UNCLASSIFIED

IDA MEMORANDUM REPORT M-459

AN ORACLE * - Ada/SQL IMPLEMENTATION

Bill R. Brykczynski
Fred Friedman
Kerry Hilliard

April 1988

Prepared for
WIS Joint Program Management Office

INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, Virginia 22311-1772

* ORACLE is a registered trademark of the Oracle Corporation

UNCLASSIFIED

IDA Log No. HQ 88-033301

DEFINITIONS

IDA publishes the following documents to report the results of its work.

Reports

Reports are the most authoritative and most carefully considered products IDA publishes. They normally embody results of major projects which (a) have a direct bearing on decisions affecting major programs, or (b) address issues of significant concern to the Executive Branch, the Congress and/or the public, or (c) address issues that have significant economic implications. IDA Reports are reviewed by outside panels of experts to ensure their high quality and relevance to the problems studied, and they are released by the President of IDA.

Papers

Papers normally address relatively restricted technical or policy issues. They communicate the results of special analyses, interim reports or phases of a task, ad hoc or quick reaction work. Papers are reviewed to ensure that they meet standards similar to those expected of refereed papers in professional journals.

Memorandum Reports

IDA Memorandum Reports are used for the convenience of the sponsors or the analysts to record substantive work done in quick reaction studies and major interactive technical support activities; to make available preliminary and tentative results of analyses or of working group and panel activities; to forward information that is essentially unanalyzed and unevaluated; or to make a record of conferences, meetings, or briefings, or of data developed in the course of an investigation. Review of Memorandum Reports is suited to their content and intended use.

The results of IDA work are also conveyed by briefings and informal memoranda to sponsors and others designated by the sponsors, when appropriate.

The work reported in this document was conducted under contract MDA 903 84 C 0031 for the Department of Defense. The publication of this IDA document does not indicate endorsement by the Department of Defense, nor should the contents be construed as reflecting the official position of that agency.

This Memorandum Report is published in order to make available the material it contains for the use and convenience of interested parties. The material has not necessarily been completely evaluated and analyzed, nor subjected to IDA review.

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION Unclassified	**1b RESTRICTIVE MARKINGS**
2a SECURITY CLASSIFICATION AUTHORITY	**3 DISTRIBUTION/AVAILABILITY OF REPORT** Approved for public release, unlimited distribution.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) IDA Memorandum Report M-459	**5 MONITORING ORGANIZATION REPORT NUMBER(S)**
6a NAME OF PERFORMING ORGANIZATION Institute for Defense Analyses **6b OFFICE SYMBOL** IDA	**7a NAME OF MONITORING ORGANIZATION** OUSDA, DIMO
6c ADDRESS (City, State, and Zip Code) 1801 N. Beauregard St. Alexandria, VA 22311	**7b ADDRESS (City, State, and Zip Code)** 1801 N. Beauregard St. Alexandria, VA 22311
8a NAME OF FUNDING/SPONSORING ORGANIZATION WIS Joint Program Management Office **8b OFFICE SYMBOL (if applicable)** WJPMO	**9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER** MDA 903 84 C 0031
8c ADDRESS (City, State, and Zip Code) WIS JPMO/DXP Room 5B19, The Pentagon Washington, D.C. 20330-6600	**10 SOURCE OF FUNDING NUMBERS** PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. T-W5-206 / WORK UNIT ACCESSION NO.
11 TITLE (Include Security Classification) An Oracle - Ada/SQL Implementation (U)	
12 PERSONAL AUTHOR(S) Bill R. Brykczynski, Fred Friedman, Kerry Hilliard	
13a TYPE OF REPORT Final **13b TIME COVERED** FROM _____ TO _____	**14 DATE OF REPORT (Year, Month, Day)** 1988 April **15 PAGE COUNT** 88
16 SUPPLEMENTARY NOTATION	
17 COSATI CODES FIELD / GROUP / SUB-GROUP	**18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)** Ada programming language; Structured Query Language (SQL); Oracle database management system (DBMS); World Wide Military Command and Control System (WWMCCS) Information System (WIS); (continued)

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The purpose of IDA Memorandum Report M-459, *An Oracle - Ada/SQL Implementation*, is to identify and describe a version of a software delivery, an Oracle - Ada/SQL (Structured Query Language) implementation, to the WIS Joint Program Management Office. The software system provides a Level 1 Ada/SQL implementation which interacts with the Oracle database management system (DBMS). It provides the ability to define Ada/SQL Data Definition Language (DDL) and Ada/SQL Data Manipulation Language (DML) which will be converted to the appropriate SQL calls required by the Oracle DBMS. Ada/SQL is a binding between Ada and the database programming language SQL. The software was developed on a VAX 8600, using VAX/VMS version 4.6, and the DEC Ada compiler, version 1.4-33.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	**21 ABSTRACT SECURITY CLASSIFICATION** Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Mr. Bill R. Brykczynski **22b TELEPHONE (Include area code)** (703) 824-5515 **22c OFFICE SYMBOL** IDA/CSED	

18 SUBJECT TERMS (Continued)

Data Definition Language (DDL); Data Manipulation Language (DML); software tools and techniques; software; prototype; VAX/VMS.

IDA MEMORANDUM REPORT M-459

AN ORACLE - Ada/SQL IMPLEMENTATION

Bill R. Brykczynski
Fred Friedman
Kerry Hilliard

April 1988

INSTITUTE FOR DEFENSE ANALYSES

Contract MDA 903 84 C 0031
Task T-W5-206

CONTENTS

Preface

The purpose of IDA Memorandum Report 459 is to identify and describe a version of a software delivery, "An Oracle - Ada/SQL Implementation," to the WIS Joint Program Management Office. The term *version* is applied to the initial release as well as to all interim changes.

This report was written to describe the software developed to satisfy deliverable 5.a under task order T-W5-206, entitled WIS Application Software Study. The purpose of this software system is to provide a Level 1 Ada/SQL implementation which interacts with the Oracle® database management system. This software provides the ability to define Ada/SQL Data Definition Language (DDL) and Ada/SQL Data Manipulation Language (DML) which will be converted into the appropriate SQL calls required by the Oracle DBMS.

ORACLE is a registered trademark of Oracle Corporation.

1. SCOPE

1.1 System Overview

The purpose of this software system is to provide a Level 1 Ada/SQL implementation which interacts with the Oracle database management system. This software provides the ability to define Ada/SQL Data Definition Language (DDL) and Ada/SQL Data Manipulation Language (DML) which will be converted into the appropriate SQL calls required by the Oracle DBMS. Ada/SQL is a binding between the Ada programming language [ADA 83] and the database programming language SQL [SQL 86]. Ada/SQL, like SQL, is comprised of two main components: a DDL and a DML. Both of these are coded using pure Ada syntax and semantics. The DDL resides in a package specification, and is used to define the data types, variable definitions, and table and column definitions. The DML is expressed as syntax very similar to the syntax of SQL DML. This expression is allowed due to a set of underlying operators and subprograms which must be 'withed' by the application. A tool which aids in the generation of these subprograms, named the application scanner, is available as IDA Memorandum Report M-460 [IDA 88]. The use of the application scanner tool is necessary for this particular prototype to function properly.

This specific version of Ada/SQL is named Level 1 Ada/SQL. The Level 1 definition can be found in [ADA 87]. The limitations and constraints placed on Ada/SQL for Level 1 reflect a desire to produce a working prototype as quickly as possible, without losing the benefit of Ada's strong typing and enumeration types. This work was based on the Ada/SQL definition found in [IDA 86].

1.2 Documentation Overview

The file [BBRYKCZYN.EXAMPLE]READ.ME is included on the magnetic tape containing the Oracle Ada/SQL implementation and the application scanner. This file contains guidelines which show the user how to create an Ada/SQL application, use the application scanner, and in what order to compile the output from the scanner. The directory located in [BBRYKCZYN.EXAMPLE] provides a comprehensive example of using the Ada/SQL system.

2. REFERENCED DOCUMENTS

[ADA 83] U.S. Department of Defense. 1983. *ANSI/MIL-STD-1815A, Military standard: Ada programming language*. Washington, D.C.: U.S. DoD.

[ANSI 86] American National Standards Institute. 1986. *ANSI X3.135.1986, Database language SQL*. New York: ANSI.

[IDA 86] Brykczynski, Bill and Fred Friedman. 1986. *Preliminary version: Ada/SQL: A standard, portable Ada-DBMS interface.* Alexandria, VA: Institute for Defense Analyses. IDA Paper P-1944.

[IDA 87] Brykczynski, Bill, Fred Friedman, and Kerry Hilliard. 1987. *Level 1 Ada/SQL database language interface user's guide.* Alexandria, VA: Institute for Defense Analyses. IDA Memorandum Report M-361.

[IDA 88] Brykczynski, Bill, and Fred Friedman, Kevin Heatwole, and Kerry Hilliard. 1988. *An Ada/SQL application scanner.* Alexandria, VA: Institute for Defense Analyses. IDA Memorandum Report M-460.

3. VERSION DESCRIPTION

3.1 Inventory of Materials Released

This prototype Ada/SQL system was developed on a VAX™ 8600, using VAX/VMS version 4.6, and the DEC Ada compiler, version 1.4-33. The magnetic tape upon which the source is located is in VAX/VMS backup format, and the save set name is ADASQL. This tape requires 8192 blocks of memory. To load the tape, allocate the tape drive desired, request a tape mount, and issue the following command: "BACKUP MUA0: [appropriate directory...]*.*.*", where MUA0 is the logical tape drive name, and appropriate directory is the directory in which you will be placing the contents of the tape.

Located on the tape is a file named READ.ME. This file contains instructions for executing the example queries found in the [.EXAMPLE] directory. Note that the use of this example assumes the use of the application scanner.

3.2 Inventory of Software Prototype Contents

The following are the files which make up the prototype Ada/SQL system. They are listed in compilation order.

DATABASE.ADA
TXTPRTS.ADA
TXTPRTB.ADA
ORDEFS.ADA
ORCUDEFS.ADA
ORHLIS.ADA
ORINTS.ADA
FUNCTIONS.ADA
ORINTB.ADA
FUNCTIONB.ADA
CURSOR_DEFINITION.ADA
SCHEMA_DEFINITION.ADA

Portions of this software were developed by SAIC Comsystems, under contract by the WIS Joint Program Management Office, and are located in the Naval Ocean Systems Center (NOSC) TECR software repository.

In addition, an example set of DDL and DML are located in the [.EXAMPLE] directory. See [.EXAMPLE]READ.ME for details.

VAX is a trademark of Digital Equipment Corporation.

3.3 Changes

Not applicable.

3.4 Adaptation Data

Not applicable.

3.5 Interface Compatibility

Not applicable.

3.6 Bibliography of Reference Documents

Date, C.J. *A guide to the SQL standard.* New York: Addison-Wesley, 1987

Brykczynski, Bill and Fred Friedman. *Ada/SQL binding specifications.* Alexandria, VA: Institute for Defense Analyses, 1988. IDA Memorandum Report M-362.

3.7 Summary of Changes

Not applicable.

3.8 Installation Instructions

To load the tape, allocate the tape drive desired, request a tape mount, and issue the following command: "BACKUP MUA0: [appropriate directory...]*.*.*", where MUA0 is the logical tape drive name, and appropriate directory is the directory in which you will be placing the contents of the tape.

3.9 User Guidelines

The following is a set of guidelines for using the VAX/VMS Level 1 Ada/SQL system. These guidelines assume that a directory exists which contains the files loaded from tape. The files on the tape were loaded from a directory named [BBRYKCZYN.ORACLE]. Of course, when a tape is loaded on another system, this path name will be different.

1) Create the ADASQL$ENV logical

There are several files which are read by the application scanner to establish a predefined environment for processing application programs. These files are DATABASE.ADA, CURSOR_DEFINITION.ADA, and STANDARD.ADA. These files are not source files that are linked with the Ada/SQL application programs. They must, however, be stored in a directory that is accessible via the VAX/VMS logical name ADASQL$ENV whenever the application

scanner is run. These files should not be compiled or otherwise used for any purpose other than that described here. To assign the logical, type in the following:

ASSIGN [BBRYKCZYN.ORACLE.STANDARDS] ADASQL$ENV

2) Copy over the AUTH_PACK.ADA file

In SQL, a module must contain an authorization identifier which identifies the user. In Ada/SQL, the authorization identifier must reside in a file called AUTH_PACK.ADA. At this time, it is necessary only to copy an AUTH_PACK.ADA from another directory and compile it into the library. A sample AUTH_PACK.ADA is located in [BBRYKCZYN.ORACLE.EXAMPLE]

3) Create the Ada/SQL application specific files

There are four files one must create in order to use Ada/SQL. These are the _TYPES.ADA, _VARIABLES.ADA, _DDL.ADA packages, and the main program. The files must be named exactly as the package name, with the addition of a '.ADA' suffix. Examples of these files are included in the [BBRYKCZYN.ORACLE.EXAMPLE] directory.

4) Create the Oracle DDL

It is necessary for Oracle to have the data definition exist prior to the running of an Ada/SQL program. If you are building an Ada/SQL program to access a pre-existing database definition, this step can be deleted. If you are building a new application, it will be necessary to invoke Oracle, and create the appropriate table and column definitions.

5) Run the scanner

To execute the application scanner, type in the following command: "RUN [BBRYKCZYN.ORACLE.APSCAN_SOURCE]APSCAN.EXE". The application scanner will prompt you with several questions:

a) **Enter DML filename:**

Here you enter the name of your Ada compilation unit which contains DML statements which you want processed by the application scanner. An output file will be generated where errors in the DDL will be reported. This file will have the name of your compilation unit's library name, suffixed with .DDLOUT. For example enter BILL.ADA here (the subprogram name is BILL) and any DDL errors will be listed in a file called BILL.DDLOUT.

b) **Enter listing filename:**

Here you enter the name of a file where the application scanner will report errors in the DML. For example if you had entered BILL.ADA for question one you could enter BILL.LST here. Only DML errors will be reported here, DML errors are in the *.DDLOUT file.

c) **Enter filename for generated functions:**

Here you enter the file name for the compilation unit which will contain the functions

necessary to make your DML compilation unit compilable. This will be an Ada compilation unit which will become a part of your program. For example if you had entered BILL.ADA for question one you could enter BILL_ADA_SQL.ADA here. The library unit name for this compilation unit will be the library unit name of your compilation unit with an extension of _ADA_SQL. The subprogram name in BILL.ADA is BILL), and the library unit name of the compilation unit BILL_ADA_SQL.ADA will be BILL_ADA_SQL. Your compilation unit must

The application scanner will then notify you:

Invoking application scanner with:
DML filename => file name you entered in #1 above
Listing filename => file name you entered in #2 above
Generated package => file name you entered in #3 above

When the application scanner is complete it will issue the message:

%ADASQL-I-SCAN, Scan completed with errors
or the message:

%ADASQL-I-SCAN, Scan completed with no errors

In the case of 'with errors' check the *.DDLOUT file to make sure the DDL scanned correctly, then check the listing file you specified in #2 above to see if there was an error in the DML. Correct the errors and run the application scanner again. In the case of 'with no errors' you must still check the *.DDLOUT file. If errors are reported in this file but not in the listing file the message at the end of the application scanner will indicate no errors.

Repeat these steps until you have generated a function package through the application scanner for all your compilation units which contain DML. The package generated by the application scanner must be withed in your compilation unit.

6) Compile the output of the scanner

When a correct version of Ada/SQL DML is processed by the scanner, a generated package will be produced which must be compiled. This package contains the various subprograms which allow the Ada/SQL DML to interact with the database.

7) Compile and link the DML package

After compiling the generated *_ADA_SQL.ADA package from the previous step, the Ada/SQL DML package may now be compiled. Continuing with the example, this file is named BILL.ADA. After compiling, the file must be linked, which, in this example, results in an executable named BILL.

8) Execute the application

3.10 Source Listings

3.10.1 package DATABASE.ADA

```
package DATABASE is
  type INT        is new STANDARD.INTEGER;
  type DOUBLE_PRECISION is new STANDARD.FLOAT;
  type CHAR       is new STANDARD.STRING;
  type CHAR_LINK is access CHAR;
  type USER_AUTHORIZATION_IDENTIFIER is new STANDARD.STRING;
  type USER_AUTHORIZATION_IDENTIFIER_LINK is access
       USER_AUTHORIZATION_IDENTIFIER;
  type COLUMN_NUMBER is new STANDARD.INTEGER;
end DATABASE;
```

3.10.2 package TXTPRTS.ADA

```
-- txtprts.ada - print utilities

with TEXT_IO;
  use TEXT_IO;
package TEXT_PRINT is

  type BUFFER_ACCESS_TYPE is access STRING;

  type QUERY_BUFFER_RECORD is
    record
      QUERY_LENGTH : INTEGER := 0;
      QUERY_BUFFER : BUFFER_ACCESS_TYPE;
    end record;

  type ACCESS_QUERY_BUFFER is access QUERY_BUFFER_RECORD;

  procedure SET_BUFFER ( BUFFER : ACCESS_QUERY_BUFFER );

  procedure UNSET_BUFFER;
  type LINE_TYPE is limited private;

  type BREAK_TYPE is (BREAK, NO_BREAK);

  type PHANTOM_TYPE is private;

  procedure CREATE_LINE(LINE : in out LINE_TYPE; LENGTH : in POSITIVE);

  procedure SET_LINE(LINE : in LINE_TYPE);

  function CURRENT_LINE return LINE_TYPE;

  procedure SET_INDENT(LINE   : in LINE_TYPE; INDENT : in NATURAL);
  procedure SET_INDENT(INDENT : in NATURAL);

  procedure SET_CONTINUATION_INDENT(LINE   : in LINE_TYPE;
```

```
                                    INDENT : in INTEGER);
procedure SET_CONTINUATION_INDENT(INDENT : in INTEGER);

function MAKE_PHANTOM(S : STRING) return PHANTOM_TYPE;

procedure SET_PHANTOMS(LINE          : in LINE_TYPE;
                       START_PHANTOM,
                       END_PHANTOM   : in PHANTOM_TYPE);

procedure SET_PHANTOMS(START_PHANTOM, END_PHANTOM : in PHANTOM_TYPE);

procedure PRINT(FILE : in FILE_TYPE;
                LINE : in LINE_TYPE;
                ITEM : in STRING;
                BRK  : in BREAK_TYPE := BREAK);
procedure PRINT(FILE : in FILE_TYPE;
                ITEM : in STRING;
                BRK  : in BREAK_TYPE := BREAK);
procedure PRINT(LINE : in LINE_TYPE;
                ITEM : in STRING;
                BRK  : in BREAK_TYPE := BREAK);
procedure PRINT(ITEM : in STRING;
                BRK  : in BREAK_TYPE := BREAK);

procedure PRINT_LINE(FILE : in FILE_TYPE; LINE : in LINE_TYPE);
procedure PRINT_LINE(FILE : in FILE_TYPE);
procedure PRINT_LINE(LINE : in LINE_TYPE);
procedure PRINT_LINE;

procedure BLANK_LINE(FILE : in FILE_TYPE; LINE : in LINE_TYPE);
procedure BLANK_LINE(FILE : in FILE_TYPE);
procedure BLANK_LINE(LINE : in LINE_TYPE);
procedure BLANK_LINE;

generic
  type NUM is range <>;
package INTEGER_PRINT is

  procedure PRINT(FILE : in FILE_TYPE;
                  LINE : in LINE_TYPE;
                  ITEM : in NUM;
                  BRK  : in BREAK_TYPE := BREAK);
  procedure PRINT(FILE : in FILE_TYPE;
                  ITEM : in NUM;
                  BRK  : in BREAK_TYPE := BREAK);
  procedure PRINT(LINE : in LINE_TYPE;
                  ITEM : in NUM;
                  BRK  : in BREAK_TYPE := BREAK);
  procedure PRINT(ITEM : in NUM;
                  BRK  : in BREAK_TYPE := BREAK);
```

```
    procedure PRINT(TO : out STRING; LAST : out NATURAL; ITEM : in NUM);

  end INTEGER_PRINT;

  generic
    type NUM is digits <>;
  package FLOAT_PRINT is

    procedure PRINT(FILE : in FILE_TYPE;
                    LINE : in LINE_TYPE;
                    ITEM : in NUM;
                    BRK  : in BREAK_TYPE := BREAK);
    procedure PRINT(FILE : in FILE_TYPE;
                    ITEM : in NUM;
                    BRK  : in BREAK_TYPE := BREAK);
    procedure PRINT(LINE : in LINE_TYPE;
                    ITEM : in NUM;
                    BRK  : in BREAK_TYPE := BREAK);
    procedure PRINT(ITEM : in NUM;
                    BRK  : in BREAK_TYPE := BREAK);

    procedure PRINT(TO : out STRING; LAST : out NATURAL; ITEM : in NUM);

  end FLOAT_PRINT;

  NULL_PHANTOM : constant PHANTOM_TYPE;

  LAYOUT_ERROR : exception renames TEXT_IO.LAYOUT_ERROR;

private

  type PHANTOM_TYPE is access STRING;

  type LINE_REC(LENGTH : INTEGER) is
    record
      USED_YET             : BOOLEAN := FALSE;
      INDENT               : INTEGER := 0;
      CONTINUATION_INDENT  : INTEGER := 2;
      BREAK                : INTEGER := 1;
      INDEX                : INTEGER := 1;
      DATA                 : STRING(1..LENGTH);
      START_PHANTOM,
      END_PHANTOM          : PHANTOM_TYPE := NULL_PHANTOM;
    end record;

  type LINE_TYPE is access LINE_REC;

  NULL_PHANTOM : constant PHANTOM_TYPE := new STRING'("");

end TEXT_PRINT;
```

