1

SQL Introduction

A database is a collection of rectangular tables of information. These tables contain columns which describe individual parameters. For example the ***Summary*** table in the ***Logbook*** database has columns for ***shot, pulse_length, max_ip, etc...*** One or more of these columns can be indexed in order to make certain classes of queries more efficient. All database accesses are broken up into **transactions** during which the database maintains a consistent view of the data for you. It is best to make your transactions as short as possible. During a **read only** transaction, the database has to maintain copies of all of the referenced tables for you so that they can not be changed out from under you. During a **read write** transaction you potentially lock out other users from accessing the tables being referenced. Transactions are finished with either a **commit** or **rollback** statement. For read only transactions they are equivalent. For read write transactions **commit** writes the changes to the database whereas **rollback** puts the database back the way it was.

It is essential that you commit or rollback all database transactions. Failure to do so can cause the CMOD shot cycle to hang.

There are canned routines and programs to do common database operations. **SQL_UPDATE** can be used to insert and modify records from **IDL. WLOGBOOK** is a general interactive database access program. It is particularly good at looking at and modifying the ***entries, shots, and runs*** tables in the **LOGBOOK** database. There is also a general interface in **IDL** which allows execution of sql statements from within IDL. Some combination of these tools can be used to perform all of the common database operations.

2

A Quick tour of the Logbook database

The Logbook database started out as a set of tables used to store textual information about the shots. It has evolved into something more general with the addition of tables of numeric information about the shots. At this time the Logbook contains the following tables:

User tables in database with filename logbook

```
  CORE**
  DATA_STORAGE
  DISRUPTIONS**
  ENTRIES*
  GOOD_SHOTS
  KEYWORDS*
  RUNS*
  SHOTS*
  SPECTROSCOPY**
  STORAGE
  SUMMARY**
  SUMMARY_UNITS
  TAG_REGISTRY
  TOPICS*
  TRANSP_SCALARS**
  TREES
```

* These tables are used by the Logbook software to store text entries provided by the users.
** These tables contain computed information about the shots.

The **CORE** table contains time slices of shots core diagnostics. The **SPECTROSCOPY** table contains information about the hirex diagnostic for each shot. The **SUMMARY** table contains summary information about each shot. Its' intent is to act as an index to the archived pulses. Looking at this table in more detail, it contains the following fields:

```
Columns for table SUMMARY:
Column Name           Data Type     Domain
-----------           ---------     -----
SHOT                  INTEGER       SHOT_DOM
IPMAX                 REAL          IPMAX_DOM
T_IPMAX               REAL          T_IPMAX_DOM
POH                   REAL          POH_DOM
```

```
NEMAX_TCI             REAL          NEMAX_TCI_DOM
T_NEMAX_TCI           REAL          T_NEMAX_TCI_DOM
TEMAX_ECE             REAL          TEMAX_ECE_DOM
NEMAX_THOMSON         REAL          NEMAX_THOMSON_DOM
TEMAX_THOMSON         REAL          TEMAX_THOMSON_DOM
ZEFF                  REAL          ZEFF_DOM
PRAD                  REAL          PRAD_DOM
TOPOLOGY              VARCHAR(3)    TOPOLOGY_DOM
DISRUPTIVITY          TINYINT       DISRUPTIVITY_DOM
T_DISRUPT             REAL          T_DISRUPT_DOM
PULSE_LENGTH          REAL          PULSE_LENGTH_DOM
TIME_OF_SHOT          DATE VMS      TIME_DOM
 Comment: Time of shot from PLCs
GAS1                  VARCHAR(6)    GAS_DOM
 Comment: Working Gas (majority species)
GAS2                  VARCHAR(6)    GAS_DOM
 Comment: Impurity species
GAS3                  VARCHAR(6)    GAS_DOM
A                     REAL          A_DOM
R                     REAL          R_DOM
DELTA_U               REAL          DELTA_U_DOM
DELTA_L               REAL          DELTA_L_DOM
KAPPA                 REAL          KAPPA_DOM
PRF                   REAL          PRF_DOM
T_PRF                 REAL          T_PRF_DOM
T_FIT                 REAL          T_FIT_DOM
WTOTMAX               REAL          WTOTMAX_DOM
IPSIGN                INTEGER       IPSIGN_DOM
IP                    DOUBLE PRECISION
 Computed: by ipmax*ipsign  Comment: Signed maximum plasma current
T_WTOTMAX             REAL          T_WTOTMAX_DOM
BTORSIGN              INTEGER       BTORSIGN_DOM
GAS_PRESSURE          REAL          GAS_PRESSURE_DOM
BTORMAX               REAL          BTORMAX_DOM
BTOR                  DOUBLE PRECISION
 Computed: by btormax*btorsign  Comment: Signed maximum toroidal
field
LPI                   TINYINT       LPI_DOM
DPI                   TINYINT       DPI_DOM
HARD_XRAY_DOSE        REAL
```

