NASA Contractor Report 191416

LARCRIM User's Guide Version 1.0

John S. Davis
William J. Heaphy

Computer Sciences Corporation
Hampton, Virginia

Contract NAS1-19038
January 1993

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-0001

FOREWORD

LARCRIM is based upon the relational algebra model for data management and has been used for both engineering and business data. It is accessible as a stand-alone system and through an application program interface. The stand-alone system may be executed in either of two modes: menu or command. The menu mode prompts the user for the input required to create, update, and/or query the database. The command mode requires the direct input of LARCRIM commands, the requirements for which are not difficult to learn.

LARCRIM derives from the Relational Information Management System (RIM) which was developed for NASA in the late 1970's. An updated version, RIM-5, released in 1981, has been available to the public through the Computer Software Information and Management Center (COSMIC). It remains available from COSMIC, as of 1992, as program #M92-10429. LARCRIM is based on the VAX version of RIM-5. LARCRIM is the first version of RIM to run under UNIX. Otherwise, the differences between the two are mostly transparent code optimization. LARCRIM is particularly suited for the storage and manipulation of scientific data because it allows arrays and matrices to be stored as such.

LARCRIM is written in FORTRAN 77. It retains a number of non-ANSI characteristics such as Hollerith character representation. This will be resolved in a future version.

Versions of LARCRIM have been released to run under UNIX on the following computers:

SUN 3 and SUN 4 (UNIX)
Convex (UNIX)
IRIS (UNIX)
Hewlett-Packard (UNIX)
CRAY 2 and CRAY Y-MP (UNICOS)

The body of this manual will discuss commands and procedures commonly valid on these computers. From an interactive viewpoint, the computer hardware is generally irrelevant. Otherwise, host dependent information is contained in Appendix F. This will generally relate to applications programming and to certain systems considerations. Future releases of LARCRIM are anticipated for the IBM PC and the Macintosh computers.

This user's guide is based on the RIM-5 User's Guide published by COSMIC. It contains substantial new material, Chapter 1 in particular, and minor to major revisions throughout..

LARCRIM and the present document were produced in support of the High Speed Airframe Integration Research (HiSAIR) project at Langley Research Center under the direction of Mr. Kennie H. Jones.

TABLE OF CONTENTS

TABLE OF CONTENTS
(continued)

COMMAND INDEX

COMMAND INDEX

1.0 A QUICK LOOK

The first section of the LARCRIM User's Guide provides a quick look at using the LARCRIM Database Management System. It is presented as an annotated example in which a database is created, relations and attributes are defined, data are loaded into the database, the database is queried, and the database is exited. Throughout this section, the following typographical conventions are observed:

THIS Typeface	Routine descriptive text, as above.
INPUT	User input to the computer, either commands or data.
OUTPUT	Computer output to the screen or to a text file.
ANNOTATION	Editorial commentary, usually in blocks of text on the right side of a page.

```
eagle% larcrim
```

Initiate LARCRIM. This may differ by host machine.

```
BEGIN LARCRIM -CONVEX   VERSION 1.0   UDxx   91/10/24    10.32.01

 LARCRIM COMMAND MODE
 ENTER "MENU" FOR MENU MODE
```

```
R>DEFINE PLANES
```

Create the database (DB) definition, also called the SCHEMA.

```
 BEGIN LARCRIM SCHEMA COMPILATION
```

```
D>OWNER AGENT
```

You are now in the DEFINE module (D>). Tell it who you are and begin giving definition commands.

```
D>ATTRIBUTES

D>WORD TEXT 8 KEY
D>WRDTYPE TEXT 9
D>DESCRIP TEXT VAR
D>MFG TEXT 10 KEY
D>MODEL TEXT 10
D>TYPE TEXT 10 KEY
D>NUMENG INT
D>ENGTYPE TEXT 10
D>CRUSALT REAL
D>CRUSSPD REAL
D>CARRIER TEXT 10 KEY
D>FLIGHTNO INT
D>STRTCITY TEXT 10
D>ENDCITY TEXT 10
D>DAYOFWK TEXT 10
D>PILOT TEXT 20
D>RATING INT
```

A relational database is made up of RELATIONS, which contain ATTRIBUTES. The attributes are defined first, then the relations. Here, some text attributes are defined with a fixed 8,9, 10, or a VARiable number of characters for each. The default is 8 characters for text and 1 value for real and integer. The KEY flags speed up querying of the finished database.

Now define 4 relations from the attributes above. A plus sign (+) at the end of a line will continue that line on the line below.

```
D>RELATIONS
D>DEFIN WITH WORD WRDTYPE DESCRIP
D>AIRLINES WITH CARRIER FLIGHTNO STRTCITY ENDCITY DAYOFWK +
MODEL
D>AIRPLANS WITH MFG MODEL TYPE NUMENG ENGTYPE CRUSALT CRUSSPD
D>CREW WITH FLIGHTNO PILOT RATING
```

Passwords are optional.

```
D>PASSWORDS
D>MODIFY PASSWORD FOR DEFIN IS DBA
D>MPW FOR AIRLINES IS AGENT
D>RPW FOR ALL IS AGENT
```

Abbreviations:
Modify password = mpw
Read " = rpw

Rules are also optional, but we'll see how these work later on.

```
D>RULES
D>DAYOFWK NE SUN
D>RATING GT 0 AND RATING LT 10
D>END
```

The "END" takes us out of the define sublevel and back to the RIM command level (R>). Now, to load the database, we first need to give the password we specified for the relation DEFIN. We do so with the "USER" command.

```
R>USER DBA
R>LOAD DEFIN

  BEGIN  -LARCRIM-  DATA LOADING
```

We are now in the "load" mode; notice the "L>" prompts

```
L>WORD ATTRIBUTE "NAMES USED IN THE PLANES DATA BASE."
L>WRDTYPE ATTRIBUTE "THE TYPE OF THE NAME:  ATTRIBUTE OR RELATION."
L>DESCRIP * "A TEXTUAL DESCRIPTION OF THE WORDS USED IN THE DATA +
BASE."
L>MFG * "THE MANUFACTURER OF THE AIRPLANE."
L>MODEL * "THE AIRPLANE MODEL NUMBER."
L>TYPE * "THE AIRPLANE TYPE, I.E., PASSENGER, CARGO, ETC."
L>NUMENG * "THE NUMBER OF ENGINES ON THE AIRPLANE."
L>ENGTYPE * "THE ENGINE TYPE, I.E., JET, PROP, ETC."
L>CRUSALT * "THE AIRPLANE'S CRUISE ALTITIUDE."
L>CRUSSPD * "THE AIRPLANE'S CRUISE SPEED."
L>CARRIER * "THE AIRLINE CARRIER WHICH USES THE PLANES."
L>FLIGHTNO * "THE FLIGHT NUMBER OF A CARRIER'S ROUTE."
L>STRTCITY * "THE START CITY OF THE FLIGHT ROUTE."
```

Note that character strings are quoted and that a plus (+) sign is used to continue long strings (over 80 characters) on the next line. The asterisk equates to a ditto mark; it repeats the corresponding field from the line above, i.e.: "ATTRIBUTE". (Any LARCRIM command may be continued on the next line, not just so-called text strings.)

```
L>ENDCITY * "THE END CITY OF THE FLIGHT ROUTE."
L>DAYOFWK * "DAY OF THE WEEK THAT THE FLIGHT ROUTE RUNS."
L>PILOT * "THE PILOT'S NAME."
L>RATING * "THE PILOT'S SKILL RATING."
L>AIRPLANS RELATION "RELATION CONTAINING THE AIRPLANE STOCK +
THAT A CARRIER OWNS ALONG WITH ALL PERTINENT DATA ON EACH +
AIRPLANE."
L>AIRLINES * "RELATION CONTAINING THE  AIRLINE'S FLIGHT +
INFORMATION."
L>CREW * "THE CREW FLIGHT INFORMATION."
L>DEFIN * "RELATION CONTAINING THE DEFINITIONS OF ALL THE+
ATTRIBUTES AND THE RELATIONS USED IN THE PLANES DATA BASE."
```

```
L>LOAD AIRPLANS
```

We defined the relation AIRPLANS with 7 attributes: MFG, MODEL, TYPE, NUMENG,ENGTYPE, CRUSALT, & CRUSSPD. Now we must load it in the same order. Note that this relation doesn't need a password; we didn't give it one. Also note that, having loaded one relation, we switch to loading another just by entering a new "load" command. We don't have to leave the LOAD mode.

```
L>BOEING B727-100 PASSENGER 3 JET 39000. 560.
L>BOEING B727-200 *5
L>BOEING B747-200 PASSENGER 4 JET 43000. 580.
L>BOEING B737-200 PASSENGER 2 JET 35000. 590.
L>BOEING B747-SP PASSENGER 4 JET 45000. 600.
L>BOEING B747-F CARGO 4 JET 50000. 650.
L>DOUGLAS DC-9 PASSENGER 2 JET 39000. 550.
L>DOUGLAS DC-10 PASSENGER 3 JET 44000. 590.
L>LOCKHEED L-1011 PASSENGER 3 JET 46000. 550.
L>BURNER BLAIR CROPDUSTER 1 PROP 500. 110.
```

```
L>LOAD AIRLINES

 BEGIN -LARCRIM- DATA LOADING
 -ERROR- UNAUTHORIZED ACCESS TO RELATION AIRLINES
 END -LARCRIM- DATA LOADING
```

Airlines requires a password. We got bumped back to the top level (R>). No sweat! We'll try again the right way.

```
R>USER AGENT


R>LOAD AIRLINES

L>UNITED 58 PORTLAND SEATTLE MON B727-200
L>UNITED 65 SEATTLE PORTLAND MON B727-100
L>UNITED 120 *2 FRI B727-200
L>UNITED 128 PORTLAND SEATTLE FRI B737-200
L>CONT 234 SEATTLE PORTLAND THU DC-9
L>CONT 235 SEATTLE PORTLAND SUN DC-9
L>CONT 187 *2 WED DC-9
L>AIRWEST 18 *2 TUE BLAIR
L>UNITED 140 SEATTLE CHICAGO FRI DC-8
L>AIRWEST 27 *2 SAT BLAIR
L>WESTERN 290 *2 SAT B707-200
L>TWA 576 SEATTLE CHICAGO MON B747-200
L>TWA 578 *2 WED B747-SP
L>TWA 624 *2 SAT B747-200
L>AMERICAN 295 SEATTLE ROCHESTER SUN B727-200
L>AMERICAN 298 ROCHESTER SEATTLE MON B727-200
L>AMERICAN 140 ROCHESTER CHICAGO TUE B727-100
L>AMERICAN 145 CHICAGO SEATTLE TUE B727-200
L>PANAM 245 SEATTLE TOKYO MON DC-10
L>PANAM 247 SEATTLE TOKYO TUE L-1011
L>PANAM 249 SEATTLE TOKYO THU L-1011
L>PANAM 246 TOKYO SEATTLE TUE DC-10
L>PANAM 248 TOKYO SEATTLE WED DC-10
L>PANAM 250 TOKYO SEATTLE SAT DC-10
L>IPAD-AIR -3 RENTON KENT THU BLAIR
L>IPAD-AIR -5 KENT RENTON FRI BLAIR
L>IPAD-AIR -6 LANGLEY RENTON MON BLAIR
L>IPAD-AIR -7 RENTON LANGLEY TUE BLAIR


L>LOAD CREW
L>-3 "FEARLESS FRED" 5
L>-6 "DARING DANIEL" 3
L>-5 "ROARING RALPH" 3
L>295 "DIEHARD DENNIS" 6
L>58 "BERNHARDT BOMBER" 1
L>18 "SMILING JACK" 0

 - ERROR- UNABLE TO PROCESS RULE 2
```

The rating 0 is against the rule "rating gt 0," so we turn off rule checking and bravely press on.

```
L>NOCHECK
L>18 "SMILING JACK" 0
L>27 "STEVE CANYON" 0
L>END
```

```
R>PRINT RULES
```

We can look at the rules only from the LARCRIM main level (R>).

```
 RULE NUMBER      1
 DAYOFWK                NE   SUN
 RULE NUMBER      2
 RATING                 GT            0 AND
 RATING                 LT           10
```

```
R>LISTREL
            EXISTING RELATIONS AS OF 91/10/24    10.33.43

                    DEFIN
                    AIRPLANS
                    AIRLINES
                    CREW
```

A plain LISTREL shows us what relations we have. Adding a relation name to the command gives us an outline of that relation.

```
R>LISTREL AIRLINES

                        RELATION : AIRLINES
     LAST MOD :    91/10/24          READ PASSWORD : YES
     SCHEMA :    PLANES              MODIFY PASSWORD : YES

       NAME            TYPE            LENGTH            KEY

       CARRIER         TEXT          10 CHARACTERS       YES
       FLIGHTNO        INT               1
       STRTCITY        TEXT          10 CHARACTERS
       ENDCITY         TEXT          10 CHARACTERS
       DAYOFWK         TEXT          10 CHARACTERS
       MODEL           TEXT          10 CHARACTERS

          CURRENT NUMBER OF ROWS =         28
```

Display 3 fields from the relation, but limit DESCRIP to 30 characters.

```
R>SELECT WORD WRDTYPE DESCRIP=30 FROM DEFIN
```

WORD	WRDTYPE	DESCRIP
WORD	ATTRIBUTE	NAMES USED IN THE PLANES DATA BASE.
WRDTYPE	ATTRIBUTE	THE TYPE OF THE NAME: ATTRIBUTE OR RELATION.
DESCRIP	ATTRIBUTE	A TEXTUAL DESCRIPTION OF THE WORDS USED IN THE DATA BASE.
MFG	ATTRIBUTE	THE MANUFACTURER OF THE AIRPLANE.
MODEL	ATTRIBUTE	The AIRPLANE MODEL NUMBER.
TYPE	ATTRIBUTE	THE AIRPLANE TYPE, I.E., PASSENGER, CARGO, ETC.
NUMENG	ATTRIBUTE	THE NUMBER OF ENGINES ON THE AIRPLANE.
ENGTYPE	ATTRIBUTE	THE ENGINE TYPE, I.E., JET, PROP, ETC.
CRUSALT	ATTRIBUTE	THE AIRPLANE'S CRUISE ALTITUDE.
CRUSSPD	ATTRIBUTE	THE AIRPLANE'S CRUISE SPEED.
CARRIER	ATTRIBUTE	THE AIRLINE CARRIER WHICH USES THE PLANES.
FLIGHTNO	ATTRIBUTE	THE FLIGHT NUMBER OF A CARRIER'S ROUTE.
STRTCITY	ATTRIBUTE	THE START CITY OF THE FLIGHT ROUTE.
ENDCITY	ATTRIBUTE	THE END CITY OF THE FLIGHT ROUTE.
DAYOFWK	ATTRIBUTE	DAY OF THE WEEK THAT THE FLIGHT ROUTE RUNS.
PILOT	ATTRIBUTE	THE PILOT'S NAME.
RATING	ATTRIBUTE	THE PILOT'S SKILL RATING.
AIRPLANS	RELATION	RELATION CONTAINING THE AIRPLANE STOCK THAT A CARRIER OWNS ALONG WITH ALL PERTINENT DATA ON EACH AIRPLANE.
AIRLINES	RELATION	RELATION CONTAINING THE AIRLINE'S FLIGHT INFORMATION.
CREW	RELATION	THE CREW FLIGHT INFORMATION.
DEFIN	RELATION	RELATION CONTAINING THE DEFINITIONS OF ALL THE ATTRIBUTES AND THE RELATIONS USED IN THE PLANES DATA BASE.

```
R>SELECT ALL FROM AIRPLANS

MFG       MODEL      TYPE        NUMENG ENGTYPE CRUSALT  CRUSSPD
-------   ---------  ---------   ------ ------- -------  -------
BOEING    B727-100   PASSENGER        3 JET     39000.   560.
BOEING    B727-200   PASSENGER        3 JET     39000.   560.
BOEING    B747-200   PASSENGER        4 JET     43000.   580.
BOEING    B737-200   PASSENGER        2 JET     35000.   590.
BOEING    B747-SP    PASSENGER        4 JET     45000.   600.
BOEING    B747-F     CARGO            4 JET     50000.   650.
DOUGLAS   DC-9       PASSENGER        2 JET     39000.   550.
DOUGLAS   DC-10      PASSENGER        3 JET     44000.   590.
LOCKHEED  L-1011     PASSENGER        3 JET     46000.   550.
BURNER    BLAiR      CROPDUSTER       1 PROP    500.     110.
```

```
R>EXIT
```

Let's stop by entering EXIT, then reenter the database.

```
eagle% LARCRIM

 BEGIN LARCRIM -CONVEX   VERSION 1.0   UD23  91/10/24    10.35.23

                (Case insensitive, except for DBname)

 LARCRIM COMMAND MODE
 ENTER "MENU" FOR MENU MODE

R>INPUT CMDFILE
```

Optional input method: This would pick up a file of LARCRIM commands and execute them immediately. But we want to see what's going on so we'll reset from the input to come from the keyboard.

```
R>INPUT TERMINAL

R>OPEN PLANES
```

The DB already exists, so we just open it. Since we defined it with all caps, we must forever more refer to it the same way.

