DB2 PROGRAM PREPARATION

- **SQL STATEMENT FORMATS (DELIMITERS)**
- **HOST VARIABLES**
- **DB2 DATATYPES Vs HOST VARIABLES**
- **COMPLIER DOESN'T UNDERSTAND SQL**
- **DB2 PROGRAM PREPARATION**
- **DCLGEN COMMAND**
- **DBRM**
- **COMPILATION AND LINK-EDIT**
- **BIND**
- **PACKAGE**
- **COLLECTION**
- **APPLICATION PLAN**
- **ISOLATION**
- **EXPLAIN**

Figure: 1.1 DB2 Program Preparations

SQL STATEMENT FORMATS (DELIMITERS)

COBOL

```
EXEC SQL
     UPDATE
     WHERE
END-EXEC.
```

PL/1

```
EXEC SQL
     UPDATE
     WHERE
;
```

Figure: 1.2 SQL Statement Formats (Delimiters)

Notes:

Whatever the language used, SQL statement will always have to be enclosed between delimiters so that they can be easily spotted and replaced by something the compilers will be able to understand.

Different delimiters are used for different languages. These are the standard delimiters used in COBOL and PL/1.

DB2 DATA TYPES - CHARACTER DATA TYPE

- CHAR(n) / CHARACTER(n)

  Fixed length string between 1 and 254 bytes

- VARCHAR(n)

  Variable length string

- LONG VARCHAR

  Variable length string

Figure 1.3 DB2 Data types CHARACTER

GRAPHIC DATA TYPE

- GRAPHIC (n)

  Fixed length graphic string (from 1 to 127)

- VARGRAPHIC(n)

  Variable length graphic string

- LONG VARGRAPHIC

  Variable length graphic string

Figure 1.4 DB2 Data types GRAPHIC

Notes :

DB2 also supports DBCS (Double Byte Character Set) data like GRAPHIC

NUMERIC DATA TYPE

- SMALLINT

  Halfword (2 Bytes)
  Whole number between + and –32K

- INTEGER / INT

  Full word (4 bytes)
  Whole number between + and – 2 GB

- NUMERIC(N)

  Max 31 digits

- DECIMAL(P,S) / DEC(P,S)

  Max 31 digits

- FLOAT(X) / REAL / DOUBLE PRECISION / FLOAT

  Fullword or double word (4 bytes)
  Floating point number between 5.4E-79 and 7.25+75

Figure: 1.5 Host Variables

Notes:

When calculations have to be performed Numeric Data Type have to be used.

DATE / TIME DATA TYPES

DATE	4 Bytes (YYYYMMDD)
TIME	3 Bytes (HHMMSS)
TIMESTAMP	10 Bytes (YYYYMMDDHHMMSSNNNNNN)

Figure: 1.6 DB2 DATE / TIME Data Type

Notes:

DB2 does the necessary verifications on the data that is entered in these types. They can be manipulated using SQL scalar functions like DAYS, MONTH, YEAR .

NULL ATTRIBUTE

NULL is a special value indicating the absence of a value.

Not consider for the column function evaluations (AVG) except COUNT.
Two nulls are not considered as equal, except for

- GROUP BY
- ORDER BY

Uniqueness of a column unless you use UNIQUE WHERE NOT NULL

Figure: 1.7 Null attribute

Notes:

DB2 considers two null values as equal when it enforces the uniqueness of columns. You can change this by using the UNIQUE WHERE NOT NULL parameter.

NULL ATTRIBUTE (Cont….)

CHOICE TO BE MADE WHILE DEFINING COLUMNS:

- (Default: NULLS allowed)
- NOT NULL
- WITH DEFAULT

Figure: 1.8 Null Attribute (Continued)

Notes:

The NULL attribute of a column is an important choice that will have to be made in cooperation with DBA.

(NOT NULL) WITH DEFAULT

- Same as NOT NULL
- System default values
  - ZERO for NUMERIC columns
  - BLANKS for fixed length CHARACTER columns
  - Blanks for fixed length GRAPHIC columns
  - Zero length string for variable length CHARACTER and GRAPHIC columns.
  - Current date for DATE data type.
  - Current time for TIME data type.
  - Current timestamp for TIMESTAMP data type.

Figure: 1.9 Not Null with DEFAULT

USER DEFINED DEFAULT VALUES

- DEFAULT CAN BE EITHER
  - Constant
  - USER (Special Register)
  - CURRENT SQLID (Special Register)
  - NULL

Figure: 1.10 User defined default values

Nores:

It is possible to provide your own default value. It will be used by DB2 if you fail to supply a value in the INSERT statement.

HOST VARIABLES

Host language variables that can be referenced in a SQL statement to supply values to DB2 or to receive values from DB2.
They must be preceded by **‘:’** when referenced in a SQL statement.

Example

```
EXEC SQL
        SELECT PHONENO INTO :PHONENO
        FROM EMP
        WHERE EMPNO = :EMPNO
END-EXEC.
```

Figure: 1.11 Host Variables

Notes:

Host variables will be used by DB2 to:

Retrieve data and put it in to host variable for use by application program
INSERT data in to a table or to update it from the data in the host variable.
Evaluate a WHERE or HAVING clause using the data in the host variable.
However, host variable cannot be used to represent a table, view or a column.
The colon(**:**) is necessary to distinguish a host variable from a column name.

HOST VARIABLES (Cont….)

Example-COBOL

```
DATA DIVISION.
    …
    WORKING-STORAGE SECTION.
    01 IOAREA.
            02 INPEMPNO PIC 9 (6).
            02 INPNAME    PIC X (15).
    PROCEDURE DIVISION.
            MOVE ‘HIGGINS’ TO INPNAME.
            MOVE 00260 TO INPEMPNO.
            EXEC SQL
                    UPDATE EMP
                    SET LASTNAME =:INPNAME
                    WHERE EMPNO =:INPEMPNO
            END-EXEC.
```

Figure: 1.12 Host Variables (Cont….)

Notes:

DB2 DATA TYPES VS HOST VARIABLES

DB2	**COBOL**	**PL/1**
SMALLINT	PIC S9 (4) COMP	BIN FIXED (15)
INTEGER	PIC S9 (9) COMP	BIN FIXED (31)

Figure: 1.13 DB2 Data types vs. Host Variables

Notes:

This table shows how to define a host variable in a program to match a given DB2 data type. It is important to use the correct data type in our program. Although DB2 will, in many cases, convert the data type to make the data type match, this could lead to bad performance and should be avoided.

