**Interview Questions**

**COBOL**

**IGATE (20/09/2014)**

**Q. Differentiate between Static and dynamic compilation?**

**ANS**

A ------- B

A----------C------D

Program A is main program calling programs B and C and C program in turn calls D.

Then if we do static compilation of A will B,C,D gets compiled automatically.

Also do we need to compile B,C,Dseperately

Static call implies that the objects modules(of called and calling) are bound together before they use each other.This bonding remains the same until one of the program gets changed and the whole set is compiled/linked again.

Dynamic call implies the calling program does not know about the called program until the call takes place.The bonding between the program happens right at the time the call is made

**Lets see in details how the calls are made and few other details**   
In case of static call, we refer to the called program name directly in the call statement.   
Ex: CALL ‘PGM1’.  
  
In case of dynamic Call, we call one working-storage variable which is populated with the called program name at run time**dynamically.**   
01 WS-PGM PIC X(08). <<== Working storage variable

Move ‘PGM1’ to WS-PGM

CALL WS-PGM                    
Here WS-PGM is the Working Storage item. In Procedure Division we are moving the called program name 'PGM1' to WS-PGM and then calling it.

**So in case of static call:**  
If we have the main program PGM-A and it calls two sub programs statically(pgm-b &pgm-c), we will have**only 1 load** **module** for all the three.Load modules of pgm-b and pgm-c will be linked edited into the load module of PGM-A making the size of the load bigger.  
Negative points for a static call   
Since the load modules exists together,it takes up more virtual space on disk    
If any changes are made to called program, then all the programs must be compiled/linkd.  
Positive point:  Execution is fast

**In case of Dynamic call:**  
As the name suggests, the program is called dynamically.The load module of the called program  will exists separately in the load module library. Thus here if PGM-A calls pgm-b and pgm-c, we will have 3 load modules our load library.  
Positive point:   
The main module does not take up unnecessary space.  
Since these are stand alone programs, we can compile/link respective programs without touching all the programs and main program.  
Negative points for a dynamic Call:   
Since it executes with*program call resolution*, the call is a bit slow than static all.

**Compiler options which can override the Static and dynamic calls**:  
If compiled with **NODYNAM** compiler option  
CALL 'PGM-A' will be handled as  a static call  
CALL WS-VARA  will be treated as dynamic call  
  
If compiled with **DYNAM** compiler option:  
CALL 'PGM-A' will be **also treated as  a dynamic call**  
CALL WS-VARA will be treated as a dynamic call

**Q. Explain the Logic for file compare using Cobol?**

**ANS.**

Read file1  
Read file2  
  
perform until end-of file1 OR end-of file2  
if file1 = file2  
write a matched record to output  
read file1  
read file2  
end-if  
if file1 < file2  
read file1  
end-if  
if file1 > file2  
read file2  
end-if   
End-perform.

**Q. Explain Working of Search and Search all?**

**ANS**

Search

Search is a linear search method, which is used to find elements inside the table. It can be performed on sorted as well as unsorted table. It is used only for tables declared by Index phrase. It starts with the initial value of index. If the searched element is not found, then the index is automatically incremented by 1 and it continues till the end of table.

**Example**

IDENTIFICATION DIVISION.

PROGRAM-ID. HELLO.

DATA DIVISION.

WORKING-STORAGE SECTION.

01 WS-TABLE.

05 WS-A PIC X(1) OCCURS 18 TIMES INDEXED BY I.

01 WS-SRCH PIC A(1) VALUE 'M'.

PROCEDURE DIVISION.

MOVE 'ABCDEFGHIJKLMNOPQR' TO WS-TABLE.

SET I TO 1.

SEARCH WS-A

AT END DISPLAY 'M NOT FOUND IN TABLE'

WHEN WS-A(I)=WS-SRCH

DISPLAY 'LETTER M FOUND IN TABLE'

END-SEARCH.

STOP RUN.

Search All is a binary search method, which is used to find elements inside the table. Table must be in sorted order for Search All option. The index does not require initialization. In binary search the table is divided into two halves and determines in which half the searched element is present. This process repeats till the element is found or the end is reached.

IDENTIFICATION DIVISION.

PROGRAM-ID. HELLO.

DATA DIVISION.

WORKING-STORAGE SECTION.