3.10.3 package TXTPRTB.ADA

```
-- txtprtb.ada - print utilities

package body TEXT_PRINT is

  DEFAULT_LINE : LINE_TYPE;

  CURRENT_BUFFER : ACCESS_QUERY_BUFFER := null;

  procedure SET_BUFFER ( BUFFER : ACCESS_QUERY_BUFFER ) is
  begin
    CURRENT_BUFFER := BUFFER;
  end SET_BUFFER;

  procedure UNSET_BUFFER is
  begin
    CURRENT_BUFFER := null;
  end UNSET_BUFFER;

  procedure CREATE_LINE(LINE : in out LINE_TYPE; LENGTH : in POSITIVE) is
  begin
    LINE := new LINE_REC(LENGTH);
  end CREATE_LINE;

  procedure SET_LINE(LINE : in LINE_TYPE) is
  begin
    DEFAULT_LINE := LINE;
  end SET_LINE;

  function CURRENT_LINE return LINE_TYPE is
  begin
    return DEFAULT_LINE;
  end CURRENT_LINE;

  procedure SET_INDENT(LINE   : in LINE_TYPE; INDENT : in NATURAL) is
  begin
    if CURRENT_BUFFER /= null then
      return;
    end if;
    if INDENT >= LINE.LENGTH then
      raise LAYOUT_ERROR;
    end if;
    if LINE.INDEX = LINE.INDENT + 1 then
      for I in 1..INDENT loop
        LINE.DATA(I) := ' ';
      end loop;
      LINE.INDEX := INDENT + 1;
    end if;
    LINE.INDENT := INDENT;
  end SET_INDENT;
```

```
procedure SET_INDENT(INDENT : in NATURAL) is
begin
  SET_INDENT(DEFAULT_LINE,INDENT);
end SET_INDENT;

procedure SET_CONTINUATION_INDENT(LINE   : in LINE_TYPE;
                                  INDENT : in INTEGER) is
begin
  if CURRENT_BUFFER /= null then
    return;
  end if;
  if LINE.INDENT + INDENT >= LINE.LENGTH or else LINE.INDENT + INDENT < 0
      then
    raise LAYOUT_ERROR;
  end if;
  LINE.CONTINUATION_INDENT := INDENT;
end SET_CONTINUATION_INDENT;

procedure SET_CONTINUATION_INDENT(INDENT : in INTEGER) is
begin
  SET_CONTINUATION_INDENT(DEFAULT_LINE,INDENT);
end SET_CONTINUATION_INDENT;

function MAKE_PHANTOM(S : STRING) return PHANTOM_TYPE is
begin
  return new STRING'(S);
end MAKE_PHANTOM;

procedure SET_PHANTOMS(LINE          : in LINE_TYPE;
                       START_PHANTOM,
                       END_PHANTOM   : in PHANTOM_TYPE) is
begin
  if CURRENT_BUFFER /= null then
    return;
  end if;
  LINE.START_PHANTOM := START_PHANTOM;
  LINE.END_PHANTOM := END_PHANTOM;
end SET_PHANTOMS;

procedure SET_PHANTOMS(START_PHANTOM, END_PHANTOM : in PHANTOM_TYPE)
  is
begin
  SET_PHANTOMS(DEFAULT_LINE,START_PHANTOM,END_PHANTOM);
end SET_PHANTOMS;

procedure PRINT(FILE : in FILE_TYPE;
                LINE : in LINE_TYPE;
                ITEM : in STRING;
                BRK  : in BREAK_TYPE := BREAK) is
  NEW_BREAK, NEW_INDEX : INTEGER;
begin
```

```
  if CURRENT_BUFFER /= null then
    NEW_INDEX :=
     CURRENT_BUFFER.QUERY_BUFFER'FIRST + CURRENT_BUFFER.QUERY_LENGTH;
    NEW_BREAK := NEW_INDEX + ITEM'LENGTH - 1;
    if NEW_BREAK > CURRENT_BUFFER.QUERY_BUFFER'LAST then
      raise LAYOUT_ERROR;
    end if;
    CURRENT_BUFFER.QUERY_BUFFER ( NEW_INDEX .. NEW_BREAK ) := ITEM;
    CURRENT_BUFFER.QUERY_LENGTH :=
     CURRENT_BUFFER.QUERY_LENGTH + ITEM'LENGTH;
    return;
  end if;
  if LINE.INDEX + ITEM'LENGTH + LINE.END_PHANTOM'LENGTH >
    LINE.LENGTH + 1
      then
    if LINE.INDENT + LINE.CONTINUATION_INDENT +
      LINE.START_PHANTOM'LENGTH +
        LINE.INDEX - LINE.BREAK + ITEM'LENGTH > LINE.LENGTH then
      raise LAYOUT_ERROR;
    end if;
    if ITEM = " " and then LINE.END_PHANTOM.all = "" then
      return;
    end if;
    PUT_LINE(FILE,LINE.DATA(1..LINE.BREAK-1) & LINE.END_PHANTOM.all);
    for I in 1..LINE.INDENT + LINE.CONTINUATION_INDENT loop
      LINE.DATA(I) := ' ';
    end loop;
    NEW_BREAK := LINE.INDENT + LINE.CONTINUATION_INDENT + 1;
    NEW_INDEX := NEW_BREAK + LINE.START_PHANTOM'LENGTH +
        LINE.INDEX - LINE.BREAK;
    LINE.DATA(NEW_BREAK..NEW_INDEX-1) := LINE.START_PHANTOM.all &
        LINE.DATA(LINE.BREAK..LINE.INDEX-1);
    LINE.BREAK := NEW_BREAK;
    LINE.INDEX := NEW_INDEX;
  end if;
  NEW_INDEX := LINE.INDEX + ITEM'LENGTH;
  LINE.DATA(LINE.INDEX..NEW_INDEX-1) := ITEM;
  LINE.INDEX := NEW_INDEX;
  if BRK = BREAK then
    LINE.BREAK := NEW_INDEX;
  end if;
  LINE.USED_YET := TRUE;
end PRINT;

procedure PRINT(FILE : in FILE_TYPE;
                ITEM : in STRING;
                BRK  : in BREAK_TYPE := BREAK) is
begin
  PRINT(FILE,DEFAULT_LINE,ITEM,BRK);
end PRINT;
```

```
procedure PRINT(LINE : in LINE_TYPE;
                ITEM : in STRING;
                BRK  : in BREAK_TYPE := BREAK) is
begin
  PRINT(CURRENT_OUTPUT,LINE,ITEM,BRK);
end PRINT;

procedure PRINT(ITEM : in STRING; BRK : in BREAK_TYPE := BREAK) is
begin
  PRINT(CURRENT_OUTPUT,DEFAULT_LINE,ITEM,BRK);
end PRINT;

procedure PRINT_LINE(FILE : in FILE_TYPE; LINE : in LINE_TYPE) is
begin
  if CURRENT_BUFFER /= null then
    if CURRENT_BUFFER.QUERY_LENGTH /= 0 then
      PRINT ( " " );
    end if;
    return;
  end if;
  if LINE.INDEX /= LINE.INDENT + 1 then
    PUT_LINE(FILE,LINE.DATA(1..LINE.INDEX-1));
  end if;
  for I in 1..LINE.INDENT loop
    LINE.DATA(I) := ' ';
  end loop;
  LINE.INDEX := LINE.INDENT + 1;
  LINE.BREAK := LINE.INDEX;
end PRINT_LINE;

procedure PRINT_LINE(FILE : in FILE_TYPE) is
begin
  PRINT_LINE(FILE,DEFAULT_LINE);
end PRINT_LINE;

procedure PRINT_LINE(LINE : in LINE_TYPE) is
begin
  PRINT_LINE(CURRENT_OUTPUT,LINE);
end PRINT_LINE;

procedure PRINT_LINE is
begin
  PRINT_LINE(CURRENT_OUTPUT,DEFAULT_LINE);
end PRINT_LINE;

procedure BLANK_LINE(FILE : in FILE_TYPE; LINE : in LINE_TYPE) is
begin
  if CURRENT_BUFFER /= null then
    return;
  end if;
  if LINE.USED_YET then
```

```
    NEW_LINE(FILE);
  end if;
end BLANK_LINE;

procedure BLANK_LINE(FILE : in FILE_TYPE) is
begin
  BLANK_LINE(FILE,DEFAULT_LINE);
end BLANK_LINE;

procedure BLANK_LINE(LINE : in LINE_TYPE) is
begin
  BLANK_LINE(CURRENT_OUTPUT,LINE);
end BLANK_LINE;

procedure BLANK_LINE is
begin
  BLANK_LINE(CURRENT_OUTPUT,DEFAULT_LINE);
end BLANK_LINE;

package body INTEGER_PRINT is

  procedure PRINT(FILE : in FILE_TYPE;
                  LINE : in LINE_TYPE;
                  ITEM : in NUM;
                  BRK  : in BREAK_TYPE := BREAK) is
    S : STRING(1..NUM'WIDTH);
    L : NATURAL;
  begin
    PRINT(S,L,ITEM);
    PRINT(FILE,LINE,S(1..L),BRK);
  end PRINT;

  procedure PRINT(FILE : in FILE_TYPE;
                  ITEM : in NUM;
                  BRK  : in BREAK_TYPE := BREAK) is
  begin
    PRINT(FILE,DEFAULT_LINE,ITEM,BRK);
  end PRINT;

  procedure PRINT(LINE : in LINE_TYPE;
                  ITEM : in NUM;
                  BRK  : in BREAK_TYPE := BREAK) is
  begin
    PRINT(CURRENT_OUTPUT,LINE,ITEM,BRK);
  end PRINT;

  procedure PRINT(ITEM : in NUM;
                  BRK  : in BREAK_TYPE := BREAK) is
  begin
    PRINT(CURRENT_OUTPUT,DEFAULT_LINE,ITEM,BRK);
  end PRINT;
```

```
    procedure PRINT(TO : out STRING; LAST : out NATURAL; ITEM : in NUM) is
      S : constant STRING := NUM'IMAGE(ITEM);
      F : NATURAL := S'FIRST; -- Bug in DG Compiler -- S'FIRST /= 1 ! ! ! ! ! !
      L : NATURAL;
    begin
      if S(F) = ' ' then
        F := F + 1;
      end if;
      if TO'LENGTH < S'LAST - F + 1 then
        raise LAYOUT_ERROR;
      end if;
      L := TO'FIRST + S'LAST - F;
      TO(TO'FIRST..L) := S(F..S'LAST);
      LAST := L;
    end PRINT;

  end INTEGER_PRINT;

  package body FLOAT_PRINT is

    package NUM_IO is new FLOAT_IO(NUM);
      use NUM_IO;

    procedure PRINT(FILE : in FILE_TYPE;
                    LINE : in LINE_TYPE;
                    ITEM : in NUM;
                    BRK  : in BREAK_TYPE := BREAK) is
      S : STRING(1..DEFAULT_FORE + DEFAULT_AFT + DEFAULT_EXP + 2);
      L : NATURAL;
    begin
      PRINT(S,L,ITEM);
      PRINT(FILE,LINE(S(1..L),BRK);
    end PRINT;

    procedure PRINT(FILE : in FILE_TYPE;
                    ITEM : in NUM;
                    BRK  : in BREAK_TYPE := BREAK) is
    begin
      PRINT(FILE,DEFAULT_LINE,ITEM,BRK);
    end PRINT;

    procedure PRINT(LINE : in LINE_TYPE;
                    ITEM : in NUM;
                    BRK  : in BREAK_TYPE := BREAK) is
    begin
      PRINT(CURRENT_OUTPUT,LINE,ITEM,BRK);
    end PRINT;

    procedure PRINT(ITEM : in NUM;
                    BRK  : in BREAK_TYPE := BREAK) is
    begin
```

```
  PRINT(CURRENT_OUTPUT,DEFAULT_LINE,ITEM,BRK);
end PRINT;

procedure PRINT(TO : out STRING; LAST : out NATURAL; ITEM : in NUM) is
  S         : STRING(1..DEFAULT_FORE + DEFAULT_AFT + DEFAULT_EXP + 2);
  EXP       : INTEGER;
  E_INDEX   : NATURAL := S'LAST - DEFAULT_EXP;
  DOT_INDEX : NATURAL := DEFAULT_FORE + 1;
  L         : NATURAL;
begin
  PUT(S,ITEM);
  EXP := INTEGER'VALUE(S(E_INDEX+1..S'LAST));
  if EXP >= 0 then
    if EXP <= DEFAULT_AFT-1 then
      S(DOT_INDEX..DOT_INDEX+EXP-1) := S(DOT_INDEX+1..DOT_INDEX+EXP);
      S(DOT_INDEX+EXP) := '.';
      for I in E_INDEX..S'LAST loop
        S(I) := ' ';
      end loop;
    end if;
  else -- EXP < 0
    if EXP >= - ( DEFAULT_EXP + 1 ) then
      S(DEFAULT_EXP+2..S'LAST) := S(1..S'LAST-DEFAULT_EXP-1);
      for I in 1..DEFAULT_EXP+1 loop
        S(I) := ' ';
      end loop;
      E_INDEX := S'LAST + 1;
      DOT_INDEX := DOT_INDEX + DEFAULT_EXP + 1;
      L := DOT_INDEX+EXP;
      for I in reverse L+1..DOT_INDEX loop
        case S(I-1) is
          when ' '    => S(I) := '0';
          when '-'    => S(I-2) := '-'; S(I) := '0';
          when others => S(I) := S(I-1);
        end case;
      end loop;
      S(L) := '.';
      case S(L-1) is
        when ' '    => S(L-1) := '0';
        when '-'    => S(L-2) := '-'; S(L-1) := '0';
        when others => null;
      end case;
    end if;
  end if;
  for I in reverse 1..E_INDEX-1 loop
    exit when S(I) /= '0' or else S(I-1) = '.';
    S(I) := ' ';
  end loop;
  L := TO'FIRST - 1;
  for I in S'RANGE loop
    if S(I) /= ' ' then
```

```
            L := L + 1;
            TO(L) := S(I);
          end if;
        end loop;
        LAST := L;
      exception
        when CONSTRAINT_ERROR =>
          raise LAYOUT_ERROR;
      end PRINT;

    end FLOAT_PRINT;

end TEXT_PRINT;
```

3.10.4 package ORDEFS.ADA

```
package RDBMS_DEFINITIONS is

  --
  -- Declare variable for reporting RDBMS error number
  --

  RDBMS_ERROR_NUM          : INTEGER;

  --
  -- Declare exceptions.
  --

  ACCESS_ERROR             : exception;
  BAD_RDBMS_NAME_ERROR     : exception;
  COLUMN_ERROR             : exception;
  COLUMN_RANGE_ERROR       : exception;
  CREATE_ERROR             : exception;
  DB_ALREADY_OPEN_ERROR    : exception;
  DBCLOSE_ERROR            : exception;
  DBOPEN_ERROR             : exception;
  DUPLICATE_KEY_ERROR      : exception;
  NO_DBOPEN_ERROR          : exception;
  NO_UPDATE_ERROR          : exception;
  NO_RETRIEVE_ERROR        : exception;
  NO_ROW_ERROR             : exception;
  NO_TABLE_ERROR           : exception;
  SYNTAX_ERROR             : exception;
  TRUNCATE_ERROR           : exception;
  TYPE_CONVERSION_ERROR    : exception;
  UNHANDLED_RDBMS_ERROR    : exception;
  UNIMPLEMENTED_ERROR      : exception;


end RDBMS_DEFINITIONS;
```

3.10.5 package ORCUDEFS.ADA

```
with TEXT_PRINT;

package ORACLE_CURSOR_DEFINITIONS is

  BUFFER_LENGTH : constant INTEGER := 4000;
  subtype BUFFER_TYPE is STRING (1 .. BUFFER_LENGTH);
  subtype BUFFER_ACCESS_TYPE is TEXT_PRINT.BUFFER_ACCESS_TYPE
                                (1..BUFFER_LENGTH);

  RETRIEVAL_BUFFER_LENGTH : constant INTEGER := 4000;
  subtype RETRIEVAL_BUFFER is STRING (1 .. RETRIEVAL_BUFFER_LENGTH);
                           --:= (others => ' ');
  subtype RETRIEVAL_LENGTH  is INTEGER;

  type FILLER_ARRAY_1 is array (1 .. 5)  of SHORT_INTEGER;
  type FILLER_ARRAY_2 is array (1 .. 25) of SHORT_INTEGER;

--  This is the declaration of the data area used by Oracle's HLI.
--  The integer used should be a 16-bit integer.
--  The telesoft compiler has integer defined as a 16-bit integer.
--  The verdix compiler has Short_Integer defined as a 16-bit integer.
--  The DEC compiler has Short_Integer defined as a 16-bit integer.

  type DATA_AREA_TYPE is
    record
      RETURN_CODE   : SHORT_INTEGER := 0;
      FILLER_DATA_1 : FILLER_ARRAY_1 := (0, 0, 0, 0, 0);
      V4_ERROR_CODE : SHORT_INTEGER := 0;
      FILLER_DATA_2 : FILLER_ARRAY_2 := (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
    end record;

  type CURSOR_DATA_TYPE is
    record
      ROWID       : STRING (1..18) := (1..18 => ' ');
      CURSOR_AREA : DATA_AREA_TYPE;
    end record;

  subtype QUERY_BUFFER_RECORD is TEXT_PRINT.QUERY_BUFFER_RECORD ;
--  type QUERY_BUFFER_RECORD is
--    record
--      QUERY_LENGTH : INTEGER := 0;
--      QUERY_BUFFER : BUFFER_ACCESS_TYPE;
--    end record;
  subtype ACCESS_QUERY_BUFFER is TEXT_PRINT.ACCESS_QUERY_BUFFER;
  type ACCESS_CURSOR_DATA_TYPE is access CURSOR_DATA_TYPE;
  type CURSOR_ORACLE is
    record
      CURSOR_DATA          : ACCESS_CURSOR_DATA_TYPE;
```

```
        CURSOR_QUERY_DATA     : ACCESS_QUERY_BUFFER;
        CURSOR_COLUMN_NUMBER  : INTEGER := 0;
        CURSOR_MAX_COLUMN     : INTEGER := 0;
        CURSOR_ROW_NUMBER     : INTEGER := 0;
        CURSOR_FIRST_FETCH    : BOOLEAN := TRUE;
        CURSOR_RETRIEVAL_LEN  : RETRIEVAL_LENGTH := 0;
        CURSOR_RETRIEVAL_BUF  : RETRIEVAL_BUFFER := (others => ' ');

    end record;

end ORACLE_CURSOR_DEFINITIONS;
```