This table is indexed by **SHOT** and contains quantities like maximum plasma current (IPMAX), time of maximum plasma current (T_IPMAX), as well as the length of the pulse (PULSE_LENGTH). It can be used to get the answers to simple queries about the existing CMOD shots. For example to get all of the shots from 1995 on with > 1.1 MAmp of plasma current using the SQL interface:

```
SQL> attach 'filename logbook prestarted transactions are off';
SQL> declare transaction read only;
SQL> select shot, ipmax, pulse_length from summary where shot >
950100000 and ipmax > 1.1E6 order by shot;
SHOT        IPMAX       PULSE_LENGTH
950209009 1.1248228E+06 1.0193000E+00
950209011 1.1377448E+06 6.1269999E-01
950209022 1.1222383E+06 1.0185000E+00
950209023 1.1248918E+06 1.4575000E+00
950209024 1.1248228E+06 1.1269000E+00
```

```
950209025 1.1222613E+06 1.0149000E+00
950209026 1.1274301E+06 1.4405000E+00
950209029 1.2101085E+06 9.9049997E-01
950210028 1.1170465E+06 9.5930004E-01
950210030 1.1196539E+06 1.2605000E+00
950210031 1.1196309E+06 1.2172999E+00
950210032 1.1248228E+06 1.1942999E+00
12 rows selected
SQL> commit;
```

A few things to notice about the above example:

1 The 'declare transaction read only;' line puts SQL into read only mode. This is a good thing to do unless you explicitly want to write into the database. The IDL interface automatically does this for you.

2 The 'commit;' line ends this transaction. This is very important to do to avoid locking up the database. See section 1 of this document.

Any of the fields in this table could have been retrieved in this select. Also any of them could have been used in the boolean expression that selected out the records or in the order by clause.

3

The IDL Interface to SQL

The IDL interface to SQL was implemented by Ken Klare of Los Alamos National Lab. It consists of SET_DATABASE, SQL_FINISH and two interfaces to pass strings to the SQL interpreter. SET_DATABASE is used to open a particular database or set of tables. SQL_FINISH is used to sever the connections to open databases. ISQL passes a string to the SQL interpreter and prints the result on SYS$OUTPUT. DSQL passes a string to the SQL interpreter and returns the result into IDL arrays. A few examples will make this fairly clear:

```
CMOD> idl

IDL. Version 3.6.1c (vms vax).
Copyright 1989-1994, Research Systems, Inc.
All rights reserved.
Unauthorized reproduction prohibited.
Installation number: 573-0.
Licensed for use by: MIT Plasma Fusion Center

IDL> set_database, 'logbook'
% Compiled module: SET_DATABASE.
% Compiled module: DSQL.
% Compiled module: SQL.
IDL> count = dsql("select shot, ipmax, pulse_length from summary
where shot > 950100000 and ipmax > 1.1E6 order by shot", shots, ips,
lengths)
IDL> print, dsql('COMMIT')
         0
IDL> print, count
        12
IDL> for i = 0, count-1 do begin print, i, ' ', shots(i), ' ',ips(i),
' ', lengths(i)
0 950209009 1.124823e+06 1.01930
1 950209011 1.137745e+06 0.612700
2 950209022 1.122238e+06 1.01850
3 950209023 1.124892e+06 1.45750
4 950209024 1.124823e+06 1.12690
5 950209025 1.122261e+06 1.01490
6 950209026 1.127430e+06 1.44050
7 950209029 1.210109e+06 0.990500
```

```
8 950210028 1.117047e+06 0.959300
9 950210030 1.119654e+06 1.26050
10 950210031 1.119631e+06 1.21730
11 950210032 1.124823e+06 1.19430
IDL> count = dsql("select shot, ipmax, pulse_length from summary
where shot between 950300000 and 950400000 order by shot",shots, ips,
lengths)
IDL> print, count
         610
IDL> print, dsql('COMMIT')
         0
IDL> for i = 0, 10 do begin print, i, ' ', shots(i), ' ', ips(i),
' ', lengths(i)
0 950301001 2607.43 0.000000
1 950301002 861235. 0.700500
2 950301003 202186. 0.143500
3 950301004 346890. 0.336100
4 950301005 481326. 0.496900
5 950301006 26422.4 0.000000
6 950301007 543375. 0.548700
7 950301008 579488. 0.490300
8 950301009 734645. 1.58270
9 950301010 23838.0 0.000000
10 950301011 393433. 0.394100
```