R>**USER AGENT**

R>**SELECT ALL FROM AIRPLINES SORTED BY STRTCITY**

CARRIER	FLIGHTNO	STRTCITY	ENDCITY	DAYOFWK	MODEL
AMERICAN	145	CHICAGO	SEATTLE	TUE	B727-200
UNITED	58	PORTLAND	SEATTLE	MON	B727-200
UNITED	128	PORTLAND	SEATTLE	FRI	B737-200
AMERICAN	298	ROCHESTER	SEATTLE	MON	B727-200
AMERICAN	140	ROCHESTER	CHICAGO	TUE	B727-100
UNITED	65	SEATTLE	PORTLAND	MON	B727-100
UNITED	120	SEATTLE	PORTLAND	FRI	B727-200
CONT	234	SEATTLE	PORTLAND	THU	DC-9
CONT	187	SEATTLE	PORTLAND	WED	DC-9
HUGHES	18	SEATTLE	PORTLAND	TUE	BLAIR
UNITED	140	SEATTLE	CHICAGO	FRI	DC-8
HUGHES	27	SEATTLE	CHICAGO	SAT	BLAIR
WESTERN	290	SEATTLE	CHICAGO	SAT	B707-200
TWA	576	SEATTLE	CHICAGO	MON	B747-200
TWA	578	SEATTLE	CHICAGO	WED	B747-SP
TWA	624	SEATTLE	CHICAGO	SAT	B747-200
PANAM	245	SEATTLE	TOKYO	MON	DC-10
PANAM	247	SEATTLE	TOKYO	TUE	L-1011
PANAM	249	SEATTLE	TOKYO	THU	L-1011
PANAM	246	TOKYO	SEATTLE	TUE	DC-10
PANAM	248	TOKYO	SEATTLE	WED	DC-10
PANAM	250	TOKYO	SEATTLE	SAT	DC-10

R>**SELECT ALL FROM AIRPLANS WHERE MFG EQ BOEING AND NUMENG GE 3**

MFG	MODEL	TYPE	NUMENG	ENGTYPE	CRUSALT	CRUSSPD
BOEING	B727-100	PASSENGER	3	JET	39000.	560.
BOEING	B727-200	PASSENGER	3	JET	39000.	560.
BOEING	B747-200	PASSENGER	4	JET	43000.	580.
BOEING	B747-SP	PASSENGER	4	JET	45000.	600.
BOEING	B747-F	CARGO	4	JET	50000.	650.

R>**OUTPUT TEMP1**

This reroutes the output from subsequent commands to the file TEMP1.

```
R>SELECT CARRIER DAYOFWK FROM AIRLINES WHERE ENDCITY EQ PORTLAND

CARRIER         DAYOFWK
-------         -------
UNITED          MON
UNITED          FRI
CONT            THU
CONT            WED
HUGHES          TUE

R>OUTPUT TERMINAL
```

Reroute output back to terminal.

```
R>TALLY STRTCITY FROM AIRLINES
```

Display each city with departing flights and the number of such flights.

```
STRTCITY        NUMBER OF OCCURRENCES
----------      --------------------
CHICAGO                          1
PORTLAND                         2
ROCHESTER                        2
SEATTLE                         14
TOKYO                            3

R>TALLY FLIGHTNO=D FROM AIRLINES WHERE FLIGHTNO GT 140

FLIGHTNO    NUMBER OF OCCURRENCES
--------    --------------------
     624                       1
     578                       1
     576                       1
     298                       1
     290                       1
     250                       1
     249                       1
     248                       1
     247                       1
     246                       1
     245                       1
     234                       1
     187                       1
     145                       1
     140                       2
     128                       1
```

This time, use FLIGHTNO to sort the list in descending order.

The PROJECT command allows you to build a "subset" relation when the full data set is not required.

```
R>PROJECT BOEINGPL FROM AIRPLANS USING MODEL TYPE NUMENG +
  CRUSALT CRUSSPD WHERE MFG EQ BOEING

 SUCCESSFUL PROJECT OPERATION       6 ROWS GENERATED

R>LISTREL BOEINGPL
                    RELATION : BOEINGPL
     LAST MOD :    91/10/24          READ PASSWORD : YES
     SCHEMA :    PLANES              MODIFY PASSWORD : NONE

       NAME           TYPE           LENGTH            KEY

       MODEL          TEXT         10 CHARACTERS
       TYPE           TEXT         10 CHARACTERS
       NUMENG         INT              1
       CRUSALT        REAL             1
       CRUSSPD        REAL             1

          CURRENT NUMBER OF ROWS =         6

R>SELECT MODEL CRUSALT CRUSSPD FROM BOEINGPL

 MODEL        CRUSALT   CRUSSPD
 ----------   --------  --------
 B727-100     39000.    560.
 B727-200     39000.    560.
 B747-200     43000.    580.
 B737-200     35000.    590.
 B747-SP      45000.    600.
 B747-F       50000.    650.
```

```
R>INTERSECT AIRPLANS WITH AIRLINES FORMING FLIGHTS

 SUCCESSFUL INTERSECT OPERATION        24 ROWS GENERATED
```

The INTERSECT operation builds a new relation from selected attributes of others.

```
R>LISTREL FLIGHTS
                    RELATION : FLIGHTS
     LAST MOD :    91/10/24          READ PASSWORD : YES
     SCHEMA :    PLANES              MODIFY PASSWORD : YES

       NAME          TYPE          LENGTH          KEY

       MFG           TEXT        10 CHARACTERS
       PLANE         TEXT        10 CHARACTERS
       TYPE          TEXT        10 CHARACTERS
       NUMENG        INT           1
       ENGTYPE       TEXT        10 CHARACTERS
       CRUSALT       REAL          1
       CRUSSPD       REAL          1
       CARRIER       TEXT        10 CHARACTERS
       FLIGHTNO      INT           1
       STRTCITY      TEXT        10 CHARACTERS
       ENDCITY       TEXT        10 CHARACTERS
       DAYOFWK       TEXT        10 CHARACTERS

           CURRENT NUMBER OF ROWS =        24

R>SELECT CRUSALT FROM FLIGHTS WHERE STRTCITY EQ ROCHESTER AND +
ENDCITY EQ SEATTLE

 CRUSALT
 -------
 39000.

R>TALLY MFG FROM FLIGHTS

 MFG           NUMBER OF OCCURRENCES
 ----------    ---------------------
 BOEING                 10
 BURNER                  2
 DOUGLAS                 6
 LOCKHEED                2
```

```
R>SUBTRACT AIRLINES FROM AIRPLANS FORMING LEFTOVER

 SUCCESSFUL SUBTRACT OPERATION        1 ROWS GENERATED
```

The SUBTRACT command finds rows in one relation that don't match any rows in the other relation.

```
R>SELECT ALL FROM LEFTOVER

MFG       MODEL      TYPE      NUMENG  ENGTYPE CRUSALT CRUSSPD
------    ------     -----      -----   ------- ------- -------
BOEING    B747-F     CARGO     4        JET      50000.  650.

R>USER AGENT
R>CHANGE CARRIER TO HUGHES IN AIRLINES WHERE CARRIER EQ AIRWEST
R>RENAME ATTRIBUTE DAYOFWK TO DAY IN FLIGHTS
R>RENAME ATTRIBUTE MODEL TO PLANE IN FLIGHTS

R>EXHIBIT DAY
 RELATIONS CONTAINING DAY
      AIRLINES
      FLIGHTS
```

How many relations use the attribute "DAY?"

```
R>RENAME RELATION DEFIN TO DATADICT
R>LISTREL DATADICT
                    RELATION : DATADICT
      LAST MOD :    91/10/24          READ PASSWORD : YES
      SCHEMA :    PLANES              MODIFY PASSWORD : YES

        NAME          TYPE            LENGTH            KEY

        WORD          TEXT            8 CHARACTERS      YES
        WRDTYPE       TEXT            9 CHARACTERS
        DESCRIP       TEXT            VARIABLE

            CURRENT NUMBER OF ROWS =         21
```

R>**PROJECT BOEINGPL FROM AIRPLANS USING ALL WHERE MFG EQS "BOE"**

SUCCESSFUL PROJECT OPERATION 6 ROWS GENERATED

R>**SELECT ALL FROM BOEINGPL**

MODEL	TYPE	NUMENG	CRUSALT	CRUSSPD
B727-100	PASSENGER	3	39000.	560.
B727-200	PASSENGER	3	39000.	560.
B747-200	PASSENGER	4	43000.	580.
B737-200	PASSENGER	2	35000.	590.
B747-SP	PASSENGER	4	45000.	600.
B747-F	CARGO	4	50000.	650.

R>**EXIT**

Standard exit. This works from the main level (R>). If it doesn't seem to work, you might be in a submode (Load, Define, etc.). Try "end", then "exit".

END LARCRIM EXECUTION 91/10/24 10.44.52

2.0 GENERAL COMMAND SYNTAX

LARCRIM is used by entering commands in response to input prompts. Each command begins with a keyword followed by the information needed to complete the command specification. Three of the commands (DEFINE, LOAD, and HELP), are used to enter sublevels which have their own sets of commands for defining a database, loading a database, and providing help to the user. The input prompt indicates which commands are expected: D> indicates the define sublevel commands, L> the load sublevel commands, H> the help sublevel commands, and R> for all other commands. In describing the LARCRIM commands, the following conventions are used:

```
relname                      name of a relation(s)
 or
relname1,relname2,...        (1-8 alphanumeric characters)

attname                      name of an attribute(s)
 or
attname1,attname2,...        (1-8 alphanumeric characters)

value                        actual value(s)
 or                          (value may be a text string,
value1,value2,...            scalar, vector, or matrix)
```

All relation and attribute names must contain at least 1 and no more than 8 alphanumeric characters. The first character must be alphabetic.

Many of the commands in LARCRIM have optional parts. These optional parts are enclosed in square brackets.

```
[THIS IS OPTIONAL]
```

Some keywords in the commands are selected from a list of acceptable keywords. These keywords are in a vertical list with the first choice enclosed in braces.

```
{CHOOSE}
 ONE
 OF
 THESE
```

LARCRIM command keywords may be abbreviated. However, at least the first 3 characters in the keyword are required.

The following keywords are interpreted as equivalent:

```
1) SELECT, SELEC, SEL

2) INTERSECT, INTER, INT

3) DELETE DUPLICATES, DEL DUPLICATES, DEL DUP
```

All LARCRIM commands are entered in a free-field format with blanks or commas as item separators. Each command may contain up to 100 items. This section presents a short description of LARCRIM conventions and data generation facilities. A more extensive description, intended for the experienced LARCRIM user, is found in Appendix E.

Keywords and data values must be separated by at least one blank or comma. If a command is too long for one 80 character line, it may be continued on succeeding lines by entering "+" as the last character of the preceding line. LARCRIM also remembers the previous command, which enables you to reuse all or part of it. An asterisk is used to indicate which item(s) of the previous command are to be reused. A single asterisk means to reuse the corresponding single item. An asterisk followed by a number *n* means to reuse the next *n* corresponding items. Two asterisks mean to reuse all remaining corresponding items.

The following are all equivalent:

```
1) THIS IS A COMMAND

2) THIS +
   IS +
   A +
   COMMAND

3) * IS, A COMMAND

4) THIS *2 COMMAND

5) THIS **
```

Multiple commands may be entered on one line separated by a semicolon or dollar sign.

```
THIS IS THE FIRST ; THIS IS THE SECOND $ THIS IS THE THIRD
```

Comments may be placed anywhere within a command by enclosing the comment between the characters *(and).

```
*(THIS IS A COMMENT) THIS IS NOT
```

When numeric data is to be interpreted as text (alphanumeric) data, the numerals must be enclosed by quotation marks.

```
"1234"
```

When entering text strings which contain embedded blanks or commas, the entire string must be enclosed by quotation marks.

```
"THIS IS A TEXT STRING"
```

When entering a text string that contains an item enclosed by quotation marks, double quotation marks are used.

```
"THERE IS A ""QUOTE"" IN THIS STRING"
```

A text string may require continuation on one or more lines. The + sign continuation can then be used within the quotation marks. The use of leading blanks in text strings is not recommended and trailing blanks are ignored.

Integer data is input as a string of digits without a decimal point. A sign may precede the digits.

```
123, -63, +56, 0
```

Real or floating point numbers must include a decimal point or E for the exponent. If a decimal point is not present, the E must be preceded by an integer.

```
1.3, .005, 0., 6.E-1, 6E-1, 0.60, -23.45
```

The absolute value of real numbers is limited to the range between 1.0E-38 and 1.0E+38. Any real number less than 1.0E-38 in absolute value will be interpreted as 0.

2.1 LARCRIM Execution

Execution of LARCRIM as a stand-alone program can be initiated in either of two modes: command mode or menu mode. The command mode is used when LARCRIM is executed in the batch environment or for interactive users who wish to bypass the menu dialog. A detailed description of the commands are given in section 2.2. The menu mode offers assistance to inexperienced users. An overview of this mode is discussed in section 2.3 and a more detailed description of the menu mode dialog is given in section 2.4. The learning curve for a new user to become capable of using the command mode is not particularly long, and the user may switch freely between menu mode and command mode.

When executing LARCRIM in the stand-alone mode, the opening command is:

```
larcrim
```

and the first output will be:

```
BEGIN LARCRIM ---- CONVEX VERSION 1.0  UDXX   YY/MM/DD HH.MM.SS

     LARCRIM COMMAND MODE
     ENTER "MENU" FOR MENU MODE
```

UDXX identifies the update level of LARCRIM and is not significant at this point. The date and time stamp indicate current date and time. At the start of execution you are in the command mode but you may switch immediately to the menu mode as indicated. And, of course, "CONVEX" would become the name of the current computer, if not Convex.

2.2 LARCRIM Commands

This section presents a summary of the LARCRIM commands. All of the commands discussed are available if no passwords are assigned to the relations and if the "owner" is known. If passwords are assigned, the use of certain commands is restricted. In addition, certain commands are available only to the database owner. See the password/command matrix below.