3.10.6 package ORHLIS.ADA

```
-- This package ORACLE_HLI contains the Oracle HLI procedures set up as
-- procedures for use with the pragma interface so that they can communicate
-- to Oracle.

with RDBMS_DEFINITIONS, SYSTEM, ORACLE_CURSOR_DEFINITIONS;
use RDBMS_DEFINITIONS, ORACLE_CURSOR_DEFINITIONS;

package ORACLE_HLI is

-- Declaration of the Oracle HLI subprograms to be called.
-- These are set up for use with the pragma interface and
-- have the following format.
--
--  procedure identifier (identifier_list : mode type_mark);

-- commit current transaction to database
  procedure OCOM (LDA : in out DATA_AREA_TYPE);

-- roll back the current transaction
  procedure OROL (LDA : in out DATA_AREA_TYPE);

-- establish communication between oracle and the user program
  procedure OLON (LDA        : in out DATA_AREA_TYPE;
--                  UID        : in BUFFER_ACCESS_TYPE;
                  UID        : in STRING;
                  UIDLEN     : in SHORT_INTEGER;
--                  PSW        : in SHORT_INTEGER := -1;
                  PSW        : in STRING;
                  PSWL       : in SHORT_INTEGER := -1;
                  AUDIT_FLAG : in SHORT_INTEGER := 0);

-- establishes a cursor to pass a SQL statement to Oracle
  procedure OOPEN (CURSOR   : in out DATA_AREA_TYPE;
                   LDA      : in out DATA_AREA_TYPE;
                   DBN      : in SHORT_INTEGER := -1;
                   DBNLEN   : in SHORT_INTEGER := -1;
```

```
                    AREASIZE : in SHORT_INTEGER :=  -1;
                    UID      : in SHORT_INTEGER :=  -1;
                    UIDLEN   : in SHORT_INTEGER :=  -1);

-- associated a SQL statement with a cursor and passes the SQL statement to
-- oracle
  procedure OSQL3 (CURSOR       : in out DATA_AREA_TYPE;
--                   SQLSTATEMENT : in BUFFER_ACCESS_TYPE;
                   SQLSTATEMENT : in STRING;
                   SQLLEN       : in SHORT_INTEGER);

-- causes the SQL statement currently associated with a cursor to be processed
  procedure OEXEC (CURSOR : in out DATA_AREA_TYPE);

--defines an output buffer for each field in a select-list in the SQL query
  procedure ODEFIN (CURSOR : in out DATA_AREA_TYPE;
                    POS    : in SHORT_INTEGER;
                    BUFFER : in SYSTEM.ADDRESS;
                    BUFL   : in SHORT_INTEGER;
                    FTYPE  : in SHORT_INTEGER;
                    SCALE  : in SHORT_INTEGER :=  -1;
                    INDP   : in SHORT_INTEGER :=  -1;
                    FMT    : in SHORT_INTEGER :=  -1;
                    FMTL   : in SHORT_INTEGER :=  -1;
                    FMTT   : in SHORT_INTEGER :=  -1;
                    RETL   : in SHORT_INTEGER :=  -1;
                    RCODE  : in SHORT_INTEGER :=  -1);

--returns internal datatype and size information for a field or expression
-- in the select_list of a query
  procedure ODSC (CURSOR   : in out DATA_AREA_TYPE;
                  POSITION : in SHORT_INTEGER;
                  DBSIZE   : in SHORT_INTEGER :=  -1;
                  FSIZE    : in SHORT_INTEGER :=  -1;
                  RCODE    : in SHORT_INTEGER :=  -1;
                  DBTYPE   : in SHORT_INTEGER :=  -1;
                  CBUF     : in SHORT_INTEGER :=  -1;
                  CBUFL    : in SHORT_INTEGER :=  -1;
                  DSIZE    : out SHORT_INTEGER);

-- retrieve the names of the columns in a select_list of a SQL query
  procedure ONAME (CURSOR   : in out DATA_AREA_TYPE;
                   POSITION : in SHORT_INTEGER;
                   TBUF     : in SHORT_INTEGER;
                   TBUF1    : in SHORT_INTEGER;
                   CBUF     : in SYSTEM.ADDRESS;
                   CBUF1    : in SYSTEM.ADDRESS);

-- returns one row of a query result
  procedure OFETCH (CURSOR : in out DATA_AREA_TYPE);
```

```
-- disconnects a cursor from oracle
  procedure OCLOSE (CURSOR : in out DATA_AREA_TYPE);

--disconnects a program from oracle
  procedure OLOGOF (LDA : in out DATA_AREA_TYPE);


private
--  pragma interface (language_name,identifier);

  pragma INTERFACE (C, OCOM);
  pragma IMPORT_PROCEDURE (OCOM, OCOM, (DATA_AREA_TYPE),
                           MECHANISM => (REFERENCE));

  pragma INTERFACE (C, OROL);
  pragma IMPORT_PROCEDURE (OROL, OROL, (DATA_AREA_TYPE),
                           MECHANISM => (REFERENCE));

  pragma INTERFACE (C, OLON);
--  pragma IMPORT_PROCEDURE (OLON, OLON, (DATA_AREA_TYPE,
      BUFFER_ACCESS_TYPE,
  pragma IMPORT_PROCEDURE (OLON, OLON, (DATA_AREA_TYPE, STRING,
--                           SHORT_INTEGER, SHORT_INTEGER, SHORT_INTEGER,
                           SHORT_INTEGER, STRING, SHORT_INTEGER,
--                           SHORT_INTEGER), MECHANISM => (REFERENCE, VALUE,
                           SHORT_INTEGER), MECHANISM => (REFERENCE, REFERENCE,
                           VALUE, REFERENCE, VALUE, VALUE));
--                           VALUE, VALUE, VALUE, VALUE));

  pragma INTERFACE (C, OOPEN);
  pragma IMPORT_PROCEDURE (OOPEN, OOPEN, (DATA_AREA_TYPE,
                           DATA_AREA_TYPE,
                           SHORT_INTEGER, SHORT_INTEGER, SHORT_INTEGER,
                           SHORT_INTEGER, SHORT_INTEGER), MECHANISM =>
                           (REFERENCE, REFERENCE, VALUE, VALUE, VALUE, VALUE,
                           VALUE));

  pragma INTERFACE (C, OSQL3);
--  pragma IMPORT_PROCEDURE (OSQL3, OSQL3, (DATA_AREA_TYPE,
      BUFFER_ACCESS_TYPE,
  pragma IMPORT_PROCEDURE (OSQL3, OSQL3, (DATA_AREA_TYPE, STRING,
                           SHORT_INTEGER), MECHANISM =>
--                           (REFERENCE, VALUE, VALUE));
                           (REFERENCE, REFERENCE, VALUE));

  pragma INTERFACE (C, ODEFIN);
  pragma IMPORT_PROCEDURE (ODEFIN, ODEFIN, (DATA_AREA_TYPE,
                           SHORT_INTEGER,
                           SYSTEM.ADDRESS, SHORT_INTEGER, SHORT_INTEGER,
                           SHORT_INTEGER, SHORT_INTEGER, SHORT_INTEGER,
                           SHORT_INTEGER, SHORT_INTEGER, SHORT_INTEGER,
```

```
                              SHORT_INTEGER), MECHANISM => (REFERENCE, VALUE,
                              VALUE, VALUE, VALUE, VALUE, VALUE, VALUE,
                              VALUE, VALUE, VALUE, VALUE));

  pragma INTERFACE (C, ODSC);
  pragma IMPORT_PROCEDURE (ODSC, ODSC, (DATA_AREA_TYPE, SHORT_INTEGER,
                              SHORT_INTEGER, SHORT_INTEGER, SHORT_INTEGER,
                              SHORT_INTEGER, SHORT_INTEGER, SHORT_INTEGER,
                              SHORT_INTEGER), MECHANISM => (REFERENCE, VALUE,
                              VALUE, VALUE, VALUE, VALUE, VALUE, VALUE,
                              REFERENCE));

  pragma INTERFACE (C, ONAME);
  pragma IMPORT_PROCEDURE (ONAME, ONAME, (DATA_AREA_TYPE, SHORT_INTEGER,
                              SHORT_INTEGER, SHORT_INTEGER, SYSTEM.ADDRESS,
                              SYSTEM.ADDRESS), MECHANISM => (REFERENCE, VALUE,
                              VALUE, VALUE, VALUE, VALUE));

  pragma INTERFACE (C, OEXEC);
  pragma IMPORT_PROCEDURE (OEXEC, OEXEC, (DATA_AREA_TYPE),
    MECHANISM => REFERENCE);

  pragma INTERFACE (C, OFETCH);
  pragma IMPORT_PROCEDURE (OFETCH, OFETCH, (DATA_AREA_TYPE),
    MECHANISM => REFERENCE);

  pragma INTERFACE (C, OCLOSE);
  pragma IMPORT_PROCEDURE (OCLOSE, OCLOSE, (DATA_AREA_TYPE),
    MECHANISM => REFERENCE);

  pragma INTERFACE (C, OLOGOF);
  pragma IMPORT_PROCEDURE (OLOGOF, OLOGOF, (DATA_AREA_TYPE),
    MECHANISM => REFERENCE);

end ORACLE_HLI;
```

3.10.7 package ORINTS.ADA

```
-- package RDBMS_INTERFACE contains the procedures utilized by the user query
-- generation function to communicate to the oracle RDBMS.

with RDBMS_DEFINITIONS, ORACLE_HLI, ORACLE_CURSOR_DEFINITIONS;
use  RDBMS_DEFINITIONS, ORACLE_HLI, ORACLE_CURSOR_DEFINITIONS;

package RDBMS_INTERFACE is

  subtype CURSOR_TYPE is CURSOR_DATA_TYPE;

  CURSOR_DATA_AREA : CURSOR_DATA_TYPE;
```

```
  type COLUMN_TYPE is
    record
      COLUMN_NAME        : STRING (1..18) := (1..18 => ' ');
      COLUMN_NAME_LENGTH : INTEGER := 0;
    end record;

  type COLUMN_NAMES_TYPE is ARRAY (1..127) of COLUMN_TYPE;

-- reap the user's data from ARI's retrieval buffer.
  procedure RDBMS_COLUMN_REAPER
            (COLUMN_NUMBER : in INTEGER;
             VAR           : out STRING;
             VAR_LEN       : out INTEGER;
             RETRIEVAL_LEN : in RETRIEVAL_LENGTH;
             RETRIEVAL_BUF : in RETRIEVAL_BUFFER);

-- reap the user's data from ARI's retrieval buffer.
  procedure RDBMS_COLUMN_REAPER
            (COLUMN_NUMBER : in INTEGER;
             VAR           : out CHARACTER;
             RETRIEVAL_LEN : in RETRIEVAL_LENGTH;
             RETRIEVAL_BUF : in RETRIEVAL_BUFFER);

-- reap the user's data from ARI's retrieval buffer.
  procedure RDBMS_COLUMN_REAPER
            (COLUMN_NUMBER : in INTEGER;
             VAR           : out INTEGER;
             RETRIEVAL_LEN : in RETRIEVAL_LENGTH;
             RETRIEVAL_BUF : in RETRIEVAL_BUFFER);

-- reap the user's data from ARI's retrieval buffer.
  procedure RDBMS_COLUMN_REAPER
            (COLUMN_NUMBER : in INTEGER;
             VAR           : out FLOAT;
             RETRIEVAL_LEN : in RETRIEVAL_LENGTH;
             RETRIEVAL_BUF : in RETRIEVAL_BUFFER);

-- executes a query on a RDBMS.
  procedure RDBMS_EXECUTE
            (CURSOR : in out DATA_AREA_TYPE);

-- logs the user off an RDBMS.
  procedure RDBMS_EXIT_DATABASE;

-- close a cursor
  procedure RDBMS_CLOSE_CURSOR
            (CURSOR : in out DATA_AREA_TYPE);

-- fetches data from an RDBMS after a retrieve query.
  procedure RDBMS_FETCH
            (CURSOR : in out CURSOR_ORACLE);
```

```
-- logs the user into an RDBMS.
  --procedure RDBMS_OPEN_DATABASE
    --       (QUERY_BUFFER : in BUFFER_ACCESS_TYPE;
      --       QUERY_LENGTH : in INTEGER);
  procedure RDBMS_OPEN_DATABASE
            (QUERY_BUFFER : in BUFFER_ACCESS_TYPE;
             QUERY_LENGTH : in INTEGER;
             QUERYX_BUFFER : in BUFFER_ACCESS_TYPE;
             QUERYX_LENGTH : in INTEGER);

-- opens a cursor for the database
  procedure RDBMS_OPEN_CURSOR
            (CURSOR : in out DATA_AREA_TYPE);

-- commit work for transaction.
  procedure RDBMS_COMMIT_WORK;

-- rollback work for transaction.
  procedure RDBMS_ROLLBACK_WORK;

-- retrieve column names used in SELECT
  procedure RDBMS_ONAME
            (CURSOR          : in out DATA_AREA_TYPE;
             NAMES           : in out COLUMN_NAMES_TYPE;
             NUMBER_OF_NAMES : out INTEGER);

-- sends a query to an RDBMS.
  procedure RDBMS_QUERY
            (QUERY_BUFFER : in out BUFFER_ACCESS_TYPE;
             QUERY_LENGTH : in INTEGER;
             CURSOR       : in out DATA_AREA_TYPE);

  ACCESS_ERROR          : exception renames RDBMS_DEFINITIONS.ACCESS_ERROR;
  BAD_RDBMS_NAME_ERROR  : exception renames
                          RDBMS_DEFINITIONS.BAD_RDBMS_NAME_ERROR;
  COLUMN_ERROR          : exception renames RDBMS_DEFINITIONS.COLUMN_ERROR;
  COLUMN_RANGE_ERROR    : exception renames
                          RDBMS_DEFINITIONS.COLUMN_RANGE_ERROR;
  DB_ALREADY_OPEN_ERROR : exception renames
                          RDBMS_DEFINITIONS.DB_ALREADY_OPEN_ERROR;
  DBCLOSE_ERROR         : exception renames RDBMS_DEFINITIONS.DBCLOSE_ERROR;
  DBOPEN_ERROR          : exception renames RDBMS_DEFINITIONS.DBOPEN_ERROR;
  DUPLICATE_KEY_ERROR   : exception renames
                          RDBMS_DEFINITIONS.DUPLICATE_KEY_ERROR;
  NO_DBOPEN_ERROR       : exception renames RDBMS_DEFINITIONS.NO_DBOPEN_ERROR;
  NO_RETRIEVE_ERROR     : exception renames
                          RDBMS_DEFINITIONS.NO_RETRIEVE_ERROR;
  NO_ROW_ERROR          : exception renames RDBMS_DEFINITIONS.NO_ROW_ERROR;
  NO_TABLE_ERROR        : exception renames RDBMS_DEFINITIONS.NO_TABLE_ERROR;
  SYNTAX_ERROR          : exception renames RDBMS_DEFINITIONS.SYNTAX_ERROR;
  TRUNCATE_ERROR        : exception renames RDBMS_DEFINITIONS.TRUNCATE_ERROR;
```

```
  TYPE_CONVERSION_ERROR : exception renames
                          RDBMS_DEFINITIONS.TYPE_CONVERSION_ERROR;
  UNHANDLED_RDBMS_ERROR : exception renames
                          RDBMS_DEFINITIONS.UNHANDLED_RDBMS_ERROR;
  UNIMPLEMENTED_ERROR   : exception renames
                          RDBMS_DEFINITIONS.UNIMPLEMENTED_ERROR;
  NO_UPDATE_ERROR       : exception renames
     RDBMS_DEFINITIONS.NO_UPDATE_ERROR;

end RDBMS_INTERFACE;
```