The first example above retrieves values from shots since January 1995 which plasma current greater than 1.1 MAmp. Note the 'commit' after the select. DSQL returns the number of records found as its' value. One variable must be provided for each item selected. In the second example, fields from all of the shots during march 1994 were retrieved.

The values of IDL variables can be substituted into the strings passed to ISQL and DSQL using '?' markers. The first example above can be rewritten with the shot and the target plasma current stored in variables:

```
...
IDL> ipmax = 1.1E6
IDL> start_shot = 950100000
IDL> count = dsql("select shot, ipmax, pulse_length from summary
where shot > ? and ipmax > ? order by shot", ipmax, start_shot,
shots, ips, lengths)
IDL> print, dsql('COMMIT')
         0
IDL> print, count
         12
IDL> for i = 0, count-1 do begin print, i, ' ', shots(i), ' ',
ips(i), ' ', lengths(i)
0 950209009 1.124823e+06 1.01930
1 950209011 1.137745e+06 0.612700
2 950209022 1.122238e+06 1.01850
3 950209023 1.124892e+06 1.45750
...
```

By default the SET_DATABASE command opens the tree read only. This means that in

order to write into the database you must explicitly set your transaction read write. This is done automatically if you use the provided routine SQL_UPDATE.

SQL_UPDATE can be used to insert or update individual records in a table. Note if you want to update more than one record at a time (e.g. update IPMAX for all shots in 1995) you can not use SQL_UPDATE. Here is the file header from SQL_UPDATE.PRO:

```
;+
; NAME:            SQL_UPDATE
; PURPOSE:         Update (optionally insert) a record in a time
;                  slice table
; CATEGORY:        MDSPLUS - SQL
; CALLING SEQUENCE:
; sql_update, table, p_key, values=values, [, key=key]
;         [, master_table=master_table]
;         [, status=status] [,/quiet] [,/debug]
;         [,mask=mask] [,/CREATE]
;         table -   name of the table to update
; p_key - struct containing the record key for the primary table.
;         for example: {shot:999999, time_slice:.05}
; values=values    struct containing the fields to set
;                  like {ninja:'ninja string', ninja_time:2.2}
; [key=key]        struct containing the values for the rest of
;                  the key of this slice, for example
;                  {side:'in', probe:12}
; [master_table=master_table] - name of the 'key' table for
;                  this table.  if not specified then this table is
;                  assumed to be a standalone database table. Like
;                  SUMMARY, or CORE...
;                  see /CREATE below
; [status=status]  optional return status
; [/QUIET]         if set no messages will be printed out
; [/DEBUG]         if set various diagnostic messages are printed
; [MASK=mask]      an array of bytes 1/field in values where
;                  -1 -> set this field to null
;                   0 -> leave this field alone
;                   1 -> update this field from values
; [/CREATE]        if set and no master table is specified then if
;                  the record to be updated is not present it will be
;                  inserted.
;
; KEYWORDS:        See above.
; OUTPUTS:         None except for status
; COMMON BLOCKS: None.
; SIDE EFFECTS:    SQL database opened or accessed.
; RESTRICTIONS:
;          -A SET_DATABASE command must have been executed prior to
;           using this routine.
;
; PROCEDURE:
;         Select the record to be updated in the table relation
;         if it is not there then
;         select the record for this shot, time from the master table
;                   if it is not there then
;         choke
;         else
```

```
;        insert this record shot, time (and any keys) into the table
;        endif
;        update the specified fields in table.
;
; MODIFICATION HISTORY:
;   April 7, 1994 - Initial version   J. Stillerman
;   May 3, 1994 - add p_key and /CREATE and change name J. Stillerman
```

