Advantage™ VISION:Interface™ for DB2® with Advantage™ VISION:Results™

Getting Started

4.0

Computer Associates®

B02143-1E

Contents

Chapter 1: Introducing VISION:Interface for DB2

Chapter 2: Usage Requirements

Chapter 3: Using the Attach Facilities

Chapter 4: Using JCL for DB2 Parameters

Chapter 5: Using SQL with VISION:Results

Chapter 6: Overviewing VISION:Interface Dynamic

Chapter 7: Overviewing VISION:Interface Static

Index

Chapter 1

Introducing VISION:Interface for DB2

This is the Getting Started Guide for Advantage VISION:Interface™ for DB2®. You can find detailed information on the product in the *VISION:Interface for DB2 with VISION:Results Reference Guide (hereafter referred to as the VISION:Interface for DB2 Reference Guide)*.

Release 4.0 of VISION:Interface for DB2 extends VISION:Results™ access capability to IBM® DB2 relational databases in the OS/390® (MVS®) environment.

You can execute a VISION:Results program with VISION:Interface for DB2 dynamically or statically with any of three DB2 attach facilities: CALL Attach, IMS Attach, and TSO Attach.

This chapter contains installing and accessing the documentation, VISION:Interface for DB2 product description, a synopsis of how VISION:Interface works, dynamic vs. static comparison, data management vs. data retrieval comparison, an overview of the enhancements, and product relationships.

Terminology

- Text, which is specific *VISION:Interface for DB2 Dynamic*, will be *italicized*, whenever it appears in chapters that apply to both *VISION:Interface for DB2 Dynamic* and VISION:Interface for DB2 Static.
- *VISION:Interface for DB2 Dynamic is the same as VISION:Interface dynamic.*
- *The IBM DB2 dynamic process may be referred to as dynamic DB2 or dynamic SQL.*
- VISION:Interface for DB2 Static is the same as VISION:Interface static.
- The IBM DB2 static process maybe referred to as static DB2 or static SQL.
- In the text, previous release refers to VISION:Interface for DB2 Release 3.5.

Deliverables, Requirements, and Documentation

Before you install the Release 4.0 of VISION:Interface for DB2 software, read this section.

Compact Disc Contents

The compact disc contains:

- VISION:Interface for DB2 with VISION:Results Reference Guide -IDREF040.PDF.
- VISION:Interface for DB2 with VISION:Results Getting Started Guide - IDGTS040.PDF
- Adobe® Acrobat® Reader software and Help

System Requirements and Installation Instructions

A printed version of the *VISION:Interface for DB2 Reference Guide* accompanies the two installation cartridge tapes. VISION:Interface dynamic and VISION:Interfacestatic are separate installation processes.

For complete information about VISION:Interface system requirements and installation instructions, see the *VISION:Interface for DB2 Reference Guide*.

About the Online Documentation

The CD-ROM contains the documentation for VISION:Interface. The documents, called books, are in Adobe Acrobat Portable Document Format (PDF) and are designed for you to read online using the Acrobat Reader.

Each online document contains a table of contents, index, and cross-references.

Installing Online Books and the Acrobat Reader

You can install the online documentation on Windows® on your local hard drive or on a network server. Alternately, you can access the documentation directly from the CD-ROM.

If you do not have Acrobat Reader installed, you can install it from the CD-ROM.

To install the online documentation, the Acrobat Reader, or both:

1. Close all application programs.
2. Insert the CD-ROM into the CD-ROM drive.
3. Click the Start menu and select Run.
4. In the Run dialog box, type: D:\Books\Setup.exe, where D:\ is the CD-ROM drive.

   To complete the Run dialog box and continue, click OK.

5. Follow the instructions. Computer Associates® recommends that you install the online documentation in the default directory (C:\ProgramFiles\Computer Associates\VISION_Interface DB2 4.0\Books\) or a directory of your choice (for example, C:\VISION_Interface DB2 4.0\Books\).

Viewing Online Books

Regardless of the location of the online documentation (on a local drive, a network server, or CD-ROM), you can view the online documentation using the following methods:

- Click the Start menu, point to Programs, point to VISION_Interface DB2 4.0, and click the document title.
- In Windows Explorer, point to the directory on the hard drive where you installed the online documentation. Double-click the PDF file name.
- In Windows Explorer, point to the Books directory on the CD-ROM drive and double-click the PDF file name.

Using Adobe Acrobat Reader

Use Acrobat Reader to view the online books, adjust the size of the page, perform searches, and print a range of pages. For more information, use the Acrobat Help menu.

VISION:Interface for DB2 Product Description

In addition to extending the reporting and data manipulation capabilities of VISION:Results to DB2 databases and associated columns, indexes, rows, tables, and views, VISION:Interface for DB2:

- Provides the ability to merge data from other sources (such as sequential files or VSAM files), with DB2 data using normal VISION:Results.
- Provides the ability to merge data from IMS™/DB databases with DB2 data using VISION:Interface for IMS DL/I with VISION:Results.
- Supports all standard IBM SQL (Structured Query Language) statements for data retrieval and data manipulation of DB2 databases. Use of SQL statements within VISION:Results means that much of the work involved in selecting, merging, sorting, retrieving, inserting, updating, and deleting data can be transferred from VISION:Results to SQL, while preserving the richness of the VISION:Results language for report writing and other tasks.

- Uses embedded SQL to access and manipulate data from DB2 tables. You code the embedded SQL as you would in COBOL, PL/I, or C programs.
- Provides for use of standard and simplified SQL statements.
  - Standard SQL refers to IBM-supported SQL statements.
  - Simplified SQL refers to special SQL statements provided with VISION:Interface.

A Synopsis of How VISION:Interface Works

VISION:Interface provides working storage and a set of syntax statements (EXEC SQL and ENDEXEC) you can use with VISION:Results to code both standard DB2 SQL statements and VISION:Interface "simplified" SQL statements.

VISION:Interface requires that you code SQL statements as you would in any other programming language such as COBOL or C. The only difference in the coding of SQL statements is that the SQL statement sequence starts with an EXEC SQL and ends with an ENDEXEC.

The special VISION:Interface simplified SQL statements are shortcuts where you use one statement rather than the standard cursor-handling SQL statements.

Dynamic vs. Static

When you run a VISION:Results program with VISION:Interface to access DB2, you have a choice on whether to run dynamic SQL or static SQL.

Note: *Text which is specific to the dynamic process is italic.*

Dynamic

Dynamic SQL prepares and executes SQL statements in a VISION:Results program while the program is running. You can change the SQL statements each time you run the application program. VISION:Interface creates dynamic SQL in the following manner:

- *VISION:Interface captures the program SQL statements into its own working storage, and then passes that working storage to its own dynamic SQL program, which is referred to as an access module.*
- *The access module uses dynamic SQL statements (such as the PREPARE and EXECUTE) to process the program SQL statement that are passed to it.*
- *The access module can only process either one non-select SQL statement or one cursor, which is comprised of the DECLARE CURSOR, OPEN, FETCH, and CLOSE statements. Therefore, during installation of the product you will create and link more than one access module. The number of access modules you establish should accommodate the largest DB2 programs you plan to code.*

Static

Static SQL requires you to precompile, compile, link edit, and bind (VISION:Results programs containing) SQL statements before you run the VISION:Results program. Any changes you make to the program means that you must repeat the precompile, compile, link edit, and bind process. VISION:Interface performs the following to create static SQL:

- VISION:Interface removes the program SQL statements from the VISION:Results program and moves them to a generated Assembler program.
- VISION:Interface replaces the SQL statements in the VISION:Results program with CALL statements. The CALL statements invoke the generated Assembler program to execute the specific SQL statement that was previously there.
- You will precompile, assemble, and link edit the generated Assembler program as a static DB2 program.

VISION:Interface Dynamic and VISION:Interface Static

Syntactically, VISION:Interface dynamic and VISION:Interface static are the same and produce the same results.

You will receive VISION:Interface dynamic and VISION:Interface static in separate libraries.

The following table compares dynamic and static SQL within the context of using VISION:Results with VISION:Interface for DB2.

VISION:Results with VISION:Interface for DB2 - Dynamic SQL	VISION:Results with VISION:Interface for DB2 - Static SQL
Prepares and binds each time the application executes.	Prepares and binds the database navigational path once, during the compile phase. The program uses the same path each time it executes.
Fast and easy to change.	
More efficient if DB2 database is expanding.	More efficient for stable DB2 databases.
	Applications run more securely.
Applications use more resources if the database is not expanding.	Applications use less resources.
Recommended for ad hoc reporting and file maintenance. Preferable for a frequently changed application or frequently modified database.	Preferable when accessing large production applications or databases that do not change frequently.
No STATSQL parameter on the VISION:Results OPTION statement.	Set the static option on the VISION:Results OPTION statement using the STATSQL parameter.

Another way to categorize SQL statements is by functionality. The next section describes data management (full function or FF) SQL and data retrieval (read only or RO) SQL.

Data Management vs. Data Retrieval

There are two categories of SQL statements.

- With data management or full function VISION:Interface SQL statements, VISION:Results programs can retrieve, insert, delete, update, and modify DB2 databases or tables.
- With data retrieval or read only VISION:Interface SQL statements, VISION:Results programs can retrieve data from DB2 databases or tables.

Full Access and Restricted Access

Your installation can have both the VISION:Interface data management version and the data retrieval version.

- To provide full access to all SQL statements, make the data management or full function version of VISION:Interface available.
- To restrict access to retrieval only SQL statements, make the data retrieval or read only version of VISION:Interface available.

Note: The VISION:Interface for DB2 data management or full function version includes both VISION:Interface dynamic and VISION:Interface static.

Supported SQL Statements

The following table shows standard SQL statements for the two categories of VISION:Interface. The following table also includes the VISION:Interface simplified SQL statements.

Data Management SQL Statements (full-function)	Data Retrieval SQL Statements (read-only)
ALTER TABLE ALTER INDEX	
CLOSE cursor-name	CLOSE cursor-name
COMMENT ON	
COMMIT	
CONNECT CONNECT TO :hostvar CONNECT TO location-name CONNECT RESET	CONNECT CONNECT TO :hostvar CONNECT TO location-name CONNECT RESET
CREATE TABLE CREATE INDEX CREATE VIEW CREATE SYNONYM	
CURSOR (WITH HOLD) SELECT fullselect INTO :hostvar$_1$, :hostvar$_2$,... (VISION:Interface simplified SQL statement.)	CURSOR (WITH HOLD) SELECT fullselect INTO :hostvar$_1$, :hostvar$_2$,... (VISION:Interface simplified SQL statement.)
DECLARE cursor-name CURSOR (WITH HOLD) FOR SELECT fullselect	DECLARE cursor-name CURSOR (WITH HOLD) FOR SELECT fullselect
DECLARE TABLE	DECLARE TABLE
DELETE FROM table-name DELETE FROM table-name WHERE CURRENT OF cursor-name	
DROP TABLE DROP INDEX DROP VIEW DROP SYNONYM	
EXPLAIN	
FETCH cursor-name INTO :hostvar$_1$, :hostvar$_2$,...	FETCH cursor-name INTO :hostvar$_1$, :hostvar$_2$,...
GRANT	
INSERT INTO table-name	
LABEL	
LOCK	
OPEN cursor-name	OPEN cursor-name
RELEASE	RELEASE
RENAME	

Data Management SQL Statements (full-function)	Data Retrieval SQL Statements (read-only)
REVOKE	
ROLLBACK	
SELECT fullselect INTO :hostvar$_1$, :hostvar$_2$,...	SELECT fullselect INTO :hostvar$_1$, :hostvar$_2$,...
SET CONNECTION SET CURRENT DEGREE SET CURRENT PACKAGESET SET CURRENT RULES SET CURRENT SQLID SET :hostvar = special register	SET CONNECTION SET CURRENT DEGREE SET CURRENT PACKAGESET SET CURRENT RULES SET CURRENT SQLID SET :hostvar = special register
UPDATE table-name UPDATE table-name WHERE CURRENT OF cursor-name	
WHENEVER condition CONTINUE WHENEVER condition GOTO host-label	WHENEVER condition CONTINUE WHENEVER condition GOTO host-label
WHENEVER condition STOP (VISION:Interface simplified SQL statement.)	WHENEVER condition STOP (VISION:Interface simplified SQL statement.)

Note: The following VISION:Interface simplified SQL statements are NOT supported by the IBM SQL processor:

- CURSOR (WITH HOLD) SELECT
- WHENEVER ... STOP

For information on using SQL statements, see the *VISION:Interface for DB2 Reference Guide*.

New and Enhanced VISION:Interface SQL Statements

The **new** SQL statements in Release 4.0 of VISION:Interface for DB2 are:

- DECLARE TABLE
- RELEASE
- RENAME
- SET CONNECTION
- SET CURRENT DEGREE
- SET CURRENT RULES

The **enhanced** SQL statement in Release 4.0 of VISION:Interface for DB2 is:

- CURSOR (WITH HOLD) SELECT fullselect INTO :hostvar$_1$, :hostvar$_2$,... (The WITH HOLD clause is new.)

Overviewing the Enhancements

For detailed information about VISION:Interface for DB2 Release 4.0 features, see the *VISION:Interface for DB2 Reference Guide*. The following is a synopsis:

- Single source capability enabled by specifying run-time DB2 parameters
- Supports SQL VARCHAR data type
- Supports IMS attach and provides synchronous rollbacks
- Supports CALL attach with both implicit and explicit connection
- Determines the attach facility at execution time (restore time)
- New and enhanced VISION:Interface SQL statements
- Additional access modules no longer needed for DDF support
- Data retrieval (read only) with CICS
- Improved diagnostics
- Improved installation process, model JCL, and sample programs (Custom patches eliminated)

Product Relationships

For additional information, see the *VISION:Interface for DB2 Reference Guide*.

Throughout this document and the *VISION:Interface for DB2 Reference Guide*, there are references to components that are delivered with associated products. In particular, the following components are delivered with the following products:

VISION:Results

See the *VISION:Results for OS/390 Installation Guide* and the *VISION:Results Reference Guide* for information on the:

- DYLINSTL macro: *DB2SYS parameter, DB2PLAN parameter,* STATSYS parameter, STATPLN parameter, CATSYS parameter, and CATPLAN parameter
- OPTION statement *DYNAMDB2 parameters: (DB2SYSID parameter and DB2PLANID parameter)*, SYSTEMID parameter, PLANID parameter, CATSYSID parameter, and CATPLANID parameter

For addition information on the DYLINSTL macro and OPTION statement, see the *VISION:Interface for DB2 Reference Guide*.

VISION:Interface for IMS DL/I

For information on the VISION:Interface for IMS DL/I DYLIDB load module (referred to as the INTERFACE.IMS.LOAD library within the JCL), see the *VISION:Interface for IMS DL/I Reference Manual*.

Contacting Total License Care (TLC)

TLC is available Monday-Friday 7 am - 9 pm Eastern Time in North America and 7 am - 7 pm United Kingdom time. Additionally, 24-hour callback service is available for after hours support. Contact TLC for all your licensing requirements.

Be prepared to provide your site ID for product activation.

To activate your product, use one of the following:

North America:	800-338-6720 (toll free) 631-342-5069	help@licensedesk.cai.com
Europe:	00800-1050-1050	euro.tlc@ca.com

If your company or local phone service does not provide international access, please call your local Computer Associates office and have them route you to the above number.

Australia:	1-800-224-852
New Zealand:	0-800-224-852
Asia Pacific:	800-224-852
Brazil:	55-11-5503-6100
Japan:	JPNTLC@ca.com (email is preferred)

Contacting Computer Associates

For further technical assistance with this product, please contact Computer Associates Technical Support on the Internet at esupport.ca.com. Technical support is available 24 hours a day, 7 days a week.

Chapter 2

Usage Requirements

For complete information on pre-installation activities and installation instructions, see the *VISION:Interface for DB2 Reference Guide*.

This chapter overviews system requirements, DB2 authorization, customizing the VISION:Results DYLINSTL macro, and the VISION:Interface installation process.

Verifying System Requirements

Verify that your system has the following software:

VISION:Results

VISION:Interface for DB2 Release 4.0 requires VISION:Results (Release 5.0 or higher).

VISION:Interface for IMS DL/I

To use IMS Attach, VISION:Interface for DB2 requires that VISION:Interface for IMS DL/I be installed. For implementation, see the *VISION:Interface for DB2 Reference Guide* and *VISION:Interface for IMS DL/I Reference Manual*.