01 WS-TABLE.

05 WS-RECORD OCCURS 10 TIMES ASCENDING KEY IS WS-NUM INDEXED BY I.

10 WS-NUM PIC 9(2).

10 WS-NAME PIC A(3).

PROCEDURE DIVISION.

MOVE '12ABC56DEF34GHI78JKL93MNO11PQR' TO WS-TABLE.

SEARCH ALL WS-RECORD

AT END DISPLAY 'RECORD NOT FOUND'

WHEN WS-NUM(I)=93

DISPLAY 'RECORD FOUND '

DISPLAY WS-NUM(I)

DISPLAY WS-NAME(I)

END-SEARCH.

**Q. What happens when we try to display comp-3 variable**

**ANS.**

You can display them, but they will appear non-numeric.   
Move the field to a WS field defined as pic +9('number of   
digits') and display the WS field to make the data user-  
friendly.

You can display the COMP-3 variable by redefining to numeric  
variable.

Note: declaration of the numeric var 01 B PIC S9(15) SIGN TRAILING SEPARATE.

**Q. Expalin how the COMP-3 variable gets stored?**

**ANS.** It is hex word if say you have +905229833 to be stored. It stores as

95283

0293C

**Q. For Arithmetic operations which PIC clause is suitable COMP-3 or COMP-1 and explain why?**

**ANS**

Comp-1 and Comp are good for arithmetic calculations since they are binary.

Comp-3 is good to store the data from SQL and so on since it saves the space N/2 +1

**HCL(01/11/2014)**

**Q. How to get a character from a read record which is stored randomly at any place in the fike**

**ANS**

Use COBOL search(linear) since we want the position of the character

SEARCH WS-A

AT END DISPLAY 'M NOT FOUND IN TABLE'

WHEN WS-A(I)=WS-SRCH

DISPLAY 'LETTER M FOUND IN TABLE'

END-SEARCH.

**NETCRACKER(31/10/2014)**

**Q.Explain how will you run IMS COBOL program**

**ANS.** WE use utility DFSRRC00 and pass the parameters ’DLI,Load program,Load PSB Name’

Later in the program there is entry for DLITCBL as first statement in Proceduredivision and then we give goback. Here PCB’s in PSB get stored into Cobol program. Note that PCB definitions are given in linkage section.

Later actual retrieval of programs take place

Sample PCB definition

01 INVENTORY-PCB-MASK.

05 **IPCB-DBD-NAME PIC X(8).**

05 **IPCB-SEGMENT-LEVEL PIC XX.**

05 **IPCB-STATUS-CODE PIC XX.**

05 **IPCB-PROC-OPTIONS PIC X(4).**

05 FILLER PIC S9(5) COMP.

05 **IPCB-SEGMENT-NAME PIC X(8).**

05 **IPCB-KEY-LENGTH PIC S9(5) COMP.**

05 **IPCB-NUMB-SENS-SEGS PIC S9(5) COMP.**

05 **IPCB-KEY PIC X(11).**

***DL/I function codes***

**01 DLI-FUNCTIONS.**

05 DLI-GU PIC X(4) VALUE ‘GU ’.

05 DLI-GHU PIC X(4) VALUE ‘GHU ’.

05 DLI-GN PIC X(4) VALUE ‘GN ’.

05 DLI-GHN PIC X(4) VALUE ‘GHN ’.

05 DLI-GNP PIC X(4) VALUE ‘GNP ’.

05 DLI-GHNP PIC X(4) VALUE ‘GHNP’.

05 DLI-ISRT PIC X(4) VALUE ‘ISRT’.

05 DLI-DLET PIC X(4) VALUE ‘DLET’.

05 DLI-REPL PIC X(4) VALUE ‘REPL’.

05 DLI-CHKP PIC X(4) VALUE ‘CHKP’.

05 DLI-XRST PIC X(4) VALUE ‘XRST’.

05 DLI-PCB PIC X(4) VALUE ‘PCB ’.

**UHG(23/11/2014)**

**Q. Explain good coding practices in Cobol in details atleast mention 10 points**

**ANS.**

The Cobol standards that we define must be able to make the code

1. Reliable
2. Maintainable
3. Structured
4. Timely
5. Efficient
6. The first two characters of program name should give the application name like the programAFUPDATE – Where AF implies it is a CAMS application program