This routine can be used the either insert new records into a table or update existing ones. The following code fragment could be used to fill in IPMAX and T_IPMAX into the SUMMARY table of the LOGBOOK. Note that the SUMMARY table is filled in automatically and you SHOULD NOT update it without talking with Josh.

```
IDL> SET_DATABASE, 'LOGBOOK'
IDL> GET_IPMAX, IPMAX, T_IPMAX  ; call some routine to get the max IP
IDL> SQL_UPDATE, 'SUMMARY', {shot:950302001}, values = {IPMAX:ipmax,
T_IPMAX:t_ipmax}, /CREATE
```

The first argument specifies the table to be operated on, in this case 'SUMMARY'. The second argument specifies the key(s) for this row of the table. Since the SUMMARY table has only one key, SHOT, this is a structure with only one field. The third argument specifies the values to update in this row. In this case IPMAX and T_IPMAX. Finally the /CREATE argument tells SQL_UPDATE to insert this row into the table if it is not already there. Detailed descriptions of the master_table, mask, and key arguments are beyond the scope of this document.

4

Some Useful SQL Snippets

Select clauses

1. A list of fields - a list of the fields or columns you want from the table. For example:
   ```
   select shot, ipmax, t_ipmax from ...
   ```
2. * - a '*' selects all of the fields of the table. This is probably only useful in the SQL interface or using ISQL from IDL.
3. Count(*) - selects the number of records of the table which match the where condition. For example:
   ```
   select count(*) from some-table where ....
   ```

Boolean expressions

1. Relational operators > < = >= <= - relational operators can be used in the where clause of a select. Both constants or other columns of the table can be used as arguments. For example:
   ```
   select a,b,c from some-table where a> 5 and b > c
   ```
2. Logical operators and, or - as in the above example terms can be connected together with ands and ors and ().
3. String operators: =, containing, like - string columns can be examined using these operators. The '=' compares for exact equality. 'LIKE' does case insensitive comparisons, and containing does substring searches.
4. Between - to select records where some column is between two values use a between clause in the where. For example:
   ```
   select a, b, c from some-table where a between 99 and 342
   ```
5. NULL, NOT NULL - table columns which do not have default values and which are uninitialized contain a special NULL value. It is sometimes useful to exclude or include these rows in a selection. For example:
   ```
   Select a, b, c from some-table where b is not null and c is null
   ```

Data Retrieval Examples

The following examples assume that LOGBOOK database has been opened. To open the LOGBOOK database use the following:

IDL> SET_DATABASE, "LOGBOOK"

1 This example retrieves the shot number and maximum plasma current for each shot taken during January 1995.

```
IDL> count = DSQL("SELECT shot, ipmax FROM summary WHERE shot BE-
TWEEN 950101000 AND 950232999", shots, ipmaxes)
IDL> dummy = DSQL("COMMIT")
```

2 There may be a few shots where plasma current was not filled in. The query could be modified to eliminate them.

```
IDL> count = DSQL("SELECT shot, ipmax FROM summary WHERE shot BE-
TWEEN 950101000 AND 950232999 AND ipmax IS NOT NULL", shots, ip-
maxes)
IDL> dummy = DSQL("COMMIT")
```

3 To find all of the shots that are missing their IPMAX entries use:

```
IDL> count = DSQL("SELECT shot FROM summary WHERE ipmax IS NULL",
shots)
IDL> dummy = DSQL("COMMIT")
```

4 To get all of them since the beginning of 1995:

```
IDL> count = DSQL("SELECT shot FROM summary WHERE shot > 950101000
and ipmax IS NULL", shots)
IDL> dummy = DSQL("COMMIT")
```

5 An order by clause can be added to the end of any of the above queries, and it can contain an asc (ascending) or desc (descending) modifier. Ascending is the default.

a.
```
IDL> count = DSQL('SELECT shot, ipmax FROM summary WHERE shot >
950101000 AND ipmax IS NOT NULL ORDER BY shot', shots, ips)
IDL> dummy = DSQL("COMMIT")
```

b.
```
IDL> count = DSQL('SELECT shot, ipmax FROM summary WHERE shot >
950101000 AND ipmax IS NOT NULL ORDER BY ipmax desc', shots,
ips)
IDL> dummy = DSQL("COMMIT")
```

6 Dates are specified in queries using standard VMS date formats in quotes.

a.
```
IDL> count = DSQL("SELECT username FROM entries WHERE entered
BETWEEN 'YESTERDAY' AND 'TODAY'", users)
IDL> dummy = DSQL("COMMIT")
```
Note that the dates are in single quotes. The easiest way to do this is to use double quotes for the query argument.