Required IBM Products

- DB2 (DATABASE 2™) Version 6 and 7.x
- IMS
- DDF (Distributed Data Facility)

DB2 Authorization

For details on DB2 authorization for VISION:Interface, see the *VISION:Interface for DB2 Reference Guide*.

This section summarizes the DB2 authorization information to install and use VISION:Interface.

Obtain DB2 Authorization

Before installing and using VISION:Interface, obtain DB2 authorization from your database administrator. You need DB2 authorization to:

- Bind a plan or package used by VISION:Interface to access DB2.
- Authorize individuals for VISION:Interface use.
- Grant user access to DB2 databases.

Bind the Plan or Package

VISION:Interface uses following DB2 plans.

- *If you use explicit connection, dynamic DB2 uses the plan called DYLDB2.*
- *If you use implicit connection, dynamic DB2 uses the plan called DYLSQI00.*
- Static DB2 uses the plan called STATDB2.

These plans are linked to the VISION:Interface modules.

VISION:Interface also uses packages:

- *Dynamic uses a package called DYLSQI.*
- Static uses default packages called STATSQI, DSN8610, DYL, and SYSIBM.

For complete information, see the *VISION:Interface for DB2 Reference Guide*.

Customizing the VISION:Results DYLINSTL Macro

You will modify the VISION:Results DYLINSTL macro parameters and run DYLINSTL before installing and using the VISION:Interface for DB2. The following DYLINSTL macro parameters need to be set:

CATPLAN	DB2NULL	DB2SYS	STATPLN
CATSYS	DB2PLAN	DYLVARP	SQLIFIF
DB2ERR	DB2SNGL	STATSYS	SUPRESQ

Notes:

- For information on using the CUSTMJCL member to run the DYLINSTL macro (You will find CUSTMJCL and DYLINSTL in the VISION:Results source library), see the *VISION:Results for OS/390 Installation Guide* and *VISION:Results Reference Guide*.
- For information on modifying the DYLINSTL parameters, see the *VISION:Interface for DB2 Reference Guide*.

Overviewing the Installation

For detailed installation information and step-by-step instructions, see the *VISION:Interface for DB2 Reference Guide*.

This section overviews the VISION:Interface for DB2 installation.

- About the installation tape
- Copying the installation tape
- Creating VISION:Interface libraries
- Installing VISION:Interface
- Editing CNTL library installation members
- Running the VISION:Interface installation jobs

About the Installation Tapes

There are two installation tapes:

- The VISION:Interface for DB2 Full Function (FF) installation tape contains JCL, modules and source libraries required to run both the dynamic and static versions of the product.
- The VISION:Interface for DB2 Read Only (RO) installation tape contains JCL, modules and source libraries required for a read only subset of the dynamic version.

The Read Only installation is the same as the Full Function installation, but without:

- The static version of VISION:Interface.
- VISION:Interface dynamic sample programs and the installation control members that perform and use data management functions.

The *VISION:Interface for DB2 Reference Guide* describes the installation of VISION:Interface full function, which contains both the dynamic and static versions of our product.

Copying the Installation Tape

VISION:Interface is distributed on a cartridge tape which contains the following data sets.

DYLDB2.REL40.COPY	IEBCOPY copies the next five data sets
DYLDB2.REL40.DYNOBJ	*VISION:Interface dynamic object library*
DYLDB2.REL40.DYNLOAD	*VISION:Interface dynamic load library*
DYLDB2.REL40.STATLOAD	VISION:Interface static load library
DYLDB2.REL40.DYNCNTL	*VISION:Interface dynamic control library*
DYLDB2.REL40.STATCNTL	VISION:Interface static control library

Copying DYLDB2.REL40.COPY From Tape

1. You will enter JCL to load DYLDB2.REL40.COPY, the first data set (using IEBGENER).
2. You will execute the IEBGENER job to copy DYLDB2.REL40.COPY from tape.
3. You will customize DYLDB2.REL40.COPY for your installation.
4. You will execute the modified DYLDB2.REL40.COPY job to copy the remaining data sets on the installation tape.

Creating VISION:Interface Libraries

For complete information, see the *VISION:Interface for DB2 Reference Guide*. This section overviews the libraries.

The first data set, DYLDB2.REL40.COPY, creates the five VISION:Interface installation libraries.

- *The second data set, DYLDB2.REL40.DYNOBJ, creates INTERFACE.DYNAMIC.OBJ, the VISION:Interface dynamic object library.*
- *The third data set, DYLDB2.REL40.DYNLOAD, creates INTERFACE.DYNAMIC.LOAD, the VISION:Interface dynamic load library.*
- The fourth data set, DYLDB2.REL40.STATLOAD, creates INTERFACE.STATIC.LOAD, the VISION:Interface static load library.
- *The fifth data set, DYLDB2.REL40.DYNCNTL, creates YOUR.DYLDB2.CNTL, the VISION:Interface dynamic control library (containing VISION:Interface dynamic installation jobs, JCL models, sample programs, sample data, and miscellaneous data sets). The installation jobs are additional jobs required to install VISION:Interface dynamic.*

- The sixth data set, DYLDB2.REL40.STATCNTL, creates YOUR.STATDB2.CNTL, the VISION:Interface static control library (containing VISION:Interface static program preparation jobs, JCL models, sample programs, sample data, and miscellaneous data sets).

  There are no additional jobs required to install VISION:Interface static. Instead, you run the program preparation jobs each time you create or change a static program.

For complete information, see the *VISION:Interface for DB2 Reference Guide*.

Installing the VISION:Interface *Dynamic*

For complete information, see the VISION:Interface for DB2 Reference Guide. This section overviews the installation of VISION:Interface dynamic.

To install* VISION:Interface *for DB2 Dynamic, you will:

1. *Edit installation members of the CNTL library.*
2. *Run the installation jobs.*
3. *Verify the installation by running the sample programs. You will need to modify the JCL models in accordance with your installation standards.*

Editing CNTL Library Installation Members (Dynamic)

Before you install VISION:Interface dynamic, you will modify the following members in the YOUR.DYLDB2.CNTL library:

DB2A	*DYLSQL10*	*DYLSQL50*	*DYLSQL90*
DYIMPCON	*DYLSQL20*	*DYLSQL60*	*PACKGE00*
DYLSQI	*DYLSQL30*	*DYLSQL70*	*PLAN00*
DYLSQL00	*DYLSQL40*	*DYLSQL80*	*PLAN01*

For a description of each member and information regarding any changes, see the VISION:Interface for DB2 Reference Guide.

Running the VISION:Interface *Dynamic Installation Jobs*

For complete information, see the VISION:Interface for DB2 Reference Guide. This section is an overview.

After you modify the CNTL library installation members, you customize and run the installation jobs.

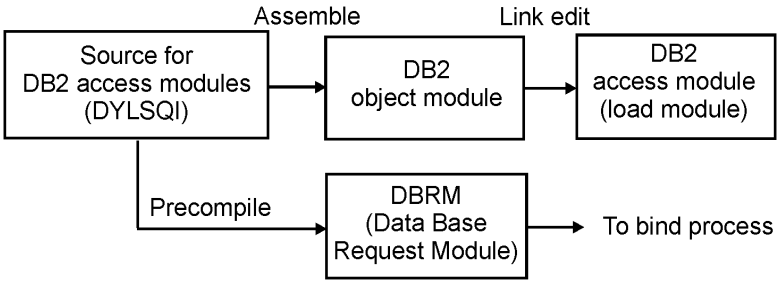
To install VISION:Interface:

1. *Modify and run CNTLPREP.*
2. *Modify and run CNTLLINK.*
3. *Modify and run CNTLLNK2.*
   - *CNTLLNK2 is optional.*
   - *CNTLLNK2 is required for using CALL Attach implicit connection.*
4. *Modify and run CNTLLNK3.*
   - *CNTLLNK3 is optional.*
   - *CNTLLNK3 is required for using VISION:Interface for IMS DL\I BMP programs with VISION:Interface for DB2.*
5. *Modify and run CNTLBIND.*

These installation jobs are instream procedures with modifiable parameters. Summary descriptions of the jobs follow.

Modify and Run CNTLPREP

Run CNTLPREP as the first job in the installation procedure. CNTLPREP creates the object code and DBRM modules necessary for CNTLBIND, CNTLLINK, CNTLLNK2 (optional), and CNTLLNK3 (optional).



CNTLPREP is a two-step process where:

1. *The DYLSQI source program, which contains VISION:Interface dynamic SQL statements, is precompiled to create the DBRM module (used as input for the bind process).*

   Note:*DBRM is the acronym for Data Base Request Module. When the installation completes there will be a DBRM for each access module.*

2. *DYLSQI is then assembled to create the object module (used as input for the link edit process).*

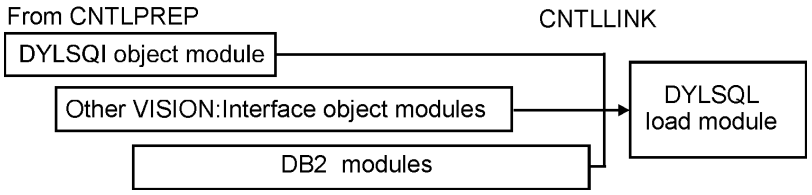
This two-step process is done repeatedly, a certain number of times (20 is the default) to create the access modules (DYLSQI00 – DYLSQInn).

- *The number of access modules, a VISION:Results program will use, depends on the number of SQL statements (a cursor with its DECLARE, OPEN, FETCH, and CLOSE statements is considered one SQL statement) that are coded in a program.*
- *Do not include DDF statements, (such as, SET, CONNECT, and RELEASE), WHENEVER statements, and DECLARE TABLE statements in the count.*
- *Assign CNTLPREP the number of access modules equivalent to the maximum number of SQL statements that can be found in a program.*

For the CNTLPREP JCL and instructions for modifying the JCL, see the VISION:Interface for DB2 Reference Guide.

Modify and Run CNTLLINK

CNTLLINK is the second job to run as part of the installation procedure. It uses the object code created from CNTLPREP as input to link with other VISION:Interface object modules and DB2 modules to create the DYLSQL load module.



For the CNTLLINK JCL and instructions for changing the JCL, see the VISION:Interface for DB2 Reference Guide.

Modify and Run CNTLLNK2 (Optional)

Run the CNTLLNK2 job if your programs use the CALL Attach implicit connection.

Once you have established this implicit connection, you can still use the explicit connection, by adding an ECONNECT DD statement in the JCL for the VISION:Results program.

The CNTLLNK2 job assembles the source member called DYIMPCON in the NEWCNTL library, and then links this member to the DYLSQL load module. Thus, alerting VISION:Interface that an implicit connection is pending, provided an ECONNECT DD statement is not present.

The CALL Attach implicit connection requires you to have:

- *A DB2 DSNEXIT library allocated to the STEPLIB of the VISION:Results program JCL. DB2 will use the DB2 subsystem found in the DSNHDECP module in the DSNEXIT library.*

- *The plan name be the same name as the first used DBRM module, which would be DYLSQI00, and if you are using COPYDB2, then the COPYDB2 plan would be DYLCAT00.*

  The CNTLBIND is set up to bind a DYLSQI00 plan, and theVISION:Results DB2INST2 source installation member is set up to bind a DYLCAT00 plan.

See the VISION:Interface for DB2 Reference Guide for information on the CNTLLNK2 JCL, instructions for changing the JCL, CALL Attach, implicit connection, and explicit connection.

Modify and Run CNTLLNK3 (Optional)

Run the CNTLLNK3 job only if your installation will be running IMS BMP (Batch Message Processing) programs with VISION:Interface for DB2.

Running IMS BMP programs requires VISION:Interface for IMS DL/I.

The CNTLLNK3 job relinks the VISION:Interface for IMS DL/I DYLIDB load module with an alias of the same name as the plan.

If you are going to use multiple plans:

- *Rerun the CNTLLNK3, but do not remove the alias entries of the previous plans that were link edited.*
- *Add the alias to the other aliases and link edit all the plans as aliases. This includes aliases of plans created for VISION:Interface for DB2 Static.*

When you run VISION:Interface BMP program JCL, the MBR parameter should be the same name as the DB2 plan that is being used.

For the CNTLLNK3 JCL and instructions for changing the JCL, see the VISION:Interface for DB2 Reference Guide.

Modify and Run CNTLBIND

CNTLBIND is the last job to run as part of the installation procedure.

- *CNTLBIND is required.*
- *CNTLBIND uses the DBRM modules created from CNTLPREP to bind these modules as packages.*
  - *There is a DBRM for each access module.*
  - *You must bind a package for each DBRM as a separate step.*

The collection ID is DYLSQI for all the packages. The plan:

- *Will be tied to the packages by means of the collection ID within the plan package list.*
- *Need only be run once for that collection ID.*
- *Will be automatically updated when new packages are created.*
- *Is also automatically updated when new packages on a remote DB2 server are created with the same collection ID. Therefore, when creating packages on a remote server for DDF (Distributed Data Facility) processing, do not run the plan steps (PLAN00 and PLAN01) within the CNTLBIND.*

There are two plan steps – PLAN00 and PLAN01.

- *PLAN00 will create the DYLDB2 plan.*
- *PLAN01 will create the DYLSQI00 plan, which is used for implicit connection, but can also be used for standard explicit connection jobs when on the DB2PARMS DD statement, DYLSQI00 is specified as the DB2 plan.*

For the CNTLBIND JCL and instructions for changing the JCL, see the VISION:Interface for DB2 Reference Guide.

Chapter

3 Using the Attach Facilities

For complete information, see the *VISION:Interface for DB2 Reference Guide*. This chapter overviews:

- Determining the attach facility at execution
- DB2 parameters with the attach facility

Note: *Text which is specific to the dynamic process is italic.*

Determining the Attach Facility at Execution

VISION:Interface for DB2 4.0 supports the following DB2 attach facilities:

- CALL Attach
- IMS Attach
- TSO Attach

VISION:Interface dynamically determines the appropriate attach facility at execution time (restore time).

The program parameters in your JCL determine the appropriate attach facility.

- If you execute VISION:Results directly as the program where the JCL step has PGM=DYL280, VISION:Interface uses CALL Attach.

  For a CALL Attach, see the information on the JCL models RUNCALL and RESTORC in the *VISION:Interface for DB2 Reference Guide*

- If the JCL step has PGM=IKJEFT01 and the SYSTSIN DD statement has a DSN command, VISION:Interface uses TSO Attach.

  For a TSO Attach, see the information on JCL models RUNTSO and RESTORT in the *VISION:Interface for DB2 Reference Guide*.

- If the JCL step contains PGM=DFSRRC00 and you are running a BMP program, VISION:Interface uses IMS Attach.

  For a BMP IMS Attach, see the information on JCL models RUNDLIB and RESTORDB in the *VISION:Interface for DB2 Reference Guide*.

- If the JCL step contains PGM=DFSRRC00 and you are running a batch program with MBR=DSNMTV01, and have a DDITV02 DD statement, VISION:Interface uses IMS Attach.

  For a batch IMS Attach, see the information on JCL models RUNDLII and RESTORDI in the *VISION:Interface for DB2 Reference Guide*.

- If the JCL step contains PGM=DFSRRC00 and you are running a batch program with MBR=DYLIDB, and have no DDITV02 DD statement, VISION:Interface uses CALL Attach.

  For a CALL Attach, see the information on JCL models RUNDLIC and RESTORDC in the *VISION:Interface for DB2 Reference Guide*.

Attach Facility Road Map

For complete information, see the *VISION:Interface for DB2 Reference Guide*.

VISION: Interface uses	Conditions	For JCL model, use
CALL Attach	If you execute VISION:Results directly as the program where the JCL step has PGM = DYL280,	*RUNCALL* *RESTORC* RESTORC
TSO Attach	If the JCL step has PGM=IKJEFT01 and the SYSTSIN DD statement has a DSN command,	*RUNTSO* *RESTORT* RESTORT
IMS Attach	If the JCL step contains PGM=DFSRRC00 and you are running a BMP program,	*RUNDLIB* *RESTORDB* RESTORDB
IMS Attach	If the JCL step contains PGM=DFSRRC00 and you are running a batch program with MBR=DSNMTV01 and have a DDITV02 DD statement,	*RUNDLII* *RESTORDI* RESTORDI
CALL Attach	If the JCL step contains PGM=DFSRRC00 and you are running a batch program with MBR=DYLIDB, and have no DDITV02 DD statement,	*RUNDLIC* *RESTORDC* RESTORDC

Notes:

- For VISION:Interface static, you no longer need to code the attach facility (CALL_ATTACH or TSO_ATTACH parameter) on the OPTION statement.
- If you code the attach facility on the OPTION statement, VISION:Interface static ignores it. The JCL determines the attach facility.