**1. Line stuff up:**

**MOVE ZEROES TO WSS-SWITCHES.                      
MOVE WSP-GCU149J-PARM-RCPT  TO GA03INDQ-VALUE     
MOVE 'D'                    TO GA03PT2C-CMD1      
MOVE 'ADC'                  TO GA03PT2C-VALUE2    
                               GA03PT2C-VALUE4    
MOVE '\*'                    TO GA03PT2C-RPAREN    
                               GA03PT2C-RPAREN3   
MOVE '+'                    TO GA03PT2C-RPAREN2   
MOVE '0'                    TO GA03PT2C-VALUE     
MOVE HIGH-VALUES            TO GA03PT2C-VALUE3.**

Instead of 

**MOVE ZEROES TO WSS-SWITCHES.                      
MOVE WSP-GCU149J-PARM-RCPT  TO GA03INDQ-VALUE     
MOVE 'D'  TO GA03PT2C-CMD1      
MOVE 'ADC'  TO GA03PT2C-VALUE2    
                               GA03PT2C-VALUE4    
MOVE '\*'   TO GA03PT2C-RPAREN    
                               GA03PT2C-RPAREN3   
MOVE '+'   TO GA03PT2C-RPAREN2   
MOVE '0'    TO GA03PT2C-VALUE     
MOVE HIGH-VALUES TO GA03PT2C-VALUE3.**

**2. don't use "NOT" unless you have to, even if it means more lines of code:**

**IF WGPC05P0-STATUS-CODE = SPACE         
    CONTINUE                            
ELSE                                    
    SET WSS-ERROR-FOUND     TO TRUE     
    MOVE 006 TO WST-ES-NEW-NUM          
    MOVE 'BAD READ FOR GS05CSLD'        
      TO WST-ES-NEW-MSG                 
    PERFORM 41500-STACK-ERRORS          
       THRU 41500-EXIT                  
END-IF.**

**3.  Do not code block sizes in programs -- use BLOCK CONTAINS 0**

For I/O optimization - there are several key concerns, such as using BLOCK CONTAINS 0 RECORDS and BLKSIZE=0 to allow the operating system to optimize the file I/O blocking factor

**4. Vertically align PIC and VALUE clauses**

**5. Use scope terminators (END-IF, END-PERFORM, etc)**

**6. Use file status codes for every file AND CHECK THEM IN YOUR PROGRAM**

**7. Avoiding the use of USAGE IS DISPLAY.**

**8. Avoiding sorts within the COBOL code, and if required providing the FASTSRT compiler option.**

It also simplifies the program. Programmers don't tend to know what SORT does or how it works in the program. There is the possibility to get the program doing diverse things, do-some-stuff-with-unsorted, sort, do-some-stuff-with-sorted, which makes it more difficult to understand and maintain.

9. Some time ago "we" (several different clients) did a bunch of "performance testing" comparing the internal sort (COBOL SORT statement) versus the external sort (invoked via jcl). Keep in mind that the external sort typically takes 3 steps (one to prepare the data to be sorted, the sort, and the process to use the sorted data) and step initiation/termination is rather expensive. For "small" to "medium" volumes, the internal sort performed as well as or better than the external sort. For "massive" amounts of data the external sort performed better - both in elapsed time and resources used.  
  
Unless something has been changed in the interface between COBOL and the sort product, the sort product handled all of the sorting i/o. The COBOL code had to read the input but once the data to be sorted was RELEASED to the sort, the sort took over. In my opinion the internal sort should Never use both the USING and GIVING options. This just wastes resources and is largely where the internal sort got a bad name years ago. USING and GIVING in the same program most often says that the developer is really lazy or is unwilling to learn better.

10.Second, make sure you define your goal before you start. Are you reducing **i/o, run time, cpu time**.Say 10,000 row table lookup might be the best way to make the program run faster, but it could use more CPU time to accomplish it. However, if you change it to a VSAM file, you CPU could go down, but the program would run longer.

COBOL sorts aren't bad. I think if we run one here, it just calls DFSORT anyway.

