Easytrieve Plus / DB2

Course Pre-requisites

This course provides no training in Easytrieve Plus nor in the use of DB2 and SQL. These topics should all be studied separately prior to this course.

SQL (Structured Query Language) provides the method of access to DB2 data structures and remains the same no matter what application language or operating system is used. It is perfectly reasonable to study DB2/SQL in a COBOL II environment before looking at its use in Easytrieve Plus.

Static SQL and Dynamic SQL

SQL is designed to allow specification of what data is required rather than how to access it. The access paths to the data is worked out by DB2. This may happen at two points in time:

- When the SQL statement is actually executed while the program is running (**Dynamic SQL**)
- In advance by using the SQL pre-processor on the source code (**Static SQL**)

The Easytrieve Plus interface uses the Dynamic and Extended Dynamic interfaces supplied by IBM. The latter of these implements Static SQL commands. Only commands which can be executed using the Dynamic and Extended Dynamic interfaces can be embedded in an Easytrieve Plus program.

NB. In Information Systems (BTC), programs may be run dynamically while in development, but must be handed over to run statically in the live environment.

Dynamic SQL

To run an Easytrieve Plus / DB2 program dynamically, simply use the **EZPGO** catalogue procedure to compile and run it all in one step.

Static SQL

Before an Easytrieve Plus / DB2 program can be run using static SQL, it is first necessary to code a **PARM LINK** statement at the top of the program. The syntax of this is as follows:

```
PARM LINK (program name [R]) +

        PLAN (plan name) +

        BIND (STATIC-ONLY)
```

The R is short for replace, and causes an existing load module of the same name to be over-written. For the first compile of a program, the R can be left off, as the default is ADD.

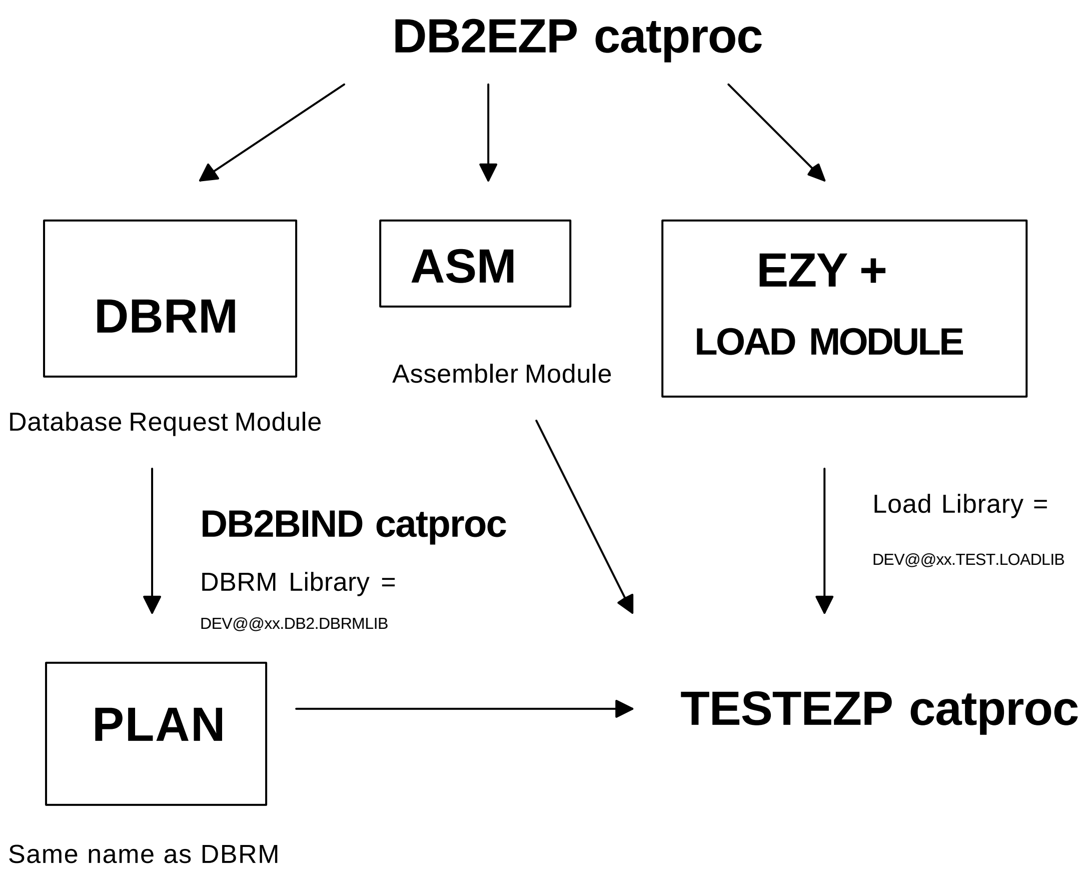
The program name and plan name MUST be different from one another.