		CURRENT PASSWORD			
SECTION	COMMAND	OWNER	MODIFY	READ	NONE
2.2.1	Defining a database schema	X			
2.2.2	Loading a relation	X	X		
2.2.3	Querying a relation	X	X	X	
2.2.4	Querying the schema	X	X*	X*	
2.2.5	Computation command	X	X	X	
2.2.6	Modification commands	X	X**		
2.2.7	Relational Algebra commands	X	X		
2.2.8	Report commands	X	X	X	
2.2.9	Key commands	X	X		
2.2.10	LARCRIM to LARCRIM commands	X	X		
2.2.11	General commands	X	X	X	X

* Except print rules **Except change owner

Password/Command Matrix

2.2.1 Defining a Database Schema

The DEFINE sublevel (prompt= D>) commands are used to define the structure of the database and are used in the sequence described below. The definition of the database is called the schema. The schema name corresponds to the name of the database and is used to form the names of the files used for the database. A schema includes attributes, relations, passwords, and rules (constraints) all of which are defined using this sublevel. To access the DEFINE sublevel enter:

```
DEFINE dbname
```

You must identify the name of the database whose definition you are going to create or expand by specifying "dbname", the 1-6 character alphanumeric schema name. The first character in "dbname" must be alphabetic. This name is used to form the name of the files used to store the database. The dbname, when augmented with a single number must be a legal filename. Once dbname is specified you must identify the owner password of the database.

```
OWNER password
```

The "password" entered must be a 1-8 character alphanumeric name. When used in the USER command, this password represents a master password to the database. It will override any individual relation passwords. If the database already exists and you want to define additional attributes or relations, "password" is checked against the existing owner password. If security is not an issue, you may specify "none" as the password.

Following the entry of the database name and owner password, the attributes, relations, relation passwords, and rules are defined. See the following section.

2.2.1.1 Defining Attributes

To enter the DEFINE submode, enter:

```
DEFINE
```

Then, to define some attributes, enter:

```
ATTRIBUTES

attname type1 [{length}] [KEY]
                  VAR

attname type2 [{row, col}][KEY]
                 row, VAR
                 VAR, VAR
    .
    .
    .
```

The attributes definitions are ended when you specify one of the DEFINE sublevel keywords RELATIONS, PASSWORDS, RULES, or END.

TYPE1 Attributes

LARCRIM supports seven "type1" data types: real (floating point), integer, text, double precision, real vectors, integer vectors and double precision vectors. You must enter REAL, INT, TEXT, DOUB, RVEC, IVEC or DVEC for type 1. The default length is one value except for TEXT which is 8 characters. The length is specified in terms of the number of values and characters respectively. VAR indicates variable length. The optional KEY specification causes an index file to be built for the attribute. This file is used by LARCRIM to quickly find qualifying rows for retrievals and updates. If KEY is not specified (nonkey attribute) the index file is not built. You should consider the cost of building and storing KEY attribute pointers on the index file versus the benefits you will obtain when deciding if a KEY declaration should be used. No specific rules are given here, experience should be used as a guide. An attribute can be changed from KEY to non-KEY or vice-versa by using the BUILD KEY and DELETE KEY commands described in section 2.2.9. For large databases (more than 1,000 rows), experience has shown that it is most efficient not to specify a KEY in the DEFINE sublevel but rather to load the data without keys and then build the index files using the BUILD KEY command. The greater the number of keys, the more efficient this method is.

TYPE2 Attributes

LARCRIM supports three "type2" data types: real matrices, integer matrices or double precision matrices. You must enter RMAT, IMAT or DMAT for type2. The matrices can be of fixed size, have variable column dimensions or variable row and column dimensions. You enter the row dimension first, followed by the column dimension. The default dimension is 1 by 1. The keyword KEY has the same meaning as for "type1" attributes.

2.2.1.2 Defining Relations

To define relations, while still in the DEFINE mode, enter:

```
RELATIONS

relname WITH attname1 [attname2.... ]
   •
   •
```

The relation definitions are ended by specifying one of the DEFINE sublevel keywords ATTRIBUTES, PASSWORDS, RULES, or END.

The attributes must be listed in the order in which they are to appear in the relation. No attributes can be used which have not been previously defined, either in the current attributes definition subsection or in a previous definition of this database. Attributes which are defined but not included in a relation will not become part of the LARCRIM schema.

2.2.1.3 Password Definition

A LARCRIM database must have attributes and relations defined, but passwords and constraint rules are optional. If read or modify passwords are desired enter:

```
PASSWORDS
{READ PASSWORD} FOR relname IS password
 RPW

{MODIFY PASSWORD} FOR relname IS password
 MPW
   •
   •
   •
```

The password definitions are ended by specifying one of the DEFINE sublevel keywords ATTRIBUTES, RELATIONS, RULES, or END.

A password can be any string of 1-8 alphanumeric characters. When you are doing queries, loads, or modifications, the current password is specified by the USER command. If this password does not match the read, modify, or owner password for a given relation, you cannot query that relation. If this password does not match the modify or owner password, you cannot load or modify the relation.

2.2.1.4 Constraint Rule Definition

Constraint rules are another optional section of the DEFINE sublevel. If rules are specified, they are used during the loading process or during CHANGE commands to screen out rows which do not meet the constraint rules. Rules may be defined for REAL, INT, and DOUB attributes of length 1 and for fixed length TEXT attributes. At most, 10 rules may be specified for a single relation.

To define constraint rules enter:

```
RULES

attname [IN relname]{ EQ} value [{AND} attname... ]
                     NE          OR
                     GT
                     GE
                     LT
                     LE

or

attname1 IN relname { EQA} attname2 IN relname [{AND}...]
                     NEA                          OR
                     GTA
                     GEA
                     LTA
                     LEA
```

```
where   EQ   =   Equals
        NE   =   Not equal to
        GT   =   Greater than
        GE   =   Greater than or equal to
        LT   =   Less than
        LE   =   Less than or equal to
        EQA  =   Equals attribute
        NEA  =   Not equal to attribute
        GTA  =   Greater than attribute
        GEA  =   Greater than or equal to attribute
        LTA  =   Less than attribute
        LEA  =   Less than or equal to attribute
```

The rule definitions are ended by specifying one of the DEFINE sublevel keywords ATTRIBUTES, RELATIONS, PASSWORDS, or END.

Attributes referenced in the rule definitions must have been previously defined. By specifying rules, you can restrict an attribute to a range of values or require that the value of an attribute in one relation have a specified relationship to the values of an attribute in the same or a different relation. The comparison operators ending in "A" are used when the comparison is to existing attribute values rather than to a specified constant. A rule expression may contain a maximum of 9 Boolean operators.

The method used for constraint checking is that the value of the first attribute mentioned in the rule is obtained from the input (LOAD or CHANGE command) and checked against the value specified or against the existing database values of the second attribute.

To complete the schema definition and leave the DEFINE sublevel you enter:

```
END
```

Example of DEFINE sublevel commands

```
DEFINE LRCDB
OWNER ME
ATTRIBUTES
   MODEL TEXT KEY
   WEIGHT REAL
   NUMPASS INT
   CARRIER TEXT 16
   FLIGHTNO INT
   NAME TEXT KEY
   AGE INT
   MATRIX IMAT 4,VAR
RELATIONS
   AIRPLANE WITH MODEL WEIGHT NUMPASS
   FLIGHTS WITH CARRIER FLIGHTNO MODEL MATRIX
   PEOPLE WITH NAME AGE
   PASSWORDS
   MPW FOR FLIGHTS IS AGENT
   RPW FOR PEOPLE IS BLUE
RULES
   MODEL IN FLIGHTS EQA MODEL IN AIRPLANE
   AGE GT 21 AND AGE LT 65
   NUMPASS IN AIRPLANE LE 350
END
```

2.2.2 Loading a Relation

The LOAD sublevel (prompt=L>) commands are used to add data to a newly defined relation or to a relation which already contains data. To access this sublevel enter:

```
LOAD relname
```

You may now load data into the relation, one row at a time, by entering data values in an order corresponding to the attribute order:

```
value1 value2 ... valuen
```

The syntax of the input values for the different attribute types is shown below:

Attribute Type	Length or Dimension	Valuei	Remark
REAL, INT DOUB, RVEC IVEC, DVEC	n n>1	(val1 ... valn)	Parentheses optional
REAL, INT DOUB, RVEC IVEC, DVEC	VAR	(val1 val2 ...)	Parentheses required
TEXT	any	"text string"	In special cases, (see Section 2.0) quotes are optional.
RMAT, IMAT DMAT	m,n	((r1c1...rmc1)(r1c2...)+ ...rmcn))	Columnwise Parentheses optional
RMAT, IMAT	m, VAR VAR, VAR	((r1c1...)(r1c2...)(...))	Columnwise Parentheses required

When data loading is complete you enter:

```
END
```

Multiple relations may be loaded from within the LOAD sublevel by re-entering the LOAD command instead of the END command.

Example of LOAD sublevel commands:

```
USER AGENT
LOAD AIRPLANE
"757" 120000. 150
DC9 87000. 110
747SP 200000. 350
LOAD PEOPLE
BOB 30
JOE 32
ALICE 29
LOAD FLIGHTS
UAL 16 "757" ((1,2,3,4)(4,5,6,7))
*3 ((2,4,5,8)(1,2,3,4)(4,5,6,7))
END
```

If the value for an attribute is unknown, you enter the characters -0- for the missing value or use two successive commas.

```
L1011 -0- -0- 250
L1011,,,250
```

These two examples have identical meaning.

2.2.3 Querying A Relation

SELECT

The SELECT command is used for displaying or printing data from a relation. The options available for this command are discussed in the following paragraphs.

To print all the data from a relation:

```
SELECT ALL FROM relname
```

To print selected attribute values from all rows in a relation:

```
SELECT attname1 [ attname2 ... attnamen] FROM relname
```

The above command will print up to 20 attributes in any order. The number of attributes may also, in a practical sense, be limited by space available in a line. As a general rule, 7 attributes may be selected when running at an 80 character interactive terminal and 11 attributes when running in the batch mode or at a 132 character terminal.

For variable length attributes or for attributes of fixed length that would otherwise not fit on a line alone or together with other attributes, you may format the output using the optional field width control:

```
SELECT attname1 [=fw1] [attname2 [=fw2]...]+

FROM relname
```

fw1 is the output field width for attname1. For a text type attribute, fw1 is the width of the output paragraph in number of characters, for other attribute types it is the number of values.

When the field width option is used, LARCRIM will use as many output lines as required for each row of data.

The field width defaults depend on the attribute type. For a fixed length attribute, other than matrix, no paragraphing is attempted and the default paragraph width is a full row. The system will use a field width equal to the attribute length or column dimension to display the value(s) of the attribute. For a variable length attribute of type TEXT, the default field width is 40 characters with paragraphing. For variable length attributes of types REAL, INT, DOUB, the default is 4 values with paragraphing. For variable length vector type attributes, the default is a display of the length and 3 values with paragraphing. For variable length matrix attributes the default is a display of the row and column dimensions and 3 values with paragraphing. Each row starts on a new line.

If a field width is specified, the system will display the dimension of variable length vectors and matrices and use one of the specified output positions for these values. However, if a field width of 1 is specified for such an attribute, the row and column dimensions will not be displayed.

When paragraphing TEXT type attributes, LARCRIM will identify substrings of text separated by blanks. The substring is placed on the current line if there is space available. If there is not space available and the current line contains less than four characters, the number of characters that fit on the line are put on the line (without hyphen) and the substring continued on the next line. If the current line contains more than four characters, the entire substring will be placed on the next line.

If the sum of the lengths (or field widths) of the attributes requested exceeds the line capacity, the data line will be truncated and a warning message displayed.

Further information about line width, number of lines per page, defaults and use specifications is given in the section on the LARCRIM report writing features (section 2.2.8).

Examples of SELECT command:

```
SELECT ivecvar FROM rel1

 DIM              IVECVAR
 ---              --------------------
  7                 1         2         3
                    4         5         6
                    7
  1                 10

SELECT imatvv FROM rel1

 ROW       COL      IMATVV
 ---       ---      --------------------
  2         5        11        12        13
                     14        15
                     21        22        23
                     24        25
  1                  1         11

SELECT textv=9 FROM rel1

TEXTV
---------
THIS IS
AN EXAMPL
E OF
WRAPAROUN
D OF TEXT
THIS IS
ANOTHER
EXAMPLE
OF TEXT
```

When entering the SELECT command, the attribute name attname1 may be replaced by its corresponding attribute number (attnum1). The attribute number is determined by the position of the attribute in the relation. Specific elements of a vector or a matrix may also be designated. The general form of the unconditional SELECT command is:

```
SELECT {attname1 [=fw1]} [attname2 [=fw2]...] +
        attnum1 [=fw1]
        attname1(i)
        attname1(i,j)
        attnum1(i)
        attnum1(i,j)
        ALL
FROM  relname
```

To print all attributes from a relation where certain conditions are met:

```
SELECT ALL FROM relname WHERE condition1 [{AND} +
                                          OR

condition2....]
```

Up to 10 conditions may be combined using the Boolean operators AND/OR. A LIMIT condition, if used, does not count as one of the 10 conditions. The conditions are combined from left to right. Each condition may be one of the following forms:

```
attname EXISTS
attname FAILS
attname EQ MAX
attname EQ MIN
attname EQ value
attname EQS value
attname NE value
attname GT value
attname GE value
attname LT value
attname LE value
attname EQ list
attname EQS list
attname NE list
attname1 EQA attname2
attname1 NEA attname2
attname1 GTA attname2
attname1 GEA attname2
attname1 LTA attname2
attname1 LEA attname2
ROWS EQ rownumber
ROWS NE rownumber
ROWS GT rownumber
ROWS GE rownumber
ROWS LT rownumber
ROWS LE rownumber
```

```
ROWS EQ list
ROWS NE list
LIMIT EQ number

where:
     EQ    =   Equals
     EQS   =   Contains the text string
     NE    =   Not equals
     GT    =   Greater than
     GE    =   Greater than or equal to
     LT    =   Less than
     LE    =   Less than or equal to
     EQA   =   Equals attribute
     NEA   =   Not equals attribute
     GTA   =   Greater than attribute
     GEA   =   Greater than or equal to attribute
     LTA   =   Less than attribute
     LEA   =   Less than or equal to attribute
     MAX   =   Maximum value
     MIN   =   Minimum value
```

Attname, attname1, attname2 may not refer to an element of a vector or a matrix, e.g., vec(3) or mat(2,3).

EXISTS will qualify those attributes which have been assigned a value. FAILS will qualify those attributes which have not been assigned a value (loaded with -0-).

MAX and MIN comparison can only be made for integer, real and double precision attributes of fixed length equal to 1.

"VALUE" in a comparison statement must follow the syntax of section 2.2.2 for vectors and matrices, i.e., if the attribute is of variable length or dimension, parentheses must be used to input a vector or a matrix value or a list of vector and matrix values.

EQS applies to text strings only. In such a comparison, "value" is a text string and the comparison is true if "value" is found as a substring anywhere within the requested attribute values.

NE comparison when applied to vectors or matrices is true if the length or dimension is different from the length or dimension of your specified vector or matrix "value" or if any vector or matrix element differs from the elements of the "value" specified.

GT and LT comparisons for vector and matrix attributes are "lexicographical", i.e., a comparison is made element by element (columnwise for matrices) and continued until a non-equal condition is detected. If no such condition is detected after the last element is checked, a false condition is assumed. Comparisons are made only for data rows where the dimensions of the attribute are the same as the dimensions of the "value" specified. For all other rows, a false condition is assumed.

GE and LE comparisons for vector and matrix attributes are similar to GT and LT comparisons except when an equal condition is detected after the last element is checked, a true condition is assumed.

Comparison rules for vector attributes also apply to real, integer and double precision attributes of fixed or variable length.

A "list" is a simple list of values: a1, a2, a3,...,an. An EQ or EQS list has the same effect as a series of OR conditions. A NE list has the same effect as a series of AND conditions. Note that the *=N+STEP option available through the LARCRIM user interface provides a more powerful list option (see Appendix E). For example, if you want a list, 0 to 100, in equal increments of 20, you may enter 0 20 40 60 80 100 or 0 *=5+20.