* Choose your variable types carefully - especially numeric
* There's a long section comparing the run-time efficiences of different types against different numeric lengths
* Just as an example: PIC S9(8) Display is 575% slower (not a typo, five hundred and seventy five percent slower) than COMP (binary) types for PERFORM Loops. And COMP-3 is 280% slower than Binary.
* Use indexes instead of subscripts - especially for large tables - and use SEARCH ALL when you can (some have set the bar for "large tables" to be at > 20 occurrences)
* For most math operations (more complicated than SUBTRACT 1 FROM XYZ-VAR) use COMPUTE, rather than a pastel of ADD/MULTIPLY/SUBTRACT/DIVIDE statements  
  + And make sure that intermediate variable decimal precision is wide enough to avoid truncating values!
* Always use Signed numeric declarations (otherwise there are extra assembler instructions inserted to sign it for you at run-time)  
  + And stay away from math on DISPLAY numeric fields
* There are also the usual Comp-Sci (common-sense) mom & apple pie homilies, such as:  
  + If you know that certain IF/ELSE tests will test true (or false) more often than others, bubble them up in the statements
  + Same goes for linear SEARCH
  + etc.

**Q. If Program A calls B, B calls C and then C calls D. B and C are DB2 programs. If program C is changed frequently then what is the efficient way of compiling the code**

**ANS.**

**Program B is dynamically compiled** in order to avoid recompilation of program B upon change of C

**Program A is statically compiled** since program B doesn’t change much.

Program C must be compiled since the code is changed.

IF Program C is Statically compiled implies D doesn’t change often

IF C is dynamically compiled then load of D is not included in C.

The optimum way of doing compilation in this case is

Prog B dynamic compilation (compiled only once even C changes many times)

Prog A static compilation(compiled only once even C changes many times)

Prog C dynamic compilation(compiled every time code is changed)

**DB2**

**IGATE (20/09/2014)**

**Q. Explain bind, package, DBRM and Collection in detail?**

**ANS** The DB2 Precompiler splits the program into two parts: a COBOL and a DB2 part. The embedded SQL is stripped out of the program and put into a partitioned data set (PDS) member, called a **DBRM**.

Just as the COBOL part has to be compiled, the **DBRM part has to go through BIND process to create the run-time executable code for the DB2 portion of the COBOL program**. To help the COBOL and DB2 part to find each other later at run time, the precompiler engraves each with identical timestamps called **consistency tokens**.

**You can BIND the DBRM into a PLAN (the old way), or you can BIND the instructions into a PACKAGE**

A PLAN is an executable module containing the access path logic produced by the DB2 optimizer**. The DBRMs of more than one program or PACKAGES can be bound into a PLAN.**

A PACKAGE is a single, bound DBRM with optimized access paths. The DBRM of a single program is bound into a PACKAGE. To execute a PACKAGE, it should be included in the package list of a PLAN. **PACKAGEs are not directly executed, they are only indirectly executed when the PLAN in which they are contained executes.**

**The relationship between a DBRM and a PLAN is one-to-many, the relationship between a DBRM and a PACKAGE is always one-to-one.**

As the number of DBRMs bound to a PLAN increases, binding the DBRM into a PLAN is not recommended. If we need to precompile and bind a new program or one of the programs changes and it is to be precompiled and bound again, all the programs (not just the modified/added program) will be rebound into the PLAN again. Then the BIND process could take hours to complete

**On the other hand, if a DBRM is bound to a PACKAGE and if the program is modified, only that PACKAGE would have to be rebound.**

A collection is simply a way of grouping PACKAGEs into meaningful groups. You could use COLLECTIONs to separate programs for different application areas, such as payroll and inventory. Another use might be to have customized set of BIND parameters associated with different COLLECTIONs

**At run time, the load module starts up and eventually hits a paragraph containing a CALL to DB2. Then the COLLECTIONs named in the PLAN are searched for the PACKAGE with the same name and consistency token. If you don't find it anywhere in DB2, you get an -805 error. If you're using the older technique of binding DBRMs directly into PLANs, then an unsuccessful search will result in an -818 error code.**

**Q. Explain errors -922 and -551**

**ANS.** -922 is authorisation failure. connection to DB2 failed due to authority for user or plan.

-551 auth id doesnot have the privilege to perform operation on Object

**HCL(01/11/2014)**

**Q. what is -811 error**

**ANS.**

Sql query returned more than one row. Cursor is needed

**UHG(15/03/2014)**

**Q. What is -803 abend**

**ANS.**

Unique index violation. The inserted record has an existing primary key in the database.