3.10.8 package FUNCTIONS.ADA

```
with ORACLE_CURSOR_DEFINITIONS, DATABASE, RDBMS_DEFINITIONS;
use  ORACLE_CURSOR_DEFINITIONS;

package ADA_SQL_FUNCTIONS is

  NO_UPDATE_ERROR : exception renames RDBMS_DEFINITIONS.NO_UPDATE_ERROR;
  NOT_FOUND_ERROR : exception renames RDBMS_DEFINITIONS.NO_ROW_ERROR;
  INTERNAL_ERROR : exception;
  UNIQUE_ERROR : exception;
  WANNA_DEBUG : BOOLEAN := FALSE;

  type SQL_OPERATION is
  ( O_AVG               , O_MAX          , O_MIN        , O_SUM               ,
    O_UNARY_PLUS        , O_UNARY_MINUS  , O_PLUS       , O_MINUS             ,
    O_TIMES             , O_DIVIDE       , O_EQ         , O_NE                ,
    O_LT                , O_GT           , O_LE         , O_GE                ,
    O_BETWEEN           , O_AND          , O_IS_IN      , O_OR                ,
    O_NOT               , O_LIKE         , O_AMPERSAND  , O_SELEC             ,
    O_SELECT_DISTINCT   , O_ASC          , O_DESC       , O_TABLE_COLUMN_LIST ,
    O_COUNT_STAR        , O_NULL_OP      , O_STAR       , O_NOT_IN            ,
    O_VALUES            , O_DECLAR );

  type SQL_OBJECT       is private;
  type TYPED_SQL_OBJECT is private;
  type TABLE_NAME       is private;
  type TABLE_LIST       is private;
  type INSERT_ITEM      is private;
  type CURSOR_NAME      is private;
  type DATABASE_NAME    is private;

  NULL_SQL_OBJECT : constant SQL_OBJECT;

  procedure INITIATE_TEST; -- ***** ONLY FOR TESTING

-- constant literal value generator

  generic
```

```
    type RESULT_TYPE is private;
    VALUE : in RESULT_TYPE;
  function CONSTANT_LITERAL return RESULT_TYPE;

-- conversion routines for SQL objects

  function L_CONVERT ( L : TYPED_SQL_OBJECT ) return SQL_OBJECT;

  function R_CONVERT ( R : TYPED_SQL_OBJECT ) return SQL_OBJECT
   renames L_CONVERT;

  function CONVERT_R ( R : SQL_OBJECT ) return TYPED_SQL_OBJECT;

  package CONVERT is

    function L_CONVERT ( L : SQL_OBJECT ) return SQL_OBJECT;

    function R_CONVERT ( R : SQL_OBJECT ) return SQL_OBJECT renames L_CONVERT;

    function CONVERT_R ( R : SQL_OBJECT ) return SQL_OBJECT renames L_CONVERT;

    function L_CONVERT ( L : TABLE_NAME ) return SQL_OBJECT;

    function R_CONVERT ( R : TABLE_NAME ) return SQL_OBJECT renames L_CONVERT;

    function CONVERT_R ( R : SQL_OBJECT ) return TABLE_NAME;

    function L_CONVERT ( L : TABLE_LIST ) return SQL_OBJECT;

    function CONVERT_R ( R : SQL_OBJECT ) return TABLE_LIST;

    function L_CONVERT ( L : INSERT_ITEM ) return SQL_OBJECT;

    function CONVERT_R ( R : SQL_OBJECT ) return INSERT_ITEM;

  end CONVERT;

-- conversion routines for user types

  -- ***** instantiate these as L_CONVERT, then rename as R_CONVERT

  generic
    type USER_TYPE is (<>);
  function INTEGER_AND_ENUMERATION_CONVERT ( VAR : USER_TYPE )
   return SQL_OBJECT;

  generic
    type USER_TYPE is digits <>;
  function FLOAT_CONVERT ( VAR : USER_TYPE ) return SQL_OBJECT;

  generic
```

```
    type INDEX_TYPE is range <>;
    type USER_TYPE is array ( INDEX_TYPE range <> ) of CHARACTER;
  function UNCONSTRAINED_CHARACTER_STRING_CONVERT ( VAR : USER_TYPE )
   return SQL_OBJECT;

  generic
    type INDEX_TYPE is range <>;
    type USER_TYPE is array ( INDEX_TYPE ) of CHARACTER;
  function CONSTRAINED_CHARACTER_STRING_CONVERT ( VAR : USER_TYPE )
   return SQL_OBJECT;

  generic
    type INDEX_TYPE is range <>;
    type COMPONENT_TYPE is (<>);
    type USER_TYPE is array ( INDEX_TYPE range <> ) of COMPONENT_TYPE;
    with function CONVERT_COMPONENT_TO_CHARACTER ( C : COMPONENT_TYPE )
     return CHARACTER is <>;
  function UNCONSTRAINED_STRING_CONVERT ( VAR : USER_TYPE )
   return SQL_OBJECT;

  -- ***** must generate CONVERT_COMPONENT_TO_CHARACTER

  generic
    type INDEX_TYPE is range <>;
    type COMPONENT_TYPE is (<>);
    type USER_TYPE is array ( INDEX_TYPE ) of COMPONENT_TYPE;
    with function CONVERT_COMPONENT_TO_CHARACTER ( C : COMPONENT_TYPE )
     return CHARACTER is <>;
  function CONSTRAINED_STRING_CONVERT ( VAR : USER_TYPE )
   return SQL_OBJECT;

-- column and table name routines

  generic
    GIVEN_NAME : in STANDARD.STRING;
  package NAME_PACKAGE is

    generic
      type SQL_OBJECT_TYPE is private;
      with function CONVERT_R ( R : SQL_OBJECT ) return SQL_OBJECT_TYPE is <>;
    function COLUMN_OR_TABLE_NAME return SQL_OBJECT_TYPE;

    generic
    function TABLE_NAME_WITH_COLUMN_LIST ( COLUMNS : SQL_OBJECT )
     return TABLE_NAME;

  end NAME_PACKAGE;

  -- ***** must generate routines for table.column (define record structure)

  -- ***** must generate package for correlation.column and correlation.table
```

```
-- value specification routines

  generic
    type USER_TYPE is private;
    type RESULT_TYPE is private;
    with function L_CONVERT ( L : USER_TYPE ) return SQL_OBJECT is <>;
    with function CONVERT_R ( R : SQL_OBJECT ) return RESULT_TYPE is <>;
  function INDICATOR_FUNCTION ( VAL : USER_TYPE ) return RESULT_TYPE;

-- generic operation routines

  generic
    GIVEN_OPERATION : in SQL_OPERATION;
    type L_TYPE is private;
    type TYPE_R is private;
    with function L_CONVERT ( L : L_TYPE ) return SQL_OBJECT is <>;
    with function CONVERT_R ( R : SQL_OBJECT ) return TYPE_R is <>;
  function UNARY_OPERATION ( L : L_TYPE ) return TYPE_R;

  generic
    GIVEN_OPERATION : in SQL_OPERATION;
    type L_TYPE is private;
    type R_TYPE is private;
    type TYPE_R is private;
    with function L_CONVERT ( L : L_TYPE ) return SQL_OBJECT is <>;
    with function R_CONVERT ( R : R_TYPE ) return SQL_OBJECT is <>;
    with function CONVERT_R ( R : SQL_OBJECT ) return TYPE_R is <>;
  function BINARY_OPERATION ( L : L_TYPE ; R : R_TYPE ) return TYPE_R;

-- set function routines

  -- ***** must also generate STAR_TYPE is '*'; function COUNT ( STAR_TYPE )
  -- ***** instantiate COUNT_STAR for DATABASE.INT or untyped

  generic
    type TYPE_R is private;
    with function CONVERT_R ( R : SQL_OBJECT ) return TYPE_R is <>;
  function COUNT_STAR return TYPE_R;

  -- instantiate UNARY_OPERATION for O_AVG, O_MAX, O_MIN, O_SUM

-- value expression routines

  -- instantiate UNARY_OPERATION for O_UNARY_PLUS, O_UNARY_MINUS

  -- instantiate BINARY_OPERATION for O_PLUS, O_MINUS, O_TIMES, O_DIVIDE

  -- ***** generate CONVERT_TO package for type conversions, calling CONVERT_R
  --  to set correct result type

-- comparison predicate routines
```

```
  -- instantiate BINARY_OPERATION for O_EQ, O_NE, O_LT, O_GT, O_LE, O_GE
-- between predicate routines
  -- instantiate BINARY_OPERATION for O_BETWEEN
  -- instantiate BINARY_OPERATION for O_AND
-- in predicate routines
  -- instantiate BINARY_OPERATION for O_IS_IN
    -- special case if <in value list> has one element
  -- instantiate BINARY_OPERATION for O_OR
    -- different instantiations for first and following ORs
  -- instantiate UNARY_OPERATION for O_NOT
-- like predicate routines
  -- instantiate BINARY_OPERATION for O_LIKE
  -- instantiate UNARY_OPERATION for O_NOT
-- search condition routines
  -- instantiate BINARY_OPERATION for O_AND, O_OR
  -- instantiate UNARY_OPERATION for O_NOT
-- from clause routines
  -- instantiate BINARY_OPERATION for O_AMPERSAND
-- group by clause routines
  -- instantiate BINARY_OPERATION for O_AMPERSAND
-- subquery routines
  generic
    SELECT_TYPE : in SQL_OPERATION;
    type WHAT_TYPE is private;
    type TYPE_R is private;
    with function L_CONVERT ( L : WHAT_TYPE ) return SQL_OBJECT is <>;
    with function CONVERT_R ( R : SQL_OBJECT ) return TYPE_R is <>;
  function SELECT_LIST_SUBQUERY
           ( WHAT      : WHAT_TYPE;
             FROM      : TABLE_LIST;
```

```
            WHERE    : SQL_OBJECT := NULL_SQL_OBJECT;
            GROUP_BY : SQL_OBJECT := NULL_SQL_OBJECT;
            HAVING   : SQL_OBJECT := NULL_SQL_OBJECT ) return TYPE_R;

  generic
    SELECT_TYPE : in SQL_OPERATION;
    type TYPE_R is private;
    with function CONVERT_R ( R : SQL_OBJECT ) return TYPE_R is <>;
  function STAR_SUBQUERY
          ( FROM     : TABLE_LIST;
            WHERE    : SQL_OBJECT := NULL_SQL_OBJECT;
            GROUP_BY : SQL_OBJECT := NULL_SQL_OBJECT;
            HAVING   : SQL_OBJECT := NULL_SQL_OBJECT ) return TYPE_R;

-- query specification routines

  -- instantiate appropriate subquery routines

  -- also instantiate BINARY_OPERATION for O_AMPERSAND

-- close routine

  procedure CLOSE ( CURSOR : in out CURSOR_NAME );

-- declare cursor routines

  procedure DECLAR
            ( CURSOR     : in out CURSOR_NAME;
              CURSOR_FOR : in     SQL_OBJECT;
              ORDER_BY   : in     SQL_OBJECT := NULL_SQL_OBJECT );

  procedure DECLAR
            ( CURSOR     : in out CURSOR_NAME;
              CURSOR_FOR : in     SQL_OBJECT;
              ORDER_BY   : in     DATABASE.COLUMN_NUMBER );

  -- instantiate BINARY_OPERATION for O_AMPERSAND

  -- instantiate UNARY_OPERATION for O_ASC and O_DESC

-- delete routines

  procedure DELETE_FROM
            ( TABLE : in TABLE_NAME;
              WHERE : in SQL_OBJECT := NULL_SQL_OBJECT );

-- fetch and into routines

  procedure FETCH ( CURSOR : in out CURSOR_NAME );
```

```
  generic
    type USER_TYPE is (<>);
  procedure INTEGER_AND_ENUMERATION_INTO
            (VAR : out USER_TYPE);

  generic
    type USER_TYPE is digits <>;
  procedure FLOAT_INTO
            (VAR : out USER_TYPE);

  generic
    type INDEX_TYPE is range <>;
    type COMPONENT_TYPE is (<>);
    type USER_TYPE is array ( INDEX_TYPE range <> ) of COMPONENT_TYPE;
    with function CONVERT_CHARACTER_TO_COMPONENT
                  (C : CHARACTER)
                   return COMPONENT_TYPE is <>;
  procedure UNCONSTRAINED_STRING_INTO
            (VAR  : out USER_TYPE;
             LAST : out INDEX_TYPE);

  generic
    type INDEX_TYPE is range <>;
    type COMPONENT_TYPE is (<>);
    type USER_TYPE is array ( INDEX_TYPE ) of COMPONENT_TYPE;
    with function CONVERT_CHARACTER_TO_COMPONENT
                  (C : CHARACTER)
                   return COMPONENT_TYPE is <>;
  procedure CONSTRAINED_STRING_INTO
            (VAR  : out USER_TYPE;
             LAST : out INDEX_TYPE);


-- insert into routines

  procedure INSERT_INTO
            ( TABLE : in TABLE_NAME;
              WHAT  : in INSERT_ITEM );

  -- instantiate BINARY_OPERATION for O_AMPERSAND

  -- see table name routines for table ( column list )

  function VALUES return INSERT_ITEM;

  -- instantiate BINARY_OPERATION for O_LE and O_AND

-- open routine

  procedure OPEN ( CURSOR : in out CURSOR_NAME );
```

```
-- select statement routines

  -- see above for fetch and into routines

  generic
    SELECT_TYPE : in SQL_OPERATION;
    type WHAT_TYPE is private;
    with function L_CONVERT ( L : WHAT_TYPE ) return SQL_OBJECT is <>;
  procedure SELECT_LIST_SELECT
            ( WHAT     : in WHAT_TYPE;
              FROM     : in TABLE_LIST;
              WHERE    : in SQL_OBJECT := NULL_SQL_OBJECT;
              GROUP_BY : in SQL_OBJECT := NULL_SQL_OBJECT;
              HAVING   : in SQL_OBJECT := NULL_SQL_OBJECT );

  generic
    SELECT_TYPE : in SQL_OPERATION;
  procedure STAR_SELECT
            ( FROM     : in TABLE_LIST;
              WHERE    : in SQL_OBJECT := NULL_SQL_OBJECT;
              GROUP_BY : in SQL_OBJECT := NULL_SQL_OBJECT;
              HAVING   : in SQL_OBJECT := NULL_SQL_OBJECT );

-- update routines

  procedure UPDATE
            ( TABLE : in TABLE_NAME;
              SET   : in SQL_OBJECT;
              WHERE : in SQL_OBJECT := NULL_SQL_OBJECT );

  procedure OPEN_DATABASE
           (DATABASE_NAME : in STRING;
            PASSWORD      : in STRING);

  procedure EXIT_DATABASE;

  -- instantiate BINARY_OPERATION for O_AND

  -- instantiate BINARY_OPERATION for O_LE

private

  type DATABASE_NAME is access STANDARD.STRING;
  type ACCESS_STRING is access STANDARD.STRING;

  type SQL_VALUE_KIND is ( INTEGER , FLOAT , STRING );

  type SQL_VALUE ( KIND : SQL_VALUE_KIND := INTEGER ) is
    record
      case KIND is
        when INTEGER =>
```

```
            INTEGER : DATABASE.INT;
          when FLOAT =>
            FLOAT : DATABASE.DOUBLE_PRECISION;
          when STRING =>
            STRING : ACCESS_STRING;
        end case;
      end record;

    type SQL_OBJECT_KIND is ( NAME , VALUE , OPERATION );

    type SQL_OBJECT_RECORD ( KIND : SQL_OBJECT_KIND );
    type TYPED_SQL_OBJECT is access SQL_OBJECT_RECORD;
    type SQL_OBJECT  is new TYPED_SQL_OBJECT;
    type TABLE_NAME  is new TYPED_SQL_OBJECT;
    type TABLE_LIST  is new TYPED_SQL_OBJECT;
    type INSERT_ITEM is new TYPED_SQL_OBJECT;

    type SQL_OBJECT_RECORD ( KIND : SQL_OBJECT_KIND ) is
      record
        ACROSS : SQL_OBJECT;
        case KIND is
          when NAME =>
            NAME : DATABASE_NAME;
          when VALUE =>
            VALUE : SQL_VALUE;
          when OPERATION =>
            OPERATION : SQL_OPERATION;
            OPERANDS  : SQL_OBJECT;
        end case;
      end record;

    NULL_SQL_OBJECT : constant SQL_OBJECT := null;

    type CURSOR_NAME_RECORD is
      record
        CURSOR_OBJECT         : SQL_OBJECT;
        CURSOR_RDBMS          : CURSOR_ORACLE;
      end record;
    type CURSOR_NAME is access CURSOR_NAME_RECORD;

end ADA_SQL_FUNCTIONS;

--with ADA_SQL_FUNCTIONS;
--package CURSOR_DEFINITION is
--   subtype CURSOR_NAME is ADA_SQL_FUNCTIONS.CURSOR_NAME;
--end CURSOR_DEFINITION;
```