DB2 Parameters Used with the Attach Facility

The JCL models direct VISION:Interface to the attach facility to use to connect to DB2. However, when the connection is made, DB2 must know the DB2 subsystem ID and DB2 plan to execute prior to processing. The attach facilities pass this information to DB2 as parameters.

COPYDB2, which copies DB2 column definitions to use as data names and host variables in the VISION:Results program, also uses the attach facilities and must identify the DB2 subsystem and plan to use.

This section explains how:

- The different attach facilities obtain the DB2 parameters for standard VISION:Interface processing and
- COPYDB2 obtains DB2 parameters for the specified attach facility.

Each attach facility uses a different method to obtain the DB2 subsystem and plan from the user to pass to DB2.

Note: If you are going to use DDF (Distributed Data Facility) processing, then the DB2 subsystem ID you specify must be the local server. The location name specified on the DB2PARMS or COPYDB2A DD statements will direct the program to use the remote server.

DB2 Parameters for TSO Attach

TSO Attach requires values for the DB2 parameters, SYSTEM and PLAN, on the SYSTSIN DD statement, as follows:

```
DSN SYSTEM(DB2A)
  RUN PROGRAM(DYL280) PLAN(DYLDB2)
END
```

COPYDB2 also uses the SYSTSIN DD statement DB2 data parameters.

DB2 Parameters for IMS Attach

Batch Processing

IMS Attach batch programs require the DB2 parameters on the DDITV02 DD statement, as follows:

DB2A,SYS1,DSNMIN10, ,R,-, ,**DYLDB2**,DYLIDB

COPYDB2 also uses the DDITV02 DD statement DB2 data parameters.

BMP Processing

IMS Attach BMP programs require the DB2 parameters be passed in two of their IMS parameters, as follows:

SSM=**DB2A** * DB2 SUBSYSTEM TO USE

MBR=**DYLDB2** * SET TO PLAN NAME

COPYDB2 also uses these IMS parameters.

For additional information on IMS Attach, see the *VISION:Interface for DB2 Reference Guide*.

DB2 Parameters for CALL Attach

This section describes DB2 parameters for CALL Attach implicit and explicit connection.

DB2 Parameters for CALL Attach Implicit Connection

The CALL Attach implicit connection:

- Obtains the DB2 subsystem from the DSNHDECP module of the DSNEXIT library within the JCL STEPLIB.
- Assumes that the DB2 plan is the same name as the first accessed DBRM module.

If COPYDB2 has been installed to use the implicit connection, then it will also use the same DB2 subsystem and the plan with the same name as the first accessed DBRM, which is always DYLCAT00.

For additional information on implicit connection, see the *VISION:Interface for DB2 Reference Guide*.

DB2 Parameters for CALL Attach Explicit Connection

The CALL Attach explicit connection, the standard attach facility used for batch processing, does not have a fixed method of obtaining the DB2 subsystem and plan. Instead, the application determines how to gather the DB2 parameter information to pass to the CALL Attach.

VISION:Interface has a variety of ways to obtain the DB2 parameters as demonstrated in the following table:

Explicit Connection Order of Precedence			
	Dynamic	**COPYDB2**	**Static**
1st	DB2PARMS DD statement	COPYDB2A DD statement	DB2PARMS DD statement
2nd	*EXEC SQL statement*		
	SSID parameter		
	PLAN parameter		
3rd	*OPTION statement*	OPTION statement	OPTION statement
	DB2SYSID parameter	CATSYSID parameter	SYSTEMID parameter
	DB2PLANID parameter	CATPLANID parameter	PLANID parameter
4th	*DYLINSTL macro*	DYLINSTL macro	DYLINSTL macro
	DB2SYS parameter	CATSYS parameter	STATSYS parameter
	DB2PLAN parameter	CATPLAN parameter	STATPLN parameter

Note: *The OPTION statement DYNAMDB2 parameters are DB2SYSID and DB2PLANID.*

Considerations for using the DB2PARMS and COPYDB2A DD statements

- If you are going to use DDF (Distributed Data Facility) processing, then the DB2 subsystem ID you specify on the DB2PARMS or COPYDB2A DD statement must be the local server. The location name specified on the DB2PARMS or COPYDB2A DD statement will direct the program to use the remote server.
- If you are not using the DB2PARMS or COPYDB2A DD statement for the DB2 subsystem ID, then the local server ID must also be the specified DB2 subsystem ID for the *DB2SYS*, CATSYS, *DB2SYSID*, SSID, and CATSYSID parameters.

For various methods of passing DB2 parameters, see the *VISION:Interface for DB2 Reference Guide*. Computer Associates recommends using the JCL to pass the DB2 parameters, that is, using the DB2PARMS and COPYDB2A DD statements.

DB2PARMS DD Statement Parameters

All of the previous methods (DYLINSTL macro, OPTION statement, and EXEC SQL statement) determine the DB2 subsystem or plan at compile or freeze time. These previous methods do not provide the means for frozen programs to change the DB2 subsystem or plan during restore unless you use a data name for the OPTION DYNAMDB2 parameters (this method is very cumbersome to use).

The DB2PARMS DD statement method obtains the DB2 parameters at execution or restore time.

The DB2PARMS DD statement method:

- Supersedes the DYLINSTL, OPTION statement, and EXEC SQL methods, which means if they are being used, they will be ignored.
- Is the only method that is the same for the VISION:Interface dynamic and VISION:Interface static.
- Allows for single source programming where you can use the same frozen module with different DB2 subsystems and with different plans.

Code the DB2 parameters on one record in the DB2PARMS DD statement, as follows:

```
//DB2PARMS  DD   *
DB2A          DYLDB2
/*
```

The DB2 parameters are positional.

- Place the DB2 subsystem ID in position 1.
- Place the DB2 plan name in position 10.

COPYDB2A DD Statement Parameters

COPYDB2 has a COPYDB2A DD statement parameter similar to the DB2PARMS DD statement. The COPYDB2A DD statement also overrides the COPYDB2 DB2 parameters used in the DYLINSTL macro or on the OPTION statement.

The COPYDB2 is always determined at compile or freeze time. The COPYDB2A DD statement method allows the user to easily change the DB2 subsystem and plan, on the fly, without changing the source program or the installation parameters.

If you have multiple COPYDB2 statements in the program, then the second COPYDB2 statement uses a COPYDB2B DD statement. If a third COPYDB2 statement is present, then a COPYDB2C DD statement will be used, and so on. Valid values for the n in the COPYDB2n names are A through Z.

Code the DB2 parameters on one record on the COPYDB2A DD statement, as follows:

```
//COPYDB2A  DD   *
DB2A          DYLDB2
/*
```

The DB2 parameters are positional.

- Place the DB2 subsystem ID in position 1.
- Place the DB2 plan name in position 10.

Chapter

4 Using JCL for DB2 Parameters

For complete information, see the *VISION:Interface for DB2 Reference Guide*. This chapter is an overview.

You can use JCL to change DB2 parameters by using the following DD statements:

- DB2PARMS
- COPYDB2A

DB2PARMS DD Statement

Using JCL, you can change the DB2 subsystem ID, DB2 plan name, and DB2 table qualifiers (location ID, authorization ID, or both) for unqualified tables, without source programming modifications during:

- Restore of frozen programs.
- Compile and go execution of programs.

COPYDB2A DD Statement

You can also change the DB2 subsystem ID, DB2 plan name, and DB2 table qualifiers (location ID, authorization ID, or both) for COPYDB2 statements during:

- Freeze of programs.
- Compile and go execution of programs.

Note: *Text that is specific to VISION:Interface dynamic is italic.*

The Differences

The capability to dynamically change DB2 parameters differs between VISION:Interface dynamic and VISION:Interface static.

- If you are using VISION:Interface dynamic, proceed to the section Changing DB2 Specifications Using VISION:Interface Dynamic.
- If you are using VISION:Interface static, proceed to the section Changing DB2 Specifications Using VISION:Interface Static.

The capability to dynamically change DB2 parameters for multiple COPYDB2 statements require different DD statements. If you require this feature for COPYDB2, see the section Changing DB2 Specifications Using COPYDB2.

Changing DB2 Specifications Using VISION:Interface Dynamic

To use this functionality, add the DB2PARMS DD statement.

```
//DB2PARMS DD *
DB2A     DYLDB2   DSN8610             LOC_ID
```

DB2PARMS DD Statement Coding Positions (Dynamic)

The DB2PARMS DD statement consists of one record with the following four parameters:

Parameter	Size	Description	Position
DB2SSID	PIC X(8).	DB2 subsystem ID	1
Filler	PIC X.		
DB2PLAN	PIC X(8).	DB2 plan name	10
Filler	PIC X.		
DB2AUTHID	*PIC X(8).*	*DB2 table authorization ID qualifier*	*19*
Filler	PIC X(11).		
DB2LOCID	PIC X(16).	DB2 table location ID qualifier (Code only if you are using DDF (Distributed Data Facility).)	38

The parameters are optional (that is, you can specify one or all of the parameters on the record) and positional (you must place them at the position indicated).

Notes:

- *VISION:Interface dynamic requires the user to have SYSADM authority to use the authorization ID.*
- VISION:Interface static does not have this restriction.

Note: If you are going to use DDF (Distributed Data Facility) processing, then the DB2SSID is the local server ID.

DB2SSID and DB2PLAN Parameters

The DB2PARMS DD statement DB2SSID and DB2PLAN parameters depend on the connection and attach facility.

- If the program uses an *explicit* connection to the CALL Attach facility (PGM=DYL280 in the JCL), the job retrieves the DB2SSID and DB2PLAN parameters from DB2PARMS DD statement.
- If the program uses an *implicit* connection to CALL Attach:
  - The DB2PLAN will always be DYLSQI00.
  - The job retrieves DB2SSID from the DSNEXIT library concatenated to the JCL STEPLIB.

- If the program uses TSO Attach, the job:
  - Retrieves the subsystem and plan you code on the SYSTSIN statement.
  - Ignores the DB2PARMS DD statement DB2SSID and DB2PLAN parameters.
- If the program uses IMS Attach, the job ignores the DB2PARMS DD statement DB2SSID and DB2PLAN parameters and retrieves the parameters as follows:
  - If the program is an IMS batch job, IMS Attach retrieves the subsystem and plan coded on the DDITV02 statement.
  - If the program is an IMS BMP job, the DB2PLAN is assumed to be the same as the MBR parameter. You specify the DB2SSID on the SSM parameter.
- If you are going to use DDF (Distributed Data Facility) processing, then the DB2SSID is the local server ID.

Note: See Changing DB2 Specifications Using COPYDB2.

DB2AUTHID Parameter

The DB2PARMS DD statement *DB2AUTHID* parameter establishes the authorization ID qualifier that DB2 uses with unqualified DB2 tables. DB2 does not change the authorization ID for tables that have the authorization ID included as part of the table name.

Note:

- *The DB2AUTHID parameter applies to VISION:Interface dynamic only.*
- The DB2PACK parameter represents the authorization ID for VISION:Interface static. For information on DB2PACK, see Changing DB2 Specifications Using VISION:Interface Static.

Different parameters are used to represent the authorization ID because different methods are used to acquire the authorization ID.

Behind the scenes, VISION:Interface executes an SQL SET CURRENT SQLID statement to direct DB2 on the authorization ID qualifier to use with the unqualified tables.

The SET CURRENT SQLID statement will be executed before any of the program SQL statements, providing that a *DB2AUTHID* parameter is present on the DB2PARMS statement.

- *In VISION:Interface dynamic, using the SQL SET CURRENT SQLID statement (that is, using the DB2AUTHID parameter) requires SYSADM authority.*
- Alternatively, your program can use VISION:Interface static where the use of the authorization ID does not require SYSADM authority.

DB2LOCID Parameter

The DB2PARMS DD statement DB2LOCID parameter establishes the location ID qualifier that DB2 uses with unqualified DB2 tables. DB2 does not change the location ID for tables that have the location ID included as part of the table name.

VISION:Interface executes an SQL CONNECT TO statement to direct DB2 to the location ID qualifier to use with unqualified tables. The CONNECT TO statement will be executed before any of the program SQL statements, providing that a DB2LOCID parameter is present on the DB2PARMS statement.

Changing DB2 Specifications Using VISION:Interface Static

To use this functionality, add the DB2PARMS DD statement.

```
//DB2PARMS DD *
DB2A      STATDB2   DSN8610              LOC_ID
```

DB2PARMS DD Statement Coding Positions (Static)

The DB2PARMS DD statement consists of one record with the following four data parameters:

Parameter	Size	Description	Position
DB2SSID	PIC X(8).	DB2 subsystem ID	1
Filler	PIC X.		
DB2PLAN	PIC X(8).	DB2 plan name	10
Filler	PIC X.		
DB2PACK	PIC X(18).	DB2 package or collection ID assigned to the authorization ID qualifier	19
Filler	PIC X.		
DB2LOCID	PIC X(16).	DB2 table location ID qualifier (Code only if you are using DDF (Distributed Data Facility).)	38

The parameters are optional, that is, you can specify one or all of the parameters on the record. The parameters are positional, that is, you must place them at the position indicated.

Note: If you are going to use DDF (Distributed Data Facility) processing, then the DB2SSID is the local server ID.

Specifying Packages for Different Authorization ID Qualifiers

IBM static DB2 does not allow a direct change to the authorization ID qualifier (unlike the IBM dynamic DB2 use of the SET CURRENT SQLID statement).

Instead, IBM static DB2 requires that you create packages for the different authorization ID qualifiers.

- For each package or collection ID, specify a unique authorization ID value on the QUALIFIER parameter.
- Include each package or collection ID in the package list of the pertinent plan.
- Your program directs VISION:Interface static to the package or collection ID to be used by means of the SET CURRENT PACKAGESET statement.
- DB2 then uses the authorization ID qualifier coded to that package.

Therefore, to direct VISION:Interface static to the authorization ID you want to use, specify the package containing the required authorization ID.

DB2SSID and DB2PLAN Parameters

The DB2PARMS DD statement DB2SSID and DB2PLAN parameters depend on the connection and attach facility.

- If the program uses an *explicit* connection to CALL Attach (PGM=DYL280 in the JCL), the job retrieves the DB2SSID and DB2PLAN parameters from DB2PARMS DD statement.
- If the program uses an *implicit* connection to CALL Attach:
  - The DB2PLAN will always be the name of the DBRM module, which is the same name as the generated Assembler source program module name.
  - The job retrieves DB2SSID from the DSNEXIT library concatenated to the JCL STEPLIB.
- If the program uses TSO Attach, the job:
  - Retrieves the plan and subsystem you code on the SYSTSIN statement.
  - Ignores DB2PARMS DD statement DB2SSID and DB2PLAN parameters.
- If the program uses IMS Attach, the job ignores the DB2PARMS DD statement DB2SSID and DB2PLAN parameters and retrieves the parameters as follows:
  - If the program is an IMS batch job, IMS Attach retrieves the subsystem and plan coded on the DDITV02 statement.
  - If program is an IMS BMP job, the DB2PLAN is assumed to be the same as the MBR parameter. You specify DB2SSID on the SSM parameter.
- If you are going to use DDF (Distributed Data Facility) processing, then the DB2SSID is the local server ID.

DB2PACK Parameter

The DB2PARMS DD statement DB2PACK parameter establishes the package or collection ID and its authorization ID qualifier that DB2 uses with unqualified DB2 tables. DB2 will not change the authorization ID for tables that have the authorization ID included as part of the table name.

Notes:

- The DB2PACK parameter applies to VISION:Interface static only.
- *The DB2AUTHID* parameter *is the authorization ID for* VISION:Interface *dynamic. For information on DB2AUTHID, see* Changing DB2 Specifications Using VISION:Interface Dynamic.

Different parameters are used to represent the authorization ID because different methods are used to acquire the authorization ID.

VISION:Interface static executes an SQL SET CURRENT PACKAGESET statement to direct DB2 on the package, and its authorization ID qualifier, to use with the unqualified tables.

The SET CURRENT PACKAGESET statement will be executed before any of the program SQL statements, providing that a DB2PACK parameter is present on the DB2PARMS statement.

DB2LOCID Parameter

The DB2PARMS DD statement DB2LOCID parameter establishes the location ID qualifier that DB2 uses with unqualified DB2 tables. DB2 will not change the location ID for tables that have the location ID included as part of the table name.