To run an Easytrieve Plus / DB2 program statically, it is necessary to use three separate catalogue procedures. The first step uses **DB2EZP** to separate the program into its Easytrieve Plus and SQL components. This step produces an Easytrieve Plus load module, which is given the program name, and a Database Request Module (**DBRM**), which resides as a member in the partitioned dataset **DEV@@xx.DB2.DBRMLIB**. Also produced, but invisible to the developer, is an Assembler module, which is called dynamically at runtime to execute the SQL.

The second step of the process uses the Boots catalogue procedure **DB2BIND** to produce a **PLAN** from the DBRM. The PLAN is given the same name as the DBRM.

Finally, the catalogued procedure **TESTEZP** is used to run the program.

These steps are illustrated on the diagram overleaf.



The stages in running an Easytrieve Plus / DB2 program under static SQL.

Field Definitions and Data Types

Any data to be retrieved by Easytrieve Plus must be defined in the Library Section of the Easytrieve Plus program. Such fields are called **Host Variables.** They can either be working storage fields or fields in an active file. Host Variables are immediately preceded by a colon when they are referred to in an SQL statement.

The following table shows SQL data types and their corresponding Easytrieve Plus field definitions:

SQL data type	Easytrieve Plus field definition
INTEGER	4 B 0
SMALLINT	2 B 0
FLOAT	none
DECIMAL (precision, scale)	x P y[1]
CHAR (length)	x A
VARCHAR	none
LONG VARCHAR	none
GRAPHIC (length)	none
VARGRAPHIC	none
LONG VARGRAPHIC	none
NUL indicator variables	2 B 0
none	x N y
none	x B y[2]
none	x U y

[1] SQL does not support packed fields with a length greater than 8.

[2] SQL does not support binary fields other than 2 B 0 and 4 B 0.

Automatic and Controlled Processing

A DB2 table can be accessed from an Easytrieve Plus program in one of two ways:

Automatic Processing

All Easytrieve Plus programs must begin with a command such as FILE, PARM or DEFINE. Where there is no file input - and therefore no FILE statement - it is necessary to code the command **DEFINE** in front of the field definitions in the program. This statement is otherwise optional and would normally be left out.

To specify that the job will have automatic input from SQL, it is first necessary to code the job statement as follows:

JOB INPUT SQL

A **SELECT** statement must then immediately follow the JOB statement. The SELECT statement will identify the rows and columns to be used as input to the job activity.

NB. Only *one* SELECT statement can be coded in each job activity.

Using this method it is possible to retrieve selected data from every row within a DB2 table.

eg. To select the contents of the column EMPNAME from the table PERSONNEL, one at a time, using automatic input in Easytrieve Plus, code the following:

```
DEFINE EMP-NAMEW 5 A

JOB INPUT SQL

        SELECT EMPNAME +

                FROM PERSONNEL +

                INTO :EMP-NAME
```

.... followed by other Easytrieve Plus commands.....

Controlled Processing

As with Automatic input, it is necessary to use the Easytrieve Plus DEFINE command on the first line of the program if there is no input file.

If controlled input is being used in an Easytrieve plus program, it is then necessary to define a **cursor** in the Library section. This is done using the **SQL DECLARE** statement, followed by a cursor name. Any host variables referred to in a DECLARE statement must already have been defined in the Library section above. All other SQL statements, such as those to open, close and fetch the cursor into a host variable are coded in the Activity section after the JOB statement, which is coded as follows:

JOB INPUT NULL

e.g.. To fetch the contents of the first row of the column EMPNAME from the table PERSONNEL, using a cursor with controlled input in Easytrieve Plus, code the following:

```
DEFINE EMP-NAME          W 15 A

SQL DECLARE C1 CURSOR FOR +

          SELECT EMPNAME +

                    FROM PERSONNEL

JOB INPUT NULL

          SQL OPEN C1

          SQL FETCH C1 +

                    INTO :EMP-NAME
```

.... followed by other Easytrieve Plus and SQL commands.....