b.
```
IDL> count = DSQL("SELECT username FROM entries WHERE entered >
'1-JAN-1995'", users)
IDL> dummy = DSQL("COMMIT")
```

7 The number of records which meet some criteria can be retrieved without actually retrieving the data.

```
IDL> dummy = DSQL("SELECT COUNT(*) FROM entries WHERE entered >
```

```
'1-JAN-1995'", num_entries)
IDL> dummy = DSQL("COMMIT")
```

8 Placeholders can be used to pass IDL variables into SQL queries.

a. **IDL>** count = DSQL("SELECT shot, ipmax FROM summary WHERE shot > ? ORDER BY SHOT", 950101000, shots, ips)
IDL> dummy = DSQL("COMMIT")

b.
```
IDL> start_shot = 950101000
IDL> working_gas = "He"
IDL> count = DSQL("SELECT shot, ipmax FROM summary WHERE shot >
? and GAS1 = ?",  start_shot, working_gas, shots, ips)
IDL> dummy = DSQL("COMMIT")
```
NOTE: if He was specified in-line in the SELECT then it would have had to have been enclosed in single quotes.

Inserting/Updating Records

It is highly recommended to use the SQL_UPDATE routine to write data to the databases. It insures that the potentially dangerous write transactions are handled correctly. For those who must modify data by hand here is a brief synopsis. The main reason to do this is to make wholesale, **read extremely dangerous**, changes to the database.

1 SET TRANSACTION - in order to write into a table of the database you must first start a read write transaction. It is easier for the database to do this if you specify the tables you are going to reference. For example:

```
IDL> dummy = DSQL('set transaction read write reserving summary
for shared write')
```

A SET TRANSACTION statement must be executed for each transaction you make to the database. So, before every INSERT, SET_TRANSACTION, after every INSERT, COMMIT.

```
IDL> dummy = DSQL("SET TRANSACTION READ WRITE RESERVING summary
FOR SHARED WRITE")
```

Notice that the set transaction is qualified by the list of tables which will be involved in the query.

2 INSERT - after doing a SET TRANSACTION statement data can be inserted into a table using an INSERT statement. For example:

```
IDL> dummy = DSQL("insert into summary (shot, ipmax, t_ipmax) val-
ues(?, ?, ?)", shot, ip, t_ip)
IDL> print, DSQL('COMMIT')  ; !!!! don't forget this !!!!!!
     0
```

When using update it is a good idea to do a select prior to doing the update using the same where clause. This way you can determine exactly how many records will be modified by the query. In the first example since we know that there is exactly one summary record per shot this step is skipped. In the remaining examples which update multiple records it is included. If records are unintentionally updated then use ROLLBACK instead of COMMIT.

```
1  IDL> print, DSQL("update summary set (ipmax=?, t_ipmax=? where
   shot=?", ip, tip, shot)
        0
   IDL> print, DSQL('COMMIT') ; !!!! don't forget this !!!!!!
        0

2  IDL> dummy = DSQL("SELECT COUNT(*) FROM summary WHERE shot between
   950101000 and 950201000", count)
   IDL> print, count
       330
   IDL> dummy = DSQL("UPDATE SUMMARY SET ipmax = ipmax*1.1 WHERE shot
   BETWEEN 950101000 and 950201000")
   IDL> print, DSQL('COMMIT') ; !!!! don't forget this !!!!!!
       0

3  IDL> dummy = DSQL("SELECT COUNT(*) FROM summary WHERE shot between
   950101000 and 950201000", count)
   IDL> print, COUNT
       330
   IDL> dummy = DSQL("UPDATE SUMMARY SET ipmax = NULL WHERE shot be-
   tween 950101000 and 950201000")
   IDL> dummy = DSQL("COMMIT")
```

5

Extra Things

Under certain circumstances the IDL interface to SQL (DSQL) will limit the number of arguments that you can pass to it. This potentially makes it difficult to retrieve large numbers of columns at a time from a table. This problem can be avoided by adding a SQL to your IDL startup.

For those really wanting to know why: When IDL compiles a procedure it binds actions to all of the external procedure references it finds in them. These bindings are static. The procedure SQL defines DSQL as a call to an external routine which accepts a large number of arguments. As a convenience, there is a .PRO called DSQL.PRO which makes the above linkage and then calls the external routine with a limited number of arguments. What happens is that in the context which DSQL is defined it always points at this dummy 'convenience' routine. The solution is to make sure that DSQL points at the external routine from the beginning. Invoking SQL at the start of your session will do this.