The comparison key words ending in A (EQA etc.) are used when comparing the value of one attribute to the value of another attribute in the same row of the relation.

ROWS refer to row numbers in a relation. Note that a relation is loaded in input order but that subsequent operations (modifications) to the database may cause the order of the rows to change.

When the LIMIT clause is used, only the first "number" of rows that qualify will actually be displayed.

Processing the WHERE condition can be speeded up greatly if key processing is used. Key processing involves using the pointers created for KEY attributes rather than looking at each row of a relation to find the rows qualified by the WHERE conditions. Key processing will be used when the following are all true:

1) The last condition uses an attribute which is KEY

2) The last condition uses EQ

3) The last condition is combined with the other conditions by AND.

The output can be sorted by specifying the attributes to sort on and the sorting order for each attribute. The sorting order default is lowest to highest.

```
SELECT ... FROM relname +
SORTED BY attname1 [{=A}] [attname2 [{A}=]...] +
                          D                  D
WHERE ... ]
```

A or D is entered to request ascending or descending sort. If a sort on more than one attribute is requested, the output will first be ordered according to the first sort attribute, the corresponding rows will be ordered by the second sort attribute, duplicates within this by the third and so on. A maximum of 5 attributes may be used, and ascending or descending order may be used in any combination. Variable length attributes may not be used as sort attributes. When fixed length attributes of length greater than 8 characters or 1 value are used as sort attributes, only the first 20 characters or the first value is used.

All the SELECT options can be described using the following general syntax:

```
SELECT {attname1 [=fw1] [... attnamen [=fwn]]} +
        attnum1...
        attname1(i)...
        attname1(i,j)...
        attnum1(i)...
        attnum1(i,j)...
        ALL
FROM relname +
[SORTED BY attname [{=A}]...] +
                    D
[WHERE condition1 [{=AND} condition2 ... ]]
                    OR
```

TALLY

The TALLY command prints the number of times each unique value of an attribute occurs in a relation. The order, ascending or descending, of the tally is user specified. Default is ascending. The WHERE clause is optional and uses the same syntax as in the SELECT command.

```
TALLY attname [{=A}] FROM relname [WHERE ... ]
                D
```

Examples of SELECT and TALLY commands:

```
SELECT ALL FROM AIRPLANE
SELECT MODEL FROM AIRPLANE
SELECT ALL FROM AIRPLANE WHERE WEIGHT GT 100000 +
AND NUMPASS LT 200
SELECT AGE FROM PEOPLE WHERE NAME EQ BOB
SELECT ALL FROM AIRPLANE SORTED BY MODEL=D
```

```
TALLY MODEL FROM FLIGHTS
TALLY MODEL FROM FLIGHTS WHERE CARRIER EQ UNITED
SELECT ALL FROM DIMENS WHERE HEIGHT GTA WIDTH
SELECT FILE TITLE=4 OWNER FROM PFDATA
```

2.2.4 Querying the Schema

When you use these commands, LARCRIM will display only the data you are authorized to access according to your current user password.

LISTREL

The purpose of LISTREL is to provide you with information about the relations in the database.

There are three options for the LISTREL command. The first consists of simply entering:

```
LISTREL
```

This option provides you with a list of all relations currently defined in your database. If you wish to display information about a specific relation, enter:

```
LISTREL relname
```

This option provides you with a display of the definition of the requested relation and a count of the number of rows of data in the relation.

```
LISTREL ALL
```

This option will display the relation definitions and count of the number of rows in each relation for all relations in the database that you are permitted to access.

EXHIBIT

The purpose of the EXHIBIT command is to allow you to query the LARCRIM schema to obtain the names of all relations having a specific set of attributes. For example, if you want to know which relations contain the attribute "attname" you would enter:

```
EXHIBIT attname
```

You would then obtain either a list of the relations having this attribute, or a message indicating that this attribute was not found in any relations in the database.

The EXHIBIT command also allows you to obtain information for a list of attributes (maximum of 10). Suppose that you want to know which relations contain both attname1 and attname2. The command would be:

```
EXHIBIT attname1 attname2 ....
```

The general syntax of this command is:

```
EXHIBIT attname1 [attname2... attnamen]
```

PRINT RULES

This command is used to obtain a complete list of the constraint rules.

```
PRINT RULES
```

PRINT RULES can be used only when the current user password matches the owner password of the database definition.

2.2.5 Computation Command

COMPUTE

The COMPUTE command is used to compute simple functional values of an attribute. The WHERE clause is optional and uses the same syntax as is used in the SELECT command.

```
COMPUTE {COUNT} attname FROM relname [WHERE ... ]
        MIN
        MAX
        AVE
        SUM
```

There are some restrictions as to the type and word length of the attribute used for these functions. All functions except COUNT exclude any -0- values when making their computations. The following table describes the attribute type and length restrictions for each function:

FUNCTION	ATTRIBUTE	ATTRIBUTE LENGTH
COUNT	any	any
MIN	any of fixed length	1 for non-text, ANY for text
MAX	any of fixed length	1 for non-text, ANY for text
AVE	int, real, double	1
SUM	int, real, double	1

Examples of COMPUTE command:

```
COMPUTE AVE NUMPASS FROM FLIGHTS
COMPUTE MAX WEIGHT FROM FLIGHTS WHERE NUMPASS LT 100
COMPUTE COUNT NAMEM FROM PEOPLE WHERE AGE GT 30
```

2.2.6 Modification Commands

These commands are used to change the contents of the database or the database definition. Your password must match the modify or owner password in order to use these commands. If the CHANGE OWNER command is used, the password must match the owner password.

CHANGE (Attribute values)

The CHANGE command is used to change the value of an attribute in a relation where certain conditions are met.

```
CHANGE {attname1} TO value [IN relname] WHERE ...
       attname(i)
       attname(i,j)
```

The "value" entered has the same syntax as described in Section 2.2.2. The WHERE clause is required and uses the same syntax as in the SELECT command. If the relation name is not specified, the attribute is changed in all relations where the attribute is found and the WHERE clause conditions are met.

CHANGE (Passwords)

The read or modify passwords may be changed by using the following command:

```
CHANGE {RPW} TO newpass FOR relname
       MPW
```

CHANGE OWNER

The CHANGE OWNER command is used to change the name of the database owner password. Only the current owner is allowed to use this command.

```
CHANGE OWNER TO newowner
```

DELETE ROWS

The DELETE ROWS command is used to delete selected rows in a relation.

```
DELETE ROW FROM relname WHERE ...
```

The name of the relation must be specified as well as a WHERE clause. The syntax for the WHERE clause is the same as in the SELECT command.

DELETE DUPLICATES

This command is used to remove any duplicate rows from a relation. It is particularly useful on relations which have been created by any of the relational algebra commands (JOIN,INTERSECT, SUBTRACT, or PROJECT). The syntax for this command is:

```
DELETE DUPLICATES [attname1 attname2 ...] from relname
```

Duplicates are checked only for the specified attribute(s). Default is to check the complete row (all attributes).

DELETE RULE

The DELETE RULE command is used to delete a specified constraint rule. The USER password must match the OWNER password for this command to be used. Rulenumber is obtained using PRINT RULES.

```
DELETE RULE rulenumber
```

RENAME ATTRIBUTE

The RENAME attribute command is used to change the name of an attribute in the database definition (schema).

```
RENAME attname1 TO attname2 [IN relname]
```

The old name is attname1 and the new name is attname2. If a relation is not specified, then the name change takes place in every relation that contains the old name. If relname is specified and attname1 occurs more than once, only the first occurrence will be changed.

RULES and KEY(s) defined for attname1 will automatically be redefined to apply to attname2.

RENAME RELATION

You may change the name of a relation by using the following command:

```
RENAME RELATION relname TO newname
```

RULES applying to relname will automatically apply to newname.

REMOVE

The REMOVE command is used to remove a relation definition and its data from the database.

```
REMOVE relname
```

2.2.7 Relational Algebra Commands

These commands allow you to create new relations from existing relations. All of these commands, except PROJECT, form rows in the new relation based upon the comparison of attributes from two parent relations. These comparisons are "exact," i.e., they do not use the user specified tolerance. All relational algebra commands require modify permission on the constituent relations.

INTERSECT

The purpose of the INTERSECT command is to allow you to combine the rows of two relations into a third relation based on common values within a set of specified attributes. The syntax of the INTERSECT command is:

```
INTERSECT relname1 WITH relname2 FORMING relname3 +
[USING attname1 [attname2...attnamen]]
```

You may INTERSECT two relations restricted to specific sets of attributes (the USING clause) or use all attributes of both relations. In either case LARCRIM will identify the common attributes and use these values to qualify rows for intersection.

If the USING clause is specified, it identifies the attributes that form the resulting relation. The attributes used in the INTERSECT process are the subset of those identified by the USING clause which are present in both relations.

For example, assume that you have the following two relations defined:

	REL-1			REL-2	
NAME	DEPT	JOB	DEPT	JOB	PAY
----	----	---	----	---	---
BOB	A	ENGR	A	ENGR	800
JIM	C	SUPR	B	ENGR	450
BOB	B	ENGR	C	ENGR	750
RAY	C	ENGR			

Suppose you want to INTERSECT the two relations using attributes DEPT, NAME, and JOB. The command for this would be:

```
INTERSECT REL-1 WITH REL-2 FORMING REL-3 USING DEPT NAME JOB
```

The result would be the new relation REL-3 shown below:

REL-3

DEPT	NAME	JOB
A	BOB	ENGR
B	BOB	ENGR
C	RAY	ENGR

In this example there are no duplicate rows in REL-3. However, it is not uncommon that the INTERSECT command creates duplicate rows. In general, duplicate rows are not desired in a relation and may be removed with the DELETE DUPLICATES command. It should also be noted that by specifying which attributes the INTERSECT is using, you restrict the number of attributes in the resulting relation to only those specified in the USING clause.

Suppose you want LARCRIM to use all the attributes in the two relations. In this instance, you would enter:

```
INTERSECT REL-1 WITH REL-2 FORMING REL-4
```

The result would be REL-4 consisting of the attributes NAME, DEPT, JOB, and PAY, shown below with the resulting rows:

REL-4

NAME	DEPT	JOB	PAY
BOB	A	ENGR	800
BOB	B	ENGR	450
RAY	C	ENGR	750

There may be situations where an INTERSECT is impossible to perform. These include:

1) The name of the resulting relation array exists

2) The two relations have no common attributes

3) The attributes in the USING clause do not exist in the relations being intersected.

4) A row of the resulting relation exceeds 1021 computer words.

If any of the above situations is encountered, you are warned of the problem and the INTERSECT command processing is stopped. In the case where common attribute names exist but there are no matching values, the operation will be successful resulting in an empty relation (0 rows).

The INTERSECT command is a powerful tool and may be used to satisfy queries which require attributes from two relations.

JOIN

The JOIN command is a function operating on two relations to form a third relation. The purpose of the JOIN is to juxtapose two relations based on a specified attribute from each. The result of the JOIN command is a third relation containing all of the attributes from both relations. Rows are generated in the new relation as a result of the comparison conditions between attributes being satisfied. The syntax of the JOIN command is:

```
JOIN relname1 USING attname1 WITH relname2 USING attname2 +
FORMING relname3 [WHERE (EQ)]
                        NE
                        GT
                        GE
                        LT
                        LE
```

The WHERE clause of the JOIN command is different from the WHERE clause of the SELECT command. In JOIN it applies only to the comparison of the two attributes upon which JOIN is based. If the WHERE clause is omitted, EQ (default) is used.

The value of attname2 in the first row of relname2 is compared to all the values of attname1 in relname1. Rows which qualify are placed in relname3. The value of attname2 in the second row of relname2 is compared to all the values of attname1, etc. Each row from relname2 may generate 0, 1, 2, or more rows in relname3.

For example, consider the relations REL1 and REL2:

REL1

A	B	C
1	2	3
4	5	6
7	8	9

REL2

D	E
3	1
6	2

The following JOIN command would produce the result shown.

```
JOIN REL1 USING B WITH REL2 USING D FORMING REL3 WHERE LT
```

REL3

A	B	C	D	E
1	2	3	3	1
1	2	3	6	2
4	5	6	6	2

The JOIN will function correctly on any comparison providing that you compare attributes of the same data type. All attribute names in the resultant relation must be unique in order for you to obtain accurate results from subsequent commands using the relation. Any duplicate attribute names should be changed using the RENAME command before doing queries or updates to the new relation. In this case, RENAME will change the first attribute name.

There may be situations where a JOIN is impossible to perform. These include:

1) The name of the resulting relation already exists

2) The attribute in the USING clause does not exist in the relation being joined

3) The attributes being compared are different data types or lengths

4) An attribute in either of the relations is greater than 300 computer words

5) A row of the resulting relation exceeds 1021 computer words.

If any of the above situations are encountered you are warned of the problem and the JOIN command processing is stopped. In the case where a legal comparison has been specified, but there is no matching values, the operation will be successful, resulting in an empty relation (0 rows).

PROJECT

The function of a PROJECT command is to create a new relation as a subset of an existing relation. You may want to create the new relation from the existing relation by removing attributes, removing rows, or both. In all cases, the parent relation remains unchanged. The syntax for the PROJECT command is:

```
PROJECT relname1 FROM relname2 USING {attname1 [attname]} +
                                      ALL
[WHERE ... ]
```

The WHERE clause is optional but if specified, it has the same syntax as in the SELECT command. You are required to specify which attributes are to be retained in the new relation. If all the attributes are to be retained, the keyword ALL may be used. The existing relation is relname2 and the new relation is relname1.

For example consider the following relation:

PEOPLE

EMP NUM	EMPNAME	BOSS	POSITION	GROUP
2181	JONES	SMITH	MANAGER	AADE
3964	ERICKSON	BUSS	APPL-MGR	ACC
6543	GRAY	PARKER	ASST-MGR	PHOTO
2233	SCHMITZ	BUSS	APPL-MGR	ACC

To create a new relation with EMPNAME and GROUP as the only attributes and where no rows contain PARKER as BOSS enter the command:

```
PROJECT TEMP1 FROM PEOPLE USING EMPNAME GROUP +
WHERE BOSS NE PARKER
```

TEMP1

EMPNAME	GROUP
JONES	AADE
ERICKSON	ACC
SCHMITZ	ACC

The PROJECT command is useful to reduce the size of a relation when only a subset of the data is needed. If duplicate rows are formed, and not desired in the new relation, you may delete them with the DELETE DUPLICATES command.

There are cases where a PROJECT is impossible to perform, for example:

1) when the name of the resulting relation already exists, or

2) when an attribute in the USING or WHERE clause is not in relname2.

If any of the above situations are encountered, you are warned of the problem and the PROJECT processing is stopped. In the case where a legal PROJECT has been specified but there are no rows that satisfy the WHERE clause, the operation will be successful, resulting in an empty relation (0 rows).

SUBTRACT

The SUBTRACT command is similar to the PROJECT command in that the new relation is a subset of an existing relation. The rows, however, are selected based on the data in another relation rather than on a WHERE clause within the same relation. Where the INTERSECT command looked for rows of two relations which matched up, the SUBTRACT command does just the opposite. It looks for rows in relname2 which do not match any rows in relname1. The syntax for the SUBTRACT command is:

```
SUBTRACT relname1 FROM relname2 FORMING relname3 +
[USING attname1 [attname2 ... attnamem]]
```

The rows in the new relation will be the rows from relname2 that do not have a match in relname1. If the USING clause is not specified, then all attributes of relname2 will be included in relname3. If a USING clause is specified at least one of the attributes in the clause must be common to both relations.