HOST STRUCTURES

COBOL

```
01      PHONEEMP.
        05      EMPNO      PIC X (06).
        05      FIRSTNAME.
                        49      FIRSTNAME-LEN          PIC S9 (4) COMP.
                        49      FIRST NAME-TEXT        PIC X (12).
        05      MIDINIT    PIC X (01).
        05      LASTNAME.
                        49      LASTNAME-LEN           PIC S9 (4) COMP.
                        49      LASTNAME-TEXT          PIC X (15).
        05      WORKDEPT           PIC X (03).
        05      PHONENO            PIC X (04).
```

PL/1

```
DCL  1      PHONEEMP
     5      EMPNO              CHAR (06),
     5      FIRSTNAME          CAHR (12) VARYING,
     5      MIDINIT            CAHR (01),
     5      LASTNAME           CAHR (15) VARYING,
     5      WORKDEPT           CHAR         (03),
     5      PHONENO            CHAR (04);
```

Figure: 1.14 Host Structure

Notes:

A host structure is a group of host variables referred to by a single name in an SQL statement. They are defined by statements of the host language.

SELECTION OF A SINGLE ROW

Using individual host variables:

```
 EXEC SQL
         SELECT    EMPNO, FIRSTNAME, MIDINIT, LASTNAME,
                   WORKDEPT, PHONENO
         INTO   :EMPNO, : FIRSTNAME, :MIDINIT, :LASTNAME,
                : WORKDEPT, : PHONENO
         FROM         EMP
        WHERE   EMPNO= :INPEMPNO
END-EXEC.
```

Using the host structure:

```
EXEC SQL
        SELECT *   INTO  :DCLEMP
        FROM     EMP
        WHERE   EMPNO =:INPEMPNO
END-EXEC.
```

SELECT ……….INTO…FROM.

is valid only for Selects that return one row

Figure: 1.15 Selection of One Row

Notes:

The examples shows how a SELECT statement can be written (to retrieve a single row), both by referring to individual host variables or by referring to a structure.

COMPILERS DON'T UNDERSTAND SQL

Who will get rid of the ‘EXEC SQL’ so that your program can be compiled?

Answer:

A PRECOMPILER will replace the EXEC SQL statements by CALLS
It will also verify whether

Your SQL statements are correct
Your host variables match the DB2 data types
But it doesn’t access DB2
You must include EXEC SQL DECLARE TABLE

Example

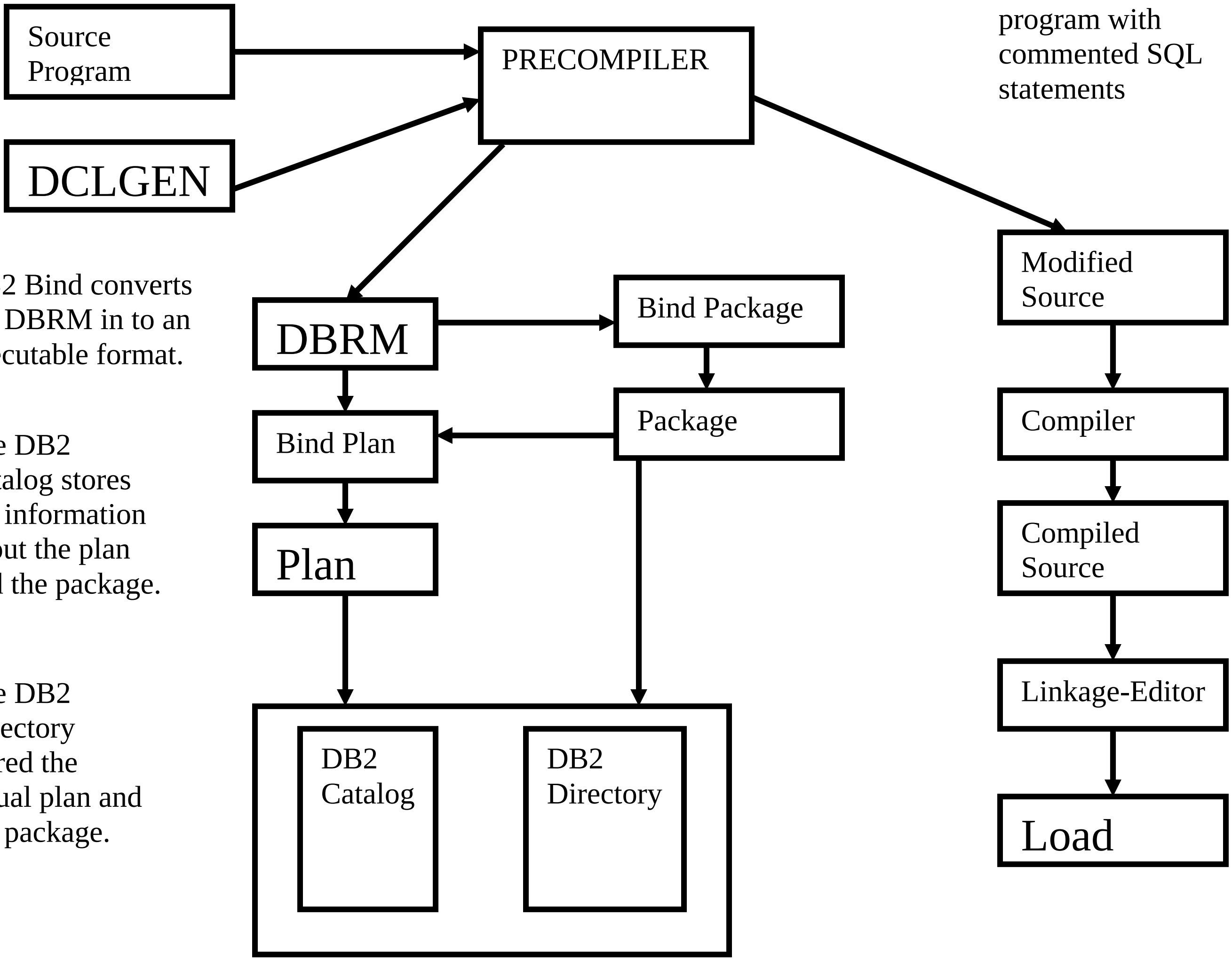
```
EXEC SQL
              DECLARE EMP TABLE
             (EMPNO CHAR (6) NOT NULL,
             LASTNAME  CHAR(10))
END-EXEC.
```

Figure: 1.16 Compiler doesn’t understand SQL

Notes:

Besides allowing the precompiled to verify the compatibility of host variables versus DB2 column data types (the precompiled doesn’t access DB2, so YOU have to provide the DB2 data type info), it is good practice to use DECLARE TABLE, because it makes your programs more readable and more easily maintainable.

DB2 PROGRAM PREPARATION STEPS



The DB2 Pre-Compiler produces a DBRM and a modified source program with commented SQL statements

DB2 Bind converts the DBRM in to an executable format.

The DB2 Catalog stores the information about the plan and the package.

The DB2 Directory stored the actual plan and the package.