**IMS DB**

**IGATE (20/09/2014)**

**Q. Explain PCB and PSB in detail?**

**ANS.**

***PCB*** *(Program Communication Block) refers to one data base.*

***PROCOPT*** *parameter specifies the program’s processing options(generally in PCB)*

***PCBs must be defined in the Linkage Section***

*Linkage Section definition of a PCB is called a ‘****PCB Mask’***

PCB stands for Program Communication Block. PCB Mask is the second parameter used in the DL/I call. It is declared in the linkage section. Given below is the syntax of a PCB Mask:

01 PCB-NAME.

05 DBD-NAME PIC X(8).

05 SEG-LEVEL PIC XX.

05 STATUS-CODE PIC XX.

05 PROC-OPTIONS PIC X(4).

05 RESERVED-DLI PIC S9(5).

05 SEG-NAME PIC X(8).

05 LENGTH-FB-KEY PIC S9(5).

05 NUMB-SENS-SEGS PIC S9(5).

05 KEY-FB-AREA PIC X(n).

Here are the key points to note:

* For each database, the DL/I maintains an area of storage that is known as the program communication block. It stores the information about the database that are accessed inside the application programs.
* The ENTRY statement creates a connection between the PCB masks in the Linkage Section and the PCBs within the program’s PSB. The PCB masks used in a DL/I call tells which database to use for operation.
* You can assume this is similar to specifying a file name in a COBOL READ statement or a record name in a COBOL write statement. No SELECT, ASSIGN, OPEN, or CLOSE statements are required.
* After each DL/I call, the DL/I stores a status code in the PCB and the program can use that code to determine whether the call succeeded or failed.

PCB Name

Points to note:

* PCB Name is the name of the area which refers to the entire structure of the PCB fields.
* PCB Name is used in program statements.
* PCB Name is not a field in the PCB.

DBD Name

Points to note:

* DBD name contains the character data. It is eight bytes long.
* The first field in the PCB is the name of the database being processed and it provides the DBD name from the library of database descriptions associated with a particular database.

Segment Level

Points to note:

* Segment level is known as Segment Hierarchy Level Indicator. It contains character data and is two bytes long.
* A segment level field stores the level of the segment that was processed. When a segment is retrieved successfully, the level number of the retrieved segment is stored here.
* A segment level field never has a value greater than 15 because that is the maximum number of levels permitted in a DL/I database.

Status Code

Points to note:

* Status code field contains two bytes of character data.
* Status code contains the DL/I status code.
* Spaces are moved to the status code field when DL/I completes the processing of calls successfully.
* Non-space values indicate that the call was not successful.
* **Status code GB indicates end-of-file and status code GE indicates that the requested segment is not found.**

Proc Options

Points to note:

* Proc options are known as processing options which contain four-character data fields.
* A Processing Option field indicates what kind of processing the program is authorized to do on the database.

Reserved DL/I

Points to note:

* Reserved DL/I is known as the reserved area of the IMS. It stores four bytes binary data.
* IMS uses this area for its own internal linkage related to an application program.

Segment Name

Points to note:

* SEG Name is known as segment name feedback area. It contains 8 bytes of character data.
* The name of the segment is stored in this field after each DL/I call.

Length FB Key

Points to note:

* Length FB key is known as the length of the key feedback area. It stores four bytes of binary data.
* This field is used to report the length of the concatenated key of the lowest level segment processed during the previous call.
* It is used with the key feedback area.

Number of Sensitivity Segments

Points to note:

* Number of sensitivity segments store four bytes binary data.
* It defines to which level an application program is sensitive. It represents a count of number of segments in the logical data structure.

Key Feedback Area

Points to note:

* Key feedback area varies in length from one PCB to another.
* It contains the longest possible concatenated key that can be used with the program’s view of the database.
* After a database operation, DL/I returns the concatenated key of the lowest level segment processed in this field, and it returns the length of the key in the key length feedback area.