When using controlled input in Easytrieve Plus, it is necessary to code a **STOP** statement at the end of the program.

The following SQL commands *cannot* be processed using Easytrieve Plus controlled SQL processing:

DECLARE STATEMENT

DECLARE TABLE

DESCRIBE

EXECUTE

EXECUTE IMMEDIATE

INCLUDE

PREPARE

SELECT.......INTO.......

WHENEVER

SQL Communications Area (SQLCA)

Whenever the Easytrieve Plus compiler encounters the **SQL** or **JOB INPUT SQL** statements, all of the SQL Communications Area fields are created in 'S' working storage. These are listed in APPENDIX A.

It is very important that the **SQLCODE** field in the SQLCA is always tested to determine whether the execution of each SQL statement is successful. If the code is not equal to 0 (successful execution) or 100 (end of table reached), then the SQLCODE should be logged and the error reported with a meaningful message in line with error processing standards.

e.g..

```
IF SQLCODE NE 0 AND +

        SQLCODE NE 100 +

        DISPLAY 'SQL ERROR - SQLCODE = ' SQLCODE

        STOP

END-IF
```

SQL Supported Commands

The following is a list of the DB2 commands that can be processed using Controlled SQL processing:

Information Retrieval / Updating Commands:

DECLARE	CONNECT
SELECT	COMMIT
OPEN	ROLLBACK
FETCH	PUT
CLOSE	UPDATE
DELETE	INSERT

Other Supported Commands:

ALTER INDEX	CREATE VIEW
ALTER STORGROUP	DROP INDEX
ALTER TABLE	DROP SYNONYM
ALTER TABLESPACE	DROP TABLE
COMMENT	EXPLAIN
CREATE DATABASE	GRANT
CREATE INDEX	LABEL
CREATE STORGROUP	LOCK
CREATE SYNONYM	REVOKE
CREATE TABLE	CREATE TABLESPACE

For further Easytrieve Plus / DB2 examples, see the Easytrieve Plus Reference Manual.

Easytrieve Plus / DB2 Exercise

Produce a report containing the fields:

Employee number

Employee's first name and surname

Department number

Salary

for those employees in the table DEV@@TR.TRTEMPL whose jobcode is greater than 52 and were born before 1948. If you don't have one already, obtain a copy of the table from a trainer before doing this exercise.

The output should be sorted into ascending surname order.

Also, give a total for all the salaries printed out.

Write programs to produce the report specified above, using:

1) Controlled input with dynamic SQL

2) Automatic input with dynamic SQL

3) Controlled input with static SQL

Appendix A - SQLCA Generated Fields

```
DEFINE SQLCA        S            136  A
DEFINE SQLCAID      SQLCA        8    A
DEFINE SQLCABC      SQLCA +8     4    B    0
DEFINE SQLCODE      SQLCA +12    4    B    0
DEFINE SQLERRM      SQLCA +16    72   A
DEFINE SQLERRML SQLCA +16   2    B    0
DEFINE SQLERRMC SQLCA +18   70   A
DEFINE SQLERRP      SQLCA +88    8    A
DEFINE SQLERRD      SQLCA +96    4    B    0    OCCURS 6
DEFINE SQLWARN      SQLCA +120   8    A
DEFINE SQLWARN0     SQLCA +120   1    A
DEFINE SQLWARN1     SQLCA +121   1    A
DEFINE SQLWARN2     SQLCA +122   1    A
DEFINE SQLWARN3     SQLCA +123   1    A
DEFINE SQLWARN4     SQLCA +124   1    A
DEFINE SQLWARN5     SQLCA +125   1    A
DEFINE SQLWARN6     SQLCA +126   1    A
DEFINE SQLWARN7     SQLCA +127   1    A
DEFINE SQLWARN8     SQLCA +128   1    A
DEFINE SQLWARN9     SQLCA +129   1    A
DEFINE SQLWARNA     SQLCA +130   1    A
```