Figure 1.17 DB2 Program Preparation Steps

PRECOMPILATION

The DB2 application program contains COBOL code with SQL statements embedded in it. The COBOL compiler will not be able to recognize the SQL statements and will give compilation errors. So before running the COBOL compiler, the SQL statements must be removed from the source code. Recompilation does the following:

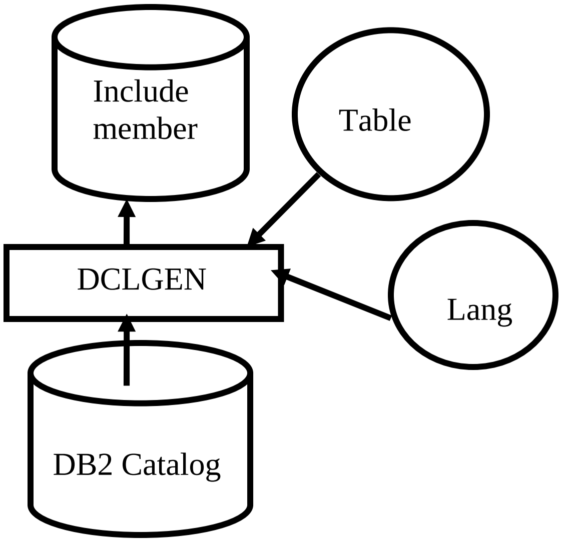
- **Searches for and expands DB2 related INCLUDE members.**
- **Searches for SQL statements in the body of the program's source code.**
- **Creates a modified version of the source program in which every SQL statement in the program is commented and replaces with a CALL to the DB2 runtime interface module, along with applicable parameters.**
- **Extracts all the SQL statements and places them in a Database Request Module (DBRM).**
- **Places A Timestamp token in the modified source and the DBRM to ensure that these two items are inextricably tied.**
- **Reports on the success of failure of the precompiled process.**

Figure 1.18 Recompilation

DCLGEN COMMAND

DCLGEN (Declaration Generator)

- **Option 2 in DB2**
- **Produces**
  - **SQL DECLARE TABLE statement**
  - **Host language variable declaration for a table or view**



(DECLARE table SQL)

```
EXEC SQL
DECLARE EMPTABLE
(EMPNO CHAR (6) NOTNULL,
 LASTNAME CHAR(15))
END-EXEC.
```

(Data declaration source code)

```
01  DCLEMP.
       05 EMPNO PIC X (6),
       05 LASTNAME  PIC X(15).
```

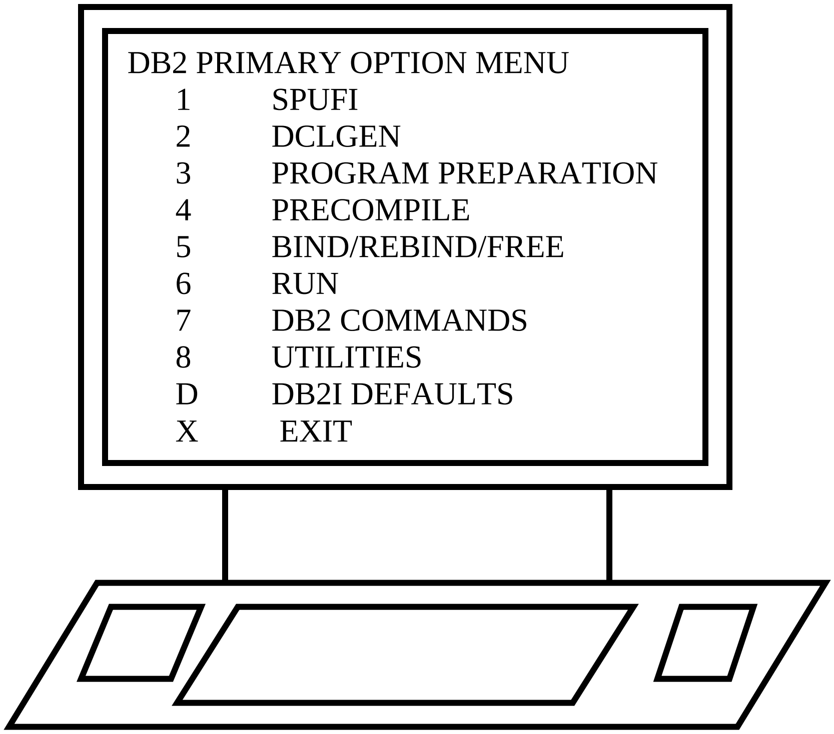
Figure1.19 DCLGEN command

Notes:

The DCLGEN tool provided with DB2I produces a COBOL copybook, which contains SQL DECLARE TABLE along with the WORKING-STORAGE host variable definitions for each column of the table. Input will be the language (COBOL, PL/1) and the table name.

DCLGEN COMMAND (Cont….)

This is the interactive DB2 environment to prepare the program for execution.



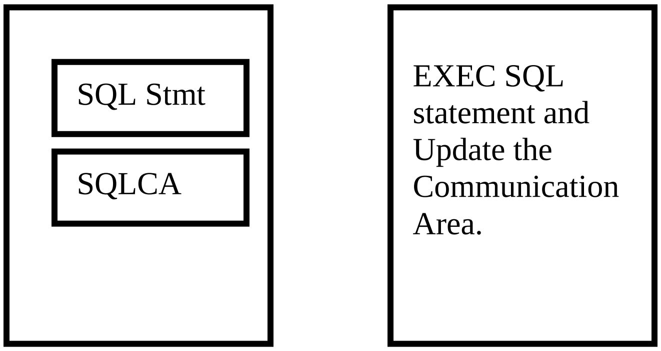
```
DB2 PRIMARY OPTION MENU
    1        SPUFI
    2        DCLGEN
    3        PROGRAM PREPARATION
    4        PRECOMPILE
    5        BIND/REBIND/FREE
    6        RUN
    7        DB2 COMMANDS
    8        UTILITIES
    D        DB2I DEFAULTS
    X         EXIT
```

Figure 1.20 DCLGEN command (Cont….)

Notes:

When DCLGEN command is issued, DB2 read the catalog to fetch the column definition for the table, and will generate an "include" member with the DECLARE TABLE and the host structure. DCLGEN command can be eliminated by hard coding in the application program. But it is a good practice to run the DCLGEN command for every table that will be embedded in a COBOL program. Then every program that accesses that table can INCLUDE that generated copybook. This reduces a lot of unnecessary coding. DCLGEN will generate the host variables with the same name as the column name and if the program uses two tables which have common column names, then edit the copybook and change the names.

SQL INCLUDE STATEMENT SQLCA



EXEC SQL
INCLUDE SQLCA
END-EXEC.

Program after compilation.

```
DCL     1   SQLCA.
        2   SQLCAID CHAR (8).
        2   SQLCABC.
        2   SQLCODE.
        2   .
        2   .
        2   .
        2   SQLSTATE CHAR (5).
```

Figure 1.21 SQL Include statement SQLCA

Notes:

SQLCA fields are updated by DB2; the application program must check value. SQCA SQLCODE and SQLSTATE fields used to check result other field used for more detailed on condition. In order to know what happened in the other side in DB2 we must provide DB2 with program storage where it can set a return code and other information DB2 wants to communicate.

DBRM

Data **B**ase **R**equest **M**odule contains the program’s source SQL statements.