VISION:Interface executes an SQL CONNECT TO statement to direct DB2 to the location ID qualifier to use with the unqualified tables. The CONNECT TO statement will be executed before any of the program SQL statements, providing a DB2LOCID parameter is present on the DB2PARMS statement.

Summarizing DB2PARMS

	Dynamic	Static (Uses packages)
DB2PARMS positional parameters	DB2PLAN, DB2SSID, *DB2AUTHID*, and DB2LOCID (*DB2AUTHID* executes a SET CURRENT SQLID. DB2LOCID executes a CONNECT TO.)	DB2PLAN, DB2SSID, DB2PACK, and DB2LOCID (DB2PACK executes a SET CURRENT PACKAGESET. DB2LOCID executes a CONNECT TO.)
Explicit connection + CALL Attach	Uses DB2PARMS statement DB2PLAN and DB2SSID positional parameters.	Uses DB2PARMS statement DB2PLAN and DB2SSID positional parameters.
Implicit connection + CALL Attach	The DB2PLAN is always DYLSQI00. Retrieves DB2SSID from the DSNEXIT library.	The DB2PLAN is always the name of the DBRM module. Retrieves DB2SSID from the DSNEXIT library
TSO Attach	Uses the plan and subsystem on the SYSTSIN statement.	Uses the plan and subsystem on the SYSTSIN statement.
IMS Attach + batch job	IMS Attach uses the plan and subsystem on the DDITV02 statement.	IMS Attach uses the plan and subsystem on the DDITV02 statement.
IMS Attach + BMP job	The DB2PLAN is assumed to be the same as the MBR parameter. The DB2SSID is specified on the SSM parameter.	The DB2PLAN is assumed to be the same as the MBR parameter. The DB2SSID is specified on the SSM parameter.

Note: If you are going to use DDF (Distributed Data Facility) processing, then the DB2SSID is the local server ID.

Notes:

- *In VISION:Interface dynamic, using the DB2AUTHID parameter (that is, using the SQL SET CURRENT SQLID statement) requires SYSADM authority.*
- Alternatively, your program can use VISION:Interface static where SYSADM authority is not required to use the DB2PACK parameter.

Changing DB2 Specifications Using COPYDB2

To use this functionality, add the COPYDB2A DD statement.

```
//COPYDB2A DD *
DB2T     DYLDB2   DSN8610              LOC_ID
```

COPYDB2 DD Statement Coding Positions

The COPYDB2A DD statement consists of one record with the following four parameters:

Parameter	Size	Description	Position
COPYDB2SSID	PIC X(8).	DB2 subsystem ID	1
Filler	PIC X.		
COPYDB2PLAN	PIC X(8).	DB2 plan name	10
Filler	PIC X.		
COPYDB2AUTHID	PIC X(8).	DB2 authorization ID qualifier	19
Filler	PIC X(1).		
COPYDB2LOCID	PIC X(16).	DB2 location ID qualifier (Code only if you are using DDF (Distributed Data Facility).)	28

The parameters are optional, that is, you can specify one or all of the parameters on the record. The parameters are positional, that is, you must place them at the position indicated.

Note: If you are going to use DDF (Distributed Data Facility) processing, then the COPYDB2SSID is the local server ID.

Multiple COPYDB2 Commands with Corresponding DD Statements

You can assign each COPYDB2 command its own DB2 subsystem ID, DB2 plan name, DB2 table authorization ID qualifier, and DB2 table location ID qualifier without modifying the VISION:Results program.

For each COPYDB2 command, you assign a unique DD statement (COPYDB2A DD statement, COPYDB2B DD statement, COPYDB2C DD statements, and so on) that contains a single record with the above positional parameters:

- The first seven characters (for each DD statement name) are COPYDB2.
- The last character in the COPYDB2n name is unique for each COPYDB2 command, and is assigned alphabetical letters (A through Z) in the order the COPYDB2 commands appear in the program.

For example:

```
COPYDB2 DSN841.EMPL       ==> //COPYDB2A DD
COPYDB2 DSN841.DEPT       ==> //COPYDB2B DD
COPYDB2 DSN841.PROJACCT   ==> //COPYDB2C DD
```

If you do not have a COPYDB2A DD statement in the JCL, there is no override for the first COPYDB2 command. If you have blanks where a positional parameter should be in the record, there is no override for that parameter. The same rules apply for the other COPYDB2 commands and their corresponding COPYDB2 DD statements.

COPYDB2SSID and COPYDB2PLAN Parameters

The COPYDB2A DD statement COPYDB2SSID and COPYDB2PLAN parameters depend on the connection and attach facility.

- If the program uses an *explicit* connection to CALL Attach (PGM=DYL280 in the JCL), the job retrieves the COPYDB2SSID and COPYDB2PLAN parameters.
- If the program uses an *implicit* connection to CALL Attach, the job:
  - Ignores COPYDB2SSID and COPYDB2PLAN parameters.
  - Retrieves DB2 subsystem ID from the DSNEXIT library concatenated to the JCL STEPLIB.
  - Always uses the DB2 plan, DYLCAT00.
- If the program uses TSO Attach, the job:
  - Retrieves the DB2 subsystem ID and DB2 plan you code on the SYSTSIN statement.
  - Ignores the COPYDB2SSID and COPYDB2PLAN parameters.
- If the program uses IMS Attach, the job ignores the COPYDB2SSID and COPYDB2PLAN parameters and retrieves the DB2 parameters as follows:
  - If the program is an IMS batch job, IMS Attach retrieves the DB2 subsystem ID and DB2 plan you code on the DDITV02 statement.
  - If the program is an IMS BMP job, the DB2 plan is assumed to be the same as the MBR parameter. You specify the DB2 subsystem ID on the SSM parameter.
- If you are going to use DDF (Distributed Data Facility) processing, then the COPYDB2SSID is the local server ID.

COPYDB2AUTHID Parameter

The COPYDB2A DD statement COPYDB2AUTHID parameter establishes the authorization ID qualifier that DB2 uses for the COPYDB2. This includes tables that have the authorization ID included as part of the table name.

COPYDB2LOCID Parameter

The COPYDB2A DD statement COPYDB2LOCID parameter establishes the location ID qualifier that DB2 uses for the COPYDB2. This includes tables that have the location ID included as part of the table name.

Chapter

5 Using SQL with VISION:Results

This chapter describes using SQL in VISION:Results with VISION:Interface. The topics include: introducing SQL, data names and data types, understanding the SELECT statement, coding data retrieval SQL statements, embedded SELECT statement, SQL cursor statements, CURSOR (WITH HOLD) SELECT statements, testing for end of cursor, coding data management SQL statements, and coding DDF (Data Distribution Facility SQL statements.

For a list of statements, see the Chapter 1, *Introducing VISION:Interface for DB2*, section Supported SQL Statements.

Notes:

- SQL is the acronym for Structure Query Language.
- *Text which is specific to the dynamic process is italic.*

Introducing SQL

VISION:Interface for DB2 requires the use of embedded SQL to access and manipulate data from DB2 tables. You code the embedded SQL statements like you would in a COBOL, PL1, or C program.

Note: See Standard SQL, Simplified SQL, and Embedded SQL

- To make effective use of VISION:Interface, you need to understand the concepts of SQL (Structured Query Language). This chapter provides an introduction to SQL.
- For complete information about SQL capabilities, see the appropriate IBM DB2 manual.

SQL is a standard language for data management and data retrieval with relational databases. SQL is a set-oriented language in that data is accessed in terms of *sets* of records (that is, relations) rather than one record at a time as with traditional languages.

Think of the database as a set of tables where relationships are represented by values in the tables. You retrieve data, from one or more tables by specifying a "result table." IBM SQL, as implemented for DB2, optimizes data retrieval and the data management.

There are two categories of SQL statements:

- **Data retrieval** SQL statements only retrieve data from a relational database. Retrieval statements do not change the data.
- **Data management** SQL statements create, update, and delete data in a relational database.

Note: For additional information on the statements in these two categories, see the *VISION:Interface for DB2 Reference Guide*.

Understanding Basics of SQL

In simplest terms, SQL requests specify data access functions, such as data retrieval, insertion, deletion, and updating, coded in an SQL statement.

Note: For additional information, see the *VISION:Interface for DB2 Reference Guide* and the section SQL Cursor Statements in this book.

Data Retrieval SQL Example

Data retrieval SQL statements define a set of data to be passed from DB2 to the VISION:Results program by means of VISION:Interface.

An SQL request for data retrieval looks like this:

```
SELECT    EMPNO, LASTNAME, SALARY
          FROM DSN8610.EMP
          WHERE SALARY > 20000
          ORDER BY LASTNAME
```

This SQL request defines a set of data that can be passed from DB2 to the VISION:Results program.

- In SQL terms (with traditional terms in parentheses), a result table (file) is created from the source table, DSN8610.EMP, containing a set of rows (records), each consisting of three columns (fields), where each row has a SALARY that exceeds $20,000.
- The ORDER BY clause sorts the LASTNAME column in ascending order.

Data Management SQL Example

A second, more complex SQL request, a data management SQL statement, looks like:

```
UPDATE    DSN8610.EMP SET SALARY = SALARY * 1.1
          WHERE EMPNO IN (SELECT EMPNO
          FROM DSN8610.EMP, DSN8610.DEPT
          WHERE (DEPTNO = 'A00' OR ADMRDEPT = 'A00')
                              AND WORKDEPT = DEPTNO)
```

This SQL request accesses two tables, DSN8610.EMP (employee table) and DSN8610.DEPT (department table) in order to increase the salaries of selected employees by ten percent.

- To select the employees (all those working in division A00), the WHERE clause of the UPDATE retrieves a list of employees that match two conditions: where either DEPTNO or ADMRDEPT (in DEPT) is A00 and the employee's work department (in EMP) is the same as a DEPTNO (in DEPT).
- The columns are not qualified since the column names are unique between the two tables. If names are ambiguous, you must fully qualify the names (that is, DSN8610.EMP.WORKDEPT).

Standard SQL, Simplified SQL, and Embedded SQL

This document uses the following SQL terms:

Standard SQL IBM-supported SQL statements.

Simplified SQL Special SQL statements which can only be processed by VISION:Results programs using VISION:Interface for DB2.

Simplified SQL statements are embedded in the same manner as standard SQL statements.

Embedded SQL SQL statements that are "embedded" in another programming language like COBOL, PL/I, VISION:Results, and so on. Embedding means that SQL statements are surrounded by an SQL-initiator (EXEC SQL) and an SQL-terminator (ENDEXEC). Embedded SQL encompasses standard SQL and simplified SQL.

VISION:Interface for DB2 provides the ability to process embedded SQL statements in VISION:Results programs in order to access DB2 databases. See the following section EXEC SQL - ENDEXEC Syntax Rules.

EXEC SQL - ENDEXEC Syntax Rules

To embed SQL statements in a VISION:Results program, you place an EXEC SQL statement in the program, follow it by SQL statements, and end the group of SQL statements with an ENDEXEC statement (that is, surround SQL statements by the EXEC SQL - ENDEXEC pair.)

Notes:

- In the text, which explains the syntax of a statement, you enter the uppercase command and keyword just as you see it.
- Lowercase implies clauses or operands that you provide.

The syntax for the EXEC SQL - ENDEXEC statement pair is:

EXEC SQL

sql statement

ENDEXEC

Where:

EXEC SQL — The SQL-initiator, EXEC SQL, begins an embedded SQL statement sequence.

The EXEC SQL phrase defines the beginning of the SQL statement or a group of statements. All source code that follows is standard SQL or simplified SQL until the SQL-terminator ENDEXEC is reached.

Do not write non-SQL statements, which are not documented by VISION:Interface, on the same line with EXEC SQL.

sql statement — All standard SQL rules apply in an SQL statement. See the standard SQL rules in IBM DB2 and SQL manuals.

Precede all VISION:Results data names (host variables in SQL terminology) by a colon (:) when they appear in SQL statements. The colon prefix distinguishes the VISION:Results data names from SQL names.

All SQL statements are completely free-form within the boundaries of an EXEC SQL - ENDEXEC pair. You can write an SQL statement on a single line:

```
EXEC SQL sql statement ENDEXEC
```

Do not use commas within numeric values inside of an SQL statement.
For example, use 4000 rather than 4,000.

This SQL requirement differs from the normal VISION:Results numeric value processing and is therefore enforced *inside* of the SQL statement environment (that is, between the EXEC SQL - ENDEXEC statement pair).

ENDEXEC — The SQL-terminator, ENDEXEC, specifies the end of the embedded SQL statement sequence.

Do not write non-SQL statements on the same line with ENDEXEC.

SQL Statements in VISION:Results

The following specifications apply to SQL statements in VISION:Results programs:

- You can use VISION:Results data names as host variables in SQL statements up to 50 characters in length. This limitation also includes qualifiers.
- You can have up to 4,000 characters in a single SQL statement.
- For VISION:Interface dynamic, you can use up to 100 executable SQL statements in a single VISION:Results program.
  Although the default of 20 should satisfy the needs of most users, you can customize this number at installation time to any number from 1 to 100.
  - For the purpose of counting executable SQL statements, all references to a specific named cursor (DECLARE, OPEN, FETCH, and CLOSE) are considered single statements.
  - DECLARE TABLE and WHENEVER statements are not executable, nor countable.
  - Do not include DDF statements, such as SET, CONNECT, and RELEASE as part of the count.
- If a program contains more than 30 extremely large SQL statements or 80 standard size SQL statements, then you must do any report writing (that is, use of the LIST statement) in a separate report within the program. Use a VISION:Results PICNSAVE statement to gather and save the data required by the report.

Data Names and Data Types

The data types are:

- SQL VARCHAR Data Type
- SQL Character Data Types
- VISION:Results Data Types

SQL VARCHAR Data Type

VISION:Interface 4.0 supports the SQL VARCHAR data type. The COPYDB2 facility determines the VARCHAR data from the DB2 catalog.

Previous to VISION:Results 5.0 and VISION:Interface 4.0, VISION:Interface retrieved a VARCHAR data type as a fixed field with the length being its maximum. Now you can use true variable length character fields, which are prefixed by a binary value, representing the actual length.

- Define an SQL VARCHAR data type as a variable length character field by coding a character field definition that includes a 2-byte binary field as part of its length. For example, if the maximum data length is 25, then the VARCHAR field definition length is 27.
- The maximum field size is 32767.
- Specify the host variable name with '_VAR' as the dataname suffix.

In the following example:

```
FIRSTNME_VAR    27    CH
    REDEFINE AT FIRSTNME_VAR
FIRSTNME_LEN    2     BI
FIRSTNME_TEXT   25    CH
    REDEFINE AT FIRSTNME_TEXT
FIRSTNME        25    CH
```

FIRSTNME_VAR has a total length of 27 bytes, where the first two bytes (FIRSTNME_LEN) contains the actual length (in binary), and the next 25 (FIRSTNME_TEXT) contain the value. When the program uses FIRSTNME as the host variable, DB2 assumes a fixed field and returns the maximum length consisting of the actual value followed by trailing spaces.

When you code an SQL statement INTO clause with :FIRSTNME_VAR:

- The call to DB2 will be set up to request the data item as a VARCHAR type field.
- The length of the VARCHAR data item will be placed in FIRSTNME_LEN.
- The data will reside in the left most portion of the FIRSTNME_TEXT field (and be available as FIRSTNME); therefore, you must interrogate or include the length of the VARCHAR data item before using FIRSTNME_TEXT or FIRSTNME in any subsequent VISION:Results statement, otherwise the remaining portion of the field could contain garbage.
- If the SQL statement INTO clause refers to host variable name FIRSTNME or FIRSTNME_TEXT, the generated call to DB2 will retrieve the data item as a fixed length field.

  Note:SQL statements containing INTO clauses are:

  CURSOR (WITH HOLD) SELECT fullselect INTO :hostvar1, :hostvar2, ...,
  FETCH cursor-name INTO :hostvar1, :hostvar2, ..., and
  SELECT column-list INTO :hostvar1, :hostvar2, ...

This feature applies to both dynamic and static VISION:Interface.

SQL Character Data Types

The following table shows valid and invalid SQL data types.

