**Is it a PLAN? Is it a PACKAGE? NO, It's a CONFUSION !!!**

No matter how long programmers have worked with DB2 for z/OS , they still have the confusion on difference between a plan and a package and what the heck is that collection anyway.

In this post i will be specific on the PLAN/PACKAGE difference and the things related to it (Such as COLLECTION and PKLIST). If you want to know from basic and detailing of what is PLAN/PACKAGE and how it get formed and why we need it then ,[THIS](http://db2guide.blogspot.com/2013/06/how-db2-application-program-runs-on.html) is the great place to start up with,rather i would recommend you to read it before going further with this post.

OK so here it goes.

The PLAN/PACKAGE are nothing but the container which contains the logic of how your SQL statement in the program should get executed.[Remember when we write any query in SQL, we say what we want and NOT how it is to be done (We don't care how DB2 internally does it)].So PLAN / PACKAGE contains the LOGIC how it is to be done.Then this PLAN / PACKAGE is used with load module (For COBOL) to run your application program smoothly.

Now why two things?? PLAN and  PACKAGE , if both are the same (Contains the LOGIC for executing your SQL) and what exactly is the difference between the two.

**Binding DBRM to a PLAN** :

* The PLAN contains the (Logic of) one or DBRM OR one or more PACKAGES OR the combination of both.
* The PLAN is executable. When i say executable it means ,it can be executed with COBOL load module to run your application program.
* Now let's consider there are 10 programs with names A TO J. So there will be 10 DBRM ( 1 DBRM for each program). I bind all 10 DBRM's to a PLAN (with name 'PLANA').Now let's say i got a requirement to change one of the Program ( let's say Program 'B') then i NOT ONLY need to bind this DBRM again to plan PLANA but all remaining 9 DBRM too (even though they didn't went under any change) and that is the biggest disadvantage of a binding a DBRM to a PLAN.NOT only this, if in my system there are various other PLAN's which uses this DBRM (The one which corresponds to Program 'B') then all those PLAN'S need to rebind along with all the other existing DBRM'S they may be having.
* There are various parameters called as Bind parameter are available, when you bind your DBRM to a PLAN. These Bind parameters helps in deciding various important aspects for you program such as when to acquire a lock for the particular resource in your program, when to release the lock, what qualifier to use for the objects refered in your application program, the owner of plan etc. So when you bind your all DBRM's (10 DBRM in this example) to PLAN you have only one set of combination of these bind parameters for all 10 DBRM (For example you can not set different isolation level for different Program/DBRM , it has to be same)

**Binding DBRM to a PACKAGE:**

* The PACKAGE contains the (Logic of) ONLY one DBRM. It's a single bound DBRM.
* The PACKAGE is not executable that means it can not be use with COBOL load module to run your Application program. In order to make your PACKAGE to be executed it has to bonded to PLAN again. So you bind your DBRM to PACKAGE and then in turn this PACKAGE to PLAN.I know what you must be thinking now, why to go this way,  isn't it better to bind DBRM to straight away to PLAN.NO IT IS NOT, i will explain you why it is so. When i say you have to bind the DBRM to PACKAGE and then this PACKAGE to PLAN again, you need not to do it for ever package. What i mean is, let's say i have 50 PACKAGES to bind to a PLAN no need to BIND it 50 times instead just bind a single package to PLAN using PKLIST(Package List) that's it ,you are done and remaining 49 PACKAGES will be automatically added to your plan (For this all your 49 Packages need to be in same PKLIST what we have bonded with PLAN earlier)
* Let's take the same example what we use above for PLAN. I have 10 programs A TO J with 10 DBRM. Now in case of PACKAGE , I will bind each DBRM to one PACKAGE so i will have 10 Packages then include all 10 PACKAGES to single PKLIST (Package List) bind the Package list to a PLAN. Now if one of my Program changes , then i have to rebind only the DBRM corresponding to the Program changing and NOT the remaining DBRM'S/PROGRAM (Which didn't went under the change)
* Regarding Bind parameter, since i can bind 10 DBRM to 10 different PACKAGES, I can have various combination of Bind parameters for all PACKAGES.(For example Isolation level of CS for 1 PACKAGE and RR to other etc.)