A DBRM

- **Contains the extracted, parsed SQL source.**
- **Is stored as a member in a partitioned dataset.**
- **One member created per precompile.**
- **Will become Input to BIND.**

Figure 1.22 SQL DBRM

Notes:

DBRM (Cont….)

One DBRM corresponds to exactly one source module. The SQL statements in the source program will be replaced by CALL to module DSNHLI statement with the following parameters.

- **DBRM name (SQLPROGN)**
- **Timestamp (SQLTIME)**
- **Statement number (SQLSTNUM)**

Other parameters are

- **Address of host variables.**
- **Address of SQLCA.**

At this stage, the two components, (DBRM and modified source), will part and won't see each other again until program execution. Therefore the CALL must include necessary information for DB2 to be able to locate the access path needed to execute the SQL statement associated with the CALL.

The information about the DBRM that have been bound in to the application plans and packages is stored in the SYSIBM.SYSDBRM in the DB2 catalog table. If a DBRM is created and is not bound cannot be referenced from this table.

When DBRM is bound to a plan, all the SQL statements are placed in to the SYSIBM.SYSTMTDB2 catalog table. When a DBRM is bound in to a package all the SQL statements are placed in to the SYSIBM.SYSPACKSTMT table.

Figure 1.23 SQL DBRM (Cont….)

COMPILATION AND LINK-EDIT

- After pre-compilation the program has to be COMPILED and LINK-EDITED. This can be done using:
- DB2I panels.
- JCL.
- PL/1 PRECOMPILER can precede the DB2 PRECOMPILER.
- CICS COMMAND TRANSTLATOR may have to be invoked.

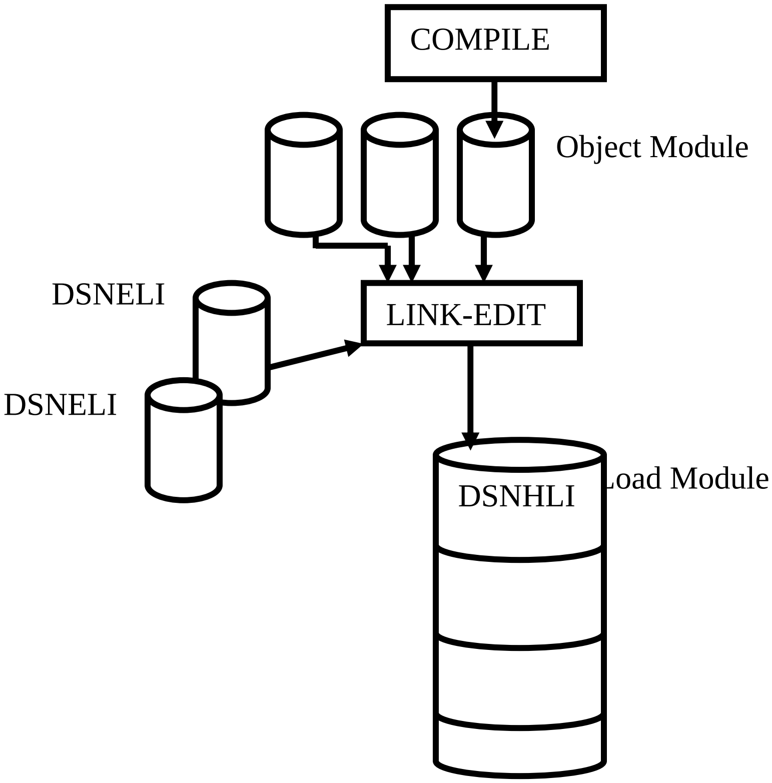


Figure 1.24 Compilation and link-Edit

Notes:

The modified program is now ready to be compiled and link-edited. The link-edit will have to include the necessary modules for the call to work properly. The appropriate LANGUAGE INTERFACE must be included. The program that has been prepared here will run in one of the many execution environments like TSO, CICS, IMS. Depending on the environment, a different interface module will have to be included in the link-edit stop. The name of the module will vary, but they will all have an entry point DSNHLI.