Valid SQL Character Data Types	
CHAR	1 - 254 Characters
VARCHAR	1 - 32767 Characters
LONGVARCHAR	1 - 32767 Characters

Valid SQL Numeric Data Types	
SMALLINT	2-byte binary integer
INTEGER	4-byte binary integer
DECIMAL	1- to 8-byte packed decimal
FLOAT	8-byte double precision floating point

Unsupported SQL Data Types	
GRAPHIC	1 - 127 Characters
VARGRAPHIC	1 - 32767 Characters

VISION:Results Data Types

The following table shows valid and invalid VISION:Results data types.

Valid VISION:Results Character Data Types

n CH	Characters (where n = 1- 32767)
2 BI n CH	VARCHAR with 2-byte binary integer length followed by characters (where n = 1- 32767)

Note: See the sample in the section SQL VARCHAR Data Type.

Valid VISION:Results Numeric Data Types

2 BI	2-byte binary integer
4 BI	4-byte binary integer
n PD	Packed decimal (where n = 1 - 16)

Unsupported VISION:Results Data Types

n NU	Zoned numeric (any size)
1 BI	1-byte binary
3 BI	3-byte binary

Rounding and Truncation Effects

- When SQL decimal data (DECIMAL or FLOAT) are passed to VISION:Results, loss of significance may occur through rounding of decimal places. This is particularly true when FLOAT is passed to a VISION:Results BI or PD with no decimal places.
- When SQL character data (CHAR and VARCHAR) are passed to VISION:Results character data names (CH), the data are truncated or padded with blanks to the VISION:Results length as necessary. VARCHAR columns are returned as fixed length if the fixed length field is the host variable. The fixed length field is the field definition that does not include the 2-byte binary field as part of its length. If it does, DB2 assumes the field to be a VARCHAR field.
- COPYDB2 automatically handles data names and data types.

Indicator Variables With and Without the INDICATOR Keyword

An indicator variable (associated with a host variable) specifies a value indicating a null condition.

Note: Indicator conditions include specifying the null value, indicating a numeric conversion or arithmetic expression error, recording the length of a truncated string, indicating that a character could not be converted, and/or recording the time if the time is truncated on assignment to the host variable.

You can specify an indicator variable in one of two ways:

- Specify a host variable name (for example, :hostvar), at least one blank, the word INDICATOR, at least one blank, and an indicator variable name (for example, :indicvar).

  :hostvar INDICATOR :indicvar

- Specify the host variable name immediately followed by (with or without intervening blanks) the indicator variable name. This previous technique will still be honored.

  :hostvar :indicvar

DYLINSTL Macro DB2NULL Parameter

When you set the DYLINSTL macro DB2NULL parameter to be used (DB2NULL=Y), COPYDB2 generates null indicator fields for all the fields that can have a null value.

- The null indicator field appears right after the field it reflects.
- It is a 2-byte binary field (required by DB2 SQL programming) and will have the same name as the field it refers to but with an additional ending suffix of IND.
- If there is no room for the ending suffix of IND, then the null indicator field will not be built for that field.

The following is an example of a COPYDB2 with the DB2NULL set to Y:

```
COPYDB2 EMP
  COPIED EMPNO                              6 CH
  COPIED FIRSTNME_VAR                      14 CH
  COPIED   REDEFINE AT    FIRSTNME_VAR
  COPIED FIRSTNME_LEN                       2 BI
  COPIED FIRSTNME_TEXT                     12 CH
  COPIED   REDEFINE AT    FIRSTNME_TEXT
  COPIED FIRSTNME                          12 CH
  COPIED MIDINIT                            1 CH
  COPIED LASTNAME_VAR                      17 CH
  COPIED   REDEFINE AT    LASTNAME_VAR
  COPIED LASTNAME_LEN                       2 BI
  COPIED LASTNAME_TEXT                     15 CH
  COPIED   REDEFINE AT    LASTNAME_TEXT
  COPIED LASTNAME                          15 CH
  COPIED WORKDEPT                           3 CH
  COPIED WORKDEPT_IND                       2 BI
  COPIED PHONENO                            4 CH
  COPIED PHONENO_IND                        2 BI
  COPIED HIREDATE                             10 CH   ;DATE COLMN
  COPIED HIREDATE_IND                       2 BI
  COPIED JOB                                8 CH
  COPIED JOB_IND                            2 BI
  COPIED EDLEVEL                            2 BI
  COPIED EDLEVEL_IND                        2 BI
  COPIED SEX                                1 CH
  COPIED SEX_IND                            2 BI
  COPIED BIRTHDATE                        10 CH  ;DATE COLMN
  COPIED BIRTHDATE_IND                        2 BI
  COPIED SALARY                               5 PD     2
  COPIED SALARY_IND                           2 BI
  COPIED BONUS                                5 PD     2
  COPIED BONUS_IND                            2 BI
  COPIED COMM                                 5 PD     2
  COPIED COMM_IND                             2 BI
```

Figure 5-1 COPYDB2 with the DB2NULL set to Y

Understanding SELECT

VISION:Interface supports the complete standard SELECT statement syntax.

- See the IBM DB2 and SQL manuals for complete information on the SELECT statement.
- This document only presents a basic form of the SELECT statement.

SELECT Statement Functions

The SELECT statement performs the following functions:

- Identify the DB2 table or tables to be used as a source for retrieval.
- Specify columns (fields) to be retrieved.
- Specify the selection conditions of the data to be retrieved.
- Specify the sort order of the data.
- Indicate whether to return summary or detail information.
- Join two or more DB2 tables into a single view.

You can qualify the selection of returned rows (records) with either SQL statements (by means of the WHERE clause of the SELECT statement) or VISION:Interface statements (using standard VISION:Results IF-ACCEPT-REJECT logic). It is almost always more efficient to have SQL do the DB2 selection.

SELECT fullselect Statement

Use the SELECT fullselect to retrieve one or more rows of data from a DB2 table into host variables, that is VISION:Results data names.

The SELECT fullselect statement syntax is:

SELECT column-list
FROM table-name
WHERE search-conditions
ORDER BY column-list

Where:

SELECT column-list

Specifies the names of the columns (fields) to be retrieved. Place commas between each name, that is, $name_1$, $name_2$, ..., $name_n$.

See the section Data Names and Data Types.

FROM table-name

Specifies the table name.

WHERE search-conditions

The search conditions used in a WHERE clause of a SELECT statement are similar to those used in the VISION:Results IF statement except that you use algebraic symbols for defining logical comparisons instead of the 2-character codes.

See the section Search Conditions.

ORDER BY column-list

Specifies the sort order by listing column names separated by commas, that is, $name_1$, $name_2$, ..., $name_n$. The default sort order is ascending.

Search Conditions

The search conditions used in a WHERE clause of a SELECT statement are similar to those used in the VISION:Results IF statement except that you use algebraic symbols for defining logical comparisons instead of the 2-character operators. The following is a list of the valid SQL comparison operators and the corresponding VISION:Results operators.

SQL Operators	Explanation	VISION:Results Operators
>	Greater than	GT
<	Less than	LT
>=	Greater than or equal to	GE
<=	Less than or equal to	LE
=	Equal to	EQ
^=	Not equal to	NE
^>	Not greater than	LE
^<	Not less than	GE

- Use SQL operators in SQL statements.
- Use VISION:Results operators in VISION:Results statements.
- Enclose all character literals used in search conditions in single quotation marks.
- Do not embed commas in numeric values (for example, write 1,234.5 as 1234.5).
- You can use VISION:Results data names as search values wherever SQL permits the use of host variables. Follow the standard SQL syntax and place a colon before the VISION:Results data name.
- When you use VISION:Results data names as search values, make the columns or values that are compared type-compatible.
  - You can compare VISION:Results character data to any valid SQL character type.
  - You can compare valid VISION:Results numeric data to any valid SQL numeric type.
- You can specify complex search conditions using the words AND or OR between two comparisons.
- Use parentheses to group a set of search conditions.
- You can use other operators to search for information including IN (list testing), LIKE (pattern matching), BETWEEN (range testing), and EXISTS (test for existing data). See the IBM DB2 and SQL manuals for further information. There are no restrictions on the use of VISION:Results search conditions with VISION:Interface.

Types of SELECT Statements

The SELECT defines the domain of the retrieved data. In order to pass the data to VISION:Results, you use the following SELECT statements:

- Embedded SELECT (SELECT INTO) statement to retrieve one row of data. (See the section Embedded SELECT Statement.)
- Standard SELECT (DECLARE CURSOR, OPEN, FETCH, and CLOSE) statements to retrieve one or more rows of data. (See the section SQL Cursor Statements.)
- Simplified SELECT (CURSOR (WITH HOLD) SELECT) statement to retrieve one or more rows of data without the requirement to code the DECLARE CURSOR, OPEN, FETCH, and CLOSE statements. (See the section CURSOR (WITH HOLD) SELECT Statement.)

Coding Data Retrieval SQL Statements

Use the basic rules for using SQL in VISION:Results programs as you would for other programming languages. This section discusses data retrieval SQL statement used by VISION:Interface to access DB2:

Embedded SELECT Statement

- Called also the SELECT INTO
- Retrieves one row

SQL Cursor Statements

Retrieves a set of rows contained in a result table. To retrieve the set of rows, code:

- DECLARE CURSOR statement
- OPEN statement
- FETCH statement
- CLOSE statement

Note: The SQL cursor is a file-like concept which retrieves data one row at a time (from a set of rows) in a manner similar to READ statements in other programming languages. Formally, the cursor is defined as named control structure used by an application program to point to a row within a set of rows, and hence retrieve, update, and/or delete rows from the set.

CURSOR (WITH HOLD) SELECT Statement

- A VISION:Interface simplified SQL statement
- A condensed version of the SQL cursor statements
- Retrieve a set of rows contained in a result table

Testing for End of Cursor Data

- WHENEVER NOT FOUND statement
- DYLSQLCODE And DYLSQLSTAT

Embedded SELECT Statement

The standard SELECT INTO, when surrounded by the EXEC SQL - ENDEXEC statement pair, is the embedded SELECT, which retrieves a single row of data from a DB2 table to host variables, that is, VISION:Results data names.

Note: IBM DB2 manuals refer to the embedded SELECT as the SELECT INTO statement.

The embedded SELECT statement syntax is:

SELECT column-list
INTO :hostvar$_1$, :hostvar$_2$, ..., :hostvar$_n$
FROM table-name
WHERE search-conditions
ORDER BY column-list

Where:

SELECT column-list

Specifies the names of the columns (fields) to be retrieved. Place commas between each name, that is, name$_1$, name$_2$, ..., name$_n$.

INTO :hostvar$_1$, :hostvar$_2$, ..., :hostvar$_n$

Defines the VISION:Results data names into which the values from the column names are stored. VISION:Results data names are equivalent to SQL host variable names (:hostvar$_1$, :hostvar$_2$, ..., :hostvar$_n$).

Make the number of column names equal to the number of host variable names, that is, specify one VISION:Results data name (in the INTO clause) for each column name you specify in the SELECT column-list clause.

VISION:Interface requires that the colon precede VISION:Results data names to distinguish them from SQL names.

Make VISION:Results data names type-compatible with the data types of the DB2 columns selected.

- Place DB2 character data into VISION:Results character variables.
- Place DB2 numeric data into VISION:Results numeric variables.
- DB2 performs the conversion of numeric data between different numeric types when the VISION:Results numeric type is valid.

Use COPYDB2 to generate the VISION:Results data fields.

FROM table-name

Specifies the table name.

WHERE search-conditions

The search conditions used in a WHERE clause of a SELECT statement are similar to those used in the VISION:Results IF statement except that you use algebraic symbols for defining logical comparisons instead of the 2-character codes.
See the section Search Conditions.

ORDER BY column-list

Specifies the sort order by listing column names separated by commas, that is, $name_1$, $name_2$, ..., $name_n$. The default sort order is ascending.

The following is a sample program using the embedded SELECT. The embedded SELECT components are highlighted.

SQL SUM FUNCTION Sample Program

```
REPORT 78 WIDE
 ;
 ; PROGRAM: SQL SUM FUNCTION
 ;
 ; DESCRIPTION: TWO RETRIEVALS FROM THE SAME TABLE WHERE
 ;              THE FIRST RETRIEVAL GETS THE DETAIL DATA
 ;              USING A SIMPLIFIED CURSOR SELECT STATEMENT
 ;              AND THE SECOND RETRIEVAL GETS TOTAL SALARY
 ;              USING AN EMBEDDED SELECT STATEMENT (ALSO
 ;              PRINTS VISION:RESULTS DERIVED TOTAL FOR COMPARISON).
 ;
 FILE XFILE DUMMY

 WORKAREA
   EMPNO     6 CH
   FNAME    12 CH
   LNAME    15 CH
   SALARY    5 PD 2 E
   TOTSAL    5 PD 2 E
   MF        1 CH VALUE "F"

 EXEC SQL
   WHENEVER NOT FOUND GOTO DONE
 ENDEXEC

 EXEC SQL
   CURSOR SELECT EMPNO, FIRSTNME, LASTNAME, SALARY
   FROM DSN8610.EMP
   WHERE SEX = :MF
   ORDER BY FIRSTNME
   INTO :EMPNO, :FNAME, :LNAME, :SALARY
 ENDEXEC

 LIST EMPNO, FNAME, LNAME, SALARY

 ACCEPT

 DONE:
```

Figure 5-2 SQL SUM FUNCTION Sample Program (Page 1 of 2)

```
  EXEC SQL
    SELECT SUM(SALARY)
    INTO :TOTSAL
    FROM DSN8610.EMP
    WHERE SEX = :MF
  ENDEXEC

  LIST ' '
  LIST 'DB2 TOTAL' AT LNAME TOTSAL
  STOP

ON FINAL LIST 'DYL TOTAL' AT LNAME SUM SALARY

T1    'VISION:INTERFACE FOR DB2 - SQL SUM FUNCTION' WITH 2 AFTER
T1+1  DYLDATE
```

Figure 5-2 SQL SUM FUNCTION Sample Program (Page 2 of 2)

SQL Cursor Statements

SQL provides the ability to access either a single row or a set of rows from a DB2 table:

The SELECT statement, in association with a DECLARE CURSOR statement, retrieves a set of rows.

By using a file-like concept called a cursor, SQL retrieves data one row at a time, in a manner very much like the READ statements of other programming languages. This type of retrieval is often referred to as cursor-based retrieval. The SELECT statement which is associated with a cursor is called a cursor-based SELECT.

Once a DECLARE CURSOR statement defines a cursor, you can use other cursor-associated statements, such as:

- OPEN Statement
- FETCH Statement
- CLOSE Statement

DECLARE CURSOR Statement

The DECLARE CURSOR statement defines the retrieval set with a standard SELECT statement. You can think of the cursor as a temporary file name for use in VISION:Results.

Notes:

- Do not use the INTO clause in a DECLARE CURSOR statement.
- There are two variations of the DECLARE CURSOR statement which you use in VISION:Results with VISION:Interface.

The syntax of the DECLARE CURSOR statements is:

DECLARE cursor-name CURSOR FOR SELECT fullselect

DECLARE cursor-name CURSOR WITH HOLD FOR SELECT fullselect

Where:

cursor-name

Each DECLARE CURSOR statement is associated with a cursor which can have any valid name.

- Do not define the cursor name as a VISION:Results data name.
- Do not invoke the same DECLARE statement more than once.
- Use the cursor name in subsequent OPEN, FETCH, and CLOSE requests to associate a particular DECLARE CURSOR statement (and its set of rows as specified in the SELECT statement) with these other DB2 requests.

WITH HOLD

The optional WITH HOLD clause allows the program to maintain the cursor position after a COMMIT.

fullselect

Use the fullselect in a DECLARE CURSOR request in any valid SELECT statement specifying the result table of the cursor.

For information on the fullselect statement, see the section SELECT fullselect Statement.

OPEN Statement

The OPEN statement activates the specified cursor. The OPEN statement syntax is:

OPEN cursor-name

- You can reposition a cursor to the beginning of the retrieved data by closing and reopening the same cursor name.
- Failure to close the cursor prior to reopening it, will result in an error.

FETCH Statement

The FETCH statement retrieves a single row (record) of data from the DB2 table defined with the corresponding DECLARE CURSOR statement. The statement returns values from each column (field) in the row to VISION:Results data names.

The requirements for use of VISION:Results data names in the FETCH request are the same as those for the embedded SELECT statement. For these requirements, see the section Embedded SELECT Statement.

The FETCH statement syntax is:

FETCH cursor-name
INTO :hostvar$_1$, :hostvar$_2$, ..., :hostvar$_n$

Where:

INTO :hostvar$_1$, :hostvar$_2$, ..., :hostvar$_n$

Defines the VISION:Results data names into which the values from the column names are stored. VISION:Results data names are equivalent to SQL host variable names (:hostvar$_1$, :hostvar$_2$, ..., :hostvar$_n$).