As an example consider the following two relations:

```
        EMPDATA
EMPNUM  EMPNAME      BOSS
------  -------      ----
2181    JONES        SMITH
3964    ERICKSON     BUSS
6543    GRAY         PARKER
8461    BROWN        WHITE
2233    SCHMITZ      BUSS
```

```
              BOSSDATA
BOSS    POSITION     GROUP
----    --------     -----
SMITH   MANAGER      AADE
PARKER  APPL-MGR     PHOTO
BUSS    APPL-MGR     ACC
```

The following command will produce a new relation by subtracting BOSSDATA from EMPDATA:

```
SUBTRACT BOSSDATA FROM EMPDATA FORMING TEMP USING EMPNAME BOSS
```

The resulting relation TEMP would contain only one row:

```
          TEMP
EMPNAME             BOSS
-------             ----
BROWN               WHITE
```

There may be situations where a SUBTRACT is impossible to perform. These include:

1) The name of the resulting relation already exists,
2) The relations have no common attributes,
3) The number of attributes in the USING clause is greater than the number in relname1,

4) An attribute in the USING clause is not in the relname2.

If any of the above situations are encountered, you are warned of the problem and the SUBTRACT processing is stopped. In the case where a legal SUBTRACT has been specified but there are no rows in relname2 which are not in relname1, the operation will be successful resulting in an empty relation (0 rows).

2.2.8 Report Commands

These commands establish a limited report generation capability.

NEWPAGE

This command causes subsequent printout to begin on a new page. It applies to batch output only.

```
NEWPAGE
```

BLANK

Blank lines can be inserted into the output stream by using the command:

```
BLANK n
```

where n is the number of blank lines written.

TITLE

This command causes the text "titlestring" to be printed centered on the line. If the length of "titlestring" is longer than current lines width, it will be truncated and a warning issued.

```
TITLE "titlestring"
```

DATE

This command will cause the current date to be printed, centered on the line.

```
DATE
```

LINES

This command controls the number of lines per page (exclusive of title).

```
LINES n
```

The number of lines per page is established by "n". Default is 56.

WIDTH

This command controls the width of a printed line.

```
WIDTH n
```

The number of characters per line is established by "n". Default is 78 if output is to a terminal, 132 if output is to a batch printer or output file. If n is specified to be less than 20, 20 will be used.

2.2.9 Key Commands

BUILD KEY

This command is used to change an attribute from non-KEY to KEY. An index is built from existing data values by making a pass through current rows of the specified relation. This index is then used and maintained just as if the attribute had been declared to be KEY in the original database definition.

```
BUILD KEY FOR attribute IN relname
```

KEYS are not transferred to relations created by relational algebra commands.

DELETE KEY

This command is used to change an attribute from KEY to non-KEY. The index file for that attribute is deactivated and no longer maintained or used one the attribute has been changed to non-KEY with this command.

```
DELETE KEY FOR attname IN relname
```

2.2.10 LARCRIM-to-LARCRIM Commands

UNLOAD

The UNLOAD command permits you to off-load a portion or all of your database onto a previously designated file (see OUTPUT command). The file will contain 80 character text records and can be read by LARCRIM as an input file on the same or on a different computer using the INPUT command. Default file name is OUTPUT. The syntax of this command is:

```
UNLOAD [dbname[= newname]]{ALL    } +
                          SCHEMA
                          DATA
  [relname1 [mpw1] relname2 [mpw2] ... ]
```

Specifying SCHEMA will off-load the schema of your database, DATA will off-load the data and ALL will off-load both schema and data.

Optionally, you may rename your database by entering dbname = newname where dbname is the name of the currently open database. By specifying relation names, you will only off-load data and/or schemas for the specific relations. If your current user password does not allow you modify access to the relation, the modify password associated with the relation must be specified.

There are implicit password restrictions to the UNLOAD command as follows: If you are the database owner, you off-load the complete schema and/or any data. If you are not the owner, you may off-load only the data and/or definitions for the relations for which you have modify permission. Your password becomes the owner of the off-loaded database. Rules, if any, will not be off-loaded.

The UNLOAD command should not be used to off-load relations which contain more than 100 values in a single row. Such records are not readable by the LARCRIM user interface.

2.2.11 General Commands

INPUT

This command is used to specify the name of a file which contains the LARCRIM commands and/or input data. Alternate input files may be assigned as often as required. The use of this command allows you to define command procedures on a file and then have LARCRIM execute the set of commands without user interaction. Typically, you might input a file you have "unloaded" on the same, or a different computer.

```
INPUT filename
```

The last command on the alternate input file should be INPUT INPUT which returns input to the batch input file or your terminal. INPUT TERMINAL will, for interactive jobs, do the same thing. If you do not use the INPUT command on your alternate input file, LARCRIM will return input to your terminal, for interactive jobs, when it encounters an end-of-file mark. Batch jobs will terminate.

OUTPUT

This command is used to specify the name of the output file. Specifying a file other than OUTPUT will result in the output from the LARCRIM commands being placed on a file with the specified file name. Error messages will continue to be directed to the terminal for an interactive job. The output file name may be changed as often as desired. The use of this command allows the interactive user to get offline hardcopy output from LARCRIM.

```
OUTPUT filename
```

OUTPUT OUTPUT will return the output to the batch output file or your terminal. OUTPUT TERMINAL will, for interactive jobs, do the same thing.

EXIT

To leave LARCRIM enter:

```
{EXIT}
 QUIT
```

This command closes your current database by copying the data in the incore working areas to the files whose names were determined by the OPEN COMMAND or by the name designated in the DEFINE sublevel.

MENU

The MENU command places you in the menu mode. It may be entered at any point when in command mode except when in the DEFINE, HELP, or LOAD sublevels. The menu mode is particularly useful for schema definition and data loading.

```
MENU
```

HELP

The HELP command allows you to obtain a description of the available LARCRIM commands, a discussion of the general command syntax, a summary of all available commands, and general news about the LARCRIM system. HELP is available at any time during execution except when in the menu mode.

To receive help when in the command mode enter:

```
HELP [{command name}]
        LARCRIM
        SYNTAX
        WHERE
        SUMMARY
        NEWS
        SORT
        INPUT FORMAT
```

The HELP sublevel (Prompt = H>) options have the following meanings:

OPTION	RESULTS
HELP	explains the previous command and its syntax or if HELP is the first command entered, it is identical to HELP LARCRIM
HELP command name	explains the indicated command name and syntax
HELP LARCRIM	lists commands for which help is available
HELP SYNTAX	describes the general command syntax
HELP WHERE	explains the LARCRIM where clauses
HELP SUMMARY	summarizes all available LARCRIM commands
HELP NEWS	provides general news about the LARCRIM system
HELP SORT	explains the LARCRIM sort options
HELP INPUT FORMAT	describes the LARCRIM user interface routines

The commands available inside the HELP sublevel are identical to the HELP commands except that the keyword HELP is omitted. The HELP sublevel displays information approximately 20 lines at a time. After each such display you will have the option to continue displaying the text or to return to the HELP sublevel by entering QUIT. You will remain in the HELP sublevel until you enter an END command which will return you to the command mode.

USER

This command is used to identify your password to LARCRIM. Your user password is used to check against read, modify, and owner passwords specified for the relations. Each time this command is issued, the new password replaces the current password. The default password is "NONE".

```
USER password
```

ECHO

This command is used to control the printing of your input commands on the output file. Default is for ECHO to be off for interactive operation and on for batch. To activate ECHO printing enter:

```
ECHO
```

NOECHO

The NOECHO command turns off ECHO printing.

```
NOECHO
```

CHECK

The CHECK command turns on the rule checking which applies to the CHANGE and LOAD commands. If rules are defined, they are enforced unless the NOCHECK command is entered. The CHECK and NOCHECK commands may be issued as many times as required anywhere in the input stream.

```
CHECK
```

NOCHECK

The NOCHECK command suppresses the rule checking.

```
NOCHECK
```

TOLERANCE

For attributes which contain floating point numbers, a tolerance may be specified for checking equality, nonequality and order. The tolerance applies to any real or double precision number you use in a WHERE clause. If A is an attribute with value a, and r is a user specified number used in a WHERE clause, and t is a tolerance (positive, zero, negative), the following are true conditions:

```
A EQ r if and only if r-t LE a LE r+t
A NE r if and only if a LE r-t or a GE r+T
A GT r if and only if a GT r-t
A GE r if and only if a GE r-t
A LT r if and only if a LT r+t
A LE r if and only if a LE r+t
```

If t is a percentage tolerance, t is to be replaced with t x r/100.0, in the above expressions to define true conditions for percentage tolerances.

```
TOLERANCE tol [PERCENT]
```

where tol is the tolerance and the presence or absence of the keyword PERCENT indicates whether tol is a percentage tolerance or an absolute tolerance. The TOLERANCE command can be used as many times as desired to reset the tolerance. A tolerance stays in effect for a session, or until a new tolerance is specified. The default value for tolerance is zero.

RELOAD

The RELOAD command permits you to rebuild your database files to recover unused space created by row deletions, relation removals, and certain attribute changes. When a row is deleted or a relation removed, its space is not reused until you issue this command. In addition, if a variable length attribute is modified so that it increases in length, the row is deleted and replaced with a new one. The old row becomes unused space. If your database has any KEY attributes, then the pointer files maintained for those attributes are also rebuilt. The syntax for this command is:

```
RELOAD
```

You should use this command with discretion and weigh the cost of rebuilding the files against the saving of mass storage space.

CLOSE

The CLOSE command permits you to close a LARCRIM database without leaving LARCRIM. This enables you to close one database, then open or define a different one, all within one LARCRIM session. However, LARCRIM automatically executes this command whenever a new database is defined or opened, therefore, this command is generally not entered by the user. This command results in update of the database files to reflect all changes you have made to the database.

```
CLOSE
```

Note: the current database will also be closed for you when you exit.

OPEN

The OPEN command is required whenever an existing database is to be used. You specify the name of the database. LARCRIM uses the name of the database to form the names of the three files which contain the database.

```
OPEN dbname
```

Only one LARCRIM database may be open at a time and LARCRIM will automatically close the current database whenever a new database is opened. The OPEN command must be issued before any commands that require data from the database can be processed.

2.3 Menu Mode Execution Overview

The options (create, update, query, command, and exit) available in menu mode are illustrated in section 2.4. This section explains the background for each option.

Execution may be terminated at anytime by entering the word QUIT, whereas, EXIT, in response to an input prompt, will return you to the top menu.

2.3.1 Database Creation Option

The purpose of this option is to construct a schema by prompting you for the database name, owner, the names of the relations, their associated attributes and read/modify passwords.

After compilation of the schema, you have the opportunity to interactively load the database and/or query the database.

2.3.2 Database Update Option

With this option you may add/modify relations and/or load additional data into the database. If additional relations are desired, you are prompted for the names of the relations, their associated attributes and read/modify passwords. If additional data is to be loaded, the list of relations in the database is displayed and you select the relations to load. The attributes of the selected relation are displayed and you enter the required data. Removal or modification of data in the database is done using the LARCRIM database modification commands in the command mode.

2.3.3 Query Option

With this option you are prompted for the database name. The command mode is then entered where the full set of LARCRIM commands is available to you for database query. In addition to query, all other database activities are available.

2.4 LARCRIM Menu Mode Interactive Dialogue

This section presents the questions and menus that appear in the menu mode. At the beginning of each subsection there is a figure that provides an interactive command/response dialogue for the options discussed in that subsection. These dialogues describe the interactive questions and menus in the sequence in which they appear during a terminal session. Whenever a user input is required, an R> appears followed by a page reference. A discussion of the question and the user response options is found on the page referenced.

The menu mode is accessed by entering "**MENU**" anytime in the command mode when a R> prompt is present.

2.4.1 Menu Dialogue

```
SELECT THE EXECUTION OPTION DESIRED
        1) CREATE A NEW DATABASE
        2) UPDATE AN EXISTING DATABASE
        3) QUERY AN EXISTING DATABASE
        4) ENTER COMMAND MODE
        5) EXIT
```

The desired execution option is selected by entering the integer 1,2,3, or 4. If the session is to be terminated, enter 5.

When option 2 is selected, the following menu appears.

```
SELECT THE UPDATE OPTION DESIRED
        1) DEFINE ADDITIONAL RELATIONS
        2) LOAD ADDITIONAL DATA
```

The desired update option is selected by entering either the integer 1, allowing the definition of additional relations, or 2, allowing the loading of additional data into the database.

2.4.2 Schema Definition

The schema, or "OUTLINE", of a database includes the names of the database, the owner, the relations, passwords and attributes, along with the type and size of each attribute. Relations are built by first defining the attributes and then by naming the relation with a set of attributes. The interactive dialog follows, with comments.

ENTER THE NAME OF THE DATABASE

The 1-6 character alphanumeric name assigned to the database is entered here. (At least one character must be alphabetic.) All future references to this database will be via the assigned database name, with case sensitivity. See also Appendix G, DATABASE FILES.

ENTER THE NAME OF THE DATABASE OWNER

The 1-8 character alphanumeric name of the database owner is entered here. This name is used as the schema password. Adding to an existing schema definition will not be permitted unless the existing owner password matches the owner password entered here. "NONE" is an acceptable owner and is used when a <CR> is entered.

ENTER THE NAME ASSIGNED TO THIS RELATION

A 1-8 character alphanumeric name assigned to the relation being defined.

ENTER THE READ PASSWORD FOR THIS RELATION

A 1-8 character alphanumeric string assigned by the owner as the read password for the relation being defined. If no read password is desired enter "NONE".

ENTER THE MODIFY PASSWORD FOR THIS RELATION

A 1-8 character alphanumeric string assigned by the owner as the modify password for the relation being defined. If no modify password is desired enter "NONE".

```
ENTER THE ATTRIBUTES OF THIS RELATION
ENTER END WHEN COMPLETE

   attname    type        length ( IF >1) "KEY" (IF KEY)

   attname    1-8 character alphanumeric string identifying
              the attribute being defined.

   type =     TEXT    (Text)
              REAL    (Real)
              INT     (Integer)
              DOUB    (Double Precision)
              RVEC    (Real Vector)
              IVEC    (Integer Vector)
              DVEC    (Double Precision Vector)
              RMAT    (Real Matrix)
              IMAT    (Integer Matrix)
              DMAT    (Double Precision Matrix)

   length =   length of the attribute in terms of the number
```

```
of characters (text attributes) or the number
of values (all other attributes).
```

The length of a text attribute is specified as a number of characters. The lengths of numeric data are specified as a number of values. Note that matrix attributes require values for both the rows and the columns. The maximum length of an attribute is dependent on other attributes in the relation. See Appendix D, LIMITATIONS.

A variable length (or length greater than one) INT, REAL, or DOUB attribute can be considered to be functionally identical to the IVEC, RVEC, or DVEC attribute.

KEY = the word "KEY" indicates a keyed attribute.

Example: To define a text attribute (TEXTST) of 60 characters, a real attribute (REAL1), an integer keyed attribute (INT-1), and a real matrix with dimensions 6x8 (MAT68), the following entries would be made:

```
TEXTST TEXT 60
REAL1 REAL
INT-1 INT KEY
MAT68 RMAT 6,8
```

To end the definition of the attributes for the current relation, the word "END" is entered.

```
DO YOU HAVE ADDITIONAL RELATIONS TO DEFINE--Y OR N
```

Additional relations may be defined by entering the character "**Y**". If no additional relations are to be defined at this time, enter "**N**".

2.4.3 Database Loading

The database may be loaded in the same session as the schema definition, or postponed. Therefore, you may safely select "**N**" below.

```
DO YOU WANT TO LOAD THE DATABASE--Y OR N

Enter "Y" if you want to load the database at this time.
Enter "N" if no data is to be loaded.

SELECT THE RELATION TO BE LOADED
```

The relations defined in the database will be listed. You select the relation to be loaded by entering the integer corresponding to the desired relation.

ENTER THE MODIFY PASSWORD FOR THIS RELATION

If a modify password is defined for the selected relation you will be prompted for the password. You will not be allowed to load data unless you enter the proper password here.