BIND PACKAGE

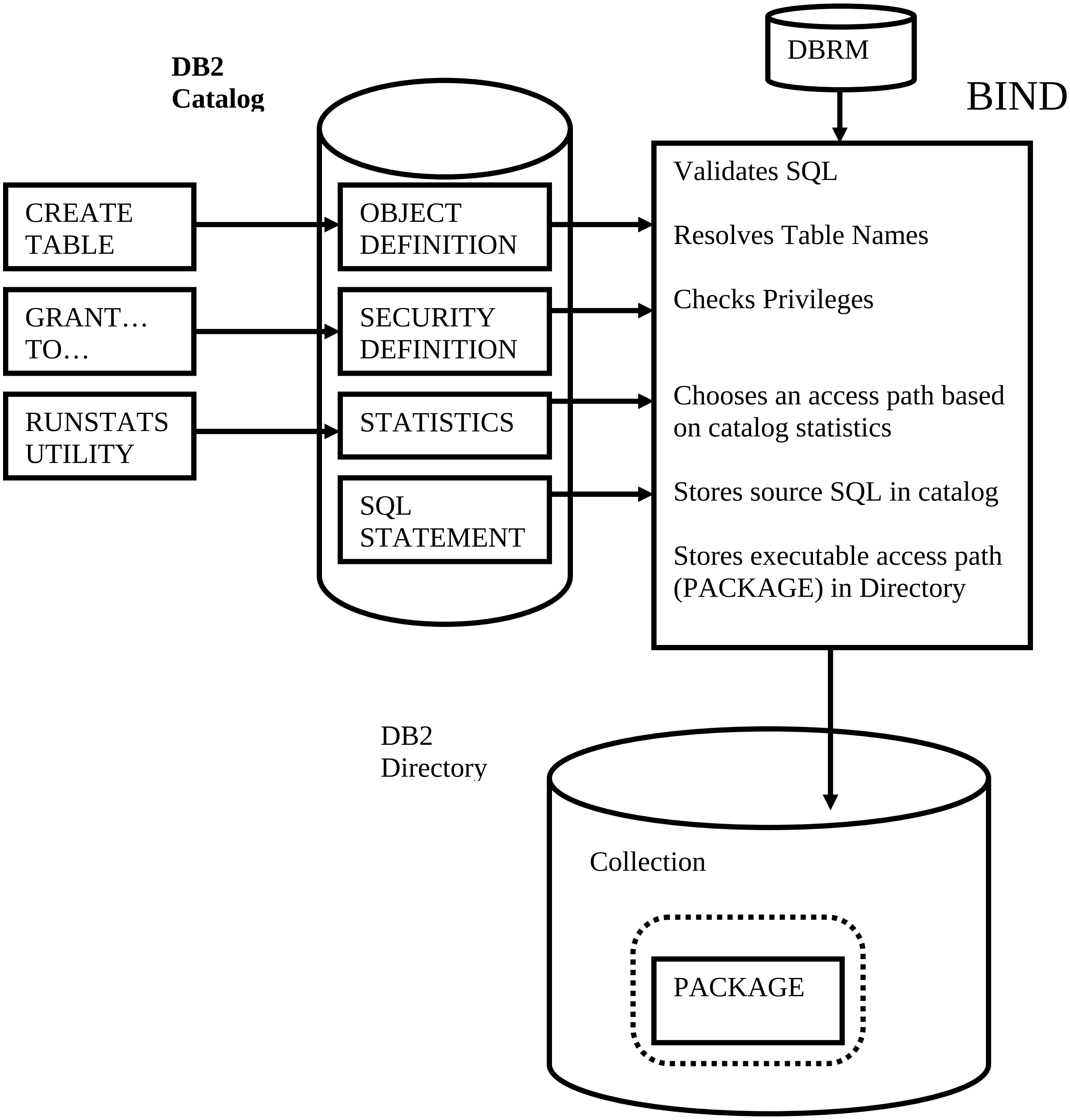


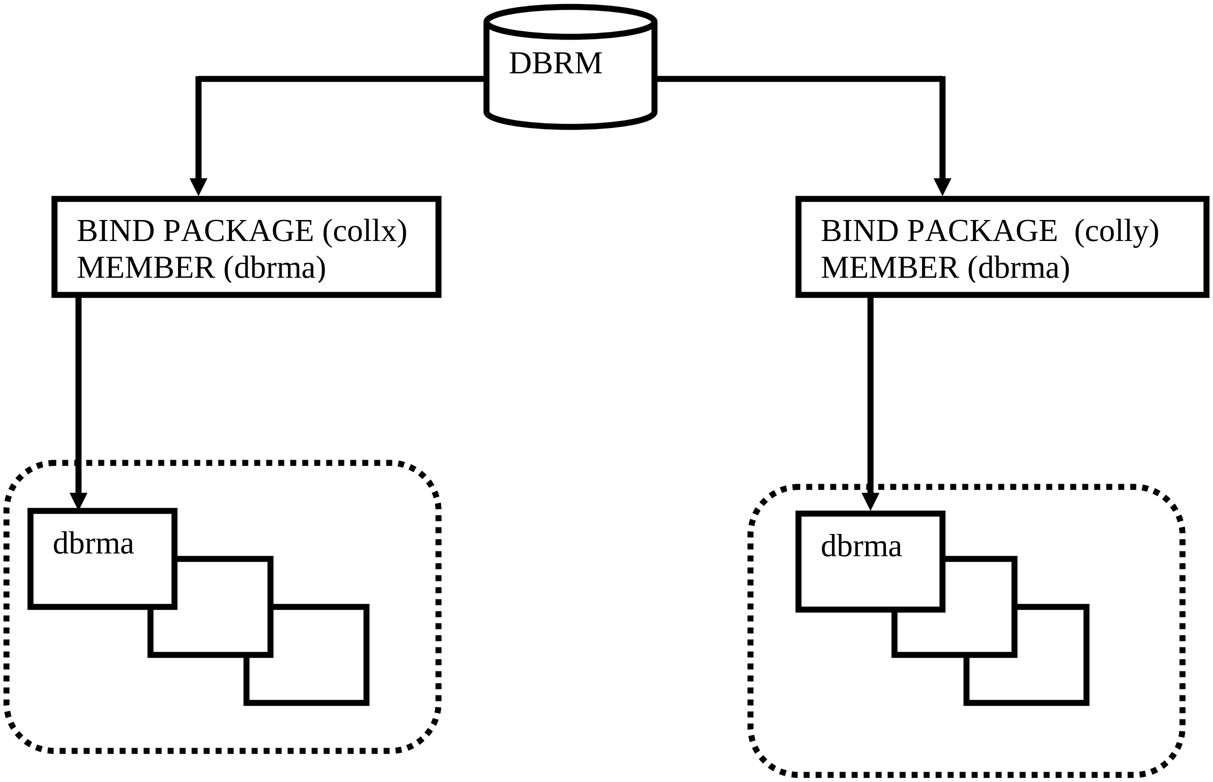
Figure 1.25 BIND PACKAGE

BIND PACKAGE (Cont….)

- **Validating: Checking whether the table exist.**
- **“Resolving user names”: Completing table names with their owners incase they were omitted.**
- **Authority checking is performed to make sure that the BIND has the authority to create a program to access the data as required.**
- **Access path selection consists of evaluating a number of different access paths and calculating their costs. The cheapest one will be retained. Those cost estimates are based on statistics, which are stored in the catalog. The RUNSTATS utility must be run to keep those statistics up-to-date.**
- **The executable access code is stored in the directory, and the source SQL statements are stored in the DB2 catalog. They may be needed if the access path were to be evaluated at some later time.**
- **BIND PACKAGE runs in TSO (Online or Batch)**

Figure 1.26 BIND PACKAGE (Cont….)

PACKAGE



Package Name = Collection_id**.**Package_id

Figure 1.27 PACKAGES

Notes:

PACKAGE (Cont….)

- **A package is a single bound DBRM with optimized access path. By using packages the table access logic is packaged at a lower level for granularity at the package or program level.**

- **To execute a package it must be first be included in the package list of a plan. Package can never be directly executed; they are only executed when the plan in which they are contained is executed.**

- **A plan can consist of one or more DBRMs, one or more packages, or a combination of packages and DBRMs.**

- **Package information is stored in its own DB2 catalog tables. When a package is bound, DB2 reads the following catalog tables:**

SYSIBM.SYSCOLDIST, SYSIBM.SYSCOLUMNS, SYSIBM.SYSFIELDS, SYSIBM.SYSINDEXES, SYSIBM.SYSPACKAGES,SYSIBM.SYSTABLES SYSIBM.SYSPACKAUTH,SYSIBM.SYSTABLESPACE and SYSIBM.SYSUSERAUTH

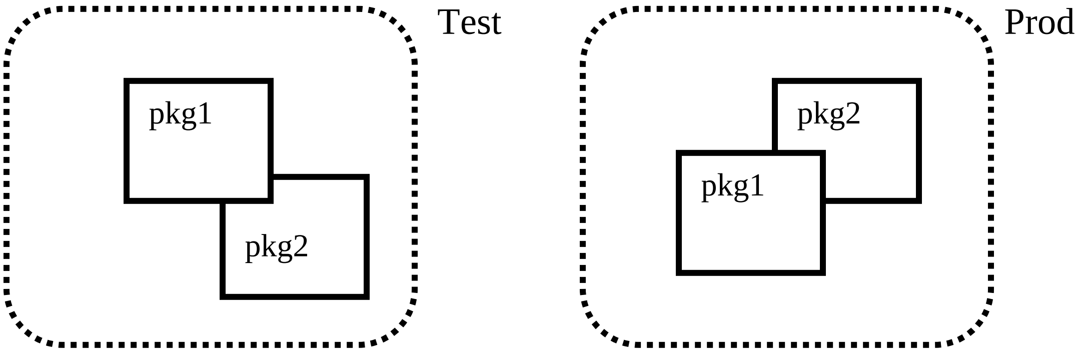
Of the above tables SYSIBM.SYSUSERAUTH table is read only for BIND ADD.

