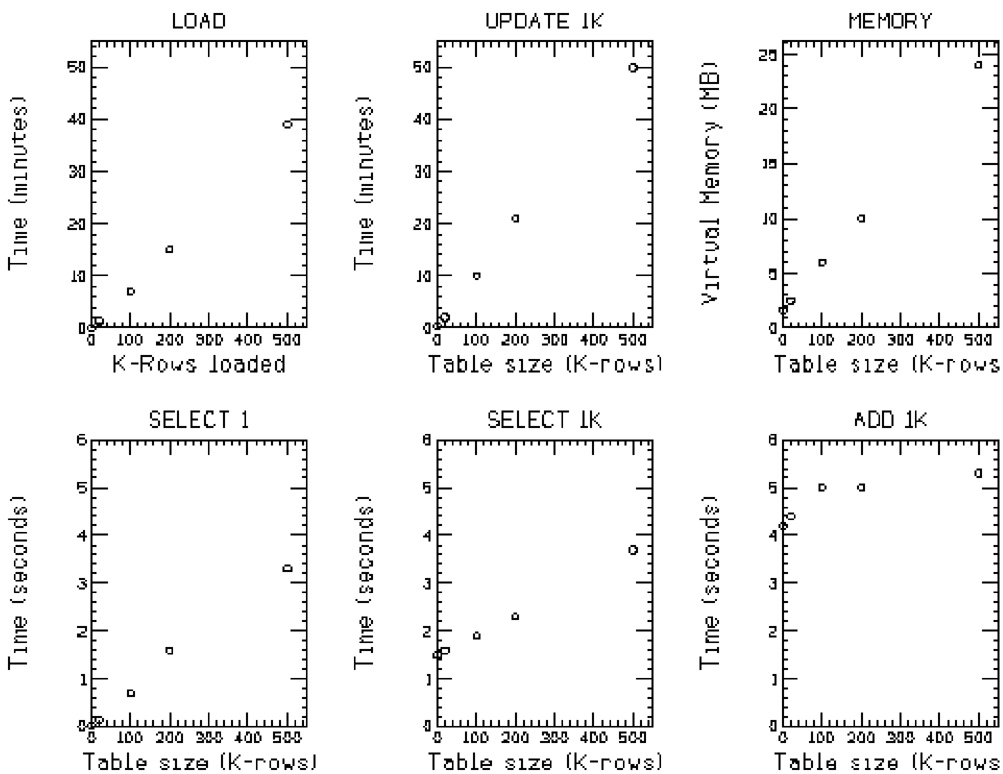
For the smaller tables we do a repeated select within the test program, in order to reduce program loading and startup overheads. By moving the server connect and disconnect code in and out of this loop we have measured the connect+disconnect overhead to be about 12 milliseconds. This is about the same time as the actual select process for a single record in a 1000 row table.

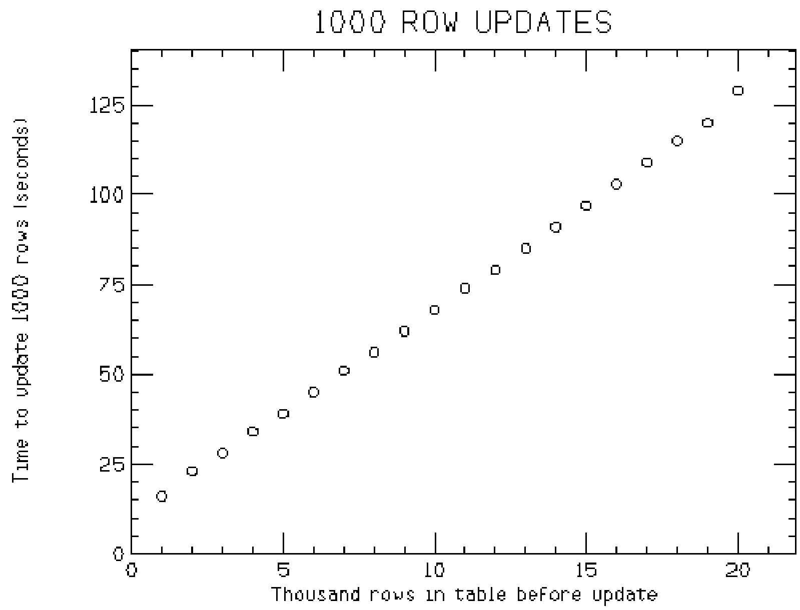
The following items are tabulated and plotted below :

- The time required to pre-load the table to its initial size
- The time required to perform 1000 row updates
- The time required to select a single row
- The time required to select 1000 rows with a single SQL query
- The time required to load 1000 additional rows
- The amount of virtual memory used by the msqld server process

Table-size Performance Study

Size (Records)	init load (minutes)	1K update (minutes)	1 select (seconds)	1K select (seconds)	1K load (seconds)	Memory (Megabytes)
1 K	.07	.3	.023	1.5	4.2	1.6 MB
20 K	1.3	2	.14	1.6	4.4	2.5 MB
100 K	7	10	0.7	1.9	5.	6 MB
200 K	15	21	1.6	2.3	5.	10 MB
500 K	39	50	3.3	3.7	5.3	24 MB




1000 ROW UPDATES
Time to update 1000 rows (seconds)
Thousand rows in table before update
0
25
50
75
100
125
0
5
10
15
20