ENTER THE ATTRIBUTE VALUES IN THE SPECIFIED SEQUENCE
ENTER END WHEN COMPLETE

Entering data values at this point loads the database. The values are entered in the order indicated and the value entered must correspond to the attribute type. If a text string contains embedded blanks, or if numeric text is entered, it must be enclosed in quotation marks. If the value of a fixed length text attribute is shorter in length than indicated by the attribute definition, LARCRIM will blank fill the input text string. It is recommended that leading blanks not be used in text strings if you plan to query the database using the text strings. Such queries (except for using EQS) are sensitive to the number of leading blanks. If vectors or matrices are loaded, all values must be specified. Enter "**END**" when data loading is complete. It is recommended that you use the application program interface for loading data for large databases and for databases that have vectors and matrices.

DO YOU HAVE ADDITIONAL RELATIONS TO LOAD--Y OR N

If you want to load another relation, enter "**Y**". If all the data to be loaded at this time has been loaded, enter "**N**".

DO YOU WANT TO QUERY THE DATABASE AT THIS TIME--Y OR N

You may switch to the command mode for query by entering "**Y**". If the query option is not desired enter "**N**".

3.0 LARCRIM EXECUTION THROUGH THE APPLICATION PROGRAM INTERFACE

Any programming language which can call FORTRAN subroutines can access and modify a LARCRIM database through subroutines contained in the LARCRIM library, LARCRIMLIB. This subroutine library constitutes the application program interface.

The application program interface requires that you manage the database files. The database files must exist as three properly named files before your program can be executed.

Password checks operate in the application program interface in much the same way as in the stand-alone system. No password permission is required for RMOPEN, RMCLOS, RMUSER, RMRULE, or RMTOL. Read permission is required for all other calls except RMLOAD and RMPUT for which modify permission is required. Modify permission implies read permission.

3.1 Initializing the Database

The RMOPEN routine initializes the internal tables used by LARCRIM and opens the specified database by reading the database control information into the working area of memory.

```
CALL RMOPEN (dbname)
```

Input parameter:

dbname name of the database

The RMCLOS routine closes the current database after copying the working areas of memory to the database files. Execution of this routine is required (if you have modified the database) before your program can access another database.

```
CALL RMCLOS
```

3.2 Status of Database Activity

When an operation on the database has been attempted, the status of the operation is returned to the application program via the RMSTAT variable in the RIMCOM common block. The value of RMSTAT should be checked after each operation. A non-zero value indicates the operation was not successful. As a result, subsequent operations may not function as expected. RIMCOM must be declared in the calling program as follows:

```
COMMON /RIMCOM/ RMSTAT
INTEGER RMSTAT
```

The RMSTAT values and meaning are as follows:

```
 -1  NO MORE DATA AVAILABLE FOR RETRIEVAL
  0  OK - OPERATION SUCCESSFUL
 10  DATABASE FILES DO NOT CONTAIN A LARCRIM DATABASE
 11  DATABASE NAME DOES NOT MATCH FILE CONTENTS
 12  INCOMPATIBLE DATABASE FILES (DATE,TIME,ETC)
 15  DATABSE FILES ARE NOT IN DIRECTORY
 16  DATABASE HAS NOT BEEN OPENED
 20  UNDEFINED RELATION
 30  UNDEFINED ATTRIBUTE
 40  MORE THAN 9 AND/OR OPERATORS IN THE WHERE CLAUSE
 41  ILLEGAL "LIMIT EQ N" CONDITION
 42  UNRECOGNIZED COMPARISON OPERATOR
 43  EQS ONLY AVAILABLE FOR TEXT ATTRIBUTES
 44  ILLEGAL USE OF MIN/MAX IN THE WHERE CLAUSE
 45  UNRECOGNIZED AND/OR OPERATOR
 46  COMPARED ATTRIBUTES MUST BE THE SAME TYPE/ LENGTH
 47  LISTS ARE VALID ONLY FOR EQ, EQS AND NE
 48  ILLEGAL WHERE CLAUSE ROW NUMBER
 50  RMFIND NOT CALLED
 60  RMGET NOT CALLED
 70  RELATION REFERENCE NUMBER OUT OF RANGE
 80  VARIABLE LENGTH ATTRIBUTES MAY NOT BE SORTED
 81  THE NUMBER OF SORTED ATTRIBUTES IS TOO LARGE
 89  SORT SYSTEM ERROR
 90  UNAUTHORIZED RELATION ACCESS
110  UNRECOGNIZED RULE RELATIONS
111  MORE THAN 10 RULES PER RELATION
112  UNABLE TO PROCESS RULES
2XX  TUPLE VIOLATES RULE XX
```

The following codes should not be encountered in normal use:

1001	BUFFER SIZE PROBLEM
1002	UNDEFINED BLOCK
1003	CANNOT FIND A LARGER B-TREE VALUE
1004	CANNOT FIND B-TREE BLOCK
21XX	RANDOM FILE ERROR XX ON FILE1
22XX	RANDOM FILE ERROR XX ON FILE2
23XX	RANDOM FILE ERROR XX ON FILE3
24XX	RANDOM FILE ERROR XX ON FILE4

3.3 General Routines

The following routines are used to set the internal switches for rule checking, to specify the database passwords, and to set the tolerance for real and double precision numbers. These routines may be called any number of times with the new value overwriting the current value.

The RMUSER routine is used to specify the password necessary for database access or modification.

```
CALL RMUSER (password)
```

Input parameters:

password	the password in Hollerith format.

The RMRULE routine turns rule checking on and off (default--on if rules are defined).

```
CALL RMRULE (switch)
```

Input Parameters:

switch	0 no rule checking (NOCHECK) 1 check rules (CHECK)

The RMTOL routine sets the tolerance for floating point numbers, (default: 0.).

```
CALL RMTOL (value,type)
```

Input Parameters:

value	the value of the tolerance (real)
type	0 if "value" is the tolerance value (int) 1 if "value" is the tolerance percent

3.4 Accessing the Schema

The following routines are used to obtain information about the database schema.

The RMLREL routine sets an implicit pointer to the first relation in the database that your password allows you to read. It must be called before RMGREL is called. If your password does not allow read access to any relations, RMSTAT will be set to 90.

```
CALL RMLREL
```

The RMGREL routine returns the data about the current relation (the relation indicated by the current implicit pointer) and increments the implied pointer to point to the next relation for which you have read permission. After a successful execution of this routine RMSTAT is set to 0. If you change passwords between calls to RMLREL and RMGREL or between successive calls to RMGREL, unpredictable results may occur. RMSTAT is set to -1 when a call is made to RMGREL and all qualified relations have been retrieved.

```
CALL RMGREL (rname,rpw,mpw,lastmod,numatt,numrows)
```

Output Parameters:

rname	relation name in Hollerith format
rpw	read password (.TRUE. or .FALSE.)
mpw	modify password (.TRUE. or .FALSE.)
lastmod	date (YY/MM/DD) of last modification of the relation data (Hollerith)
numatt	number of attributes in the relation
numrows	number of rows of data in the relation

The following example shows how to use RMLREL and RMGREL to obtain the data about all relations in the database.

```
COMMON /RIMCOM/ RMSTAT
INTEGER RMSTAT
                .
                .
                .
CALL RMOPEN(dbname)
CALL RMUSER (password)
                .
                .
CALL RMLREL
IF (RMSTAT.EQ.0) GO TO 100
                .
```

```
Print message that no relations are available using the
current password.

      GO TO 200
100   CONTINUE
      CALL RMGREL(rname,rpw,mpw,lastmod,numatt,numrows)
      IF(RMSTAT.NE.0) GO TO 200
                        .
                        .
print the data about the relation, etc.........
                        .
                        .
      GO TO 100
200   CONTINUE
```

The RMLATT routine sets an implied pointer to the first attribute of the specified relation. This routine must be called prior to calls to RMGATT. If the relation exists and your password allows read access to it, RMSTAT is set to 0.

```
CALL RMLATT (rname)
```

Input Parameters:

rname	relation name in Hollerith format.

The RMGATT routine returns the data about the current attribute (the attribute indicated by the implied pointer) and increments the implied pointer to point to the next attribute. RMSTAT is set to -1 when a call is made to RMGATT and all the attributes for the relation have been retrieved.

```
CALL RMGATT(aname,type,matvec,var,len1,len2,column,key)
```

Output Parameters:

aname	attribute name in Hollerith format
type	attribute type (3HINT,4HREAL,4HDOUB,4HTEXT)
matvec	attribute type (3HVEC,3HMAT or blank)
var	variable length attribute (.TRUE. or .FALSE.)
len1	attribute length data as follows:
	TEXT — number of characters
	INT,REAL,DOUB,VEC — number of values
	MAT — row dimension
len2	column dimension of MAT attributes (otherwise 0)
column	attribute column location in the relation
key	keyed attribute (.TRUE. or .FALSE.).

The following example shows the use of RMLREL, RMGREL, RMLATT, and RMGATT to obtain the data about all attributes for all relations. (The equivalent of LISTREL ALL)

```
                .
                .
                .
      COMMON /RIMCOM/ RMSTAT
      INTEGER RMSTAT
                .
                .
                .
      CALL RMOPEN(dbname)
      CALL RMUSER(password)
                .
                .
                .
      CALL RMLREL
100   CONTINUE
      CALL RMGREL(rname,rpw,mpw,lastmod,numatt,numrows)
      IF(RMSTAT.NE.0) GO TO 300
      CALL RMLATT(rname)
      DO 200 K=1,numatt
      CALL REMGATT(aname,type,matvec,var,len1,len2,column,key)
      IF(RMSTAT.NE.0) GO TO 999
                .
                .
                .
C     printout the relation and attribute data, etc....
                .
                .
                .
200   CONTINUE
      GO TO 100
300   CONTINUE
              .
              .
              .
999   CONTINUE
      error while retrieving attribute data
              .
              .
```

3.5 Accessing the Database

The routines which access the database allow the following operations:

1) GET an existing row of data from a specified relation and store it in a local array (must be preceded by a RMFIND).

2) LOAD a new row of data from a local array to the bottom of a specified relation (must be preceded by a RMFIND).

3) PUT an existing row of data back into a specified relation after it has been modified (must be preceded by a RMGET).

4) DELETE an existing row of data from a specified relation (must be preceded by a RMGET).

Each of the above operations work on one row of data at a time. Get (RMGET), put (RMPUT) and delete (RMDEL) use an implied row pointer which is initialized by RMFIND and incremented by RMGET. You have 6 implied pointers referenced by numbers 0-5 at your disposition. You must not use more than one pointer for a given relation; unpredictable results may occur. RMPUT, which must be preceded by RMGET, simply puts a row back where it was taken from. Load (RMLOAD), while not specifically using the pointers established by RMFIND, requires a call to RMFIND. In this case, RMFIND simply identifies the relation.

A call to RMFIND establishes an implied pointer chain to all the rows of a relation in the internal order of the rows. You may restrict the number of rows of the implied pointer chain by a call to RMWHER following the RMFIND call. You may also sort the rows qualified by the implied pointer chain by a call to RMSORT. Thus, for each pointer you wish to use, you must first call RMFIND, then, if you wish to restrict qualifying rows, call RMWHER and, if you desire sorted order, call RMSORT. You must not call RMGET, RMLOAD, RMPUT, or RMDEL between the calls to RMFIND, RMWHER, and RMSORT unless a new pointer chain is being established.

A pointer may be redefined as many times as desired. Each time RMFIND is called, the pointer, if previously defined, is reset. The logical sequence of RMFIND, RMWHER (optional), RMSORT (optional) calls must be set up prior to the first call to RMGET, etc. Note that a new qualification (RMWHER), a new sort ordering (RMSORT), or both requires a new call to RMFIND.

The RMFIND routine establishes the initial pointer for a relation and associates the user assigned number with the pointer. A call to RMFIND must be made before calls to routines which further define the pointer (RMWHER, RMSORT) as well as before calls to routines which use the pointer.

```
CALL RMFIND (number,relname)
```

Input Parameters:

number	user assigned integer number (0-5) used to reference the pointer for the relation (int)
relname	relation name in Hollerith format

The RMWHER routine qualifies a set of rows for retrieval and corresponds to the stand-alone WHERE clause.

```
CALL RMWHER (number,attname,operator,value,numval,nextboo,
            numboo)
```

Input Parameters:

number	number (0-5) which identifies the relation pointer previously established by RMFIND
attname	array of attributes names (may also be attribute numbers, ROWS or LIMIT) where the attname(n) corresponds to the nth condition of the WHERE clause
operator	array of operators (2HEQ,3HEQA, etc.) where operator(n) is the operator for the nth condition of the WHERE clause
value	2-dimensional array of WHERE clause "values" where the nth row corresponds tothe nth condition of the WHERE clause.

The organization of the array is dependent on the attribute type and length. Let v represent an attribute value and let v(i) represent the ith value in a list of values (WHERE clause list option). The rows of value are organized as follows:

Fixed length attributes:

v(1),v(2),.....,v(x) where x is equal to the number of values in the list

Variable length attributes:

TEXT -
c(1),0,v(1),c(2),0,v(2) .....,
c(x),0,v(x)
where c is the number of characters in the corresponding value and x is equal to the number of "values" in the list.

INT,REAL,DOUB,VEC -
items(1),0,v(1), items(2),0,v(2) .....,
items(x),0,v(x)
where items is the number of items in the corresponding value and x is equal to the number of "values" in the list.

MAT -
rows(1),cols(1),v(1),rows(2),cols(2),v(2),...,rows(x),cols(x),v(x)
where rows is the number of rows and cols is the number of columns in the corresponding matrix value x is equal to the number of "values" in the list.

numval number of "values" in the list of values (v(1),v(2),....,v(x)). numval(n) equals x and corresponds to the nth condition of the WHERE clause. Note that numval(n) may be greater than1 only for EQ, EQS or NE conditions (see the SELECT command).

nextboo array of "AND" or "OR" operators in Hollerith format.

numboo number of WHERE clause conditions.

Note that a call to RMWHER for a specified pointer (number) must be preceded by a call to RMFIND. For example, if the following WHERE clause were required:

```
WHERE ATT1 EQ 4 7 12 OR ATT2 EQS "TEXT STRING" AND +
ATT3 GT 5. AND ATT3 EQA ATT4
(ATT1 -- integer length 1)
(ATT2 -- text variable length)
(ATT3 -- real length 1)
(ATT4 -- real length 1)
```

The arrays would be dimensioned:

```
attname(4)
operator(4)
value(4,5)
numval(4)
nextboo(4)
```

The arrays would contain:

```
attname(1)     = 4HATT1
attname(2)     = 4HATT2
attname(3)     = 4HATT3
attname(4)     = 4HATT3
operator(1)    = 2HEQ
operator(2)    = 3HEQS
operator(3)    = 2HGT
operator(4)    = 3HEQA
value(1,1)     = 4
value(1,2)     = 7
value(1,3)     = 12
value(1,4)     = 0
value(1,5)     = 0
```

```
value(2,1)      = 11
value(2,2)      = 0
value(2,3)      = 4HTEXT
value(2,4)      = 4H STR
value(2,5)      = 3HING
value(3,1)      = 5.
value(3,2)      = value(3,3) = value(3,4) = value(3,5) = 0
value(4,1)      = 4HATT4
value(4,2)      = value(4,3) = value(4,4) = value(4,5) = 0
numval(1)       = 3
numval(2)       = 1
numval(3)       = 1
numval(4)       = 1
nextboo(1)      = 2HOR
nextboo(2)      = 3HAND
nextboo(3)      = 3HAND
nextboo(4)      = 0
numboo          = 4
```

The RMSORT routine sorts the data prior to retrieval and is equivalent to the stand-alone SORTED BY clause.

```
CALL RMSORT (number,attname,numsort,sortype)
```

Input Parameters:

number	number (0-5) which identifies the relation pointer previously established by RMFIND (int).
attname	array of sort attribute names
numsort	number of attributes to sort (int)
sortype	sort control numbers, corresponding to attname(n)
sortype(n)	= -1 for descending sort
sortype(n)	= 1 for ascending sort

For example, if the following SORTED BY clause were required:

```
SORTED BY ATT1=A ATT2=A ATT3=D
```

The arrays would contain:

```
attname(1) = 4HATT1
attname(2) = 4HATT2
attname(3) = 4HATT3
numsort = 3
sortype(1) =  1
sortype(2) =  1
sortype(3) = -1
```

The RMGET routine gets a row of data from the specified relation and advances the pointer to the next qualifying row (as determined by RMFIND and optionally by RMWHER and RMSORT conditions). RMSTAT is set to -1 when a call is made to RMGET and all qualified rows have been retrieved.

```
CALL RMGET (number,array)
```

Input Parameters:

number — number (0-5) which identifies the relation from which a row is to be retrieved (int).