The DB2 catalog stores information only about the packages. The executable form of the package is stored as a skeleton package table in the DB2 directory in the SYSIBM.SPTOI1 table. A package also contains a location identifier, a collection identifier and a package identifier. These are identifiers used to uniquely identify the packages.

Figure 1.28 PACKAGES (Cont….)

COLLECTIONS

A collection is a set of Packages.



A collection is IMPLICITLY created a first BIND PACKAGE referring to that collection.

BIND PACKAGE (test) MEMBER (pkg1)
BIND PACKAGE (test) MEMBER (pkg2)

Figure 1.29 COLLECTIONS

Notes:

A collection is a user-defined name (1 to 18 characters) that the programmer must specify for every package. A collection is not an actual, physical database object. A collection is a grouping of DB2 packages. By specifying different collection identifier, for a package, the same DBRM can be bound to different packages. This capability permits the programmer to use the same DBRM for different packages, enabling easy access to tables that have the same structure, but different owners.

APPLICATION PLAN

- **A plan is an executable module containing the access path, logic provided by the DB2 optimiser. It can be composed of one or more DBRMs and packages.**

- **Plans are created by the BIND command. When a plan is bound, DB2 reads the following catalog tables:**

SYSIBM.SYSCOLDIST,SYSIBM.SYSCOLUMNS,SYSIBM.SYSFIELDS, SYSIBM.SYSINDEXES,SYSIBM.SYSPLANS,SYSIBM.SYSPLANAUTH, SYSIBM.SYSTABLES, SYSIBM.SYSTABLESPACE and SYSIBM.SYSUSERAUTH

The SYSIBM.SYSUSERAUTH table is read only for BIND ADD.

Figure 1.30 Application Plan

CONSISTENCY TOKEN – PROGRAM PREPARATION

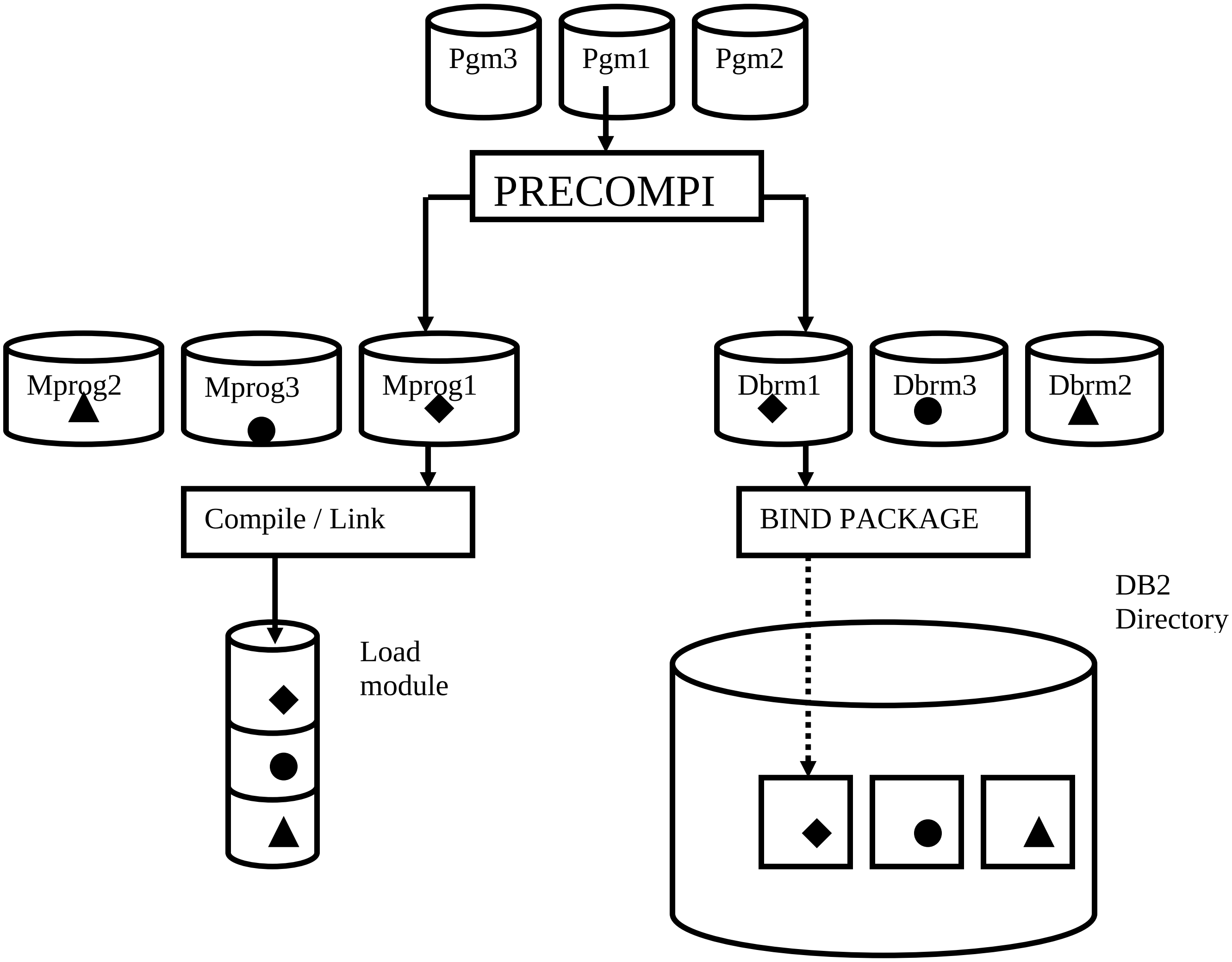


Figure 1.31 Consistency Token – Program Preparation

Notes:

The precompilation TIMESTAMP is included in each generated CALL in the modified source. The other twin brother (the DBRM) also contains this timestamp. It will be used by DB2 at program execution time to locate the correct package for the correct load module.