3.10.9 package ORINTB.ADA

```
-- package RDBMS_INTERFACE contains the procedures utilized by the user query
```

```
-- generation function to communicate to the Oracle RDBMS.

with TEXT_IO, SYSTEM, UNCHECKED_DEALLOCATION;
use TEXT_IO, SYSTEM;

package body RDBMS_INTERFACE is


  DB_OPEN_FLAG     : BOOLEAN := FALSE; -- indicates if a database is open
  LOGON_DATA_AREA : DATA_AREA_TYPE;
  --FIRST_FETCH      : BOOLEAN := FALSE;

  function STRING_TO_INTEGER
          (STR : in STRING)
           return   INTEGER;
  function STRING_TO_FLOAT
          (STR : in STRING)
           return   FLOAT;
  procedure FREE is new UNCHECKED_DEALLOCATION
                       (BUFFER_TYPE, BUFFER_ACCESS_TYPE);

-------------------------------------------------------------------------
-- RDBMS_COLUMN_REAPER - extraction of a string column from the retrieval
-- buffer.

  procedure RDBMS_COLUMN_REAPER
           (COLUMN_NUMBER : in INTEGER;
            VAR           : out STRING;
            VAR_LEN       : out INTEGER;
            RETRIEVAL_LEN : in RETRIEVAL_LENGTH;
            RETRIEVAL_BUF : in RETRIEVAL_BUFFER) is

    COLUMN_COUNTER : INTEGER;
    COLUMN_START   : INTEGER;
    COLUMN_END     : INTEGER;

  begin
    COLUMN_COUNTER := 0;
    COLUMN_START := 1;
    COLUMN_END := COLUMN_START;
    VAR_LEN := 0;
    loop
      if COLUMN_END > RETRIEVAL_LEN then
        raise COLUMN_RANGE_ERROR;
      end if;
      if RETRIEVAL_BUF (COLUMN_END) = '|' then
        COLUMN_COUNTER := COLUMN_COUNTER + 1;
      end if;
      if COLUMN_COUNTER = COLUMN_NUMBER then
        if VAR'LAST - VAR'FIRST + 1 < COLUMN_END - COLUMN_START then
          VAR (VAR'FIRST .. VAR'LAST) := RETRIEVAL_BUF
```

```
            (COLUMN_START .. COLUMN_START + VAR'LAST - VAR'FIRST);
          VAR_LEN := VAR'LAST - VAR'FIRST + 1;
          raise TRUNCATE_ERROR;
        else
          VAR (VAR'FIRST .. VAR'FIRST + COLUMN_END - COLUMN_START - 1) :=
            RETRIEVAL_BUF (COLUMN_START .. COLUMN_END - 1);
          VAR_LEN := (VAR'FIRST + COLUMN_END - COLUMN_START - 1) -
                     VAR'FIRST + 1;
        end if;
        exit;
      elsif RETRIEVAL_BUF (COLUMN_END) = '|' then
        COLUMN_START := COLUMN_END + 1;
        COLUMN_END := COLUMN_START;
      else
        COLUMN_END := COLUMN_END + 1;
      end if;
    end loop;
  end RDBMS_COLUMN_REAPER;

-------------------------------------------------------------------------
-- RDBMS_COLUMN_REAPER - extraction of a character column from the retrieval
-- buffer.

  procedure RDBMS_COLUMN_REAPER
            (COLUMN_NUMBER : in INTEGER;
             VAR           : out CHARACTER;
             RETRIEVAL_LEN : in RETRIEVAL_LENGTH;
             RETRIEVAL_BUF : in RETRIEVAL_BUFFER) is

    COLUMN_COUNTER : INTEGER;
    COLUMN_START   : INTEGER;
    COLUMN_END     : INTEGER;

  begin
    COLUMN_COUNTER := 0;
    COLUMN_START := 1;
    COLUMN_END := COLUMN_START;
    loop
      if COLUMN_END > RETRIEVAL_LEN then
        raise COLUMN_RANGE_ERROR;
      end if;
      if RETRIEVAL_BUF (COLUMN_END) = '|' then
        COLUMN_COUNTER := COLUMN_COUNTER + 1;
      end if;
      if COLUMN_COUNTER = COLUMN_NUMBER then
        if COLUMN_END - COLUMN_START > 1 then
          VAR := RETRIEVAL_BUF (COLUMN_START);
          raise TRUNCATE_ERROR;
        else
          VAR := RETRIEVAL_BUF (COLUMN_START);
        end if;
```

```
        exit;
      elsif RETRIEVAL_BUF (COLUMN_END) = '|' then
        COLUMN_START := COLUMN_END + 1;
        COLUMN_END := COLUMN_START;
      else
        COLUMN_END := COLUMN_END + 1;
      end if;
    end loop;
  end RDBMS_COLUMN_REAPER;

-------------------------------------------------------------------------
-- RDBMS_COLUMN_REAPER - extraction of an integer column from the retrieval
-- buffer.

  procedure RDBMS_COLUMN_REAPER
           (COLUMN_NUMBER : in INTEGER;
            VAR           : out INTEGER;
            RETRIEVAL_LEN : in RETRIEVAL_LENGTH;
            RETRIEVAL_BUF : in RETRIEVAL_BUFFER) is


    COLUMN_COUNTER : INTEGER;
    COLUMN_START   : INTEGER;
    COLUMN_END     : INTEGER;

  begin
    COLUMN_COUNTER := 0;
    COLUMN_START := 1;
    COLUMN_END := COLUMN_START;
    loop
      if COLUMN_END > RETRIEVAL_LEN then
        raise COLUMN_RANGE_ERROR;
      end if;
      if RETRIEVAL_BUF (COLUMN_END) = '|' then
        COLUMN_COUNTER := COLUMN_COUNTER + 1;
      end if;
      if COLUMN_COUNTER = COLUMN_NUMBER then
        VAR := STRING_TO_INTEGER
               (RETRIEVAL_BUF (COLUMN_START .. COLUMN_END - 1));
        exit;
      elsif RETRIEVAL_BUF (COLUMN_END) = '|' then
        COLUMN_START := COLUMN_END + 1;
        COLUMN_END := COLUMN_START;
      else
        COLUMN_END := COLUMN_END + 1;
      end if;
    end loop;

  exception
    when COLUMN_RANGE_ERROR => raise COLUMN_RANGE_ERROR;
    when others => raise TYPE_CONVERSION_ERROR;
```

```
  end RDBMS_COLUMN_REAPER;

-------------------------------------------------------------------------
-- RDBMS_COLUMN_REAPER - extraction of a float column from the retrieval
-- buffer.

  procedure RDBMS_COLUMN_REAPER
           (COLUMN_NUMBER : in INTEGER;
            VAR           : out FLOAT;
            RETRIEVAL_LEN : in RETRIEVAL_LENGTH;
            RETRIEVAL_BUF : in RETRIEVAL_BUFFER) is


    COLUMN_COUNTER : INTEGER;
    COLUMN_START   : INTEGER;
    COLUMN_END     : INTEGER;

  begin
    COLUMN_COUNTER := 0;
    COLUMN_START := 1;
    COLUMN_END := COLUMN_START;
    loop
      if COLUMN_END > RETRIEVAL_LEN then
        raise COLUMN_RANGE_ERROR;
      end if;
      if RETRIEVAL_BUF (COLUMN_END) = '|' then
        COLUMN_COUNTER := COLUMN_COUNTER + 1;
      end if;
      if COLUMN_COUNTER = COLUMN_NUMBER then
        VAR := STRING_TO_FLOAT
              (RETRIEVAL_BUF (COLUMN_START .. COLUMN_END - 1));
        exit;
      elsif RETRIEVAL_BUF (COLUMN_END) = '|' then
        COLUMN_START := COLUMN_END + 1;
        COLUMN_END := COLUMN_START;
      else
        COLUMN_END := COLUMN_END + 1;
      end if;
    end loop;

--exception
--  when others  =>  raise TYPE_CONVERSION_ERROR;
  end RDBMS_COLUMN_REAPER;

-------------------------------------------------------------------------
-- RDBMS_EXECUTE - execute a query which has been communicated to Oracle.

  procedure RDBMS_EXECUTE
           (CURSOR : in out DATA_AREA_TYPE) is

    FILE : FILE_TYPE;
```

```
  begin
-- SQL statement associated with cursor is processed
    OEXEC (CURSOR);
    RDBMS_ERROR_NUM := INTEGER (CURSOR.RETURN_CODE);
    if CURSOR.RETURN_CODE /= 0 then
      if CURSOR.RETURN_CODE =  -1001 then
        raise NO_DBOPEN_ERROR;
      elsif CURSOR.V4_ERROR_CODE = 1722 then
        raise SYNTAX_ERROR;
      elsif CURSOR.RETURN_CODE =  -110 then
-- commit current transaction to database
        OCOM (LOGON_DATA_AREA);
-- SQL statement associated with cursor is processed
        OEXEC (CURSOR);
        RDBMS_ERROR_NUM := INTEGER (CURSOR.RETURN_CODE);
        if CURSOR.RETURN_CODE /= 0 then
          if CURSOR.RETURN_CODE =  -1001 then
            raise NO_DBOPEN_ERROR;
          elsif CURSOR.V4_ERROR_CODE = 1722 then
            raise SYNTAX_ERROR;
          else
            raise UNHANDLED_RDBMS_ERROR;
          end if;
        elsif CURSOR.FILLER_DATA_1 (2) = 0 then
          if CURSOR.FILLER_DATA_1 (1) = 5 or CURSOR.FILLER_DATA_1 (1) = 9 then
            raise NO_UPDATE_ERROR;
          end if;
        end if;
      elsif CURSOR.RETURN_CODE = -9 then
        raise DUPLICATE_KEY_ERROR;
      else
        raise UNHANDLED_RDBMS_ERROR;
      end if;
    elsif CURSOR.FILLER_DATA_1 (2) = 0 then
      if CURSOR.FILLER_DATA_1 (1) = 5 or CURSOR.FILLER_DATA_1 (1) = 9 then
        raise NO_UPDATE_ERROR;
      end if;
    end if;
  end RDBMS_EXECUTE;

-------------------------------------------------------------------------------
-- RDBMS_EXIT_DATABASE - log the user off a database.

  procedure RDBMS_EXIT_DATABASE is
  begin
    if not DB_OPEN_FLAG then
      raise NO_DBOPEN_ERROR;
    else
      RDBMS_CLOSE_CURSOR (CURSOR_DATA_AREA.CURSOR_AREA);
-- disconnects a program from oracle
      OLOGOF (LOGON_DATA_AREA);
```

```
      RDBMS_ERROR_NUM := INTEGER (LOGON_DATA_AREA.RETURN_CODE);
      if LOGON_DATA_AREA.RETURN_CODE /= 0 then
        if LOGON_DATA_AREA.RETURN_CODE =  -1012 then
          raise NO_DBOPEN_ERROR;
        else
          raise UNHANDLED_RDBMS_ERROR;
        end if;
      end if;
    end if;
    DB_OPEN_FLAG := FALSE;    -- Set flag to indicate database is not open
  end RDBMS_EXIT_DATABASE;

-------------------------------------------------------------------------------
-- RDBMS_CLOSE_CURSOR - close a cursor

  procedure RDBMS_CLOSE_CURSOR
           (CURSOR : in out DATA_AREA_TYPE) is
  begin
    if not DB_OPEN_FLAG then
      raise NO_DBOPEN_ERROR;
    else
-- disconnects a cursor from oracle
      OCLOSE (CURSOR);
      RDBMS_ERROR_NUM := INTEGER (CURSOR.RETURN_CODE);
      if CURSOR.RETURN_CODE /= 0 then
        if CURSOR.RETURN_CODE =  -1001 then
          raise NO_DBOPEN_ERROR;
        else
          raise UNHANDLED_RDBMS_ERROR;
        end if;
      end if;
    end if;
  end RDBMS_CLOSE_CURSOR;

-------------------------------------------------------------------------------
-- RDBMS_FETCH - fetch data into the user-program.

  procedure RDBMS_FETCH
           (CURSOR            : in out CURSOR_ORACLE) is

    COL_LENGTH      : array (1 .. 127) of SHORT_INTEGER;
    COL_LENGTH_INT  : array (1 .. 127) of INTEGER;
    FTYPE           : INTEGER;
    FIELD_NUM       : INTEGER;
    FIELD_LENGTH    : INTEGER;

  begin
    if not DB_OPEN_FLAG then
      raise NO_DBOPEN_ERROR;
    else
      if CURSOR.CURSOR_FIRST_FETCH then
```

```
        FIELD_NUM := 1;
        while FIELD_NUM <= 127 loop
-- return type & size of field of SQL query
          ODSC (CURSOR.CURSOR_DATA.CURSOR_AREA, SHORT_INTEGER
           (FIELD_NUM),
                DSIZE => COL_LENGTH (FIELD_NUM));
          exit when CURSOR.CURSOR_DATA.CURSOR_AREA.RETURN_CODE /= 0;
          FIELD_NUM := FIELD_NUM + 1;
        end loop;
        FIELD_LENGTH := 1;
        CURSOR.CURSOR_MAX_COLUMN := FIELD_NUM - 1;
        for COUNTER in 1 .. FIELD_NUM - 1 loop
-- define output buffer for one field of SQL query
          ODEFIN (CURSOR.CURSOR_DATA.CURSOR_AREA, SHORT_INTEGER (COUNTER),
                  CURSOR.CURSOR_RETRIEVAL_BUF (FIELD_LENGTH)'ADDRESS,
                  COL_LENGTH (COUNTER), 1);
          FIELD_LENGTH := FIELD_LENGTH + INTEGER (COL_LENGTH (COUNTER));
          CURSOR.CURSOR_RETRIEVAL_BUF (FIELD_LENGTH) := '|';
          FIELD_LENGTH := FIELD_LENGTH + 1;
        end loop;
        CURSOR.CURSOR_RETRIEVAL_LEN := FIELD_LENGTH;
        CURSOR.CURSOR_FIRST_FETCH := FALSE;
      end if;
-- return one row of a query result
      OFETCH (CURSOR.CURSOR_DATA.CURSOR_AREA);
      RDBMS_ERROR_NUM :=
        INTEGER (CURSOR.CURSOR_DATA.CURSOR_AREA.RETURN_CODE);
      if CURSOR.CURSOR_DATA.CURSOR_AREA.RETURN_CODE /= 0 then
        if CURSOR.CURSOR_DATA.CURSOR_AREA.V4_ERROR_CODE = 1002 or
           CURSOR.CURSOR_DATA.CURSOR_AREA.V4_ERROR_CODE = 1003 then
          raise NO_RETRIEVE_ERROR;
        elsif CURSOR.CURSOR_DATA.CURSOR_AREA.V4_ERROR_CODE = 1403 then
          raise NO_ROW_ERROR;
        elsif CURSOR.CURSOR_DATA.CURSOR_AREA.V4_ERROR_CODE = 1406 then
          RDBMS_ERROR_NUM := 0;
        else
          raise UNHANDLED_RDBMS_ERROR;
        end if;
      end if;
    end if;
  end RDBMS_FETCH;

-------------------------------------------------------------------------------
-- RDBMS_OPEN_DATABASE - log the user onto a database.

  procedure RDBMS_OPEN_DATABASE
            (QUERY_BUFFER : in BUFFER_ACCESS_TYPE;
             QUERY_LENGTH : in INTEGER;
             QUERYX_BUFFER : in BUFFER_ACCESS_TYPE;
             QUERYX_LENGTH : in INTEGER) is
  begin
```

```
    if DB_OPEN_FLAG then
      raise DB_ALREADY_OPEN_ERROR;
    else
-- establish communication between oracle and the user program
      OLON (LOGON_DATA_AREA, QUERY_BUFFER.all, SHORT_INTEGER (QUERY_LENGTH),
            QUERYX_BUFFER.all, SHORT_INTEGER (QUERYX_LENGTH));
      RDBMS_ERROR_NUM := INTEGER (LOGON_DATA_AREA.RETURN_CODE);
      if LOGON_DATA_AREA.RETURN_CODE /= 0 then
        if LOGON_DATA_AREA.RETURN_CODE =  -1017 then
          raise DBOPEN_ERROR;
        else
          raise UNHANDLED_RDBMS_ERROR;
        end if;
      end if;
      RDBMS_OPEN_CURSOR (CURSOR_DATA_AREA.CURSOR_AREA);
    end if;
    DB_OPEN_FLAG := TRUE;
  end RDBMS_OPEN_DATABASE;

-------------------------------------------------------------------------------
-- RDBMS_OPEN_CURSOR - open a cursor under a particular database.

  procedure RDBMS_OPEN_CURSOR
          (CURSOR : in out DATA_AREA_TYPE) is
  begin
-- establishes a cursor to pass a SQL statement to Oracle
    OOPEN (CURSOR, LOGON_DATA_AREA);
    RDBMS_ERROR_NUM := INTEGER (CURSOR.RETURN_CODE);
    if CURSOR_DATA_AREA.CURSOR_AREA.RETURN_CODE /= 0 then
      raise DBOPEN_ERROR;
    end if;
  end RDBMS_OPEN_CURSOR;

-------------------------------------------------------------------------------
-- RDBMS_COMMIT_WORK - communicate a commit work

  procedure RDBMS_COMMIT_WORK is
  begin
    if not DB_OPEN_FLAG then
      raise NO_DBOPEN_ERROR;
    else
-- commit current transaction to database
      OCOM (LOGON_DATA_AREA);
    end if;
  end RDBMS_COMMIT_WORK;

-------------------------------------------------------------------------------
-- RDBMS_ROLLBACK_WORK - communicate a rollback transaction

  procedure RDBMS_ROLLBACK_WORK is
  begin
```

```
    if not DB_OPEN_FLAG then
      raise NO_DBOPEN_ERROR;
    else
-- roll back the current transaction
      OROL (LOGON_DATA_AREA);
    end if;
  end RDBMS_ROLLBACK_WORK;

-------------------------------------------------------------------------------
-- RDBMS_ONAME - retrieve the names of the columns used in a SELECT clause of
-- a query.

  procedure RDBMS_ONAME
            (CURSOR          : in out DATA_AREA_TYPE;
             NAMES           : in out COLUMN_NAMES_TYPE;
             NUMBER_OF_NAMES : out INTEGER) is

    FIELD_NUM          : INTEGER;

  begin
    if not DB_OPEN_FLAG then
      raise NO_DBOPEN_ERROR;
    else
      FIELD_NUM := 1;
      while FIELD_NUM <= 127 loop
        NAMES (FIELD_NUM).COLUMN_NAME_LENGTH := 18;
-- get names of columns of SQL query
        ONAME (CURSOR, SHORT_INTEGER (FIELD_NUM), -1, -1,
          NAMES (FIELD_NUM).COLUMN_NAME'ADDRESS,
          NAMES (FIELD_NUM).COLUMN_NAME_LENGTH'ADDRESS);
        exit when CURSOR.RETURN_CODE /= 0;
        FIELD_NUM := FIELD_NUM + 1;
      end loop;
      NUMBER_OF_NAMES := FIELD_NUM - 1;
    end if;
  end RDBMS_ONAME;

-------------------------------------------------------------------------------
-- RDBMS_QUERY - communicate a query

  procedure RDBMS_QUERY
            (QUERY_BUFFER : in out BUFFER_ACCESS_TYPE;
             QUERY_LENGTH : in INTEGER;
             CURSOR       : in out DATA_AREA_TYPE) is

    FILE : FILE_TYPE;

  begin
    if not DB_OPEN_FLAG then
      raise NO_DBOPEN_ERROR;
    else
```

```
-- associated SQL statement with cursor & pass it to oracle
      OSQL3 (CURSOR, QUERY_BUFFER.all, SHORT_INTEGER (QUERY_LENGTH));
      RDBMS_ERROR_NUM := INTEGER (CURSOR.RETURN_CODE);
      if CURSOR.RETURN_CODE /= 0 then
        if CURSOR.RETURN_CODE =  -1001 then
          raise NO_DBOPEN_ERROR;
        elsif CURSOR.V4_ERROR_CODE = 704 then
          raise COLUMN_ERROR;
        elsif CURSOR.V4_ERROR_CODE = 901 or
              CURSOR.V4_ERROR_CODE = 955 then
          raise CREATE_ERROR;
        elsif CURSOR.V4_ERROR_CODE = 942 then
          raise NO_TABLE_ERROR;
        elsif CURSOR.V4_ERROR_CODE = 902 or
              CURSOR.V4_ERROR_CODE = 984 or
              CURSOR.V4_ERROR_CODE = 1722 then
          raise SYNTAX_ERROR;
        elsif CURSOR.RETURN_CODE = -1747 then
          raise ACCESS_ERROR;
        else
          raise UNHANDLED_RDBMS_ERROR;
        end if;
      end if;
      RDBMS_EXECUTE (CURSOR);
    end if;
    -- FREE (QUERY_BUFFER);
  end RDBMS_QUERY;

-----------------------------------------------------------------------
-- STRING_TO_INTEGER - convert a string to an integer

  function STRING_TO_INTEGER
          (STR : in STRING)
           return   INTEGER is

    VAL_INT          : INTEGER;
    I                : INTEGER;
    EXP              : INTEGER;
    NEGATIVE_FLAG    : BOOLEAN;

  begin
    NEGATIVE_FLAG := FALSE;
    EXP := 10;
    I := 0;
    VAL_INT := 0;
    for J in reverse STR'range loop
      if STR (J) = '0' then
        I := 0;
      elsif STR (J) = '1' then
        I := 1;
      elsif STR (J) = '2' then
```

```
        I := 2;
      elsif STR (J) = '3' then
        I := 3;
      elsif STR (J) = '4' then
        I := 4;
      elsif STR (J) = '5' then
        I := 5;
      elsif STR (J) = '6' then
        I := 6;
      elsif STR (J) = '7' then
        I := 7;
      elsif STR (J) = '8' then
        I := 8;
      elsif STR (J) = '9' then
        I := 9;
      elsif STR (J) = '-' then
        NEGATIVE_FLAG := TRUE;
        exit;
      elsif STR (J) = ' ' or STR (J) = '+' then
        exit;
      else
        raise TYPE_CONVERSION_ERROR;
      end if;
      VAL_INT := VAL_INT + (I * (EXP ** (STR'LAST - J)));
    end loop;
    if NEGATIVE_FLAG then
      VAL_INT := VAL_INT * (-1);
    end if;
    return VAL_INT;
  end STRING_TO_INTEGER;

-------------------------------------------------------------------------------
-- STRING_TO_FLOAT - convert a string to a float

  function STRING_TO_FLOAT
          (STR : in STRING)
           return   FLOAT is

    package FLT_IO is new FLOAT_IO (LONG_LONG_FLOAT);
    use FLT_IO;
    VAL_FLT : LONG_LONG_FLOAT;
    LAST    : INTEGER;
    DOT     : BOOLEAN;

  begin
    for I in STR'FIRST .. STR'LAST loop
      if STR (I) /= ' ' then
        if STR (I) = '.' then
          GET ("0" & STR (I .. STR'LAST), VAL_FLT, LAST);
        else
          DOT := FALSE;
```

```
          for J in I .. STR'LAST loop
            if STR (J) = '.' then
              DOT := TRUE;
              exit;
            end if;
          end loop;
          if DOT then
            GET (STR (I .. STR'LAST), VAL_FLT, LAST);
          else
            GET (STR (I .. STR'LAST) & ".0", VAL_FLT, LAST);
          end if;
        end if;
        exit;
      end if;
    end loop;
    return FLOAT (VAL_FLT);
  end STRING_TO_FLOAT;

-------------------------------------------------------------------------------
-- RANGE_COUNT

  function RANGE_COUNT
          (STR : in STRING)
           return   INTEGER is

    RANGE_OF : constant STRING (1 .. 8) := "range of";
    COUNT    : INTEGER := 1;

  begin
    for I in STR'FIRST .. STR'LAST - 8 loop
      if STR (I .. I + 7) = RANGE_OF then
        COUNT := COUNT + 1;
      end if;
    end loop;
    return COUNT;
  end RANGE_COUNT;

end RDBMS_INTERFACE;
```