------------------

**Q. Explain while using FAI can we retrieve records with partial key**

**ANS.**

**Yes we can, usw contains (CO)**

**JCL**

**IGATE (20/09/2014)**

**Q. Explain declaration of Temporary Dataset**

**ANS**

//SORTOUT DD DSN=&&TEMP2,DISP=(NEW,PASS)

**Q. If a job fails to run in the middle and one of the steps have temp dataset then what is restart procedure**

**Ans.**

Restart the job from the beginning

**Q. Declaration of sortworkfile**

**ANS.**

//SORTWK01 DD DISP=(NEW,DELETE),

// UNIT=SYSDA,SPACE=(CYL,(3335,3335))

//SORTWK01 DD UNIT=DISK,SPACE=(CYL,(20,5),RLSE)

**HCL(01/11/2014)**

**Q. Time card for Long running Jobs?**

**ANS.**

TIME=NOLIMIT/1440/MAXIMUM

**Q. GDG options scratch and no scratch. How to restore the gdg versions deleted from the catalogue?**

**ANS.**

By using the UNIT and VOL serial parameters in the dataset DD statement.

**EMPTY/NOEMPTY**- These two parameters are mutually exclusive. EMPTY

specifies that all existing generations of the GDG

are to be uncataloguedwhenever the generations of GDG

reached the maximum limit NOEMPTY specifies that only

the oldest generation of the GDG is to be uncatalogued

if the limit is reached

**SCRATCH/NOSCRATCH**- These two parameters are mutually exclusive. SCRATCH

parameter specifies that whenever entry of the GDG is

removed from the index, it should be deleted physically

anduncatalogued. NOSCRATCH parameter specifies that

whenever entry of the GDG is removed from the index, it

should be uncatalogued, not physically deleted

**Q. How to eliminate duplicates from the file using sort**

**ANS**

//SYSIN DD \*

SORT FIELDS=(1,22,CH,A)

SUM FIELDS=NONE

/\*

**NETCRACKER(31/10/2014)**

**Q. Explain the space abends SB37, SE37, SD37**

**ANS**

**Q. Explain the conversion from Packed decimal to zone in sort**

**ANS**

//SYSIN DD \*

SORT FIELDS=COPY

OUTFIL,BUILD=(6,3,PD,TO=ZD,LENGTH=5,

9,5,PD,TO=ZD,LENGTH=9,

16,2,

18,2,

80,3,PD,TO=ZD,LENGTH=5,

57X)

/\*

**Q. Explain the difference b/w outrec and inrec usage in sort**

**ANS.**

**INREC** - Used to reformat the records before applying sort.   
**OUTREC** - Used to reformat the records after applying sort.

**CAPGEMINI(26/04/2014)**

**Q. explain creation of GDG steps**

**ANS.**

//MYJOB JOB (W234),'RAMESH'

//STEP1 EXEC PGM=IDCAMS

**//SYSIN DD \***

**DEFINE GDG(NAME(MYLIB.LIB.TEST) -**

**LIMIT(10) -**

**NOEMPTY -**

**SCRATCH)**

**/\***

//

**Project Related Questions**

**IGATE (20/09/2014)**

**NAB**

**Q. What is MQ series how oracle people communicate with mainframe?**

**ANS**.

MQSeries is an IBM software family whose components are used to tie together other software applications so that they can work together.

MQSeries is an IBM software family whose components are used to tie together other software applications so that they can work together. This type of application is often known as business integration software or middleware.

MQSeries consists of three products:

1. MQSeries Messaging, which provides the communication mechanism between applications on different platforms
2. MQSeries Integrator, which centralizes and applies business operations rules
3. MQSeries Workflow, which enables the capture, visualization, and automation of business processes

The point of business integration is to connect different computer systems, diverse geographical locations, and dissimilar IT infrastructures so that a seamless operation can be run. IBM's MQSeries supplies communications between applications, or between users and a set of applications on dissimilar systems. It has grown in popularity as applications are made available over the Internet because of its support of over 35 platforms and its ability to integrate disparate automation systems.

An additional helpful feature is that its messaging scheme requires the application that receives the the message to confirm receipt. If no confirmation materializes, the message is re-sent by the MQSeries.

IBM asserts that MQSeries can connect any two commercial systems that are in current business use.

TCS

1. Types of Performs
2. Importance of inspect
3. How do you restart if job abends in a proc step
4. How do you create GDG and explain each parameters in it
5. Explain on -904 error
6. Explain about static and dynamic call of COBOL
7. How GDG created in the same JCL is referenced in later steps
8. WHAT are different space abends and explain