Output Parameters:

array — array to receive the row of data. The array used may be unique for this relation or it may be a general array used for all data access activities. In either case the array must be large enough to hold one row of data. LARCRIM does not check the array dimension. An integer array is recommended.

The attribute values are stored in the array according to the conventions discussed below. For this discussion, let "coli" be the column number in the relations that corresponds to the ith attribute.

Fixed length attributes:

Array(coli) contains the start of the value for the i-th attribute.

Variable length attributes:

Array(coli) contains the pointer "p" which points to the start of the attribute data in the array.

The pointer word, array(p) contains one of the following values:

attribute	array(p)
TEXT	number of characters
INT,REAL,DOUB,VEC	number of items
MAT	row dimension

Array(p+1) contains one of the following values:

attribute type	array(p+1)
TEXT	0
INT,REAL,DOUB,VEC	0
MAT	column dimension

Array (p+2),... contains the attribute values

For a more complete explanation of the data array, refer to Section 3.6.

The RMLOAD routine loads a row of data into a specified relation. Calls to RMLOAD may be repeated with each row being loaded at the bottom of the relation.

```
CALL RMLOAD (number,array)
```

Input Parameters:

number	number (0-5) which identifies the relation to load.
array	array containing the row of data to load (see RMGET for a description of array).

The RMPUT routine, following a call to RMGET, will modify a row of data in a specified relation.

```
CALL RMGET (number,array)
        .
        .
        .
CALL RMPUT (number,array)
```

Input Parameters:

number	number (0-5) which identifies the relation to be modified . The row to be modified was established by the call to RMGET.
array	array containing the modified row of data (see RMGET for a description of array).

The RMDEL routine, following a call to RMGET, will delete a row of data in a specified relation.

```
CALL RMGET (number,array)
                          .
                          .
                          .
CALL RMDEL (number)
```

Input Parameters:

number	number (0-5) which identifies the relation which is to be modified The row to be deleted was established by the call to RMGET.

Calls to RMPUT and RMDEL must be preceded by calls to RMGET. The use of several intermixed modify and load operators using the same pointer should be avoided since unpredictable results may occur.

3.6 The Data Array

An application program passes data to and receives data from a LARCRIM database one row at a time. In order to receive, load, or modify data, a buffer is used by the application program. This buffer, a multi-word array, must be large enough to contain the largest accessed row. An understanding of the organization of the data within this array is crucial, because for each attribute's data, LARCRIM either places the data in or expects it to be in a particular position within the array. This position is determined by the data type of an attribute.

The structure of the data array is easiest to understand within the context of an example. Suppose a relation has the following definition:

Attribute Name	Data Type	Length
PARTNO	INT	1
DESC	TEXT	20
OPTEMP	RVEC	3
FREQRNG	IMAT	2,3

Values in the data array appear in the same order as they are specified in the relation's LARCRIM schema definition. In this example relation, the value for PARTNO will occupy the first postion in the data array, followed in order by DESC, OPTEMP, and FREQRNG.

The number of words each attribute occupies is determined by its data type. Each single valued, non-text attribute will occupy one word in the data array, for example, PARTNO. Text attributes can occupy more than one word, depending upon the number of characters in the text and the computer' s word size. Assuming a word size of four bytes, DESC will require one word for each four characters. Because DESC is defined with a fixed length of 20, it will occupy 5 words in the data array. If DESC had been defined with a length of 17 characters, it would still occupy five words. Attribute values are aligned on word boundries within the data array. Vectors and matrices require one word for each element in the attribute. Thus OPTEMP will occupy three words and FREQRNG will require six words in the data array (2 x 3). The data array for this example relation would require a minimun of fifteen words, but could have more.

Let's assume that one row in this relation contains the following data:

Attribute	Value
PARTNO	4039
DESC	"LOW GAIN ANTENNA"
OPTEMP	325, 437, 665
FREQRNG	10.2, 20.5, 40.8
	24.6, 38.2, 91.4

By knowing the position and number of words occupied by each attribute in this relation, the data array for the row above would look like:

Attribute	Word	Data Array Values
PARTNO	1	403
DESC	2	"LOW "
"	3	"GAIN"
"	4	" ANT"
"	5	"ENNA"
"	6	" "
OPTEMP	7	325
"	8	437
"	9	665
FREQRNG	10	10.2
"	11	24.6
"	12	20.5
"	13	38.2
"	14	40.8
"	15	91.4

Notice that the values for FREQRNG, a matrix attribute, are stored in the FORTRAN columnwise convention, i.e., row1/col1, row2/col1, row1/col2, row2/col2, row1/col3, row2/col3. Word six is filled with blanks, because the length of DESC is 20 and the text is only 16 characters long. Five words must be allocated for DESC even though there is not enough text to fill up the 20 characters.

Now let's look at a variable length text attribute and its characteristics within the data array. We will use the same example relation, but change DESC to a variable length attribute. The definition of the relation becomes:

Attribute Name	Data Type	Length
PARTNO	INT	1
DESC	TEXT	VAR
OPTEMP	RVEC	3
FREQRNG	IMAT	2,3

For a variable length attribute, a data location pointer is placed in the data array at the position where the attribute's value would normally begin. This pointer indicates where in the data array the variable length attribute's value is actually located. Data for variable length attribute is always located at the end of the data array. The data location pointer is an integer which represents the position (word number) in the data array where information about the variable length attribute begins. Following this information is the attribute's value.

The information at the data array location indicated by the pointer consists of two words. What these words represent depends upon the data type of the attribute. The following table describes what this information means for each data type.

Attribute Type	Contents of Word 1	Contents of Word 2
TEXT	Number of characters	0
INT	Number of values	0
REAL	Number of values	0
DOUB	Number of values	0
IVEC	Number of values	0
RVEC	Number of values	0
DVEC	Number of values	0
IMAT	Number of values	0
RMAT	Row dimension	Column dimension
DMAT	Row dimension	Column dimension

Now in our example, the data array would look like:

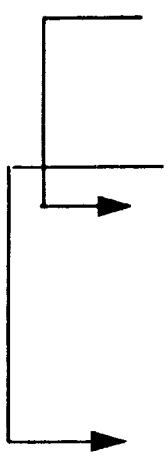
Attribute	Word	Data Array Values	
PARTNO	1	403	
DESC	2	12	(Pointer to word 12)
OPTEMP	3	325	
"	4	437	
"	5	665	
FREQRNG	6	10.2	
"	7	24.6	
"	8	20.5	
"	9	38.2	
"	10	40.8	
"	11	91.4	
DESC	12	16	(Number of characters)
"	13	0	(Always 0 for TEXT)
"	14	"LOW "	(Beginning of data)
"	15	"GAIN"	
"	16	" ANT"	
"	17	"ENNA"	

Now the value of DESC only occupies four words (16 characters at 4 characters per word), but the data array needs to be 17 words long instead of 15 words. For each variable length attribute, the number of words needed in the data array is three more than the number of words needed for the actual value (one for the pointer plus two for the information about variable length attribute).

For the last example, FREQRNG will become a variable length matrix. Both the number of rows and the number of columns will be variable. The definition of the example relation is now given by:

Attribute Name	Data Type	Length
PARTNO	INT	1
DESC	TEXT	VAR
OPTEMP	RVEC	3
FREQRNG	IMAT	VAR, VAR

Leaving the example data the same, the data array now would look like:



Attribute	Word	Data Array Values	
PARTNO	1	403	
DESC	2	7	(Pointer to word 7)
OPTEMP	3	325	
"	4	437	
"	5	665	
FREQRNG	6	13	(Pointer to word 13)
DESC	7	16	(Number of characters)
"	8	0	(Always 0 for TEXT)
"	9	"LOW "	(Beginning of data)
"	10	"GAIN"	
"	11	" ANT"	
"	12	"ENNA"	
FREQRNG	13	2	(Number of rows)
"	14	3	(Number of columns)
"	15	10.2	(Beginning of data)
"	16	24.6	
"	17	20.5	
"	18	38.2	
"	19	40.8	
"	20	91.4	

Because of the addition of the second variable length atribute, the data array now needs to be 20 words long to hold the data plus the information about each variable length attribute.

The data array should be a FORTRAN integer array. In order to transfer real and character data to and from the data array without data conversion, a real and/or a character array can be equivalenced to the data array. For the example above, the following FORTRAN code could be used:

```
INTEGER      IARRAY
REAL         RARRAY
CHARACTER*4  CARRAY
DIMENSION    IARRAY(20), RARRAY(20), CARRAY(20)
EQUIVALENCE  ( IARRAY(1), RARRAY(1), CARRAY(1) )
```

Using this technique, integer data is transferred via the IARRAY, real data via the RARRAY, and character data via the CARRAY. When calling LARCRIM subroutines, it's recommended that the data array parameter be a FORTRAN integer array.

In summary, remember the following points about the data array.

1. Attribute values in the data array are in the same order as declared in the relation definition.

2. The number of words required for each attribute depends upon the data type and size of the attribute and upon the computer's word size.

3. For variable length attributes, the position in tha data array normally used for the attribute's data, contains a pointer to another location within the data array. This new location contains information about the attribute followed by the attribute's data.

4. EQUIVALENCE statements are used to transfer non-integer data to and from the data array.

5. The data array should be dimensioned with sufficient space to hold the longest row in a relation. LARCRIM does not determine if the data array was dimensioned with sufficient space.

6. The recommended way to pass data between and application program and a LARCRIM database is via an integer array.

APPENDIX A: SUMMARY OF LARCRIM COMMANDS

Defining a database schema

```
DEFINE dbname
OWNER password
ATTRIBUTES
attname {REAL} [{length}] [KEY]
         INT      VAR
         TEXT
         DOUB
         RVEC
         IVEC
         DVEC

attname {RMAT}  {row,col} [KEY]
         IMAT    row,VAR
         DMAT    VAR,VAR

RELATIONS
relname WITH attname1 [attname2 ... ]
PASSWORDS
{READ PASSWORD} FOR {relname} IS password
 RPW                   ALL
{MODIFY PASSWORD} FOR {relname} IS password
 MPW                    ALL
RULES
attname [IN relname] {EQ} value [{AND} ...]
                      NE          OR
                      GT
                      GE
                      LT
                      LE
attname IN relname {EQA} attname IN relname [{AND} ...]
                    NEA                          OR
                    GTA
                    GEA
                    LTA
                    LEA
END
```

Loading a relation

```
LOAD relname
value1  value2 ... valuen
END
```

```
value: SCALARS val1
       TEXT "text string"
       VECTOR(val1, val2, ...)
       MATRICES ((r1c1,r2c1, ...)(r1c2,r2c2, ...) ...)
```

Querying a relation

```
SELECT {attname1[=fw1],attname2[=fw2], ... } FROM relname +
       attnum1 [=fw1], ...
       attname1(i), ...
       attname1(i,j),...
       ALL
       [SORTED BY attname1 [={A}], [attname2 [={A}], ...]]+
                                D                    D

       [WHERE ...]

TALLY attname [={A} FROM relname [WHERE ... ]]
                D

WHERE attname {EXISTS}              [{ AND} ... ]
              FAILS                   OR
              EQS                   value
              EQ                    value
              NE                      MAX
              GT                      MIN
              LT
              LE
WHERE attname {EQA} attname      [{AND} ... ]
              NEA                   OR
              GTA
              GEA
              LTA
              LEA
WHERE ROWS    {EQ}   rownumber    [{AND} ...]
              NE                    OR
              LT
              LE
              GE
              GT
WHERE {attname}{EQS}   list       [{AND} ... ]
      ROWS    EQ                    OR
              NE
WHERE  LIMIT  EQ      number    [AND ...]
```

Querying the schema

```
LISTREL [(relname)]
          ALL
EXHIBIT attname1 [attname2 ...]
PRINT RULES
```

Computation commands

```
COMPUTE  {COUNT} attname FROM relname [WHERE ... ]
          MIN
          MAX
          AVE
          SUM
```

Modification commands

```
CHANGE {attname} TO value [IN relname] WHERE ...
        attname(i)
        attname(i,j)
CHANGE {RPW} TO newpass FOR relname
        MPW
CHANGE OWNER TO newowner
DELETE ROWS FROM relname WHERE ...
DELETE DUPLICATES [attname1,attname2,...] FROM relname
DELETE RULE rulenumber
RENAME ATTRIBUTE attname TO newname [IN relname]
RENAME RELATION  relname TO newname
REMOVE relname
```

Relational algebra commands

```
INTERSECT relname1 WITH relname2 FORMING relname3 +
          [USING attname1 [attname2, ... ]]

JOIN relname1 USING attname1 WITH relname2 USING attname2 +
     FORMING relname3 [WHERE {EQ}]
                              NE
                              GT
                              GE
                              LT
                              LE

SUBTRACT relname1 FROM relname2 FORMING relname3 +
         [USING attname1 [attname2, ... ]]
PROJECT  relname1 FROM relname2 USING +
         {attname1,[attname2, ...]} [WHERE ... ]
          ALL
```

Report Commands

```
NEWPAGE
BLANK n
TITLE "title"
DATE
LINES n
WIDTH n
```

KEY Commands

```
BUILD  KEY FOR attname IN relname
DELETE KEY FOR attname IN relname
```

LARCRIM-TO-LARCRIM Commands

```
UNLOAD [dbname [=newdbname ]] { SCHEMA} [relname1 [=mpw]]+
                               DATA
                               ALL
       [relname2 [=mpw], ... ]
```

General Commands

```
INPUT      {filename}
            TERMINAL
OUTPUT     {filename}
            TERMINAL
EXIT
QUIT
MENU
HELP [command name]
USER password
ECHO
NOECHO
CHECK
NOCHECK
TOLERANCE tol [PERCENT]
RELOAD
CLOSE
OPEN dbname
```

APPENDIX B: SUMMARY OF THE APPLICATION PROGRAM INTERFACE

INITIALIZING THE DATABASE

```
CALL RMOPEN (dbname)
```

Input Parameter:

dbname	the name of the database in Hollerith format

```
CALL RMCLOS
```

GENERAL ROUTINES

```
CALL RMUSER (password)
```

Input Parameters:

password	the password in Hollerith format.

```
CALL RMRULE (switch)
```

Input Parameters:

switch	0 no rule checking (NOCHECK) 1 check rules (CHECK)

```
CALL RMTOL (value,switch)
```

Input Parameters:

value	the value of the tolerance (real)
switch	0 if value is the tolerance value (int) 1 if value is the tolerance percent

ACCESSING THE SCHEMA

```
CALL RMLREL
```

```
CALL RMGREL (rname,rpw,mpw,lastmod,numatt,numrows)
```

Output Parameters:

rname	relation name in Hollerith format.
rpw	read password (.TRUE. or .FALSE.)
mpw	modify password (.TRUE. or .FALSE.)
lastmod	date of last modification of relation data in YY/MM/DD format
numatt	number of attributes in the relation
numrows	number of rows of data in the relation

```
CALL RMLATT (rname)
```

Input Parameters:

rname	relation name in Hollerith format.

```
CALL RMGATT(aname,type,matvec,var,len1,len2,column,key)
```

Output Parameters:

aname	attribute name in Hollerith format
type	attribute type (INT, REAL, DOUB, or TEXT)
matvec	attribute type (VEC or MAT - otherwise blank)
var	variable length attribute (.TRUE. or .FALSE.)
len1	attribute length data as follows (int): TEXT - number of characters INT, REAL, DOUB, VEC - number of items MAT - row dimension
len2	column dimension of MAT attributes (otherwise 0)
column	attribute column location in the relation (int)
key	keyed attribute (.TRUE. or .FALSE.)