3.10.10 package FUNCTIONB.ADA

```
with TEXT_PRINT, TEXT_IO, RDBMS_INTERFACE;
 use TEXT_PRINT, TEXT_IO, RDBMS_INTERFACE;

package body ADA_SQL_FUNCTIONS is

  INDENT : STANDARD.INTEGER;
  FETCH_CURSOR : CURSOR_NAME;
  OPERATION_CURSOR : CURSOR_NAME := new CURSOR_NAME_RECORD;
  DOING_A_SELECT : BOOLEAN := FALSE;
```

```
  package DOUBLE_PRECISION_PRINT is new
   FLOAT_PRINT ( DATABASE.DOUBLE_PRECISION );

  package INT_PRINT is new INTEGER_PRINT ( DATABASE.INT );

  use DOUBLE_PRECISION_PRINT , INT_PRINT;

  LINE : LINE_TYPE;

-- declarations for print routines (since some are recursive and mutually
--  recursive)

  procedure SHOW_VALUE_SPECIFICATION ( S : in SQL_OBJECT );
  procedure SHOW_ALL_SET_FUNCTION    ( S : in SQL_OBJECT );
  procedure SHOW_VALUE_EXPRESSION    ( S : in SQL_OBJECT );
  procedure SHOW_BETWEEN_PREDICATE   ( S : in SQL_OBJECT );
  procedure SHOW_IN_VALUE_LIST       ( S : in SQL_OBJECT );
  procedure SHOW_LIKE_PREDICATE      ( S : in SQL_OBJECT );
  procedure SHOW_SEARCH_CONDITION    ( S : in SQL_OBJECT );
  procedure SHOW_TABLE_EXPRESSION    ( S : in SQL_OBJECT );
  procedure SHOW_QUERY_SPECIFICATION ( S : in SQL_OBJECT );
  procedure SHOW_SELECT_LIST         ( S : in SQL_OBJECT );
  procedure SHOW_ORDER_BY_CLAUSE     ( S : in SQL_OBJECT );
  procedure SHOW_INSERT_VALUE_LIST   ( S : in SQL_OBJECT );
  procedure SHOW_SET_CLAUSES         ( S : in SQL_OBJECT );
  procedure SHOW_COMPARISON_PREDICATE
            ( S : in SQL_OBJECT ; P : in STANDARD.STRING );
  procedure SHOW_IN_PREDICATE
            ( S : in SQL_OBJECT ; P : in STANDARD.STRING );

  procedure INITIATE_TEST is -- ***** FOR TESTING ONLY
  begin
    CREATE_LINE ( LINE , 79 );
    SET_LINE ( LINE );
    SET_CONTINUATION_INDENT ( 7 );
  end INITIATE_TEST;

-- constant literal value generator

  function CONSTANT_LITERAL return RESULT_TYPE is
  begin
    return VALUE;
  end CONSTANT_LITERAL;

-- conversion routines for SQL objects

  function L_CONVERT ( L : TYPED_SQL_OBJECT ) return SQL_OBJECT is
  begin
    return SQL_OBJECT ( L );
  end L_CONVERT;
```

```
  function CONVERT_R ( R : SQL_OBJECT ) return TYPED_SQL_OBJECT is
  begin
    return TYPED_SQL_OBJECT ( R );
  end CONVERT_R;

  package body CONVERT is

    function L_CONVERT ( L : SQL_OBJECT ) return SQL_OBJECT is
    begin
      return L;
    end L_CONVERT;

    function L_CONVERT ( L : TABLE_NAME ) return SQL_OBJECT is
    begin
      return SQL_OBJECT ( L );
    end L_CONVERT;

    function CONVERT_R ( R : SQL_OBJECT ) return TABLE_NAME is
    begin
      return TABLE_NAME ( R );
    end CONVERT_R;

    function L_CONVERT ( L : TABLE_LIST ) return SQL_OBJECT is
    begin
      return SQL_OBJECT ( L );
    end L_CONVERT;

    function CONVERT_R ( R : SQL_OBJECT ) return TABLE_LIST is
    begin
      return TABLE_LIST ( R );
    end CONVERT_R;

    function L_CONVERT ( L : INSERT_ITEM ) return SQL_OBJECT is
    begin
      return SQL_OBJECT ( L );
    end L_CONVERT;

    function CONVERT_R ( R : SQL_OBJECT ) return INSERT_ITEM is
    begin
      return INSERT_ITEM ( R );
    end CONVERT_R;

  end CONVERT;

-- conversion routines for user types

  function INTEGER_AND_ENUMERATION_CONVERT ( VAR : USER_TYPE )
   return SQL_OBJECT is
  begin
    return
     new SQL_OBJECT_RECORD'
```

```
      ( VALUE , null , ( INTEGER , USER_TYPE'POS ( VAR ) ) );
end INTEGER_AND_ENUMERATION_CONVERT;

function FLOAT_CONVERT ( VAR : USER_TYPE ) return SQL_OBJECT is
begin
  return
   new SQL_OBJECT_RECORD'
       ( VALUE , null , ( FLOAT , DATABASE.DOUBLE_PRECISION ( VAR ) ) );
end FLOAT_CONVERT;

function NUMBER_OF_QUOTES_IN ( S : STANDARD.STRING )
         return STANDARD.INTEGER is
  C : STANDARD.INTEGER := 0;
begin
  for I in S'RANGE loop
    if S(I) = ''' then
      C := C + 1;
    end if;
  end loop;
  return C;
end NUMBER_OF_QUOTES_IN;

function SQL_OBJECT_FOR_STRING ( S : in STANDARD.STRING )
 return SQL_OBJECT is
  L : STANDARD.INTEGER       := NUMBER_OF_QUOTES_IN ( S );
  J : POSITIVE       := 1;
  R : ACCESS_STRING := new STANDARD.STRING ( 1 .. S'LENGTH + L );
begin
  if L = 0 then
    R.all := S;
  else
    for I in S'RANGE loop
      if S(I) = ''' then
        R ( J .. J+1 ) := """";
        J := J + 2;
      else
        R(J) := S(I);
        J := J + 1;
      end if;
    end loop;
  end if;
  return new SQL_OBJECT_RECORD' ( VALUE , null , ( STRING , R ) );
end SQL_OBJECT_FOR_STRING;

function UNCONSTRAINED_CHARACTER_STRING_CONVERT ( VAR : USER_TYPE )
 return SQL_OBJECT is
begin
  return SQL_OBJECT_FOR_STRING ( STANDARD.STRING ( VAR ) );
end UNCONSTRAINED_CHARACTER_STRING_CONVERT;

function CONSTRAINED_CHARACTER_STRING_CONVERT ( VAR : USER_TYPE )
```

```
    return SQL_OBJECT is
  begin
    return SQL_OBJECT_FOR_STRING ( STANDARD.STRING ( VAR ) );
  end CONSTRAINED_CHARACTER_STRING_CONVERT;

  function UNCONSTRAINED_STRING_CONVERT ( VAR : USER_TYPE )
    return SQL_OBJECT is
    S : STANDARD.STRING ( 1 .. VAR'LENGTH );
    I : POSITIVE := 1;
  begin
    for J in VAR'RANGE loop
      S(I) := CONVERT_COMPONENT_TO_CHARACTER ( VAR(J) );
      I := I + 1;
    end loop;
    return SQL_OBJECT_FOR_STRING ( S );
  end UNCONSTRAINED_STRING_CONVERT;

  function CONSTRAINED_STRING_CONVERT ( VAR : USER_TYPE )
    return SQL_OBJECT is
    S : STANDARD.STRING ( 1 .. VAR'LENGTH );
    I : POSITIVE := 1;
  begin
    for J in VAR'RANGE loop
      S(I) := CONVERT_COMPONENT_TO_CHARACTER ( VAR(J) );
      I := I + 1;
    end loop;
    return SQL_OBJECT_FOR_STRING ( S );
  end CONSTRAINED_STRING_CONVERT;

-- column and table name routines

  package body NAME_PACKAGE is

    NAME_P : constant DATABASE_NAME := new STANDARD.STRING' ( GIVEN_NAME );

    NAME_O : constant SQL_OBJECT :=
      new SQL_OBJECT_RECORD' ( NAME , null , NAME_P );

    function COLUMN_OR_TABLE_NAME return SQL_OBJECT_TYPE is
    begin
      return CONVERT_R ( NAME_O );
    end COLUMN_OR_TABLE_NAME;

    function TABLE_NAME_WITH_COLUMN_LIST ( COLUMNS : SQL_OBJECT )
      return TABLE_NAME is
      N : SQL_OBJECT := new SQL_OBJECT_RECORD' ( NAME , COLUMNS , NAME_P );
    begin
      return
        new SQL_OBJECT_RECORD' ( OPERATION , null , O_TABLE_COLUMN_LIST , N );
    end TABLE_NAME_WITH_COLUMN_LIST;
```

```
  end NAME_PACKAGE;

-- value specification routines

  function INDICATOR_FUNCTION ( VAL : USER_TYPE ) return RESULT_TYPE is
  begin
    return CONVERT_R ( L_CONVERT ( VAL ) );
  end INDICATOR_FUNCTION;

-- generic operation routines

  function COPY_NAME ( OBJECT : SQL_OBJECT ) return SQL_OBJECT is
  begin
    if OBJECT /= null and then OBJECT.KIND = NAME then
      return new SQL_OBJECT_RECORD' ( OBJECT.all );
    else
      return OBJECT;
    end if;
  end COPY_NAME;

  function UNARY_OPERATION ( L : L_TYPE ) return TYPE_R is
  begin
    return
     CONVERT_R
     ( new SQL_OBJECT_RECORD'
       ( OPERATION , null , GIVEN_OPERATION , L_CONVERT ( L ) ) );
  end UNARY_OPERATION;

  function BINARY_OPERATION ( L : L_TYPE ; R : R_TYPE ) return TYPE_R is
    LEFT : SQL_OBJECT := COPY_NAME ( L_CONVERT ( L ) );
  begin
    LEFT.ACROSS := R_CONVERT ( R );
    return
     CONVERT_R
     ( new SQL_OBJECT_RECORD' ( OPERATION , null , GIVEN_OPERATION , LEFT ) );
  end BINARY_OPERATION;

-- set function routines

  function COUNT_STAR return TYPE_R is
  begin
    return
     CONVERT_R
     ( new SQL_OBJECT_RECORD' ( OPERATION , null , O_COUNT_STAR , null ) );
  end COUNT_STAR;

-- subquery routines

  function NEW_TAIL ( L , R : SQL_OBJECT ) return SQL_OBJECT is
  begin
    if R = null then
```

```
    L.ACROSS :=
     new SQL_OBJECT_RECORD' ( OPERATION , null , O_NULL_OP , null );
  else
    L.ACROSS := R;
  end if;
  return L.ACROSS;
end NEW_TAIL;

function BUILD_SELECT
         ( SELECT_TYPE                : SQL_OPERATION;
           WHAT                       : SQL_OBJECT;
           FROM                       : TABLE_LIST;
           WHERE , GROUP_BY , HAVING  : SQL_OBJECT )
 return SQL_OBJECT is
  F     : SQL_OBJECT := COPY_NAME ( SQL_OBJECT ( FROM ) );
  W     : SQL_OBJECT := COPY_NAME ( WHAT );
  TAIL  : SQL_OBJECT :=
   NEW_TAIL
   ( NEW_TAIL
     ( NEW_TAIL ( F , WHERE ) , COPY_NAME ( GROUP_BY ) ) , HAVING );
begin
  W.ACROSS := F;
  return new SQL_OBJECT_RECORD' ( OPERATION , null , SELECT_TYPE , W );
end BUILD_SELECT;

function SELECT_LIST_SUBQUERY
         ( WHAT      : WHAT_TYPE;
           FROM      : TABLE_LIST;
           WHERE     : SQL_OBJECT := NULL_SQL_OBJECT;
           GROUP_BY  : SQL_OBJECT := NULL_SQL_OBJECT;
           HAVING    : SQL_OBJECT := NULL_SQL_OBJECT ) return TYPE_R is
begin
  return
   CONVERT_R
   ( BUILD_SELECT
     ( SELECT_TYPE,
       L_CONVERT ( WHAT ) , FROM , WHERE , GROUP_BY , HAVING ) );
end SELECT_LIST_SUBQUERY;

function STAR_SUBQUERY
         ( FROM      : TABLE_LIST;
           WHERE     : SQL_OBJECT := NULL_SQL_OBJECT;
           GROUP_BY  : SQL_OBJECT := NULL_SQL_OBJECT;
           HAVING    : SQL_OBJECT := NULL_SQL_OBJECT ) return TYPE_R is
begin
  return
   CONVERT_R
   ( BUILD_SELECT
     ( SELECT_TYPE,
       new SQL_OBJECT_RECORD' ( OPERATION , null , O_STAR , null ),
       FROM , WHERE , GROUP_BY , HAVING ) );
```

```
  end STAR_SUBQUERY;

-- print routines

  -- 5.6.1 <value specification>

  procedure SHOW_VALUE_SPECIFICATION ( S : in SQL_OBJECT ) is
  begin
    case S.VALUE.KIND is
      when INTEGER => PRINT ( S.VALUE.INTEGER );
      when FLOAT   => PRINT ( S.VALUE.FLOAT );
      when STRING  => PRINT ( "'" & S.VALUE.STRING.all & "'" );
    end case;
  end SHOW_VALUE_SPECIFICATION;

  -- 5.8.3 <all set function>

  procedure SHOW_ALL_SET_FUNCTION ( S : in SQL_OBJECT ) is
  begin
    case S.OPERATION is
      when O_AVG  => PRINT ( "AVG( " );
      when O_MAX  => PRINT ( "MAX( " );
      when O_MIN  => PRINT ( "MIN( " );
      when O_SUM  => PRINT ( "SUM( " );
      when others => raise INTERNAL_ERROR;
    end case;
    SHOW_VALUE_EXPRESSION (  S.OPERANDS );
    PRINT ( " )" );
  end SHOW_ALL_SET_FUNCTION;

  -- 5.9.1 <value expression>

  procedure PARENTHESIZE_ADDING_OPERANDS
            ( S : in SQL_OBJECT ; P : in STANDARD.STRING ) is
  begin
    SHOW_VALUE_EXPRESSION ( S );
    PRINT ( P );
    if S.ACROSS.KIND = OPERATION then
      case S.ACROSS.OPERATION is
        when O_UNARY_MINUS | O_PLUS | O_MINUS =>
          PRINT ( "( " );
          SHOW_VALUE_EXPRESSION ( S.ACROSS );
          PRINT ( " )" );
        when others =>
          SHOW_VALUE_EXPRESSION ( S.ACROSS );
      end case;
    else
      SHOW_VALUE_EXPRESSION ( S.ACROSS );
    end if;
  end PARENTHESIZE_ADDING_OPERANDS;
```

```
procedure PARENTHESIZE_MULTIPLYING_OPERANDS
          ( S : in SQL_OBJECT ; P : in STANDARD.STRING ) is
begin
  if S.KIND = OPERATION then
    case S.OPERATION is
      when O_UNARY_MINUS | O_PLUS | O_MINUS =>
        PRINT ( "( " );
        SHOW_VALUE_EXPRESSION ( S );
        PRINT ( " )" );
      when others =>
        SHOW_VALUE_EXPRESSION ( S );
    end case;
  else
    SHOW_VALUE_EXPRESSION ( S );
  end if;
  PRINT ( P );
  if S.ACROSS.KIND = OPERATION then
    case S.ACROSS.OPERATION is
      when O_UNARY_MINUS | O_PLUS | O_MINUS | O_TIMES | O_DIVIDE =>
        PRINT ( "( " );
        SHOW_VALUE_EXPRESSION ( S.ACROSS );
        PRINT ( " )" );
      when others =>
        SHOW_VALUE_EXPRESSION ( S.ACROSS );
    end case;
  else
    SHOW_VALUE_EXPRESSION ( S.ACROSS );
  end if;
end PARENTHESIZE_MULTIPLYING_OPERANDS;

procedure SHOW_VALUE_EXPRESSION ( S : in SQL_OBJECT ) is
begin
  case S.KIND is
    when VALUE =>
      SHOW_VALUE_SPECIFICATION ( S );
    when NAME =>
      PRINT ( S.NAME.all );
    when OPERATION =>
      case S.OPERATION is
        when O_AVG | O_MAX | O_MIN | O_SUM =>
          SHOW_ALL_SET_FUNCTION ( S );
        when O_COUNT_STAR =>
          PRINT ( "COUNT(*)" );
        when O_UNARY_PLUS =>
          SHOW_VALUE_EXPRESSION ( S.OPERANDS );
        when O_UNARY_MINUS =>
          PRINT ( " - " );
          if S.OPERANDS.KIND = OPERATION then
            case S.OPERANDS.OPERATION is
              when O_UNARY_MINUS | O_PLUS | O_MINUS | O_TIMES | O_DIVIDE =>
                PRINT ( "( " );
```

```
                SHOW_VALUE_EXPRESSION ( S.OPERANDS );
                PRINT ( " )" );
              when others => SHOW_VALUE_EXPRESSION ( S.OPERANDS );
            end case;
          else
            SHOW_VALUE_EXPRESSION ( S.OPERANDS );
          end if;
        when O_PLUS =>
          PARENTHESIZE_ADDING_OPERANDS ( S.OPERANDS , " + " );
        when O_MINUS =>
          PARENTHESIZE_ADDING_OPERANDS ( S.OPERANDS , " - " );
        when O_TIMES =>
          PARENTHESIZE_MULTIPLYING_OPERANDS ( S.OPERANDS , " * " );
        when O_DIVIDE =>
          PARENTHESIZE_MULTIPLYING_OPERANDS ( S.OPERANDS , " / " );
        when others => raise INTERNAL_ERROR;
      end case;
  end case;
end SHOW_VALUE_EXPRESSION;

-- 5.11.1 <comparison predicate>

procedure SHOW_COMPARISON_PREDICATE
          ( S : in SQL_OBJECT ; P : in STANDARD.STRING ) is
begin
  SHOW_VALUE_EXPRESSION ( S );
  PRINT ( P );
  if S.ACROSS.KIND = OPERATION then
    case S.ACROSS.OPERATION is
      when O_SELEC | O_SELECT_DISTINCT =>
        SHOW_QUERY_SPECIFICATION ( S.ACROSS );
      when others =>
        SHOW_VALUE_EXPRESSION ( S.ACROSS );
    end case;
  else
    SHOW_VALUE_EXPRESSION ( S.ACROSS );
  end if;
end SHOW_COMPARISON_PREDICATE;

-- 5.12.1 <between predicate>

procedure SHOW_BETWEEN_PREDICATE ( S : in SQL_OBJECT ) is
  OPERAND : SQL_OBJECT := S.ACROSS.OPERANDS; -- first operand of AND
begin
  SHOW_VALUE_EXPRESSION ( S );
  PRINT ( " BETWEEN " );
  SHOW_VALUE_EXPRESSION ( OPERAND );
  PRINT ( " AND " );
  SHOW_VALUE_EXPRESSION ( OPERAND.ACROSS );
end SHOW_BETWEEN_PREDICATE;
```

```
-- 5.13.1 <in predicate>

procedure SHOW_IN_PREDICATE
          ( S : in SQL_OBJECT ; P : in STANDARD.STRING ) is
begin
  PRINT ( P );
  SHOW_VALUE_EXPRESSION ( S );
  PRINT ( " IN " );
  if S.ACROSS.KIND = OPERATION then
    case S.ACROSS.OPERATION is
      when O_SELEC | O_SELECT_DISTINCT =>
        SHOW_QUERY_SPECIFICATION ( S.ACROSS );
        return;
      when others =>
        null;
    end case;
  end if;
  PRINT ( "( " ); --( "< " );
  SHOW_IN_VALUE_LIST ( S.ACROSS ); PRINT ( " )" ); --( " >" );
end SHOW_IN_PREDICATE;

-- 5.13.2 <in value list>

procedure SHOW_IN_VALUE_LIST ( S : in SQL_OBJECT ) is
begin
  case S.KIND is
    when VALUE =>
      SHOW_VALUE_SPECIFICATION ( S );
    when OPERATION =>
      if S.OPERATION /= O_OR then
        raise INTERNAL_ERROR;
      end if;
      SHOW_IN_VALUE_LIST ( S.OPERANDS );
      PRINT ( ", " );
      SHOW_IN_VALUE_LIST ( S.OPERANDS.ACROSS );
    when others =>
      raise INTERNAL_ERROR;
  end case;
end SHOW_IN_VALUE_LIST;

-- 5.14.1 <like predicate>

procedure SHOW_LIKE_PREDICATE ( S : in SQL_OBJECT ) is
  P : ACCESS_STRING := S.ACROSS.VALUE.STRING; -- must be of right type
begin
  PRINT ( S.NAME.all ); PRINT ( " LIKE " ); --( " = " );
  --for I in P'RANGE loop
    --case P(I) is
      --when '_'    => P(I) := '?';
      --when '%'    => P(I) := '*';
      --when others => null;
```

```
    --end case;
  --end loop;
  SHOW_VALUE_SPECIFICATION  ( S.ACROSS );
end SHOW_LIKE_PREDICATE;

-- 5.18.1 <search condition>

procedure PARENTHESIZE_RELATIONAL_OPERATORS
          ( S : in SQL_OBJECT ; P : in STANDARD.STRING ) is
  OPERAND : SQL_OBJECT := S.OPERANDS;
begin
  case OPERAND.OPERATION is -- must be operation
    when O_AND | O_OR =>
      if OPERAND.OPERATION /= S.OPERATION then
        PRINT ( "( " );   --( "[ " );
        SHOW_SEARCH_CONDITION ( OPERAND ); PRINT ( ") " );  --( " ]" )
      else
        SHOW_SEARCH_CONDITION ( OPERAND );
      end if;
    when others => SHOW_SEARCH_CONDITION ( OPERAND );
  end case;
  PRINT_LINE; PRINT ( P );
  OPERAND := OPERAND.ACROSS;
  case OPERAND.OPERATION is -- again, must be operation
    when O_AND | O_OR =>
      PRINT ( "( " );   --( "[ " );
      SHOW_SEARCH_CONDITION ( OPERAND ); PRINT ( ") " );  --( " ]" );
    when others =>
      SHOW_SEARCH_CONDITION ( OPERAND );
  end case;
end PARENTHESIZE_RELATIONAL_OPERATORS;

procedure SHOW_SEARCH_CONDITION ( S : in SQL_OBJECT ) is
begin
  case S.OPERATION is
    when O_EQ      => SHOW_COMPARISON_PREDICATE ( S.OPERANDS , " = " );
    when O_NE      => SHOW_COMPARISON_PREDICATE ( S.OPERANDS , " ^= " );
    when O_LT      => SHOW_COMPARISON_PREDICATE ( S.OPERANDS , " < " );
    when O_GT      => SHOW_COMPARISON_PREDICATE ( S.OPERANDS , " > " );
    when O_LE      => SHOW_COMPARISON_PREDICATE ( S.OPERANDS , " <= " );
    when O_GE      => SHOW_COMPARISON_PREDICATE ( S.OPERANDS , " >= " );
    when O_BETWEEN => SHOW_BETWEEN_PREDICATE    ( S.OPERANDS );
    when O_IS_IN   => SHOW_IN_PREDICATE         ( S.OPERANDS , "" );
    when O_NOT_IN  => SHOW_IN_PREDICATE         ( S.OPERANDS , "NOT" );
    when O_LIKE    => SHOW_LIKE_PREDICATE       ( S.OPERANDS );
    when O_AND     => PARENTHESIZE_RELATIONAL_OPERATORS ( S , "AND    " );
    when O_OR      => PARENTHESIZE_RELATIONAL_OPERATORS ( S , "OR     " );
    when O_NOT =>
      PRINT ( "NOT " );
      case S.OPERANDS.OPERATION is -- must be operation
        when O_AND | O_OR =>
```

```
            PRINT ( "( " );   --( "[ " );
            SHOW_SEARCH_CONDITION ( S.OPERANDS );
            PRINT ( ") " );  --( " ]" );
          when others =>
            SHOW_SEARCH_CONDITION ( S.OPERANDS );
        end case;
      when others => raise INTERNAL_ERROR;
    end case;
  end SHOW_SEARCH_CONDITION;

  -- 5.19.1 <table expression>

  procedure SHOW_TABLE_EXPRESSION ( S : in SQL_OBJECT ) is
    CLAUSE : SQL_OBJECT := S.ACROSS;
  begin
    PRINT ( "FROM   " ); SHOW_SELECT_LIST ( S );
    if CLAUSE.OPERATION /= O_NULL_OP then -- WHERE must have operation on top
      PRINT_LINE; PRINT ( "WHERE  " ); SHOW_SEARCH_CONDITION ( CLAUSE );
    end if;
    CLAUSE := CLAUSE.ACROSS;
    if CLAUSE.KIND /= OPERATION or else CLAUSE.OPERATION /= O_NULL_OP then
      PRINT_LINE; PRINT ( "GROUP  BY " ); SHOW_SELECT_LIST ( CLAUSE );
    end if;
    CLAUSE := CLAUSE.ACROSS;
    if CLAUSE.OPERATION /= O_NULL_OP then -- same as WHERE
      PRINT_LINE; PRINT ( "HAVING " ); SHOW_SEARCH_CONDITION ( CLAUSE );
    end if;
  end SHOW_TABLE_EXPRESSION;

  -- 5.25.1 <query specification>

  procedure SHOW_QUERY_SPECIFICATION ( S : in SQL_OBJECT) is
    CLAUSE : SQL_OBJECT := S.OPERANDS;
  begin
    INDENT := INDENT + 7; PRINT_LINE;
    if INDENT > 0 then
      SET_INDENT ( INDENT - 2 );
      PRINT ( "( " );
    end if;
    SET_INDENT ( INDENT );
    PRINT ( "SELECT " );
    case S.OPERATION is
      when O_SELEC           => null;
      when O_SELECT_DISTINCT => PRINT ( "UNIQUE ");
      when others            => raise INTERNAL_ERROR;
    end case;
    SHOW_SELECT_LIST ( CLAUSE );
    PRINT_LINE;
    SHOW_TABLE_EXPRESSION ( CLAUSE.ACROSS );
    INDENT := INDENT - 7;
    if INDENT >= 0 then
```

```
    PRINT ( " )" );
    SET_INDENT ( INDENT );
  end if;
end SHOW_QUERY_SPECIFICATION;

-- 5.25.2 <select list>

procedure SHOW_SELECT_LIST ( S : in SQL_OBJECT ) is
begin
  case S.KIND is
    when NAME | VALUE =>
      SHOW_VALUE_EXPRESSION ( S );
    when OPERATION =>
      case S.OPERATION is
        when O_STAR =>
          PRINT ( "*" );
        when O_AMPERSAND =>
          SHOW_SELECT_LIST ( S.OPERANDS );
          PRINT ( ", " );
          SHOW_SELECT_LIST ( S.OPERANDS.ACROSS );
        when others =>
          SHOW_VALUE_EXPRESSION ( S );
      end case;
  end case;
end SHOW_SELECT_LIST;

-- 8.3.5 <order by clause>

procedure SHOW_ORDER_BY_CLAUSE ( S : in SQL_OBJECT ) is
begin
  case S.KIND is
    when NAME | VALUE =>
      SHOW_VALUE_EXPRESSION ( S );
    when OPERATION =>
      case S.OPERATION is