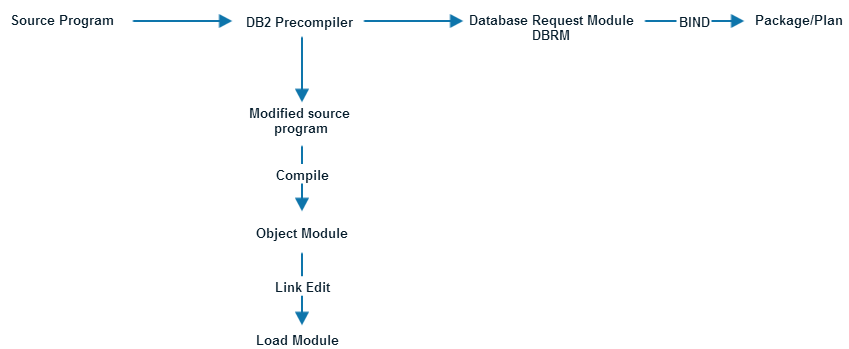
Having said that, you will wonder, you have only the advantages of binding DBRM to PACKAGE and don't have any reason to go for binding DBRM to a PLAN.

When we write a program that needs data from DB2 tables, we code SQL statements in the source program. SQL statements are **guest** to the host Cobol program. The SQL statements are enclosed by **EXEC SQL** and **END-EXEC** delimiters. Between the delimiters, we also code the **INCLUDE** dclgen (which is similar to coding Cobol COPY). For each table referenced in the SQL query, we must code INCLUDE dclgen. The DCLGEN has the definition of the table, just how the table looks in DB2. The DCLGEN also has host Cobol variables, that are a warehouse for the rows delivered from the SQL query.

**Pre-process SQL**

Once the program is written, this source program containing SQL code blocks is passed through the DB2 pre-compiler. You see, the pre-compiler is **not connected** to DB2. So, it does a modest syntax check on the SQL queries. The tables are compared against their definitions in the INCLUDE dclgen.

The **pre-compiler** then extracts the SQL and creates a Database Request Module**(DBRM)** containing all the SQL in the program. A modified source program is also created with the SQL commented out and calls to DB2 substituted. **Arnold** - the modified source program and **Danny** - the DBRM are two twins born out of pre-processor.



Arnold - the "pure" Cobol source and Danny - the DBRM go down separate paths, until they re-unite in the future. The pre-compiler puts an identical **tattoo** on both Arnold and Danny. A few years from now, Arnold graduates to become a load-module. Danny grows into a package/plan. We'll look at the life of Arnold and Danny in a while. But, Arnold and Danny carry their tattoos with them throughout their life.

**Arnold's journey**

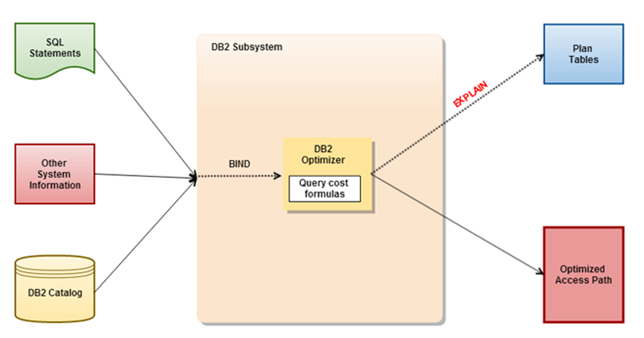
The normal compile/link process is executed on, the newly generated, modified Cobol source code. The Cobol **COMPILE** produces an object program. The object program is **LINK EDIT**ed with DB2 routines. The resulting load module(with the tattoo on it) is put into a **LOAD** library.

**The life of Danny**

The SQL in the DBRM must now go through a process similar to **COMPILE** to produce run-time code. The DB2 **BIND** process transforms the SQL into DB2 interpretable form. The output of BIND is called a **package**. To convert the DBRM into executable code, the BIND process **connects** to **DB2**. It reads the SQL statements in the DBRM and does a much, much more, **thorough** syntax check. The pre-compiler did a decent check on the SQL, only as **accurate** as the **DCLGEN**. The BIND process validates the DB2 tables and columns being accessed against the corresponding **DB2 catalog** information.

The **optimizer** is the **heart** and **soul** of DB2. It basically works like an expert system. The optimizer analyzes the SQL and determines the most efficient access path for satisfying the SQL. The optimizer queries the **statistics** stored in the DB2 catalog to determine an access path. Statistics used by optimizer include information about the current status of tables, indexes, columns etc. The optimizer plugs this information into **query cost** formulas. The optimizer also looks at how many CPU's are being used, size of your bufferpools and much, much more.

A doctor applies his knowledge of medicine, combined with the symptoms of the ailment to prescribe medication. Likewise, the optimizer applies a standard set of rules combined with situational data housed in the DB2 catalog and recommends **data retrieval methods**.

[](http://lh3.ggpht.com/-u6rkaZqPmhg/UwnQjbMIuoI/AAAAAAAAFwU/sKLZT9N3pB8/s1600-h/image%25255B5%25255D.png)