ACCESSING THE DATABASE

```
CALL RMFIND (number,relname)
```

Input Parameters:

number	user assigned number (0-5) used to reference the pointer for the relation the pointer for the relation
relname	relation name Hollerith format

```
CALL RMWHER(number,attname,operator,value,numval,nextboo,numboo)
```

Input Parameters:

number	number (0-5) which identifies the relation pointer for this operation
attname	array of attribute names, attribute numbers,the keyword ROWS, or LIMIT
operator	array of operators (EQ, GT, EQA, EQS, etc.)
value	2-dimensional array of WHERE clause "values"
numvall	number of "values' in the list of values.
nextboo	array of "AND" "OR" operators.
numboo	number of WHERE conditions (int).

```
CALL RMSORT (number,attname,numsort,sortype)
```

Input Parameters:

number	number (0-5) which identifies the relation pointer for this operation
attname	array of "numsort" attribute names to sort on
numsort	number of attributes to sort
sortype	sort control numbers -1 = descending sort 1 = ascending sort

```
CALL RMGET (number,array)
```

Input Parameters:

number	number (0-5) which identifies the relation pointer for this operation

Output Parameters:

array	array to receive the row of data.

```
CALL RMLOAD (number,array)
```

Input Parameters:

number	number (0-5) which identifies the relation to load.
array	array containing the row of data to load

```
CALL RMPUT (number,array)
```

Input Parameters:

number	number (0-5) which identifies the relation to be modified (int).
array	array containing the modified row of data

```
CALL RMDEL (number)
```

Input Parameters:

number	number (0-5) identifies the relation from which rows are to be deleted.

APPENDIX C: SAMPLE APPLICATION PROGRAM

PROGRAM SAMPLE

```
C
C
C     This sample FORTRAN program shows the use of the
C     application program interface to access the AERODB data-
C     base and print the following:
C
C     1) All the information about the schema ----------------
C
C                        (LISTREL ALL)
C
C     2)    The data in the relation REL300 for the airports in
C           Brazil sorted by descending altitude -- CITYNAME is
C           variable length
C           (SELECT ALL FROM REL300 SORTED BY ALTITUDE=D +
C            WHERE CITYNAME EQS "BRAZIL")
C
      LOGICAL RPW,MPW,VAR,KEY
      COMMON /RIMCOM/RMSTAT
      INTEGER RMSTAT
      REAL*8 NAME,LASTMD,NAMEA,NAMEC,IVAR,DBNAME
      DIMENSION NVAL(20)
      DIMENSION NAMEQS(5)

C         Open the database
C
        DBNAME=6HAERODB
        CALL RMOPEN(DBNAME)
C
C       LISTREL ALL
C


        CALL RMLREL
100     CONTINUE
        CALL RMGREL(NAME,RPW,MPW,LASTMD,NUMATT,NUMROW)
        IF(RMSTAT.NE.0) GO TO 200
        LRP = 3HNO
        IF (RPW) LRP=3HYESS
        MRP = 3HNO
        IF(MPW) MRP = 3HYES
        WRITE(6,110) NAME,LRP,MRP,LASTMD,NUMATT,NUMROW
110     FORMAT(1X,A8,2(21X,A4),1X,A8,sI8)
        CALL RMLATT(NAME)

120     CONTINUE
        CALL RMGATT(NAMEA,ITYPE,MAT,VAR,LEN1,LEN2,NCOL,KEY)
```

```
      IF(RMSTAT.NE.0) GO TO 100
      IVAR = 5HFIXED
      IF(VAR) IVAR = 8HVARIABLE
      IKEY = 2HNO
      IF(KEY) IKEY = 3HYES
      WRITE(6,130) NAMEA,ITYPE,MAT,IVAR,LEN1,LEN2,NCOL,IKEY
130   FORMAT(IX,A8,2(1X,A5),1X,A8,3I8,1X,A3)
      GO TO 120
200   CONTINUE
C
C        SELECT ALL FROM REL300 SORTED BY ALTITUDE=D +
C         WHERE CITYNAME EQS "BRAZIL"
C
      NAME=6HREL300
      CALL RMFIND(1,NAME)
      IF(RMSTAT.NE.0) GO TO 999
      NAMEQS(1) = 6
      NAMEQS(2) = 04
      NAMEQS(3) = HBRAZ
      NAMEQS(4)=2HIL
      NAMEC=8HCITYNAME
      IBOOP=3HEQS
      CALL RMWHER(1,NAMEC,IBOOP,NAMEQS,1,0,1)
      IF(RMSTAT.NE.0) GO TO 500
      NAMEA=8HALTITUDE
      CALL RMSORT(1,NAMEA,1,-1)
      IF(RMSTAT.NE.0) GO TO 999
300   CONTINIUE
      CALL RMGET(1,NVAL)
      IF(RMSTAT.NE.0) TO TO 500
      NUMX = (NVAL(5)-1)/10 + 1
      NUMP = 6 + NUMX
      WRITE(6,400) (NVAL(K),K=1,NUMP)
400   FORMAT(A4,5I6,2X,3A10)
      GO TO 300
500   CONTINUE
      IF (RMSTAT .LT.0) GO TO 1000
999   CONTINUE
      WRITE(6,9001) RMSTAT
 9001 FORMAT(6HRMSTAT,I5)
C
C     Close the database
C
 1000 CONTINUE
      CALL RMCLOS
      END
```

APPENDIX D: LIMITATIONS

1. There is no limit on the size of a relation (number of rows) or the number of relations other than hardware or operating system imposed, e.g., mass storage availability.

2. A row in a relation must fit in 1021 computer words. If len(i) is the fixed length (in words) of the ith attribute, and if var(j) is the length (in words) of the jth variable length attribute, then:

   $$(\Sigma\ \text{len}(i) + \Sigma\ (\text{var}(j)+3)) < 1021$$

   Note that if a relation fits on a CYBER (60 bit machine), it may not fit on a 32 bit machine. This is important to consider when using LARCRIM-to-LARCRIM communications (UNLOAD command).

3. A relation or attribute name must not begin with the character string RMRUL.

4. The following words may not be used as attribute or relation names:

   TO, FROM, BY, USING, WHERE, IN, FORMING, ROWS, LIMIT, DUPLICATE

   Also, names must not be a substring of the above which is 3 characters or more in length starting with the first character (LARCRIM substring). For example, FOR and FORM are not allowed, however FORT is legal.

5. In loading data, the value of the first attribute, if it is TEXT, is limited as follows:

   If the relation contains only one or two attributes, then the following text strings and their LARCRIM substrings are not allowed as values for the first attribute:

   CHECK, NOCHECK, ECHO, NOECHO, END, HELP, INPUT, OUTPUT,QUIT

   If the relation contains three attributes, then the value for the first attribute may not be:

   HELP, HEL

6. The number of items in one command may not exceed 100.

7. The number of rules specified for one relation may not exceed 10.

8. The number of conditions used in the SELECT WHERE clause may not exceed 10.

APPENDIX E: ENTERING INPUT WITH THE LARCRIM USER INTERFACE

The following is a detailed discussion of the algorithms used for reading and parsing LARCRIM commands and data in the stand-alone system. It is intended to detail the significant flexibility available to the experienced LARCRIM user.

The LARCRIM user interface is a free-field input routine, used by the LARCRIM stand-alone system, which separates user input into items which are grouped into records.

TERMINOLOGY

LINE - One line of information with a maximum of 80 characters including blanks.

ITEM - One piece of information. An item may be a real number, an integer or text. Items are delimited by blanks or commas. Multiple blanks count as a single blank. Multiple commas generate null items (see section on multiple commas).

RECORD -A collection or list of up to 100 items entered in response to a single request for data by the calling program.

INTEGER -All characters must be numeric except the first one which may be + or -. For example: -1 23 +10000

REAL - An item of the form I1.I2EI3 where I1 and I3 may be signed integers and I2 is an unsigned integer. The entire form is not necessary but at least one digit and the . or two digits separated by the E must be present. For example: 1. 1E-3 -2.7E+4 .0 The size of real numbers are limited to the range between 1.0E+38 to 1.0E-38.

TEXT - Any single item which is not an integer or real. If a text item looks like an integer or real or if it contains blanks or commas, it must be enclosed in quotes (").

COMPOSING RECORDS

Ordinarily records consist of one line. However, multiple records may be put on one line by separating them with dollars or semicolons. Alternatively, a record may span several lines by ending all but the last line with a plus. Items must be wholly contained on one line with the exception of quoted text items and comments.

SPECIAL ITEMS - =,(,)

Equals and left and right parentheses are treated as single items unless enclosed in quoted text items. Thus a=3. is 3 items (two text and one real) rather than one item. "a=3." is one text item. This allows more convenient parsing of many commands.

MULTIPLE COMMAS

If more than one comma separates two items, each additional comma will generate a text item with three characters "-0-". thus, ,,abc,,2.5 is equivalent to -0-,abc,-0-,2.5.

RULES FOR TEXT ITEMS

A quoted text item is terminated by a record separator (dollar or semi-colon). Quoted text items may be continued on multiple lines. If the trailing quote is omitted on the last item in a record, the quoted item is terminated at the record separator, if any, or the last non-blank character on the line. Quotes may be included in quoted text items by doubling the quotes (e.g. "a," "b yields, a, "b as a text string).

SOME EXAMPLES

```
    1,2. ABC "2."
```
This record has four items - integer,real, and two text

```
    1 $ 2
```
This line is two records - each one integer

```
    1 +
    2
```
This is one record on two lines with two integers

COMMENTS

Comments may be included anywhere in the input stream by enclosing them between *(and). For example *(this is a comment). Comments are completely ignored by the user interface. Empty lines between records are also ignored and may be used to paragraph input. An alternative form of comment is */..../ where slashes replace the parentheses. This may be used if parentheses are needed in the comment.

DATA GENERATION

Activities such as entering large volumes of data, repeating similar records and reentering mis-typed records can be eased by using the LARCRIM user interface data generation facilities.

REPEATING ITEMS ON PREVIOUS RECORD - *N,**,*

A data item of the form *n where n is an unsigned integer indicates that the next n items are identical to the corresponding n items in the preceding record. An isolated * is treated as *1. Double asterisks (**) indicate that the remaining items in the previous record are to be copied into the current record.

REPEATING AN ITEM IN THE CURRENT RECORD - *=N *=N+STEP

An item of the form *=n, where n is an unsigned integer, indicates that the next n items are identical to the immediately preceding item. An item of the form *=n+step or *=n-step where step is an unsigned real or integer, indicates that the next n items are to be generated by consecutively incrementing the immediately preced- ing item.

GENERATING MULTIPLE RECORDS - *+N

A record beginning with *+n where n is an unsigned integer indi- cates that the next n records are to be generated from the preceding record. Each item of the generated record is formed by adding an item of the *+n record to the corresponding item of the immediately preceding input or generated record. A zero (integer) item should be inserted in an *+n record for text items in the preceding record. The number of items after the *+n must match the number in the preceding record.

NOTE ON GENERATING ITEMS

When increments are specified, either on the *+n record or as step on an *=n+step item they must match the item they are incrementing in type. It should be noted that the *+n record generation option is based on the expanded representation of the previous record. The generation does not operate on the preceding record if it contains data generation items. Therefore, it is not possible to repeat or increment an asterisk-type item.

EXAMPLES

Consider the following seven input records to illustrate the data generation features.

```
1 2 3 4 5 6 7 8 9 10 11 12
2 1 *2 4 *=2 1 *=2+2 **
*+1 0 *=3 0 *=5 **
*+1 0 *=11
*+1 *12
*+1 **
**
```

Twelve data items are defined by each of these records. Each of the last six records is translated into the same internal record which is:

```
2 1 3 4 4 4 4 1 3 5 11 12
```

Note - the last five records could be replaced by the single record:

```
*+5 **
```

CHANGING SPECIAL CHARACTERS

It is possible to change the special characters the user interface uses to break apart records. These special characters may either be changed to another character or set to null so that they are ignored. This is useful for reading specially formatted files or to allow special characters to be input as text items. To change special characters enter the following special comment as the only entry on a line between records.

```
*(SET KEYWORD=newvalue)
```

where KEYWORD can be
```
DOLLAR
SEMI
QUOTES
BLANK
PLUS
COMMA
```

and newvalue is either the word null or the new special character. For example, if one wanted to use dollars to delimit items rather than records and to not have commas delimit items, the following two lines could be entered.

```
*(SET DOLLAR=NULL)
*(SET COMMA=$)
```

Note that commas could now be used in unquoted text strings and dollars could now be included in quoted text strings. Also, note that it is really the function that is being altered, not the character. Changing plus only changes the line continuation character, not the representation of real numbers. To restore the original condition after the above example, the following could be entered.

```
*(SET DOLLAR=$)
*(SET COMMA=,)
```

Warning - using the same character for multiple functions will produce undefined results.

APPENDIX F: HOST DEPENDENT INSTRUCTIONS

Reserved for future use.

APPENDIX G : DATABASE FILES

Each database consists of three LARCRIM-generated files whose logical names are formed by appending a 1, 2, or 3 to the 1-6 character database name "dbname". The first file contains the database definition (schema) data, the second file contains the actual data for each relation, and the third file contains the pointers for the "keyed" attributes. LARCRIM uses units 5 and 6 for input and output.

If your database files reside on your directory with names different from the logical database file names, you must use assign control statements prior to the LARCRIM execution to assign required names to your database files.

References:

F. P. Gray, S. O. Wahlstrom; Boeing Commercial Airplane Co.; *USER GUIDE, RIM 5.0, VAX VMS*, Vol 1 of 2; 1982. (Available through COSMIC)

Recommended Reading:

C. J. Date; *DATABASE, A Primer*, Addison-Wesley Publishing Co.; 1983

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	January 1993	Contractor Report

4. TITLE AND SUBTITLE
LARCRIM User's Guide, Version 1.0

5. FUNDING NUMBERS
C NAS1-19038
WU 505-90-53-02

6. AUTHOR(S)
John S. Davis
William J. Heaphy

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Computer Sciences Corporation
3217 N. Armistead Ave.
Hampton, VA 23666-1379

8. PERFORMING ORGANIZATION REPORT NUMBER
TAO 60306

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23681-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA CR-191416

11. SUPPLEMENTARY NOTES
Langley Technical Monitor: Kennie H. Jones

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified -Unlimited

Subject Category 61

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

LARCRIM is a relational database management system (RDBMS) which performs the conventional duties of an RDBMS with the added feature that it can store attributes which consist of arrays or matrices. This makes it particularly valuable for scientific data management. It is accessible as a stand-alone system and through an application program interface. The standalone system may be executed in two modes: menu or command. The menu mode prompts the user for the input required to create, update, and/or query the database. The command mode requires the direct input of LARCRIM commands. Although LARCRIM is an update of an old database family, its performance on modern computers is quite satisfactory. LARCRIM is written in FORTRAN 77 and runs under the UNIX operating system. Versions have been released for the following computers: SUN (3 & 4), Convex, IRIS, Hewlett-Packard, CRAY 2 & Y-MP.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Database Management System; Relational DBMS; Scientific and engineering DBMS	103
	16. PRICE CODE A06

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	