Make each VISION:Results data name correspond to one of the DB2 columns being retrieved in the SELECT statement. Make the number of VISION:Results data names the same as the number of DB2 columns.

Prefix each VISION:Results data name with a colon (:), that is, :hostvar$_1$, :hostvar$_2$, ..., :hostvar$_n$.

Maintain type-compatibility between DB2 columns and VISION:Results data names by using COPYDB2.

CLOSE Statement

Use the CLOSE statement to terminate use of the cursor and free up DB2 memory.

- Closing the cursor is recommended but not required.
- If you are going to reopen a cursor, you must close it (reopening the cursor will retrieve the same data).

  However, if the WHERE keyword is using host variables and you change the host variable value, then DB2 will retrieve a whole new set of data.

The CLOSE statement syntax is:

CLOSE cursor-name

The following is a sample using the SQL cursor statements. The SQL cursor statements are highlighted.

SALARY COMPARISON Sample Program

```
REPORT 78 WIDE
;
;
;
; PROGRAM:     SALARY COMPARISON
;
; DESCRIPTION: SINGLE TABLE DB2 EXTRACTION REPORT FROM
;              DSN8610.EMP USING STANDARD EMBEDDED SQL
;              AND HOST VARIABLES FOR SELECTION VALUES.

FILE XFILE DUMMY

WORKAREA
  EMPNO     6 CH
  FNAME    12 CH
  LNAME    15 CH
  SALARY    5 PD 2 E
  SELSAL    5 PD 2 E VALUE 20000

ON ONE

  EXEC SQL
    DECLARE EMPCSR CURSOR FOR
    SELECT EMPNO, FIRSTNME, LASTNAME, SALARY
    FROM DSN8610.EMP
    WHERE SALARY > :SELSAL
    ORDER BY LASTNAME
  ENDEXEC

  EXEC SQL
    OPEN EMPCSR
  ENDEXEC

ENDONE

EXEC SQL
  FETCH EMPCSR INTO :EMPNO, :FNAME, :LNAME, :SALARY
ENDEXEC

IF DYLSQLSTAT EQ 'Y'
  THEN LIST EMPNO, FNAME, LNAME, SALARY
  ELSE
    EXEC SQL
      CLOSE EMPCSR
    ENDEXEC
    STOP
ENDIF

T1    'VISION:INTERFACE FOR DB2 - SALARY COMPARISON' WITH 2 AFTER
T1+1  DYLDATE
T1+70 DYLETIME
```

Figure 5-3 SALARY COMPARISON Sample Program

CURSOR (WITH HOLD) SELECT Statement

The CURSOR (WITH HOLD) SELECT statement is one of two simplified SQL statements provided to enhance the use of SQL in VISION:Results with VISION:Interface.

Note: The simplified SQL statements are to simplify using SQL in conjunction with VISION:Results. They are not valid in any IBM-supported DB2 environment.

The CURSOR (WITH HOLD) SELECT statement combines the functions of the DECLARE CURSOR, OPEN, FETCH, and CLOSE statements into a single statement.

- The word CURSOR is not used by IBM as the initial word of any SQL statement and has been designated for use in VISION:Results as an extension to standard SQL.
- The standard SELECT INTO (embedded SELECT) returns only a single row, whereas the CURSOR (WITH HOLD) SELECT returns a set of rows.
- The way to code a CURSOR SELECT is to code the keyword CURSOR (WITH HOLD is optional) in front of the SELECT statement and add the INTO clause (found on the FETCH statement) at the end of the SELECT statement.

The CURSOR (WITH HOLD) SELECT statement syntax is:

CURSOR (WITH HOLD) SELECT fullselect

INTO :hostvar$_1$, :hostvar$_2$, ..., hostvar$_n$

Where:

CURSOR (WITH HOLD) SELECT fullselect

The fullselect syntax is any valid SELECT statement, except that the INTO clause must be last. It should define the table of data that is to be retrieved. For an explanation of the fullselect, see the section SELECT fullselect Statement.

The WITH HOLD clause maintains the cursor position after a COMMIT.

Position the CURSOR (WITH HOLD) SELECT in the VISION:Results program where the DB2 data should be retrieved.

The cursor is automatically opened and the first row from the selected table is placed into the specified VISION:Results data names in the same way as the FETCH statement.

Subsequent executions of the CURSOR (WITH HOLD) SELECT statement retrieve additional rows without executing the fullselect again.

When the last row is retrieved, the cursor automatically closes. You can find information on test error conditions and end of data in the *VISION:Interface for DB2 Reference Guide*.

INTO :hostvar$_1$, :hostvar$_2$, ..., :hostvar$_n$

Place the INTO clause after the fullselect. The INTO clause must be after the fullselect. The ENDEXEC keyword must follow the INTO clause.

Make each VISION:Results data name correspond to one of the DB2 columns being retrieved in the fullselect.

Make the same number of VISION:Results data names as there are DB2 columns.

Prefix each VISION:Results data name with a colon (:), that is :hostvar$_1$, :hostvar$_2$, ..., :hostvar$_n$.

Maintain type-compatibility between DB2 columns and VISION:Results data names.

The following is a sample program using the CURSOR (WITH HOLD) SELECT statement. The statement is highlighted.

Simplified SQL Sample Program

```
REPORT 78 WIDE
 ;
 ;
 ;
 ;
 ; PROGRAM:     SIMPLIFIED SQL
 ;
 ; DESCRIPTION: SINGLE TABLE DB2 EXTRACTION REPORT
 ;              FROM DSN8610.EMP USING SIMPLIFIED SQL
 ;              AND DB2 SELECTION CONDITIONS
 ;

 FILE XFILE DUMMY

 WORKAREA
   EMPNO     6 CH
   FNAME    12 CH
   LNAME    15 CH
   SALARY    5 PD 2 E

 EXEC SQL
   CURSOR SELECT EMPNO, FIRSTNME, LASTNAME, SALARY
   FROM DSN8610.EMP
   WHERE SALARY > 20000
   ORDER BY SALARY
   INTO :EMPNO, :FNAME, :LNAME, :SALARY
 ENDEXEC

 IF DYLSQLSTAT EQ 'E' THEN STOP ENDIF

 LIST EMPNO, FNAME, LNAME, SALARY

 T1    'VISION:INTERFACE FOR DB2 - SIMPLIFIED SQL' WITH 2 AFTER
 T1+1  DYLDATE
 T1+70 DYLETIME
```

Figure 5-4 SIMPLIFIED SQL Sample Program

Testing for End of Cursor Data

Fetching multiple rows is like reading a file, eventually there will not be any more rows to fetch. When there are no more rows to fetch, DB2 returns an SQL code of 100, which means the cursor needs to be closed.

- If you code an embedded SELECT statement or CURSOR SELECT statement, VISION:Interface automatically closes the cursor.
- However, if you coded the standard SQL cursor statements (DECLARE, OPEN, FETCH, CLOSE), it is your responsibility to check for an SQL code of 100 and issue a CLOSE statement to close the cursor.

There are two methods of checking for the end of cursor. The first method involves using the WHENEVER NOT FOUND statement. The second involves using one of the VISION:Interface SQL status fields: DYLSQLCODE and DYLSQLSTAT.

WHENEVER NOT FOUND Statement

The WHENEVER NOT FOUND statement specifies what the program will do when a NOT FOUND condition occurs during a fetch.

- WHENEVER statements are not executed. The normal flow of control through the program during execution has no effect on which WHENEVER statement is used for a particular SQL statement.
- WHENEVER statements are processed when the VISION:Results program is compiled and are applied to all SQL statements following the particular WHENEVER NOT FOUND statement until another WHENEVER NOT FOUND statement is coded.
- Place a single WHENEVER NOT FOUND statement at the top of a VISION:Results program to designate the action for all subsequent EXEC SQL statements.
- However, if you use multiple cursors in the program, then code multiple WHENEVER NOT FOUND statements before the FETCH statements directing the program to a location that will close the appropriate cursor.

WHENEVER NOT FOUND Statement

The WHENEVER NOT FOUND statement syntax is:

WHENEVER NOT FOUND { **GOTO host-label** | **CONTINUE** | **STOP** }

Where:

GOTO host-label

The GOTO host-label clause defines the target location in the VISION:Results program to which the program transfers when the "not found" condition occurs. You must define the host-label as a label or tag in the VISION:Results program.

CONTINUE

The CONTINUE keyword prevents the "not found" condition from branching to a previously specified label, which was set up from a previous WHENEVER NOT FOUND statement.

The CONTINUE keyword for a WHENEVER NOT FOUND statement is the default if a WHENEVER NOT FOUND statement is never coded in the program.

STOP

The STOP keyword is a simplified WHENEVER statement used to simplify end of data testing when used in the following form:

WHENEVER NOT FOUND STOP

The STOP causes VISION:Results to enter its normal end of job processing (including ON FINAL) when the end of data has been reached on an SQL FETCH or simplified CURSOR SELECT statement without requiring a special label to be coded.

The following is a sample program using WHENEVER NOT FOUND. The WHENEVER NOT FOUND is highlighted.

WHENEVER NOT FOUND Sample Program

The WHENEVER NOT FOUND program uses standard SQL cursor statements to DECLARE CURSOR and OPEN the cursor.

- DYLSQLCODE contains the save value as SQLCODE in a standard embedded SQL program written in another language.
- VISION:Interface translates DYLSQLCODE into a single character status variable called DYLSQLSTAT.
- The program detects the last retrieved row with the SQL WHENEVER statement.
- When the NOT FOUND condition occurs, the WHENEVER statement directs VISION:Results to the label DONE.

```
REPORT 78 WIDE
 ;
 ;
 ; PROGRAM: WHENEVER NOT FOUND PROGRAM
 ;
 ; DESCRIPTION: SINGLE TABLE DB2 EXTRACTION REPORT FROM
 ;              DSN8610.EMP USING STANDARD EMBEDDED SQL
 ;              TESTING FOR END OF DATA WITH WHENEVER.
 ;

 FILE XFILE DUMMY

 WORKAREA
   EMPNO     6 CH
   FNAME    12 CH
   LNAME    15 CH
   SALARY    5 PD 2 E

 EXEC SQL
   WHENEVER NOT FOUND GOTO DONE
 ENDEXEC

 ON ONE

   EXEC SQL
     DECLARE EMPCSR CURSOR FOR
     SELECT EMPNO, FIRSTNME, LASTNAME, SALARY
     FROM DSN8610.EMP
     ORDER BY LASTNAME
   ENDEXEC

   EXEC SQL
     OPEN EMPCSR
   ENDEXEC

 ENDONE

 EXEC SQL
   FETCH EMPCSR INTO :EMPNO, :FNAME, :LNAME, :SALARY
 ENDEXEC

 LIST EMPNO, FNAME, LNAME, SALARY

 ACCEPT
```

Figure 5-5 WHENEVER NOT FOUND Sample Program (Page 1 of 2)

```
DONE:

  EXEC SQL
    CLOSE EMPCSR
  ENDEXEC

  STOP

T1    'VISION:INTERFACE FOR DB2 - WHENEVER NOT FOUND' WITH 2 AFTER
T1+1  DYLDATE
```

Figure 5-5 WHENEVER NOT FOUND Sample Program (Page 2 of 2)

DYLSQLCODE And DYLSQLSTAT

As an alternative to using WHENEVER statements for end of cursor data processing, you can inspect two special status variables supplied with VISION:Interface, DYLSQLCODE and DYLSQLSTAT:

- DYLSQLCODE contains the save value as SQLCODE in a standard embedded SQL program written in another language.
- VISION:Interface translates DYLSQLCODE into a single character status variable called DYLSQLSTAT.
- DYLSQLCODE has the value of 100 when end of data occurs.
- DYLSQLSTAT contains a value of 'E' when end of data occurs.
- You can use either field to check for end of data.
- When the program uses the standard cursor statements, it is the responsibility of the programmer to code the close of the cursor. However, if the program uses the CURSOR SELECT statement, then VISION:Interface automatically closes the cursor.

The following is a sample program using DYLSQLSTAT. The DYLSQLSTAT statement is highlighted.

DYLSQLSTAT Sample Program

The DYLSQLSTAT program uses standard SQL cursor statements to DECLARE CURSOR and OPEN the cursor.

- These SQL statements are placed inside an ON ONE — ENDONE pair to prevent opening the cursor more than once.
- The automatic cycle of VISION:Results repeatedly executes the FETCH request which retrieves the rows of the table into VISION:Results host variables.
- The program detects the last row of the table by checking the reserved VISION:Results variable DYLSQLSTAT for a code that is not the letter Y.
- When the program fetches the last row, the program closes the cursor and terminates the VISION:Results run with the STOP command.

```
REPORT 78 WIDE
 ;
 ;
 ; PROGRAM:     DYLSQLSTAT PROGRAM
 ;
 ; DESCRIPTION: SINGLE TABLE DB2 EXTRACTION REPORT FROM
 ;              DSN8610.EMP USING STANDARD EMBEDDED SQL
 ;              TESTING FOR END OF DATA WITH DYLSQLSTAT.
 ;

 FILE XFILE DUMMY

 WORKAREA
   EMPNO     6 CH
   FNAME    12 CH
   LNAME    15 CH
   SALARY    5 PD 2 E

 ON ONE

   EXEC SQL
     DECLARE EMPCSR CURSOR FOR
     SELECT EMPNO, FIRSTNME, LASTNAME, SALARY
     FROM DSN8610.EMP
     ORDER BY LASTNAME
   ENDEXEC

   EXEC SQL
     OPEN EMPCSR
   ENDEXEC

 ENDONE

 EXEC SQL
   FETCH EMPCSR INTO :EMPNO, :FNAME, :LNAME, :SALARY
 ENDEXEC

 IF DYLSQLSTAT EQ 'Y'
   THEN LIST EMPNO, FNAME, LNAME, SALARY
```

Figure 5-6 DYLSQLSTAT Sample Program (Page 1 of 2)

```
  ELSE
    EXEC SQL
      CLOSE EMPCSR
    ENDEXEC
    STOP
ENDIF

T1    'VISION:INTERFACE FOR DB2 - DYLSQLSTAT' WITH 2 AFTER
T1+1  DYLDATE
T1+70 DYLETIME
```

Figure 5-6 DYLSQLSTAT Sample Program (Page 2 of 2)

Coding Data Management SQL Statements

Data management or full function VISION:Interface SQL statements are those statements which can not only retrieve data but also can insert data, delete data, update rows, and modify the table structure. You can use the following statements to manipulate DB2 tables from VISION:Results programs:

ALTER TABLE/INDEX	GRANT
COMMENT ON	INSERT INTO
COMMIT	LABEL
CREATE TABLE/INDEX/VIEW/SYNONYM	LOCK
DELETE FROM table-name	REVOKE
DROP TABLE/INDEX/VIEW/SYNONYM	ROLLBACK
EXPLAIN	UPDATE

See the IBM DB2 and SQL manuals for information and syntax of each statement.

The following is a sample program using data management statements. These statements are highlighted.