        when O_AMPERSAND =>
          SHOW_ORDER_BY_CLAUSE ( S.OPERANDS );
          PRINT ( ", " );
          SHOW_ORDER_BY_CLAUSE ( S.OPERANDS.ACROSS );
        when O_ASC =>
          SHOW_VALUE_EXPRESSION ( S.OPERANDS );
        when O_DESC =>
          SHOW_VALUE_EXPRESSION ( S.OPERANDS );
          PRINT ( " DESC" );
        when others =>
          raise INTERNAL_ERROR;
      end case;
  end case;
end SHOW_ORDER_BY_CLAUSE;

-- 8.7.3 <insert value list>
```

```
  procedure SHOW_INSERT_VALUE_LIST ( S : in SQL_OBJECT ) is
  begin
    case S.KIND is
      when VALUE =>
        SHOW_VALUE_SPECIFICATION ( S );
      when OPERATION =>
        case S.OPERATION is
          when O_AND =>
            SHOW_INSERT_VALUE_LIST ( S.OPERANDS );
            PRINT ( ", " );
          when O_LE =>
            null;
          when others =>
            raise INTERNAL_ERROR;
        end case;
        SHOW_INSERT_VALUE_LIST ( S.OPERANDS.ACROSS );
      when others =>
        raise INTERNAL_ERROR;
    end case;
  end SHOW_INSERT_VALUE_LIST;

  -- 8.11.2 <set clause>

  procedure SHOW_SET_CLAUSES ( S : in SQL_OBJECT ) is
  begin
    case S.OPERATION is -- must be operation
      when O_AND =>
        SHOW_SET_CLAUSES ( S.OPERANDS ); PRINT ( "," ); PRINT_LINE;
        SHOW_SET_CLAUSES ( S.OPERANDS.ACROSS );
      when O_LE =>
        PRINT ( S.OPERANDS.NAME.all & " = " );
        SHOW_VALUE_EXPRESSION ( S.OPERANDS.ACROSS );
      when others =>
        raise INTERNAL_ERROR;
    end case;
  end SHOW_SET_CLAUSES;

-- routine to show a cursor

  procedure SHOW_CURSOR
            ( CURSOR  : in CURSOR_NAME ;
              MESSAGE : in STANDARD.STRING) is
  begin
    BLANK_LINE; SET_INDENT ( 0 ); PRINT ( MESSAGE ); PRINT_LINE;
    INDENT := -7;
    SHOW_QUERY_SPECIFICATION ( SQL_OBJECT (
      CURSOR.CURSOR_OBJECT.OPERANDS ) );
    if CURSOR.CURSOR_OBJECT.OPERANDS.ACROSS /= null then
      PRINT_LINE;
      PRINT ( "ORDER  BY " );
      SHOW_ORDER_BY_CLAUSE ( CURSOR.CURSOR_OBJECT.OPERANDS.ACROSS );
```

```
    end if;
    --PRINT ( " ;");   -- (" /");
    PRINT_LINE;
--  exception
  --  when others => raise INTERNAL_ERROR;
  end SHOW_CURSOR;

-- close routine


  procedure CLOSE   -- x
          ( CURSOR : in out CURSOR_NAME ) is
  begin
-- DEBUG print it out
    if WANNA_DEBUG then
      SHOW_CURSOR ( CURSOR, "Cursor closed for:");
    end if;
    RDBMS_CLOSE_CURSOR
      (CURSOR.CURSOR_RDBMS.CURSOR_DATA.CURSOR_AREA);
  end CLOSE;

-- declare cursor routines

  procedure DECLAR  -- x
            ( CURSOR     : in out CURSOR_NAME;
              CURSOR_FOR : in     SQL_OBJECT;
              ORDER_BY   : in     SQL_OBJECT := NULL_SQL_OBJECT ) is
  begin
    if CURSOR = null then
      CURSOR := new CURSOR_NAME_RECORD;
    end if;
    CURSOR.CURSOR_OBJECT := new
    SQL_OBJECT_RECORD' ( OPERATION , null , O_DECLAR , CURSOR_FOR );
    CURSOR_FOR.ACROSS := ORDER_BY;
-- DEBUG print it out
    if WANNA_DEBUG then
      SHOW_CURSOR ( CURSOR , "Cursor declared for:");
    end if;
  end DECLAR;

  procedure DECLAR  -- x
            ( CURSOR     : in out CURSOR_NAME;
              CURSOR_FOR : in     SQL_OBJECT;
              ORDER_BY   : in     DATABASE.COLUMN_NUMBER ) is
  begin
    if CURSOR = null then
      CURSOR := new CURSOR_NAME_RECORD;
    end if;
    CURSOR.CURSOR_OBJECT := new
     SQL_OBJECT_RECORD' ( OPERATION , null , O_DECLAR , CURSOR_FOR );
    CURSOR_FOR.ACROSS :=
```

```
      new SQL_OBJECT_RECORD'
          ( VALUE , null , ( INTEGER , DATABASE.INT ( ORDER_BY ) ) );
-- DEBUG print it out
    if WANNA_DEBUG then
      SHOW_CURSOR ( CURSOR , "Cursor declared for:");
    end if;
  end DECLAR;

-- delete routines


  procedure DELETE_FROM_SUB
            ( TABLE : in TABLE_NAME;
              WHERE : in SQL_OBJECT := NULL_SQL_OBJECT ) is
  begin
    SET_INDENT ( 0 ); PRINT ( "DELETE " & TABLE.NAME.all );
    if WHERE /= null then
      INDENT := 0; PRINT_LINE; PRINT ( "WHERE  " );
      SHOW_SEARCH_CONDITION ( WHERE );
    end if;
    --PRINT ( " ;" ); -- (" /")
    PRINT_LINE;
  end DELETE_FROM_SUB;

  procedure DELETE_FROM  -- x
            ( TABLE : in TABLE_NAME;
              WHERE : in SQL_OBJECT := NULL_SQL_OBJECT ) is
  begin
-- DEBUG print it out
    if WANNA_DEBUG then
      BLANK_LINE; SET_INDENT ( 0 ); PRINT ("Delete from performed on:");
      PRINT_LINE; DELETE_FROM_SUB (TABLE, WHERE);
    end if;
-- now print to buffer
    SET_BUFFER (OPERATION_CURSOR.CURSOR_RDBMS.CURSOR_QUERY_DATA);
    OPERATION_CURSOR.CURSOR_RDBMS.CURSOR_QUERY_DATA.QUERY_LENGTH := 0;
    DELETE_FROM_SUB (TABLE, WHERE);
    UNSET_BUFFER;
    RDBMS_OPEN_CURSOR (OPERATION_CURSOR.CURSOR_RDBMS.CURSOR_DATA.
     CURSOR_AREA);
    RDBMS_QUERY (OPERATION_CURSOR.CURSOR_RDBMS.CURSOR_QUERY_DATA.
     QUERY_BUFFER, OPERATION_CURSOR.CURSOR_RDBMS.
      CURSOR_QUERY_DATA.QUERY_LENGTH, OPERATION_CURSOR.
       CURSOR_RDBMS.CURSOR_DATA.CURSOR_AREA);
    RDBMS_CLOSE_CURSOR (OPERATION_CURSOR.CURSOR_RDBMS.
     CURSOR_DATA.CURSOR_AREA);
  end DELETE_FROM;

-- fetch and into routines

  procedure FETCH -- x
```

```
          ( CURSOR : in out CURSOR_NAME ) is
  begin
    FETCH_CURSOR := CURSOR;
    DOING_A_SELECT := FALSE;
-- DEBUG print it out
    if WANNA_DEBUG then
      SHOW_CURSOR ( CURSOR , "Fetch performed on:" );
    end if;
    RDBMS_FETCH (CURSOR.CURSOR_RDBMS);
    CURSOR.CURSOR_RDBMS.CURSOR_COLUMN_NUMBER := 1;
    CURSOR.CURSOR_RDBMS.CURSOR_ROW_NUMBER :=
                CURSOR.CURSOR_RDBMS.CURSOR_ROW_NUMBER + 1;
  end FETCH;

  procedure INTEGER_AND_ENUMERATION_INTO  -- x
           (VAR : out USER_TYPE) is

    TMP : STANDARD.INTEGER := 0;

  begin
-- DEBUG print it out
    if WANNA_DEBUG then
      PRINT ( "INTO with integer or enumeration argument" ); PRINT_LINE;
      VAR := USER_TYPE'FIRST; -- ***** FOR TEST PURPOSES
    end if;
    RDBMS_COLUMN_REAPER (FETCH_CURSOR.CURSOR_RDBMS.
     CURSOR_COLUMN_NUMBER, TMP,
       FETCH_CURSOR.CURSOR_RDBMS.CURSOR_RETRIEVAL_LEN,
       FETCH_CURSOR.CURSOR_RDBMS.CURSOR_RETRIEVAL_BUF);
    VAR := USER_TYPE'VAL (TMP);
    FETCH_CURSOR.CURSOR_RDBMS.CURSOR_COLUMN_NUMBER :=
             FETCH_CURSOR.CURSOR_RDBMS.CURSOR_COLUMN_NUMBER + 1;
    if DOING_A_SELECT and
       FETCH_CURSOR.CURSOR_RDBMS.CURSOR_COLUMN_NUMBER >
                FETCH_CURSOR.CURSOR_RDBMS.CURSOR_MAX_COLUMN then
      CLOSE (FETCH_CURSOR);
      DOING_A_SELECT := FALSE;
    end if;
  end INTEGER_AND_ENUMERATION_INTO;

  procedure FLOAT_INTO  -- x
           (VAR : out USER_TYPE) is

    RESULT : STANDARD.FLOAT;

  begin
-- DEBUG print it out
    if WANNA_DEBUG then
      PRINT ( "INTO with float argument" ); PRINT_LINE;
      VAR := USER_TYPE'SMALL; -- FOR TEST PURPOSES
    end if;
```

```
    RDBMS_COLUMN_REAPER (FETCH_CURSOR.CURSOR_RDBMS.
      CURSOR_COLUMN_NUMBER,
                          RESULT,
                          FETCH_CURSOR.CURSOR_RDBMS.CURSOR_RETRIEVAL_LEN,
                          FETCH_CURSOR.CURSOR_RDBMS.CURSOR_RETRIEVAL_BUF);
    FETCH_CURSOR.CURSOR_RDBMS.CURSOR_COLUMN_NUMBER :=
                      FETCH_CURSOR.CURSOR_RDBMS.CURSOR_COLUMN_NUMBER + 1;
    VAR := USER_TYPE (RESULT);
    if DOING_A_SELECT and
       FETCH_CURSOR.CURSOR_RDBMS.CURSOR_COLUMN_NUMBER >
            FETCH_CURSOR.CURSOR_RDBMS.CURSOR_MAX_COLUMN then
      CLOSE (FETCH_CURSOR);
      DOING_A_SELECT := FALSE;
    end if;
  end FLOAT_INTO;

  procedure UNCONSTRAINED_STRING_INTO  -- x
           (VAR  : out USER_TYPE;
            LAST : out INDEX_TYPE) is

    V             : INDEX_TYPE := VAR'FIRST;
    RESULT        : STANDARD.STRING (1..100);
    RESULT_LENGTH : STANDARD.INTEGER;

  begin
-- DEBUG print it out
    if WANNA_DEBUG then
      PRINT ( "INTO with unconstrained string argument" ); PRINT_LINE;
      LAST := INDEX_TYPE'FIRST; -- FOR TEST PURPOSES
    end if;
    RDBMS_COLUMN_REAPER (FETCH_CURSOR.CURSOR_RDBMS.
       CURSOR_COLUMN_NUMBER,
                          RESULT, RESULT_LENGTH,
                          FETCH_CURSOR.CURSOR_RDBMS.CURSOR_RETRIEVAL_LEN,
                          FETCH_CURSOR.CURSOR_RDBMS.CURSOR_RETRIEVAL_BUF);
    FETCH_CURSOR.CURSOR_RDBMS.CURSOR_COLUMN_NUMBER :=
              FETCH_CURSOR.CURSOR_RDBMS.CURSOR_COLUMN_NUMBER + 1;
    LAST := INDEX_TYPE (RESULT_LENGTH) + VAR'FIRST - 1;
    for I in 1..RESULT_LENGTH loop
      VAR (V) := CONVERT_CHARACTER_TO_COMPONENT (RESULT (I));
      if V < INDEX_TYPE'LAST then
        V := V + 1;
      end if;
    end loop;
    if DOING_A_SELECT and
       FETCH_CURSOR.CURSOR_RDBMS.CURSOR_COLUMN_NUMBER >
                    FETCH_CURSOR.CURSOR_RDBMS.CURSOR_MAX_COLUMN then
      CLOSE (FETCH_CURSOR);
      DOING_A_SELECT := FALSE;
    end if;
  end UNCONSTRAINED_STRING_INTO;
```

```
  procedure CONSTRAINED_STRING_INTO  -- x
           (VAR  : out USER_TYPE;
            LAST : out INDEX_TYPE) is

    V  : INDEX_TYPE := VAR'FIRST;
    RESULT        : STANDARD.STRING (1..100);
    RESULT_LENGTH : STANDARD.INTEGER;

  begin
-- DEBUG print it out
    if WANNA_DEBUG then
      PRINT ( "INTO with constrained string argument" ); PRINT_LINE;
      LAST := INDEX_TYPE'FIRST; -- FOR TEST PURPOSES
    end if;
    RDBMS_COLUMN_REAPER (FETCH_CURSOR.CURSOR_RDBMS.
     CURSOR_COLUMN_NUMBER,
                         RESULT, RESULT_LENGTH,
                         FETCH_CURSOR.CURSOR_RDBMS.CURSOR_RETRIEVAL_LEN,
                         FETCH_CURSOR.CURSOR_RDBMS.CURSOR_RETRIEVAL_BUF);
    FETCH_CURSOR.CURSOR_RDBMS.CURSOR_COLUMN_NUMBER :=
                  FETCH_CURSOR.CURSOR_RDBMS.CURSOR_COLUMN_NUMBER + 1;
    LAST := INDEX_TYPE (RESULT_LENGTH) + VAR'FIRST - 1;
    for I in 1..RESULT_LENGTH loop
      VAR (V) := CONVERT_CHARACTER_TO_COMPONENT (RESULT (I));
      if V < INDEX_TYPE'LAST then
        V := V + 1;
      end if;
    end loop;
    if DOING_A_SELECT and
       FETCH_CURSOR.CURSOR_RDBMS.CURSOR_COLUMN_NUMBER >
                FETCH_CURSOR.CURSOR_RDBMS.CURSOR_MAX_COLUMN then
      CLOSE (FETCH_CURSOR);
      DOING_A_SELECT := FALSE;
    end if;
  end CONSTRAINED_STRING_INTO;

-- insert into routines

  procedure INSERT_INTO_SUB
            ( TABLE : in TABLE_NAME;
              WHAT  : in INSERT_ITEM ) is
  begin
    SET_INDENT ( 0 ); PRINT ( "INSERT INTO " );
    if TABLE.KIND = NAME then
      PRINT ( TABLE.NAME.all );
    else -- must be O_TABLE_COLUMN_LIST
      PRINT ( TABLE.OPERANDS.NAME.all );
      PRINT ( "( " );
      SHOW_SELECT_LIST ( TABLE.OPERANDS.ACROSS );
      PRINT ( " )" );
    end if;
```

```
    PRINT_LINE;
    case WHAT.OPERATION is -- must be an operation
      when O_SELEC | O_SELECT_DISTINCT =>
        INDENT := -7; SHOW_QUERY_SPECIFICATION ( SQL_OBJECT ( WHAT ) );
      when O_LE | O_AND =>
        PRINT ( "VALUES ( " ); --( "< " );
        SHOW_INSERT_VALUE_LIST ( SQL_OBJECT ( WHAT ) );
        PRINT ( " )" );  --( " >" );
      when others =>
        raise INTERNAL_ERROR;
    end case;
    --PRINT ( " ;" ); -- (" /")
    PRINT_LINE;
  end INSERT_INTO_SUB;

  procedure INSERT_INTO  -- x
            ( TABLE : in TABLE_NAME;
              WHAT  : in INSERT_ITEM ) is
  begin
-- DEBUG print it out
    if WANNA_DEBUG then
      BLANK_LINE; SET_INDENT ( 0 ); PRINT ("Insert into performed on:");
      PRINT_LINE; INSERT_INTO_SUB (TABLE, WHAT);
    end if;
-- now for real
    SET_BUFFER (OPERATION_CURSOR.CURSOR_RDBMS.CURSOR_QUERY_DATA);
    OPERATION_CURSOR.CURSOR_RDBMS.CURSOR_QUERY_DATA.QUERY_LENGTH := 0;
    INSERT_INTO_SUB (TABLE, WHAT);
    UNSET_BUFFER;
    RDBMS_OPEN_CURSOR (OPERATION_CURSOR.CURSOR_RDBMS.
     CURSOR_DATA.CURSOR_AREA);
    RDBMS_QUERY (OPERATION_CURSOR.CURSOR_RDBMS.
     CURSOR_QUERY_DATA.QUERY_BUFFER,
     OPERATION_CURSOR.CURSOR_RDBMS.CURSOR_QUERY_DATA.QUERY_LENGTH,
     OPERATION_CURSOR.CURSOR_RDBMS.CURSOR_DATA.CURSOR_AREA);
    RDBMS_CLOSE_CURSOR (OPERATION_CURSOR.CURSOR_RDBMS.
     CURSOR_DATA.CURSOR_AREA);
  end INSERT_INTO;

  function VALUES return INSERT_ITEM is
  begin
    return new SQL_OBJECT_RECORD' ( OPERATION , null , O_VALUES , null );
  end VALUES;

-- open routine

  procedure OPEN   -- x
          ( CURSOR : in out CURSOR_NAME ) is

    X1 : STANDARD.INTEGER := 0;
    X2 : STANDARD.INTEGER := 0;
```

```
  begin
    if CURSOR.CURSOR_RDBMS.CURSOR_DATA = null then
      CURSOR.CURSOR_RDBMS.CURSOR_DATA := new CURSOR_DATA_TYPE;
    end if;
    CURSOR.CURSOR_RDBMS.CURSOR_DATA.ROWID := (1..18 => ' ');
    CURSOR.CURSOR_RDBMS.CURSOR_DATA.CURSOR_AREA.RETURN_CODE := 0;
    CURSOR.CURSOR_RDBMS.CURSOR_DATA.CURSOR_AREA.FILLER_DATA_1 :=
     (0, 0, 0, 0, 0);
    CURSOR.CURSOR_RDBMS.CURSOR_DATA.CURSOR_AREA.V4_ERROR_CODE := 0 ;
    CURSOR.CURSOR_RDBMS.CURSOR_DATA.CURSOR_AREA.FILLER_DATA_2 :=
                        (0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
    if CURSOR.CURSOR_RDBMS.CURSOR_QUERY_DATA = null then
      CURSOR.CURSOR_RDBMS.CURSOR_QUERY_DATA :=
                  new ORACLE_CURSOR_DEFINITIONS.QUERY_BUFFER_RECORD;
      CURSOR.CURSOR_RDBMS.CURSOR_QUERY_DATA.QUERY_BUFFER :=
                  new ORACLE_CURSOR_DEFINITIONS.BUFFER_TYPE;
    end if;
    if CURSOR.CURSOR_RDBMS.CURSOR_QUERY_DATA.QUERY_BUFFER = null then
      CURSOR.CURSOR_RDBMS.CURSOR_QUERY_DATA.QUERY_BUFFER :=
                  new ORACLE_CURSOR_DEFINITIONS.BUFFER_TYPE;
    end if;
    CURSOR.CURSOR_RDBMS.CURSOR_QUERY_DATA.QUERY_LENGTH := 0;
    CURSOR.CURSOR_RDBMS.CURSOR_COLUMN_NUMBER := 0;
    CURSOR.CURSOR_RDBMS.CURSOR_MAX_COLUMN := 0;
    CURSOR.CURSOR_RDBMS.CURSOR_ROW_NUMBER := 0;
    CURSOR.CURSOR_RDBMS.CURSOR_FIRST_FETCH := TRUE;
    CURSOR.CURSOR_RDBMS.CURSOR_RETRIEVAL_LEN := 0;
    CURSOR.CURSOR_RDBMS.CURSOR_RETRIEVAL_BUF := (others => ' ');
-- DEBUG print it out
    if WANNA_DEBUG then
      SHOW_CURSOR ( CURSOR , "Cursor opened for:");
    end if;
-- now to the buffer
    SET_BUFFER (CURSOR.CURSOR_RDBMS.CURSOR_QUERY_DATA);
    CURSOR.CURSOR_RDBMS.CURSOR_QUERY_DATA.QUERY_LENGTH := 0;
    X1 := CURSOR.CURSOR_RDBMS.CURSOR_QUERY_DATA.QUERY_BUFFER'FIRST;
    X2 := CURSOR.CURSOR_RDBMS.CURSOR_QUERY_DATA.QUERY_BUFFER'LAST;
    SHOW_CURSOR ( CURSOR , "");
    UNSET_BUFFER;
    RDBMS_OPEN_CURSOR
      (CURSOR.CURSOR_RDBMS.CURSOR_DATA.CURSOR_AREA);
    RDBMS_QUERY (CURSOR.CURSOR_RDBMS.CURSOR_QUERY_DATA.QUERY_BUFFER,
                 CURSOR.CURSOR_RDBMS.CURSOR_QUERY_DATA.QUERY_LENGTH,
                 CURSOR.CURSOR_RDBMS.CURSOR_DATA.CURSOR_AREA);
    CURSOR.CURSOR_RDBMS.CURSOR_FIRST_FETCH := TRUE;
  end OPEN;

-- select statement routines

  procedure SHOW_SELECT ( S : in SQL_OBJECT ) is
```

```
  begin
    INDENT := -7;
    SHOW_QUERY_SPECIFICATION ( S );
    --PRINT ( " ;" ); -- (" /")
    PRINT_LINE;
  exception
    when others => raise INTERNAL_ERROR;
  end SHOW_SELECT;

  procedure SELECT_LIST_SELECT  -- x
            ( WHAT     : in WHAT_TYPE;
              FROM     : in TABLE_LIST;
              WHERE    : in SQL_OBJECT := NULL_SQL_OBJECT;
              GROUP_BY : in SQL_OBJECT := NULL_SQL_OBJECT;
              HAVING   : in SQL_OBJECT := NULL_SQL_OBJECT ) is
    TMP           : SQL_OBJECT;

  begin
    TMP :=
    ( BUILD_SELECT
      ( SELECT_TYPE,
        L_CONVERT ( WHAT ) , FROM , WHERE , GROUP_BY , HAVING ) );
-- DEBUG print it out
    if WANNA_DEBUG then
      BLANK_LINE; SET_INDENT ( 0 ); PRINT ("Select performed on:");
      PRINT_LINE; SHOW_SELECT (TMP);
    end if;
-- open will print to output for debug and to the query too
    DECLAR (OPERATION_CURSOR, TMP);
    OPEN (OPERATION_CURSOR);
    if OPERATION_CURSOR.CURSOR_RDBMS.CURSOR_DATA.CURSOR_AREA.
     FILLER_DATA_1 (2) > 1 then
      raise UNIQUE_ERROR;
    end if;
    FETCH (OPERATION_CURSOR);
    DOING_A_SELECT := TRUE;
  end SELECT_LIST_SELECT;

  procedure STAR_SELECT  -- x
            ( FROM     : in TABLE_LIST;
              WHERE    : in SQL_OBJECT := NULL_SQL_OBJECT;
              GROUP_BY : in SQL_OBJECT := NULL_SQL_OBJECT;
              HAVING   : in SQL_OBJECT := NULL_SQL_OBJECT ) is
    TMP           : SQL_OBJECT;

  begin
    TMP :=
    ( BUILD_SELECT
      ( SELECT_TYPE,
        new SQL_OBJECT_RECORD' ( OPERATION , null , O_STAR , null ),
        FROM , WHERE , GROUP_BY , HAVING ) );
```

```
-- DEBUG print it out
    if WANNA_DEBUG then
      BLANK_LINE; SET_INDENT ( 0 ); PRINT ("Select performed on:");
      PRINT_LINE; SHOW_SELECT (TMP);
    end if;
-- open will print for debug and to the buffer too
    DECLAR (OPERATION_CURSOR, TMP);
    OPEN (OPERATION_CURSOR);
    if OPERATION_CURSOR.CURSOR_RDBMS.CURSOR_DATA.CURSOR_AREA.
     FILLER_DATA_1 (2) > 1 then
      raise UNIQUE_ERROR;
    end if;
    FETCH (OPERATION_CURSOR);
    DOING_A_SELECT := TRUE;
  end STAR_SELECT;

-- update routines

  procedure UPDATE_SUB
            ( TABLE : in TABLE_NAME;
              SET   : in SQL_OBJECT;
              WHERE : in SQL_OBJECT := NULL_SQL_OBJECT ) is
  begin
    SET_INDENT ( 0 ); PRINT ( "UPDATE " & TABLE.NAME.all );
    PRINT_LINE; PRINT ( "SET " ); SET_INDENT ( 4 ); SHOW_SET_CLAUSES ( SET );
    if WHERE /= null then
      INDENT := 0; SET_INDENT ( 0 ); PRINT_LINE; PRINT ( "WHERE  ");
      SHOW_SEARCH_CONDITION ( WHERE );
    end if;
    --PRINT ( " ;" ); -- (" /")
    PRINT_LINE;
  end UPDATE_SUB;

  procedure UPDATE  -- x
            ( TABLE : in TABLE_NAME;
              SET   : in SQL_OBJECT;
              WHERE : in SQL_OBJECT := NULL_SQL_OBJECT ) is
  begin
-- DEBUG print it out
    if WANNA_DEBUG then
      BLANK_LINE; SET_INDENT ( 0 ); PRINT ("Update performed on:");
      PRINT_LINE; UPDATE_SUB (TABLE, SET, WHERE);
    end if;
-- now into the buffer
    SET_BUFFER (OPERATION_CURSOR.CURSOR_RDBMS.CURSOR_QUERY_DATA);
    OPERATION_CURSOR.CURSOR_RDBMS.CURSOR_QUERY_DATA.QUERY_LENGTH := 0;
    UPDATE_SUB (TABLE, SET, WHERE);
    UNSET_BUFFER;
    RDBMS_OPEN_CURSOR (OPERATION_CURSOR.CURSOR_RDBMS.CURSOR_DATA.
     CURSOR_AREA);
    RDBMS_QUERY (OPERATION_CURSOR.CURSOR_RDBMS.CURSOR_QUERY_DATA.
```

```
      QUERY_BUFFER,
      OPERATION_CURSOR.CURSOR_RDBMS.CURSOR_QUERY_DATA.QUERY_LENGTH,
      OPERATION_CURSOR.CURSOR_RDBMS.CURSOR_DATA.CURSOR_AREA);
     RDBMS_CLOSE_CURSOR (OPERATION_CURSOR.CURSOR_RDBMS.CURSOR_DATA.
      CURSOR_AREA);
  end UPDATE;

  procedure OPEN_DATABASE
            (DATABASE_NAME : in STANDARD.STRING;
             PASSWORD      : in STANDARD.STRING) is

    BUF       : ORACLE_CURSOR_DEFINITIONS.BUFFER_ACCESS_TYPE := null;
    BUF_LEN   : STANDARD.INTEGER := 0;
    BUFX      : ORACLE_CURSOR_DEFINITIONS.BUFFER_ACCESS_TYPE := null;
    BUFX_LEN  : STANDARD.INTEGER := 0;

  begin
    if BUF = null then
      BUF := new BUFFER_TYPE;
    end if;
    if BUFX = null then
      BUFX := new BUFFER_TYPE;
    end if;
--    BUF_LEN := DATABASE_NAME'LENGTH + 1 + PASSWORD'LENGTH;
--    BUF (1..BUF_LEN) := DATABASE_NAME & "/" & PASSWORD;
    BUF_LEN := DATABASE_NAME'LENGTH;
    BUF (1..BUF_LEN) := DATABASE_NAME;
    BUFX_LEN := PASSWORD'LENGTH;
    BUFX (1..BUFX_LEN) := PASSWORD;
-- DEBUG print it out
    if WANNA_DEBUG then
      BLANK_LINE; SET_INDENT ( 0 ); PRINT ( "OPEN DATABASE for " &
        BUF (1..BUF_LEN) & " " & BUFX (1..BUFX_LEN)); PRINT_LINE;
    end if;
    RDBMS_OPEN_DATABASE (BUF, BUF_LEN, BUFX, BUFX_LEN);
  end OPEN_DATABASE;

  procedure EXIT_DATABASE is
  begin
-- DEBUG print it out
    if WANNA_DEBUG then
      BLANK_LINE; SET_INDENT ( 0 ); PRINT ( "CLOSE DATABASE"); PRINT_LINE;
    end if;
    RDBMS_EXIT_DATABASE;
  end EXIT_DATABASE;

begin

  OPERATION_CURSOR.CURSOR_RDBMS.CURSOR_QUERY_DATA :=
              new ORACLE_CURSOR_DEFINITIONS.QUERY_BUFFER_RECORD;
  OPERATION_CURSOR.CURSOR_RDBMS.CURSOR_QUERY_DATA.QUERY_BUFFER :=
```

```
            new ORACLE_CURSOR_DEFINITIONS.BUFFER_TYPE;
  OPERATION_CURSOR.CURSOR_RDBMS.CURSOR_DATA := new CURSOR_DATA_TYPE;

end ADA_SQL_FUNCTIONS;
```