LOCATING PACKAGES AT EXECUTION TIME

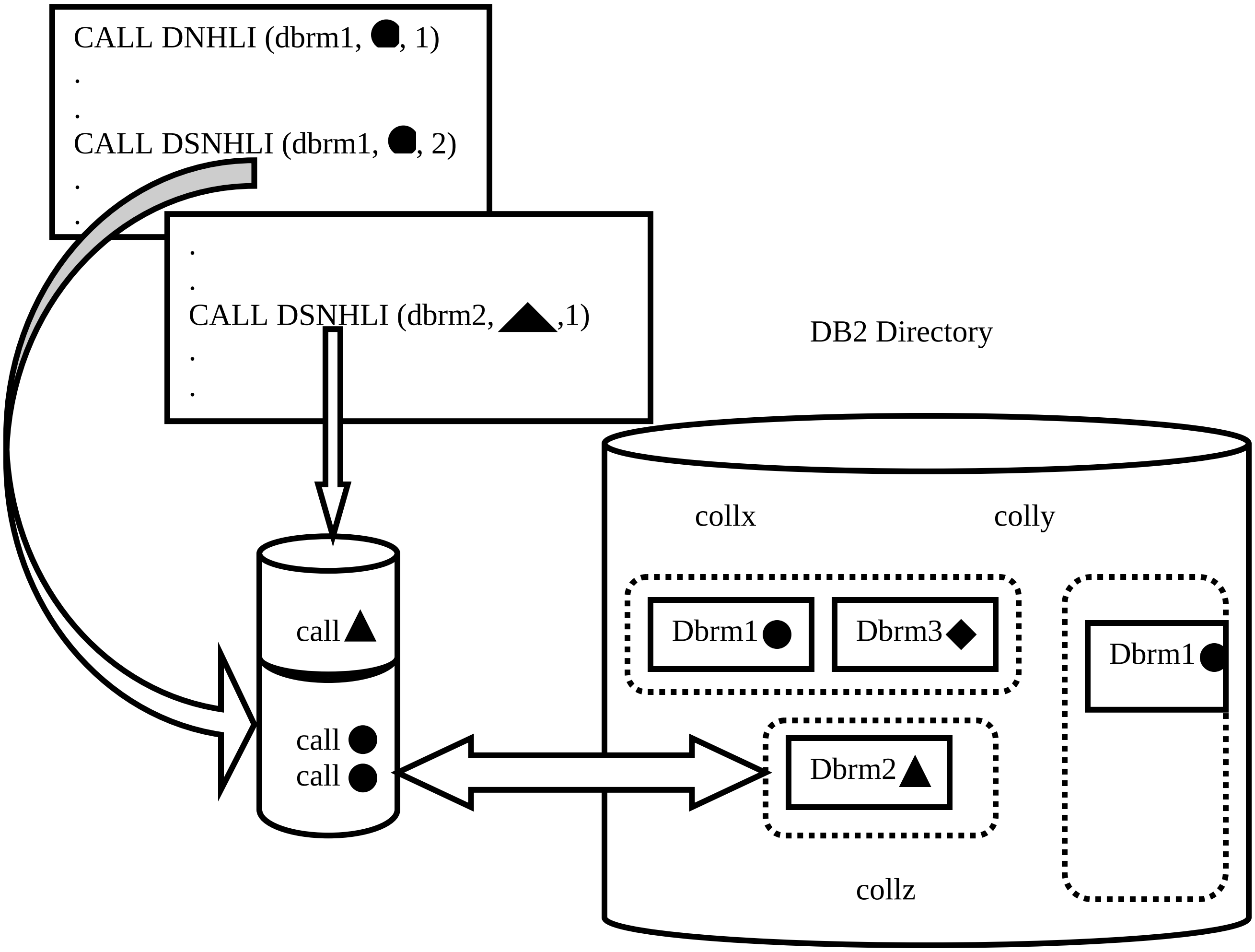


Figure 1.32 Locating Packages At Execution Time

THE MISSING LINK

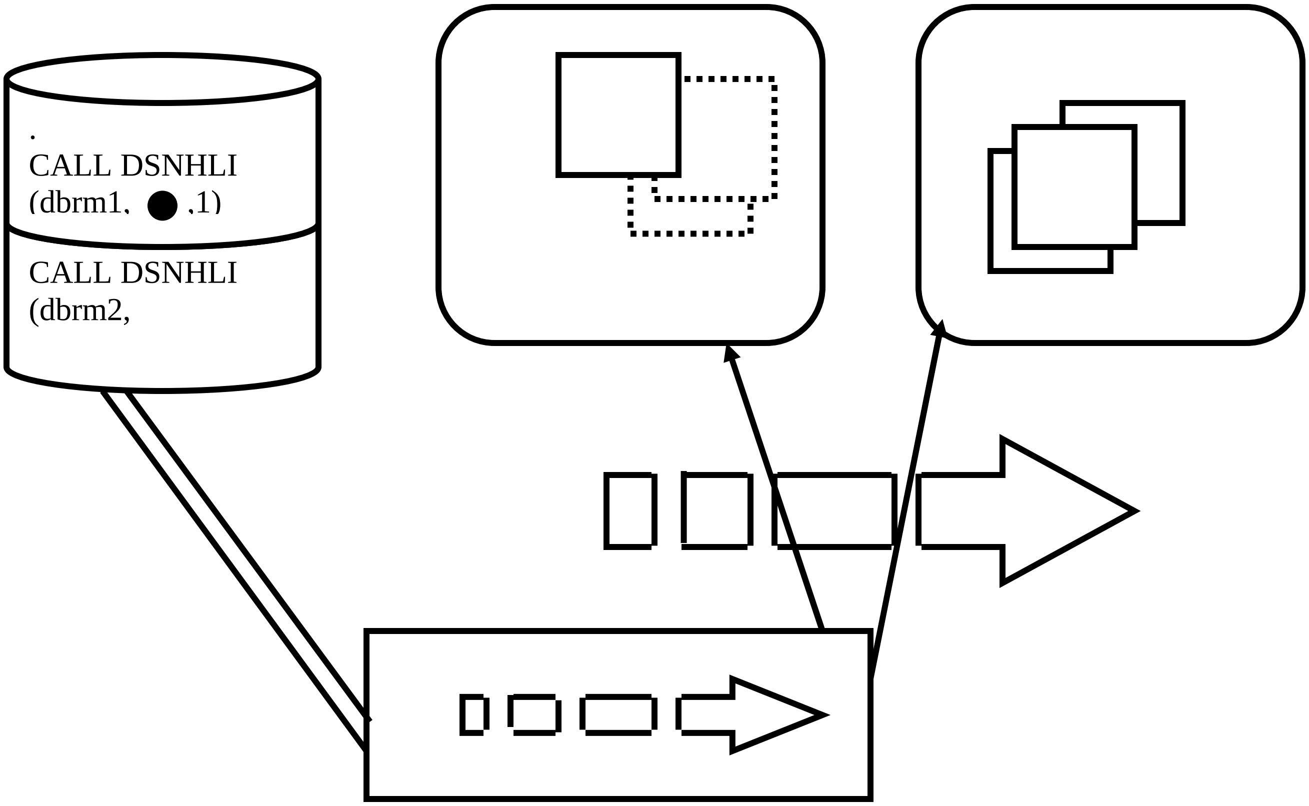


Figure 1.33 The Missing Link

Notes:

In between the load module and the packages there is a need for an additional structure that will guide DB2 in its searches for the appropriate package. This structure is called a plan and it will contain a package list, a list of logical pointers to packages. DB2 will look for the right (same timestamp) package by using each entry in the package list in turn until it has a hit. The mechanism is very similar to the search mechanism by the operating system to locate a member of a PDS on a concatenation of PDSs. Running a DB2 program is done by associating the plan with the load module. This is done outside the program.

PLAN AND PACKAGES

- **A PACKAGE can be located and executed only via a PLAN**
- **A PLAN contains a PACKAGE list a list of pointers to packages.**
- **The DBRM name and the TIMESTAMP provided with the CALL are used to locate the correct package via the package list.**

Figure 1.34 Plan And Packages

Notes:

The package list determines which package will be used.