CREATE TEMPORARY TABLE Sample Program

The sample program creates a temporary table, TEMPL, from the DSN8610.EMP table. You will need DB2 authorization to create and update TEMPL.

```
REPORT 78 WIDE
 ;
 ;
 ; PROGRAM:     CREATE TEMPORARY TABLE
 ;
 ; DESCRIPTION: CREATE AND LOAD TEST TABLE TEMPL.
 ;              DB2 SAMPLE EMPLOYEE (EMP) TABLE IS THE INPUT.
 ;
 ;              THE TEMPL TABLE IS CODED AS AN UNQUALIFIED TABLE.
 ;
 ;
FILE XFILE DUMMY
WORKAREA
EMPNO                              6 CH
FIRSTNME_VAR                      14 CH
  REDEFINE AT    FIRSTNME_VAR
FIRSTNME_LEN                       2 BI
FIRSTNME_TEXT                     12 CH
  REDEFINE AT    FIRSTNME_TEXT
FIRSTNME                          12 CH
MIDINIT                            1 CH
LASTNAME_VAR                      17 CH
  REDEFINE AT    LASTNAME_VAR
LASTNAME_LEN                       2 BI
LASTNAME_TEXT                     15 CH
  REDEFINE AT    LASTNAME_TEXT
LASTNAME                          15 CH
WORKDEPT                           3 CH
PHONENO                            4 CH
HIREDATE                          10 CH          ;DATE COLMN
JOB                                8 CH
EDLEVEL                            2 BI
SEX                                1 CH
BIRTHDATE                         10 CH          ;DATE COLMN
SALARY                             5 PD     2
BONUS                              5 PD     2
COMM                               5 PD     2

WORKAREA
MESSAGE                           40 CH  VALUE
                      'TABLE SUCCESSFULLY CREATED AND POPULATED'

ON ONE

    EXEC SQL
      WHENEVER NOT FOUND GOTO DONE
    ENDEXEC

   EXEC SQL
      CREATE TABLE TEMPL
           ( EMPNO                          CHAR(6) NOT NULL,
             FIRSTNME                       VARCHAR(12) NOT NULL,
             MIDINIT                        CHAR(1) NOT NULL,
             LASTNAME                       VARCHAR(15) NOT NULL,
             WORKDEPT                       CHAR(3),
             PHONENO                        CHAR(4),
             HIREDATE                       DATE,
             JOB                            CHAR(8),
```

Figure 5-7 CREATE TEMPORARY TABLE Sample Program (Page 1 of 2)

```
            EDLEVEL                         SMALLINT,
            SEX                             CHAR(1),
            BIRTHDATE                       DATE,
            SALARY                          DECIMAL(9, 2),
            BONUS                           DECIMAL(9, 2),
            COMM                            DECIMAL(9, 2) )
   ENDEXEC

   EXEC SQL
      CREATE UNIQUE INDEX XEMPL1 ON TEMPL (EMPNO)
   ENDEXEC

   EXEC SQL
      CREATE INDEX XEMPL2 ON TEMPL (WORKDEPT)
   ENDEXEC

   EXEC SQL
      GRANT SELECT, INSERT, DELETE, UPDATE ON TEMPL TO PUBLIC
   ENDEXEC

  EXEC SQL
    DECLARE EMPCSR CURSOR FOR
    SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT,
           PHONENO, HIREDATE, JOB, EDLEVEL, SEX, BIRTHDATE,
           SALARY, BONUS, COMM
    FROM DSN8610.EMP
  ENDEXEC

  EXEC SQL
    OPEN EMPCSR
  ENDEXEC

ENDONE
EXEC SQL
  FETCH EMPCSR INTO :EMPNO, :FIRSTNME_VAR, :MIDINIT, :LASTNAME_VAR,
           :WORKDEPT, :PHONENO, :HIREDATE, :JOB, :EDLEVEL, :SEX,
           :BIRTHDATE, :SALARY, :BONUS, :COMM
ENDEXEC

EXEC SQL
   INSERT INTO TEMPL
     VALUES (:EMPNO, :FIRSTNME_VAR, :MIDINIT, :LASTNAME_VAR,
             :WORKDEPT, :PHONENO, :HIREDATE, :JOB, :EDLEVEL, :SEX,
             :BIRTHDATE, :SALARY, :BONUS, :COMM)
ENDEXEC

ACCEPT
DONE:
  EXEC SQL
    CLOSE EMPCSR
  ENDEXEC

  LIST MESSAGE ( ) WITH 2 BEFORE

STOP

T1    'VISION:INTERFACE FOR DB2 - CREATE TEMPORARY TABLE' WITH 2 AFTER
T1+1  DYLDATE
```

Figure 5-7 CREATE TEMPORARY TABLE Sample Program (Page 2 of 2)

COMMIT Statement

The COMMIT statement ends a unit of recovery and commits any changes to the DB2 databases. This means that if an SQL error occurs later, the automatic rollback that VISION:Interface performs does not include the changes made before the program issues the COMMIT statement.

Normally, the job commits any changes to the DB2 databases when it ends, and, therefore, the COMMIT statement is not required. You should only code the COMMIT statement in a program to ensure necessary updates are not undone when an error occurs.

If you code the COMMIT statement in a program, make it the last SQL statement.

- If you do not code the COMMIT statement as the last SQL statement, then any SQL statements that follows the COMMIT statement will not generate the proper VISION:Results instructions and fields. When the program executes the COMMIT statement and SQL statements physically follow, the program will end with one of the following errors:

  DYLD011I (DYLSQLCODE = -9011) INVALID STATUS – PREPARE

  - OR –

  DYLD013I (DYLSQLCODE = -9013) INVALID STATUS – EXECUTE

- Place the COMMIT statement in a separate routine at the end of the program and then use a VISION:Results PERFORM statement to invoke it as the following example demonstrates:

COMMIT STATEMENT Sample Program

```
REPORT 78 WIDE
 ;
 ;
 ; PROGRAM: COMMIT STATEMENT
 ;
 ; DESCRIPTION: AN EXAMPLE WHERE TO CODE A COMMIT WORK STATEMENT
 ;              IN YOUR PROGRAM.  THE COMMIT WORK STATEMENT MUST
 ;              ALWAYS BE THE LAST SQL STATEMENT PHYSICALLY CODED
 ;              IN THE PROGRAM. IT SHOULD HAVE ITS OWN ROUTINE.
 ;              A PERFORM SHOULD BE USED TO INVOKE IT.
 ;

 OPTION STRUCTURED2

 FILE EMPFILE FB 80 3120 STATUS EMP_EOF
   EMPNO     6  1 CH
   FNAME    12 11 CH
   MINIT     1 26 CH
   LNAME    15 31 CH

 WORKAREA
   EN        6 CH
   LN       15 CH
```

Figure 5-8 COMMIT STATEMENT Sample Program (Page 1 of 2)

```
EXEC SQL
  INSERT INTO TEMPL
  (EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT)
  VALUES (:EMPNO, :FNAME, :MINIT, :LNAME, 'XXX')
ENDEXEC

ON END OF INPUT

  PERFORM COMMIT

  EXEC SQL
    DECLARE EMPCSR CURSOR FOR
    SELECT EMPNO, LASTNAME FROM TEMPL
    WHERE WORKDEPT = 'XXX' ORDER BY LASTNAME
  ENDEXEC

  EXEC SQL OPEN EMPCSR ENDEXEC

  EXEC SQL FETCH EMPCSR INTO :EN, :LN ENDEXEC

  DOWHILE DYLSQLSTAT EQ 'Y'

    LIST EN, LN

    EXEC SQL FETCH EMPCSR INTO :EN, :LN ENDEXEC

  ENDDO

  EXEC SQL CLOSE EMPCSR ENDEXEC

  STOP

COMMIT:

  EXEC SQL
    COMMIT WORK
  ENDEXEC

T1    'VISION:INTERFACE FOR DB2 - COMMIT STATEMENT' WITH 2 AFTER
T1+1  DYLDATE
T1+70 DYLETIME
```

Figure 5-8 COMMIT STATEMENT Sample Program (Page 2 of 2)

Coding DDF (Distributed Data Facility) SQL Statements

DDF (Distributed Data Facility) SQL statements are the statements that set up the local DBMS (Data Base Management System) server to access data on tables that reside on a remote DBMS server. The following are DDF SQL statements:

- CONNECT
- SET
- RELEASE

See the IBM manuals for a detail explanation on the various forms of these statements and their use.

A DBMS is known to your DB2 system by its location name or ID.

- Each DBMS system has a unique location name, which is the initial qualifier of all the tables on that DBMS system. All tables have three part table names consisting of the location name ID, authorization name ID, and the table name.
- DB2 knows where the table resides by its location ID qualifier. When the table is qualified with the location ID (referred to by IBM as the DB2 Private Protocol Access), then DB2 knows to go to that remote DBMS server to get the data. However, DB2 Private Protocol Access does not support all SQL statements.

When the table does not include the location ID (that is, the table is unqualified), then DB2 uses the local servers location name as the default.

DB2 provides you the option to change the default location name for unqualified tables by means of the SQL CONNECT TO statement. On the CONNECT TO statement, you specify the location name for unqualified tables.

Using the CONNECT statement, DB2 then connects to the DBMS server with that location name. Using this method to switch from one DBMS server to another is called DRDA® (Distributed Relational Database Architecture) access.

> DRDA access works with all SQL statements and should be used instead of a three part table name.

VISION:Interface facilitates and simplifies the use of accessing distributed data by means of the DB2PARMS and COPYDB2A DD statements. These two DD statements remove the programming requirements for accessing data that resides on a remote DBMS server. Instead, the user specifies the location ID of the unqualified tables on a record belonging to the DB2PARMS and COPYDB2A DD statements.

- VISION:Interface will internally do the explicit connect to the DBMS server that owns the location ID.
- The COPYDB2A DD statement is used with COPYDB2 statements and therefore is optional.

Using the WHENEVER Statement for Errors

The WHENEVER statement specifies a location for the program to branch to when selected DB2 errors occur.

- WHENEVER statements are not executed. These statements define the location where processing continues if any subsequent SQL statement causes an error.
- A single occurrence of a WHENEVER statement at the top of a VISION:Results program provides error handling for all subsequent SQL statements, if not modified later in the program.
- The normal flow of control through the program during execution has no effect on which WHENEVER statement is used for a particular SQL statement.
- WHENEVER statements are processed when the VISION:Results program is compiled and are applied to all SQL statements following the particular WHENEVER statement without regard to other VISION:Results logic (such as ON ONE — ENDONE or IF — ENDIF).
- If there are no WHENEVER statements, VISION:Results provides automatic job termination when SQLERROR conditions are encountered. A 4-line message prints after the CONTROL TOTALS section of the programs listing and then the program halts.
- If you provide your own SQLERROR handling (that is, you have set the DYLINSTL macro DB2ERR parameter to N, or WHENEVER SQLERROR CONTINUE statement is in effect), VISION:Interface continues to process when SQLERROR occurs.

  It is the responsibility of the programmer to code the program to display the SQLERROR, to perform the DB2 rollback, and to terminate the program. The defaults do not occur.

  It is important to terminate processing in this situation; further calls to DB2 could result in serious program errors.
- You can override your own SQLERROR handling by coding a WHENEVER SQLERROR STOP statement.
- IF you do your own SQLERROR handling, use the standard SQL cursor statements (DECLARE CURSOR, OPEN, FETCH, and CLOSE) instead of the CURSOR SELECT statement to continue fetching rows.

WHENEVER Statement Syntax

The WHENEVER condition STOP is a VISION:Interface simplified SQL statement. The WHENEVER statement syntax is:

WHENEVER condition GOTO host-label

or

WHENEVER condition CONTINUE

or

WHENEVER condition STOP

Where:

WHENEVER condition

The WHENEVER condition clause traps or acknowledges the following three types of DB2 errors:

NOT FOUND	After the last row has been retrieved.
SQLWARNING	When a non-fatal DB2 error occurs.
SQLERROR	When a fatal DB2 error occurs.

GOTO host-label

The GOTO host-label clause defines the target location in the VISION:Results program to which the program transfers when one of the three specified error conditions occurs. You must define host-label as a label in the VISION:Results program.

CONTINUE

The CONTINUE command (for a particular condition) resets error-handling for the condition so that processing will not branch to the previously specified label in the event of a subsequent error.

STOP

The STOP key word is a simplified WHENEVER condition and is used in the following forms:

WHENEVER NOT FOUND STOP

The STOP causes VISION:Results to enter its normal end of job processing (including ON FINAL) when the end of data has been reached on an SQL FETCH or simplified CURSOR SELECT statement without requiring a special label to be coded just to specify STOP.

WHENEVER SQLWARNING STOP

There is no restriction on STOP used with the SQLWARNING condition, but its usefulness is limited for this situation.

WHENEVER SQLERROR STOP

Using the STOP with the SQLERROR condition, which is the default, ensures that VISION:Interface handles the SQLERROR condition.

Notes:

- WHENEVER condition STOP is a VISION:Interface simplified SQL statement.
- The simplified SQL extension WHENEVER condition STOP is identical to the syntax supported in SQL/DS, IBM's other relational data base management system which is used in VSE and VM environments.

Using DYLSQLCODE and DYLSQLSTAT for Errors

As an alternative to using WHENEVER statements for error handling, you can inspect two special status variables supplied as part of VISION:Interface: DYLSQLCODE and DYLSQLSTAT.

- DYLSQLCODE contains the same value as SQLCODE in a standard embedded SQL program written in another language.
- VISION:Interface translates DYLSQLCODE into a single character status variable called DYLSQLSTAT.
- DYLSQLCODE and DYLSQLSTAT are assigned one of the following values after each SQL statement:

DYLSQLCODE	DYLSQLSTAT	Explanation
0	Y	Valid completion of SQL statement.
100	E	End of cursor data or no data found.
>0	W	Warning condition (except 100).
<0	N	Fatal error condition.

- You can find SQLCODE error codes (in the range -999 to 999 and -20000 to -30999) in IBM DB2 and SQL manuals.
- VISION:Interface codes (in the range -9000 to -9099) are contained in the *VISION:Interface for DB2 Reference Guide*.

Chapter 6

Overviewing VISION:Interface Dynamic

For detailed information on running VISION:Interface dynamic and static programs, see the *VISION:Interface for DB2 Reference Guide*.

This chapter overviews how to run VISION:Interface dynamic programs. The topics include introducing VISION:Interface dynamic, using the JCL models. JCL models for compile and go jobs, JCL models for restore jobs, modifying the common parameters in the JCL models, and modifying the JCL to run the freeze job.

Introducing VISION:Interface Dynamic

For complete information, see the *VISION:Interface for DB2 Reference Guide*. The following is an overview.

VISION:Interface for DB2 Dynamic provides execution JCL that allows the same unmodified VISION:Results source program to:

- Run under different DB2 subsystems.
- Run using different DB2 attach facilities (CALL Attach, TSO Attach, and IMS Attach).
- Run either VISION:Interface dynamic or VISION:Interface static programs.

  For a VISION:Interface static program to run with VISION:Interface dynamic execution JCL, remove the STATSQL parameter on the OPTION statement.

  If the optional parameters, SYSTEMID and PLANID, are present on the OPTION statement, remove them also.
- Allow a compile and go or a restore of a frozen program to use different DB2 parameters (DB2 subsystem ID, DB2 plan, and DB2 table qualifiers, such as location ID and authorization ID, for unqualified tables), without the need to change the source program.

  Note: You must refreeze any programs frozen with a release prior to VISION:Interface 4.0.

The execution JCL consists of different JCL jobs or models for use with different DB2 attach facilities. The JCL models use instream procedures with modifiable parameters.

The JCL models allocate two JCL DD statements, DB2PARMS and COPYDB2A.

VISION:Interface uses the DB2PARMS and COPYDB2A DD statements to dynamically change the DB2 parameters (such as DB2 subsystem ID, plan name, location ID of unqualified tables, and authorization ID of unqualified tables) used in a VISION:Results program.

For additional information on the use of JCL models, see the *VISION:Interface for DB2 Reference Guide*.

Using the JCL Models

For complete information, see the *VISION:Interface for DB2 Reference Guide*. The following is an overview.

The product comes with a JCL model for each attach facility (CALL Attach, IMS Attach, and TSO Attach).

- These JCL models are set up for the following job types:
  - Compile and go, which are referred to as the run jobs.
  - Freeze and restore, which are referred to as the restore jobs.
- You should be able to run the programs for each job type (run, restore, or both) and under each attach facility.

The JCL models are for:

- The installer to adapt the JCL models (modify in accordance with the installation requirements) and run the sample programs to verify the VISION:Interface installation.
- The user to adapt the JCL models to run production and non-production programs.

If the program contains a large number of SQL statements, you should use the freeze and restore models. It is during the compile that there is a large usage of resources (CPU) and by freezing the program, the large usage does not occur when you run the restore.

For specific information, see the *VISION:Interface for DB2 Reference Guide*. For an overview, see the following sections:

JCL Models for Compile and Go Models Jobs

This section overviews the JCL for the compile and go JCL models (run jobs): RUNCALL, RUNDLIB, RUNDLIC, RUNDLII, and RUNTSO.

RUNCALL

VISION:Interface for DB2 Dynamic program using CALL Attach with the default explicit connection or optional implicit connection.

RUNDLIB

VISION:Interface for IMS DL/I BMP (Batch Message Processing) program with VISION:Interface for DB2 Dynamic using IMS Attach.

RUNDLIC

VISION:Interface for IMS DL/I batch program with VISION:Interface for DB2 Dynamic using CALL Attach.

RUNDLII

VISION:Interface for IMS DL/I batch program with VISION:Interface for DB2 Dynamic using IMS Attach.

RUNTSO

VISION:Interface for DB2 Dynamic program using TSO Attach.