3.10.11 package CURSOR_DEFINITION.ADA

```
with ADA_SQL_FUNCTIONS;
package CURSOR_DEFINITION is
  subtype CURSOR_NAME is ADA_SQL_FUNCTIONS.CURSOR_NAME;
end CURSOR_DEFINITION;
```

3.10.12 package SCHEMA_DEFINITION.ADA

```
package SCHEMA_DEFINITION is
  type IDENTIFIER is range 1..5;
  generic
  function AUTHORIZATION_IDENTIFIER return IDENTIFIER;
end SCHEMA_DEFINITION;

package body SCHEMA_DEFINITION is
  function AUTHORIZATION_IDENTIFIER return IDENTIFIER is
  begin
    return 1;
  end AUTHORIZATION_IDENTIFIER;
end SCHEMA_DEFINITION;
```

Distribution List for IDA Memorandum Report M-459

NAME AND ADDRESS	NUMBER OF COPIES
Sponsor	
CPT Stephen Myatt WIS JPMO/DXP Room 5B19, The Pentagon Washington, D.C. 20330-6600	5
Other	
Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Mr. Bill Allen 311 Park Place Blvd. Suite 360 Clearwater, FL 34619	1
Mr. Fred Friedman P.O. Box 576 Annandale, VA 22003	1
Ms. Kerry Hilliard 7321 Franklin Road Annandale, VA 22003	1
Ms. Linn Roller General Dynamics P.O. Box 748 M-2 1786 Ft. Worth, TX, 76101	1
Mr. Eugen Vasilescu 35 Chestnut St. Malverne, Long Island, NY 11565	1

NAME AND ADDRESS	NUMBER OF COPIES
CSED Review Panel	
Dr. Dan Alpert, Director Program in Science, Technology & Society University of Illinois Room 201 912-1/2 West Illinois Street Urbana, Illinois 61801	1
Dr. Barry W. Boehm TRW Defense Systems Group MS R2-1094 One Space Park Redondo Beach, CA 90278	1
Dr. Ruth Davis The Pymatuning Group, Inc. 2000 N. 15th Street, Suite 707 Arlington, VA 22201	1
Dr. C.E. Hutchinson, Dean Thayer School of Engineering Dartmouth College Hanover, NH 03755	1
Mr. A.J. Jordano Manager, Systems & Software Engineering Headquarters Federal Systems Division 6600 Rockledge Dr. Bethesda, MD 20817	1
Mr. Robert K. Lehto Mainstay 302 Mill St. Occoquan, VA 22125	1
Dr. John M. Palms, Vice President Academic Affairs & Professor of Physics Emory University Atlanta, GA 30322	1

NAME AND ADDRESS	NUMBER OF COPIES
Mr. Oliver Selfridge 45 Percy Road Lexington, MA 02173	1
Mr. Keith Uncapher University of Southern California Olin Hall 330A University Park Los Angeles, CA 90089-1454	1
IDA	
General W.Y. Smith, HQ	1
Mr. Philip Major, HQ	1
Dr. Robert E. Roberts, HQ	1
Mr. Bill Brykczynski, CSED	5
Ms. Anne Douville, CSED	1
Ms. Audrey A. Hook, CSED	1
Dr. John F. Kramer, CSED	1
Mr. Terry Mayfield, CSED	1
Ms. Katydean Price, CSED	2
IDA Control & Distribution Vault	3