BIND PLAN – PACKAGE LIST

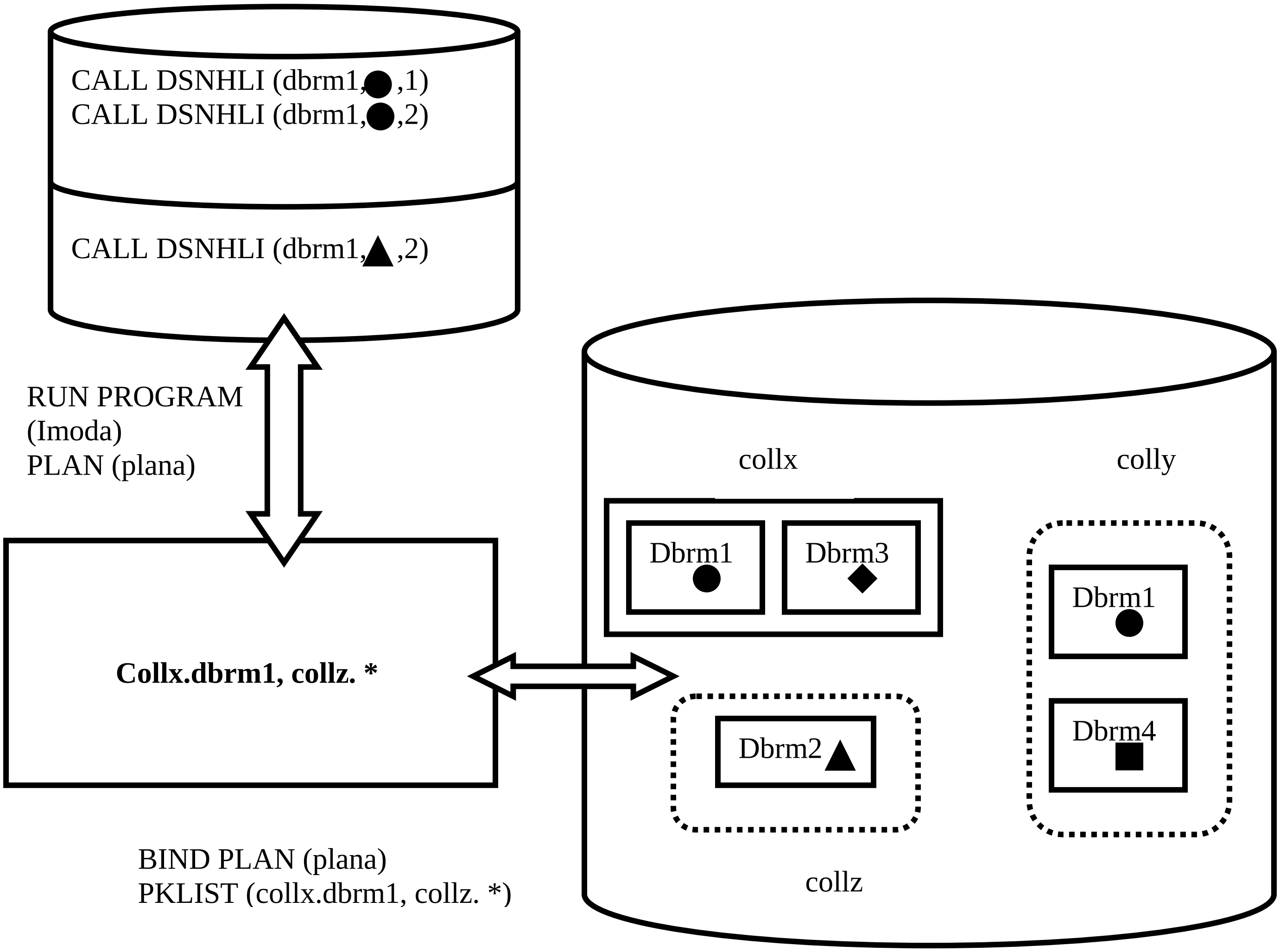
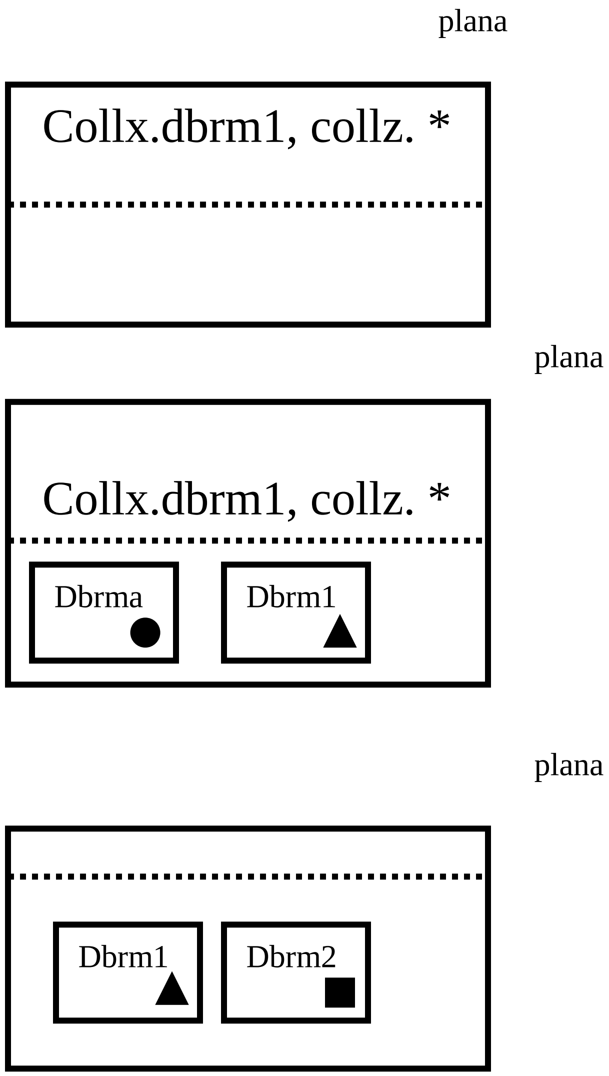


Figure 1.35 Bind Plan – Package List

Notes:

A PACKAGE LIST is a list of references to individual packages (E.G. COLLX.DBRM1) or to sets of packages (e.g. COLLY. *). A package list makes specific parts of the directory eligible for the package search.

BIND PLAN DBRMs



BIND PLAN (plana)
PLSIST (collx.dbrm1, collz. *)

BIND PLAN (plana)
PKLIST (collx.dbrm1, collz. *)
MEMBER (dbrma, dbrm1)

BIND PLAN (plana)
MEMBER (dbrm1, dbrm2)

Figure 1.36 Bind Plan DBRMs

Notes:

Packages were introduced with DB2 V2.3, but plans already existed. Previously, the DBRMs were bound directly into the plan structure itself. For compatibility reasons, DB2 also supports this today.