You will select the appropriate JCL model for your job, modify the JCL, and submit the job for execution.

For the JCL models and information on modifying the JCL, see the *VISION:Interface for DB2 Reference Guide*.

JCL Models for the Restore Jobs

This section overviews the JCL for restore jobs: RESTORC, RESTORDB, RESTORDC, RESTORDI, and RESTORT.

RESTORC

VISION:Interface for DB2 Dynamic program using CALL Attach with default explicit connection or optional implicit connection.

RESTORDB

VISION:Interface for IMS DL/I BMP (Batch Message Processing) program with VISION:Interface for DB2 Dynamic using IMS Attach.

RESTORDC

VISION:Interface for IMS DL/I batch program with VISION:Interface for DB2 Dynamic using CALL Attach.

RESTORDI

VISION:Interface for IMS DL/I batch program with VISION:Interface for DB2 Dynamic using IMS Attach.

RESTORT

VISION:Interface for DB2 Dynamic program using TSO Attach.

See the *VISION:Interface for DB2 Reference Guide*, for:

- The JCL models and information on modifying the models.
- Complete information on the freeze job.

Modifying the Common Parameters in the JCL Models

For complete information, see the *VISION:Interface for DB2 Reference Guide*.

The following summarizes the modifications common to all the JCL models.

1. Change the JOB statement in accordance with your installation requirements.
2. Make appropriate changes to the default symbolic parameters in the procedure.
3. Specify the DB2PARMS DD statement.

   If the authorization ID and location ID are not hard coded, the following are passed to DB2 by means of the DB2PARMS DD statement, regardless of the attach facility being used.

   - Authorization ID
   - Location ID

   If the authorization ID and location ID are not hard coded and if the CALL Attach explicit connection is being used, the following are passed to DB2 by means of the DB2PARMS DD statement:

   - Subsystem ID
   - Plan name
   - Authorization ID
   - Location ID

   Notes:

   - *VISION:Interface* dynamic requires the user to have SYSADM authority to use the authorization ID.
   - VISION:Interface static does not have this restriction.

4. Run the job.

Modifying JCL Model to Run the Freeze Job

For complete information, see the *VISION:Interface for DB2 Reference Guide*.

This section overviews the modifications you will make to the freeze JCL model for production and non-production jobs.

1. Change the JOB statement in accordance with your installation requirements.
2. Make appropriate changes to the default symbolic parameters.
3. CALL Attach and TSO Attach use the same freeze and generate VISION:Results code that is interchangeable, in that you can use the frozen module during the restore to run either CALL or TSO Attach restore JCL.

   To freeze a program with IMS Attach generated code, uncomment the DDITV02 DD statement (which is commented out) in the job.

   - If you freeze a program using IMS Attach, you must use the restore JCL that uses IMS Attach (RESTORDB or RESTORDI).
   - Conversely, if you freeze a program as either CALL Attach or TSO Attach (DDITV02 DD statement must be commented out), you must use the restore JCL for CALL Attach (RESTORC or RESTORDC) or for TSO Attach (RESTORT).

4. Run the FREEZE job.

Chapter 7

Overviewing VISION:Interface Static

For detailed information on running VISION:Interface dynamic and static programs, see the *VISION:Interface for DB2 Reference Guide*.

This chapter overviews how to run VISION:Interface static programs. The topics include introducing VISION:Interface static, overviewing of running static SQL, using the program preparation jobs and JCL models, JCL models for program preparation (freeze, prepare, and bind) jobs, JCL models for the restore jobs, and modifying the common parameters in the JCL models.

Note: You freeze (CNTLFREZ), prepare (CNTLPREP), and bind (CNTLBIND) all static programs.

Introducing VISION:Interface Static

For complete information, see the *VISION:Interface for DB2 Reference Guide*. The following is an overview.

VISION:Interface for DB2 Static provides execution JCL that allows the same unmodified VISION:Results source program to:

- Run under different DB2 subsystems.
- Run using different DB2 attach facilities (CALL Attach, TSO Attach, and IMS Attach).
- Run either VISION:Interface static or VISION:Interface dynamic source programs.

  For a VISION:Interface dynamic program to run with VISION:Interface static execution JCL, add the STATSQL parameter on the OPTION statement.

  If the optional DYNAMDB2 parameters are present on the OPTION statement, remove them. The DB2 system ID parameter, SYSTEMID, and the plan parameter, PLANID, are not required on the OPTION statement.
- Allow a compile and go program or a restore of a frozen program to use different DB2 parameters (DB2 subsystem ID, DB2 plan, and DB2 table qualifiers, such as location ID and authorization ID, for unqualified tables), without the need to change the source program.

Note: You must refreeze any programs frozen with a release prior to VISION:Interface 4.0.

The execution JCL consists of different JCL jobs or models for use with different DB2 attach facilities. The JCL models use instream procedures with modifiable parameters.

The freeze JCL model allocates the JCL DD statement COPYDB2A.
The restore JCL model allocates the JCL DD statement, DB2PARMS.

VISION:Interface uses the DB2PARMS and COPYDB2A DD statements to dynamically change the DB2 parameters (such as DB2 subsystem ID, plan name, location ID of unqualified tables, and authorization ID of unqualified tables) used in a VISION:Results program.

Additional information on the use of JCL models follows the next section.

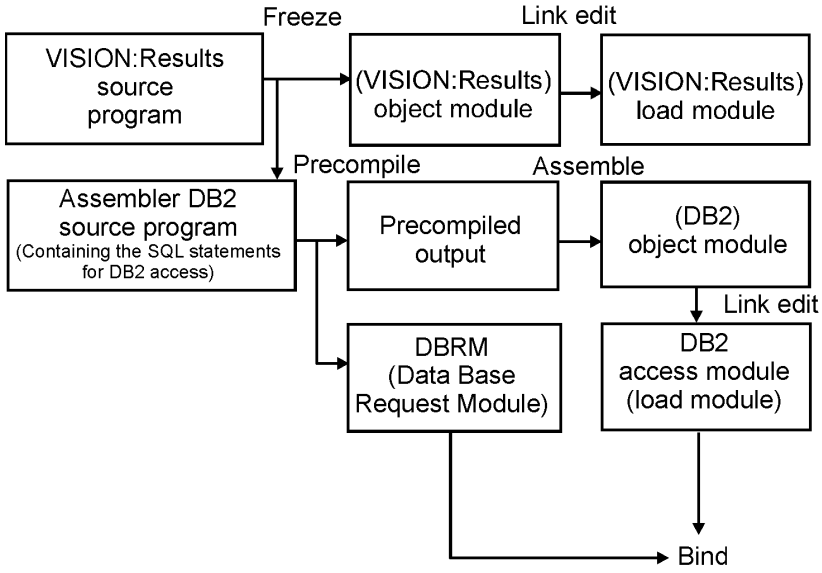
Overview of Running Static SQL with VISION:Results

For complete information, see the *VISION:Interface for DB2 Reference Guide*.
The following is an overview.

To execute VISION:Results with embedded SQL statements in static SQL, you will:

- Freeze the VISION:Results program.
- Prepare the generated Assembler DB2 source program.
- Bind the DBRM.
- Run the restore of the frozen VISION:Results program.

The following figure shows the process:



1. Freeze the VISION:Results program with additional DD statements.

You will freeze the VISION:Results program with two additional DD statements:

- STAT280 DD statement
- SQLTEMP DD statement

The freeze process calls VISION:Interface. VISION:Interface uses the STAT280 and SQLTEMP DD statements to generate a tailored Assembler DB2 source program, which will contain the SQL statements originally coded in the VISION:Results program. Next, VISION:Interface replaces the SQL statements in the VISION:Results program with CALL statements to the Assembler DB2 program.

The next step prepares the Assembler DB2 program to run in static SQL mode, becoming the Assembler DB2 database access module for the VISION:Results program.

You will include the following OPTION statement in the VISION:Results program to signify static SQL processing:

OPTION FREEZE wwwwwwww STATSQL xxxxxxxx

where:

FREEZE wwwwwwww	Specifies the VISION:Results program name.
STATSQL xxxxxxxx	Specifies the DB2 database access module name.

STAT280 DDNAME

The process places the generated Assembler language DB2 database access module in the Assembler source library defined by the STAT280 DDNAME.

- You will specify the member name by coding its name immediately after STATSQL parameter on the OPTION statement. This member is the DB2 database access module name.
- You will keep the STAT280 DD statement library and its members for as long as you use VISION:Results programs.

SQLTEMP DDNAME

The process uses SQLTEMP DDNAME which you allocate to a temporary file with a logical record length of 80 bytes and a block size of 1600 bytes. You can delete the SQLTEMP file when the freeze step completes.

Multiple Reports/Requests

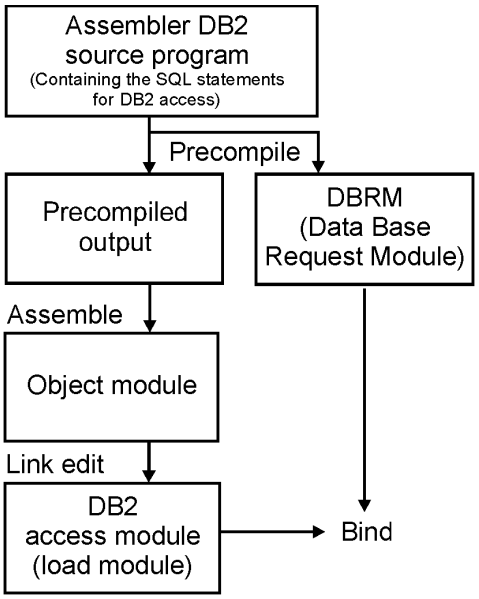
When you freeze multiple reports/requests with SQL statements present in the different reports, then, for each report that has SQL statements, you will need to specify an OPTION statement with the STATSQL parameter followed by a unique DB2 database access module name.

Only the first report will have the FREEZE parameter.

Each STATSQL parameter generates an Assembler DB2 source program which produces a DB2 database access module which will be linked to one VISION:Results frozen module.

2. Preparing the Assembler DB2 source program.

To prepare the Assembler DB2 source program as a static DB2 access module, you will:



Note: DBRM is the acronym for Data Base Request Module.

When the installation completes there will be a DBRM for each access module.

a. Precompile the Assembler DB2 source program from the STAT280 library (generating precompiled output and a DBRM).

b. Assemble the precompiled output generating an object module.

c. Link edit the object module generating the DB2 access module (load module).

d. If you are using the CALL Attach implicit connection, then run the CNTLLNK2 job.

e. If you are running an IMS BMP (Batch Message Processing) program, then run the CNTLLNK3 job.

If multiple Assembler DB2 source programs are generated from a multiple reports/requests program, then each Assembler DB2 source program will be prepared as a static DB2 access module.

3. Bind the DBRM.

You will bind the DBRM, that was generated in the precompile step, as a package with a collection ID that has been or will be included in the package list of a plan.

If a multiple reports/requests program generates multiple Assembler DB2 source programs, then you will bind each Assembler DB2 source program DBRM as a package.

4. Running the restore of the frozen VISION:Results program.

You will run the restore of the frozen VISION:Results program by submitting an:

OPTION RESTORE wwwwwwww

where:

RESTORE wwwwwwww	Specifies the name of the frozen VISION:Results program.

If frozen program has multiple reports/requests, then list the number of reports after the program name on the OPTION RESTORE statement as follows:

OPTION RESTORE TEST01 3 ←===Three reports

Using the Program Preparation Jobs and JCL Models

For complete information, see the *VISION:Interface for DB2 Reference Guide*. The following is an overview.

The product comes with program preparation jobs (freeze, prepare, and bind) and JCL models for each attach facility (CALL Attach, TSO Attach, or IMS Attach).

- The program preparation jobs:
  - Freeze (CNTLFREZ) the VISION:Results program and create the Assembler source program.
  - Prepare (CNTLPREP) the Assembler source program.

    If you are using CALL Attach implicit connection, then, after running CNTLPREP, you would run the CNTLLNK2 job.

    If you are running an IMS BMP (Batch Message Processing) program, then, after running CNTLPREP, you would run the CNTLLNK3 job.
  - Bind (CNTLBIND) the Assembler source program.
- The JCL models are set up for running the restore on the VISION:Results program with JCL provided for each attach facility.
- You should be able to run the programs for each job type and under each attach facility.

To run a new or changed VISION:Interface program, you must rerun the freeze, prepare, and bind jobs, then run the VISION:Interface execution JCL. For subsequent runs, you run just the VISION:Interface execution JCL (until you change the program).

The program preparation job and JCL models are for:

- The installer to adapt the JCL models (modify in accordance with the installation requirements) and run the sample test programs to verify the VISION:Interface installation.
- The user to adapt the program preparation jobs and JCL models to run production and non-production programs.

For specific information, see the *VISION:Interface for DB2 Reference Guide*. For an overview, see the following sections:

- JCL Models for the Program Preparation (Freeze, Prepare, and Bind) Jobs
- JCL Models for the Restore Jobs
- Modifying JCL Model to Run the CNTLFREZ Job
- Modify the Common Parameters in the JCL Models

JCL Models for the Program Preparation (Freeze, Prepare, and Bind) Jobs

The section overviews the JCL models for program preparation jobs.

CNTLFREZ

Required for every VISION:Interface program.

CNTLPREP

Required for every VISION:Interface program.

CNTLLNK2

Optional, required only if programs require CALL Attach implicit connection.

CNTLLNK3

Optional, required only if you are using
VISION:Interface for IMS DL/I BMP programs with VISION:Interface for DB2.

CNTLBIND

Required for every VISION:Interface program.

You will select the appropriate JCL model for your job, modify the JCL, and submit the job for execution.

For the JCL models and information on modifying the JCL, see the *VISION:Interface for DB2 Reference Guide*.

JCL Models for the Restore Jobs

This section overviews the restore JCL models (restore jobs): RESTORC, RESTORDB, RESTORDC, RESTORDI, and RESTORT.

RESTORC

VISION:Interface for DB2 Static program using CALL Attach with default explicit connection or optional implicit connection.

RESTORDB

VISION:Interface for IMS DL/I BMP (Batch Message Processing) program with VISION:Interface for DB2 Static using IMS Attach.

RESTORDC

VISION:Interface for IMS DL/I batch restore program with VISION:Interface for DB2 Static using CALL Attach.

RESTORDI

VISION:Interface for IMS DL/I batch program with VISION:Interface for DB2 Static using IMS Attach.

RESTORT

VISION:Interface for DB2 Static program using TSO Attach.

You will select the appropriate JCL model for your job, modify the JCL, and submit the job for execution.

For the JCL models and information on modifying the JCL, see the *VISION:Interface for DB2 Reference Guide*.

Modifying JCL Model to Run the CNTLFREZ Job

For complete information, see the *VISION:Interface for DB2 Reference Guide*.

This section overviews the modifications you will make to the freeze JCL model for production and non-production jobs.

1. Change the JOB statement in accordance with your installation requirements.
2. Make appropriate changes to the default symbolic parameters.
3. CALL Attach and TSO Attach use the same freeze and generate VISION:Results code that is interchangeable, in that you can use the frozen module during the restore to run either CALL or TSO Attach restore JCL.

   To freeze a program with IMS Attach generated code, uncomment the DDITV02 DD statement (which is commented out) in the job.

   - If you freeze a program using IMS Attach, you must use the restore JCL that uses IMS Attach (RESTORDB or RESTORDI).
   - Conversely, if you freeze a program as either CALL Attach or TSO Attach (DDITV02 DD statement must be commented out), you must use the restore JCL for CALL Attach (RESTORC or RESTORDC) or for TSO Attach (RESTORT).

4. Run the CNTLFREZ job.

Modify the Common Parameters in the JCL Models

For complete information, see the *VISION:Interface for DB2 Reference Guide*.

The following summarizes the modifications common to all the JCL models.

1. Change the JOB statement in accordance with your installation job requirements.
2. Make appropriate changes to the default symbolic parameters.
3. Specify the DB2PARMS DD statement.

   If the authorization ID and location ID are not hard coded, the following are passed to DB2 by means of the DB2PARMS DD statement, regardless of the attach facility being used.

   - Authorization ID
   - Location ID

   If the authorization ID and location ID are not hard coded and if the CALL Attach explicit connection is being used, the following are passed to DB2 by means of the DB2PARMS DD statement:

   - Subsystem ID
   - Plan name
   - Authorization ID
   - Location ID

4. Run the job.

Index

Symbols

Numerics

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Z